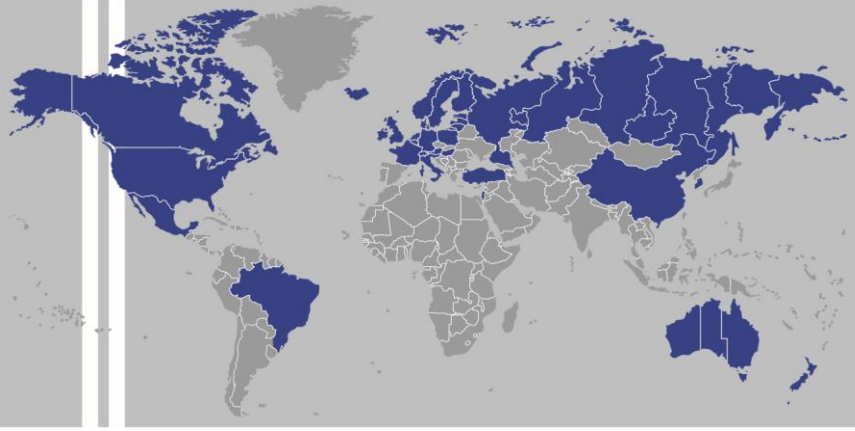

dunkermotoren
advanced motion solutions

DME 230

Part Program
for servo drives series
- BN6773

Typ:
DME 230x4-I/O

Part No:
81703.00101

Publication Ref: 160616

Dunkermotoren GmbH | Allmendstraße 11 | D-79848 Bonndorf/ Schwarzwald
Phone +49 (0) 7703 930-0 | Fax +49 (0) 7703 930-210/ 212 | info@dunkermotoren.com

AMETEK®
PRECISION MOTION CONTROL

TrioDrive D/xS / MidiDrive D/xS
TrioDrive D / MidiDrive D
MaxiDrive

Digital Servo Drives
for Direct Mains Connection

Part Program (Positioning Control)

Operating Instructions 6710.231, V 8.5

These operating instructions apply to
- TrioDrive D/xS servo drives, BN 6755 to BN 6758 with option B2 or higher
- MidiDrive D/xS servo drives, BN 6745 to BN 6749 with option B2 or higher
- TrioDrive D servo drives, BN 6751 to BN 6753 with option B2 or higher
- MidiDrive D servo drives, BN 6741 to BN 6743 with option B2 or higher
- MaxiDrive servo drives, BN 6721 to BN 6725

These operating instructions apply together with
- Operating Instructions 6710.201 (Functions and Parameters)
- Operating Instructions 6755.202, 6745.202, 6750.202, 6740.202, or 6710.202 (Connection and Commissioning)
- Operating Instructions 6710.207 (SPP Windows Command and Commissioning Software)

ESR Pollmeier GmbH
Lindenstr. 20

64372 Ober-Ramstadt

Federal Republic of Germany

Phone +49 6167 9306-0
Fax +49 6167 9306-77

E-Mail info@esr-pollmeier.de
www.esr-pollmeier.de

These operating instructions have been prepared with care. However, ESR Pollmeier GmbH can accept no liability for any errors in these operating instructions or possible consequences. Neither can any liability be accepted for direct or indirect damage resulting from abuse of the device.

The relevant regulations concerning safety technology and electromagnetic compatibility must be complied with when using the device.

Subject to alteration.

Contents

Please, also refer to the index at the end of this document.

1 Preliminary Remarks

1.1 About this Description

These Operating Instructions 6710.231 explain the functions of the part program which can be run by the integrated positioning control of the TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, and MidiDrive D servo drives with option B2 or higher and the MaxiDrive servo drive independent of a higher-level controller.

These operating instructions apply together with

- Operating Instructions “Functions and Parameters” of the servo drives (included in the scope of delivery)
  – Operating Instructions 6710.201
- Operating Instructions “Connection and Commissioning” of the servo drive (included in the scope of delivery)
  – Operating Instructions 6755.202 (TrioDrive D/xS),
  – Operating Instructions 6745.202 (MidiDrive D/xS),
  – Operating Instructions 6750.202 (TrioDrive D),
  – Operating Instructions 6740.202 (MidiDrive D) or
  – Operating Instructions 6710.202 (MaxiDrive)
- Operating Instructions “SPP Windows Command and Commissioning Software” (supplied with the optional SPP Windows command and commissioning software)
  – Operating Instructions 6710.207

A PC with the SPP Windows command and commissioning software (full version) is required for creating and editing part programs. Please, make sure that this requirement is met and the above-mentioned operating instructions are available.

2 Safety Instructions

In any case, observe the safety instructions as well as the warnings and hints in the margins of the corresponding operating instructions “Connection and Commissioning” (6755.202, 6745.202, 6750.202, 6740.202, or 6710.202) and all other operating instructions.

Commissioning and parameterization of the servo drives may trigger drive movements. If drive system and/or machine have not been set up and secured properly, health and life of persons may be endangered.

Therefore, working with the drive system is prohibited until the requirements of the machine directive have been met.

In bus systems (CANopen®, EtherCAT, Ethernet, Interbus, Profibus, etc.), a bus participant can be influenced invisibly from outside. This can lead to an unexpected (uncontrollable) system behavior. Do not put the bus into operation unless you have made sure that all participants are properly connected and configured.

2.1 Type of Instructions

Please, observe the warnings and hints in the margin:

- **Danger** to heath and life due to electrical shock or motion of the drive.
- **Caution**: Noncompliance violates the safety regulations or statutory provisions and can lead to personal injury or material damage.
- **Check**: Prior to commissioning and in case of failures or problems, check these items first.
- **Tip**, useful hint.

3 Overview

In device operating modes **program automatic** and **program single step**, part programs can be used for running motion sequences by the integrated positioning control of the servo drives independent of a higher-level controller.

The software of a higher-level controller and part programs can communicate and synchronize their sequences via special commands or digital inputs/outputs.

A part program consists of a sequence of blocks (also called program commands) used for defining the motions. Each of these blocks is of a defined block type with data fields (selection fields or numerical fields) that can be edited.

In these operating instructions, the blocks or block types as well as the corresponding variables are printed in italics. For explanations regarding the different block types, please see section Part Program Block Types 11.

A part program may look as follows:

```
Block   0: Label  100
Block   1: Wait until I  1 equals [xxxx xxx1]
Block   2: Go to X reference point
Block   3: X  F +5000.0
Block   4: Label  110
Block   5: X +100.000 A  F% 100.0  A  0.0 [x] after pos.
Block   6: Label  120
Block   7: X  +10.000 I  F%  20.0  A  1.0 [±] after pos.
Block   8: From label  120 repeat  4 times
Block   9: Jump to label  110 if I  1 equals [xxxx xx1x]
Block  10: Program end
```

All major events of starting and stopping part programs are described by the part program state machine. See section Part Program State Machine 8 for more detailed information.

Part programs can be created and edited using the SPP Windows software running on PCs. This software can also be used for accessing the program variables and for setting the machine data of the servo drives.

Prerequisite for a proper operation of the software is a correct parameterization of the servo drive. For that, the machine data, e. g. for adjusting the controller, must have been transmitted.

4 Part Program State Machine

The part program state machine describes all key processes for starting and stopping part programs. It is controlled and queried via axis control word and axis status word.

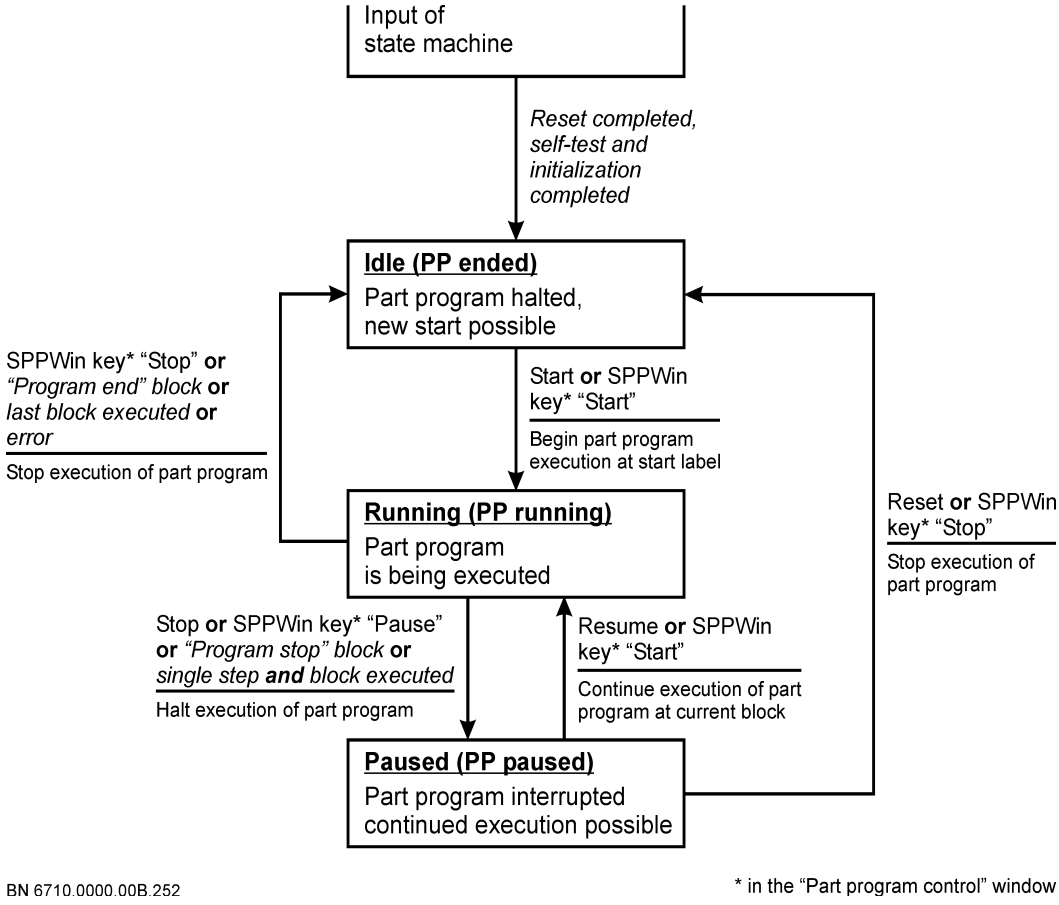

Figure 1: Part Program State Machine

The part program control is always in a defined state displayed via two bits in the axis control word. The following table shows the coding of the states.

Coding of states of the part program state machine										
State name	**Axis Status Word** (highest-value byte)									
	Bit	15	14	13	12	11	10	9	8	Typical Values
Idle (out of operation)		1	0	–	–	–	–	–	–	80_{hex}, $B0_{hex}$, …
Running		0	1	–	–	–	–	–	–	40_{hex}, 70_{hex}, …
Stopped		0	0	–	–	–	–	–	–	00_{hex}, 30_{hex}, …

– = Bit assigned to other functions

Bit names: bit 15 = "Idle"
bit 14 = "Running"

For a complete overview of the individual bits in the axis status word, see Operating Instructions 6710.201 (Functions and Parameters).

In case of an event, the program state machine changes between two states (state transition). If required, an action is carried out during the change.

An event can be the setting of a control command or an internal event. Control commands can be set via the axis control word or the enable input. The following table shows the coding of the control commands.

Control commands are only effective in the device operating modes of the program mode.

Coding of the control commands of the part program state machine									
Control command name	**Axis Control Word** (highest-value byte)								
	Bit	15	14	13	12	11	10	9	8	Typical values
Start		x	1	–	–	–	–	–	–	40_{hex}, $C0_{hex}$, …
Stop		0	0	–	–	–	–	–	–	00_{hex}, 30_{hex}, …
Resume		0	1	–	–	–	–	–	–	40_{hex}, 70_{hex}, …
Reset		1	0	–	–	–	–	–	–	80_{hex}, $B0_{hex}$, …

x = Bit may be 1 or 0

– = Bit assigned to other functions

Bit names: bit 15 = “Reset”
bit 14 = “Start”

For a complete overview of the individual bits in the axis status word, see Operating Instructions 6710.201 (Functions and Parameters).

For setting control commands via the enable input, the enable selection code in the machine data of the axis control must be set accordingly.

Information on state machines can be found in Appendix A 40.

4.1 Starting and Stopping the Part Program

The part program can be started and stopped via the enable input, with SPP Windows, or from a higher-level controller using the axis control word or the axis status word.

- Starting and stopping via the enable input

  For starting and stopping the part program via the enable input, select the corresponding enable action in window “Operate / Parameterization / Axis data” of the SPP Windows software. With “PPS-IDLE / Start part program”, the part program is re-started from label 9999 on. With “Stop part program / Resume PP + start PP”, the part program is resumed at the position it was stopped.

- Starting and stopping with SPP Windows

  In the SPP Windows command and commissioning software, the part pro-

gram is controlled via window “Part program control” (“Operate / Part program” or button “Part Prog.” in the device control window).

The enable action in window “Operate / Parameterization / Axis data” must not be set to one of the above-mentioned values related to the part program.

For that, any type of start label can be set.

- Starting and stopping with the axis control word

  In this case, the part program is started or stopped directly e. g. from a higher-level controller by coding the respective bits correspondingly (see table “Coding of the control commands” in the previous section).

  In this case, the *program start label* variable is used as start label.

5 Part Program Block Types

This section describes all block types that can be used in part programs.

One block in the part program corresponds to one line. The block is always of a certain type, the block type. The part program editor of the SPP Windows command and commissioning software supports the user when creating the part program and ensures that only blocks of a permitted block type are entered (for operating the SPP Windows software, see Operating Instructions 6710.207). In the editor, the consecutive blocks of the part program appear as lines.

One line in the part program corresponds to one block. For each line, the user can select a block type. In addition to the words and characters typical for each block type, the line includes data fields that can be edited (underlined in these operating instructions). These fields may be:

- Selection fields

  For each field, a value can be selected from a variety of defined values. The values are usually displayed as a character or a short word. Examples: axis designation with values “X”, “Y”, “Z”, “U” (for the servo drive, only value “X” is permitted) or selection of the point of time of actuating an output before or after positioning in type position blocks.

- Numerical fields

  These fields can be used for entering numbers. Each numerical field (e. g. target position, target velocity, or input byte number) has an individual value range determining the number of digits and display with or without sign.

  The value can be set in two ways:

  – by direct specification of a number (e. g. position +10,000) as value of the numerical field (standard)

  – by indirect specification via a part program variable which permits the external specification of values for a numerical field. Reading and writing access to the part program variables is possible via both the part program as well as the communication interfaces. Part program variables can be used e. g. for parameterizing motion sequences of a part program from a PC.

    A part program variable is displayed in a field by a number with prefix “@” (e. g. “@23”). 255 part program variables `@1` to `@255` can be used.

Function	Block type designation	Example of display
	Empty (NOP, no function)	–
D I V	*Position*	X -760.255 A F% 100.0 O _1.4 [1] after pos.
D V	*Feed*	X F +5000.0
I V	*M function*	O _1 [xx0± 1110]
P	*Label*	Label 4310
P	*Jump to label*	Jump to label 4310
P V	*Repeat*	From label 4310 repeat 13 times (counter=@01)
P I V	*Jump on input*	Jump to label 4310 if I_9 equals [xx0x x011]
P V	*Wait*	Wait 120.0 seconds
P I V	*Wait for input*	Wait until I_1 equals [xx0x x011]
P	*Subroutine call*	Call subroutine at label 4310
P	*Return from subroutine*	Return from subroutine
D	*Go to reference point*	Go to X reference point
P I V	*Jump destination for input value*	Jump destination for input value at I_9: [xx0x x011]
P I V	*Wait for input value, jump*	Wait for input value at I_9 and jump
P	*Program halt*	Program end / stop
D I V	*Object access*	Write: index 5EF0 subindex 0 value 30
M V	*Mathematical operation*	@51 = @50 * 1000
P M V	*Jump dependent on comparison*	Jump to label 7500 if @51 == 20000
D	*Select axis operating mode*	Axis operating mode X: velocity mode
D	*Select axis state*	Axis state X: operation enabled
D P I V	*Fast axis start*	Start positioning axis X if I 1.0 equals 1
I V	*Logical operation*	O 15 = _ I 14 & ~ I _2
I V	*Logical operation with constant*	O 15 = _ I _2 & 11110000
D V	*Velocity profile individual definition*	Velocity profile axis X 3000 VU ramp = 100 ms duration = 100 ms
D V	*Velocity profile data group*	Velocity profile axis X from variable 1
D V	*Velocity profile data group with auto increment*	Velocity profile axis X from variable @1 with autoincrement
P	*Velocity profile end*	Velocity profile axis X end
D V	*Velocity profile spindle positioning*	Velocity profile spindle positioning axis X target = 1000 PU
D	*Velocity profile spindle positioning end*	Velocity profile spindle positioning axis X end

D = influences drive
P = influences the program run
I = queries or changes I/Os and flags
M = mathematical function
V = variables as operands possible

The table gives an overview of all block types. Each block type is described with its designation and an example of the representation in the part program editor. Additionally, the functions affected by the block type are stated (see explanations at the end of the table).

The following sections describe the block types in detail. Additionally, the variables of the axis and device functions used by the respective block type and the error messages (axis fault codes) that may occur in the block sequence are listed.

5.1 Empty (NOP, no function)

Example: –

This block type serves as a dummy and does not have an effect on the execution of the part program. In case of time-critical applications, please remember that even a type *empty* block requires a minimum processing time.

A type *empty* block can be used as stopgap for temporarily not required blocks or for a better optical delimitation of program sections used in the editor. A comment of up to 16 characters can be entered in a type *empty* block using the SPP Windows command and commissioning software (this comment is stored in the device together with the part program).

Used variables:

- none

Possible axis error codes:

- none

5.2 Position

Example: `X –760.255 A F% 100.0 A 1.4 [1] after pos.`

This block type serves for actual positioning. The positioning can be implemented as an absolute (A) or an incremental dimension (I) (incremental or relative positioning).

Prerequisite: a target velocity has to be set with at type *feed* block before the first *position* block.

The first character identifies the axis and has to be “X” for TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, MidiDrive D, and MaxiDrive as only one axis exists. This is followed by the position setpoint in position units. This setpoint in conjunction with the subsequent character for the measuring system (A or I) states the position to be approached during the execution of the part program.

“F%” specifies a factor (in percent) determining the velocity of this positioning. The value refers to the velocity specified before with a type *feed* block. If a part program variable is used for this factor, it is stated in 0.1% (1000 = 100%).

For positioning, the target velocity must not be preset by the analog input set-

point. Otherwise, it would be set by two different sources (part program interpreter and analog input read out routine), which will lead to undefined behavior. Therefore, machine data target velocity source has to be set to variable *target velocity digital*.

With the other fields, a digital output can be actuated. The two numbers after the second “O” specify the number of digital output O X.Y with byte number X (2 digits) and bit number Y (1 digit). The character in the square brackets specifies the type of actuation of the output:

x no actuation

0 set output to logical “0”

1 set output to logical “1”

± invert the state of the output

n negative pulse (set output briefly to logical “0”, then immediately to “1” – should not be used as the pulse width is not defined)

p positive pulse (set output briefly to logical “1”, then immediately to “0” – should not be used as the pulse width is not defined)

The last field specifies whether the actuation of the output should take place before or after positioning.

A type *position* block changes the axis operating mode to **target position mode**.

Used variables:

- *axis operating mode, axis control word, axis status word, target position, target velocity, digital outputs*

Possible axis error codes:

- 6250_{hex} axis access fault
- 6270_{hex} internal I/O access fault

5.3 Feed

Example: `X F +5000.0`

This block type specifies the velocity for positioning the selected axis in velocity units. In type *position* blocks, this velocity can be changed by entering a percentage factor.

For positioning, the target velocity must not be set by the analog setpoint output. Machine data target velocity source has to be set to variable *target velocity (digital)*.

Used variables:

- *target velocity*

Possible axis error codes:

- 6250_{hex} axis access fault

5.4 Machine Operation

Example: `O 1 [xx0± 1110]`

This block type can be used for actuating several digital outputs within a single output byte simultaneously. The first digit after “O” specifies the byte number (2 digits). The 8 characters in the square brackets specify the type of actuation for each output of the output byte (O X.7 far left, O X.0 far right):

x no actuation

0 set output to logical “0”

1 set output to logical “1”

± invert the state of the output

n negative pulse (set output briefly to logical “0”, then immediately to “1” – should not be used as the pulse width is not defined)

p positive pulse (set output briefly to logical “1”, then immediately to “0” – should not be used as the pulse width is not defined)

Used variables:

- *digital outputs*

Possible axis error codes:

- 6270_{hex} internal I/O access fault

5.5 Label

Example: `Label 4310`

This block type can be used for marking specific positions in the program by means of a number with up to four digits, called *label*. With a *label*, the start of part programs or sections of part programs can be marked. Additionally, *labels* are required for marking part program sections to be jumped to from type *jump to label*, *repeat*, *jump on input*, or *subroutine call* blocks.

A part program cannot be started unless a type *label* block with a value equal to the one in variable *program start label* exists at its beginning.

Label 9999 has a special meaning in case the part program is started via the enable input (enable selection code in the machine data of the axis control system must have been set accordingly): in this case, the program is started at *label* 9999 which has to be present. The program start label does not have an effect.

Used variables:

- none

Possible axis error codes:

- none

5.6 Jump to Label

Example: `Jump to label 4310`

If a block of this type is reached during the execution of the part program, the program is not continued at the block with the next consecutive number but at the block with the specified *label*.

Used variables:

- none

Possible axis error codes:

- 6220_{hex} label not found

5.7 Repeat

Example: `From label 4310 repeat 13 times (counter=@1)`

With iterations, sections of a program can be repeated cyclically up to 999 times. Prerequisite for that is that the program section to be repeated starts with a *label* and ends with a type *repeat* block containing the same label.

Iterations can be nested in any way, however, the number of simultaneously active repeat loops is limited to 10.

During the execution of the part program, the current counter status for the iterations is written in the last field of the block. If, as in the above example, a program variable is entered in this field, the counter status can be queried, e. g. from a PC, via a communication interface. This type of counter variable must not be used by several nested loops simultaneously as this will lead to conflicts.

Used variables:

- none

Possible axis error codes:

- 6220_{hex} label not found
- 6233_{hex} repeat list overflow fault

5.8 Jump on Input

Example: `Jump to label 4310 if I 9 equals [xx0x x011]`

This block type can be used for conditional branching of the program due to an external condition, i. e. the current state of a digital input byte.

The number after “label” (4 digits) specifies the jump destination, i. e. the block at

which the execution of the part program will be continued when the condition is fulfilled. If the condition is not fulfilled, the program continues with the block following the *jump on input* type block.

The number after “I” specifies the number of the digital input byte (2 digits).

The jump condition is fulfilled when the state of all 8 inputs of the input byte corresponds to the mask in the square brackets (I X.7 far left, I X.0 far right):

x input is ignored (i.e. input is always treated as matching)

0 input must not be activated

1 input must be activated (+24 V must be applied)

Used variables:

- *digital inputs*

Possible axis error codes:

- 6220_{hex} label not found
- 6270_{hex} internal I/O access fault

5.9 Wait

Example: `Wait 120.0 seconds`

This block type pauses the program execution for the specified time. Once the time has expired, the program execution continues at the next block. The maximum waiting time is 6553.5 seconds (almost 1 h 50 min.).

If a part program variable is used for the waiting time, it is specified in tenths of a second (“10” = 1 second).

Used variables:

- none

Possible axis error codes:

- 6280_{hex} internal timer fault

5.10 Wait for Input

Example: `Wait until I 1 equals [xx0x x011]`

This block type pauses the program execution until the state of all 8 inputs of the input byte corresponds to the mask in the square brackets (I X.7 far left, I X.0 far right):

x input is ignored (i. e. input is always treated as matching)

0 input must not be activated

1 input must be activated (+24 V must be applied)

n falling edge at the input (from logical “1” to “0”)S

p rising edge at the input (from logical “0” to “1”)

The number after “I” (2 digits) specifies the number of the digital input byte.

If this matches, the program execution is continued at the block with the next consecutive number.

Used variables:

- *digital inputs*

Possible axis error codes:

- 6270_{hex} internal I/O access fault

5.11 Subroutine Call

Example: `Call subroutine at label 4310`

With this block type, the same program section (subroutine) can be run from various positions in the part program. For that, the desired processing has to be written in a program section beginning with the specified label and ending with a type *return from subroutine* block.

During program execution, when the type *subroutine call* block is reached, the program always jumps to a block with the specified label, the beginning of the subroutine. If block *return from subroutine* is found there when the subroutine has been executed, the program will continue with the block following the subroutine call.

Calling another subroutine from a subroutine is possible (nesting). A maximum of 10 type *subroutine call* blocks can be executed without a *return from subroutine*. If this limit is exceeded, a stack overflow fault occurs.

Used variables:

- none

Possible axis error codes:

- 6220_{hex} label not found
- 6231_{hex} subroutine stack overflow fault

5.12 Return from Subroutine

Example: `Return from subroutine`

The meaning of this block type has been described in connection with the subroutine call.

Used variables:

- none

Possible axis error codes:

- 6232_{hex} subroutine stack underflow fault

5.13 Go to Reference Point

Example: `Go to X reference point`

This block type can be used for driving the specified axis to the mechanically defined zero point and setting the actual position to zero. The way the zero point is reached is determined via the *homing selection code* and the *homing velocity* of the machine data.

A type *go to reference point* block changes the axis operating mode to **homing mode**.

Used variables:

- *axis operating mode, axis control word, axis status word, homing selection code, homing velocity, home offset*

Possible axis error codes:

- 6250_{hex} axis access fault
- 6255_{hex} homing fault

5.14 Jump Destination for Input Value

Example: `Jump destination for input value at I 9: [xx0x x011]`

Similar to the *labels*, this block type identifies certain positions in the program which can be jumped to depending on the status of the digital input byte using a type *wait for input value and jump* block.

In block type *jump destination for input value*, the number after "I" (2 digits) states the number of the digital input byte and the square brackets contain a mask for the status of all 8 inputs of the input bytes (I X.7 far left, I X.0 far right):

x input is ignored (i.e. input is always treated as matching)

0 input must not be activated

1 input must be activated (+24 V must be applied)

In a type *wait for input value and jump* block, the state of the digital input byte indicated there is continuously compared with the masks of the type *jump to destination* for input values block containing the same input number byte. If this input byte matches a mask, the program is continued at the corresponding block.

If a type *jump destination for input value* block is not reached as a result of a jump from a type *wait for input value and jump* block but on completion of the previous block, this block is no longer of significance and the part program is continued at the following block.

Used variables:

- none

Possible axis error codes:

- none

5.15 Wait for Input Value and Jump

Example: `Wait for input value at I_9 and jump`

During the execution of a block of this type, the digital input byte stated after “I” is checked cyclically. This check is continued until it matches the mask specified in a type jump *destination for input value* block and has the same input byte number. The program execution is continued at this block.

In contrast to a jump to a label during which the jump destination is reached by programming the corresponding blocks of the part program, the jump destination here is selected by an external controller via the digital inputs.

Used variables:

- *digital inputs*

Possible axis error codes:

- 6270_{hex} internal I/O access fault

5.16 Program Halt

Example: `Program end`

Blocks of this type end the execution of the part program. After having reached this block, the program has to be restarted, if required. The field after “Program” defines the way of ending. For that, values “end” and “stop” can be selected:

- After “Program end”, the status of the part program state machine is “idle”. The part program can only be re-started beginning with the start label (for further details, see section Overview 7).
- “Program stop” pauses the program execution temporarily, similar to the execution of a block in the **program single step** operating mode. After that, the status of the part program state machine is stopped. The part program can be continued using the resume command. This block type can be used e. g. for synchronizing part program sections with higher-level PLC or for test purposes. In operating mode **program single step**, this block type does not have an effect as the program is stopped after each block, anyway (for further details, see section Overview 7).

The execution of the part program is also ended if it exceeds the last block of the part program memory. In the part program state machine, this will always result in idle state.

Used variables:

- none

Possible axis error codes:

- none

5.17 Object Access

Example: `Write: index 5EF0 subindex 0 value 30`

With this block type, variables of the axis and device functions can be accessed directly using the part program. These can be all variables listed in section Variable Descriptions of Operating Instructions 6710.201 (Functions and Parameters).

Object access is particularly useful for changing variables for which a specific block type does not exist. For example, changing the machine data using the part program is usually not possible (e. g. *position control loop Kp* variable). With block type *object access*, this variable can be changed.

The first word (“read” or “write”) indicates whether the variable with the stated index (4 digits, hexadecimal) and subindex (3 digits, decimal) is to be read or written.

The meaning of the number stated after “value” (10 digits, decimal, with sign, corresponding to 4 bytes integer32) depends on whether it is to be read or written:

- In the case of “write”, the number is transferred to the variable. The stated number has to be within the value range of the variable. In above example, *position control loop Kp* (index $5EF0_{hex}$, subindex 0) is to be set to 30. This number must not exceed the range 0 to 32767.
- In the case of “read”, it is useful to insert part program variables in the field after “value” (for that see section Parameters of Part Program Operation 37) to be able to continue with that at another position of the part program. During that, the part program variable is overwritten with the value of the axis or device function variable stated by index and subindex.

  Storage of the value directly in the part program after “value” is possible, e. g. for test purposes.

Axis status word and axis control word can be accessed via digital inputs and outputs, see section Access to Axis Control Word and Axis Status Word via Digital Inputs and Outputs 31.

Used variables:

- depending on selected index and subindex

Possible axis error codes:

- 6235_{hex} object access fault
- 6236_{hex} object does not exist
- 6237_{hex} object too long

5.18 Mathematical Operation

Example: `@51 = @50 * 1000`

The *mathematical operation* can be used for simple calculations. For that, constants or variables can be used. The result is stored to the left of the equals sign. The following calculation operations are possible:

+ add

– subtract

* multiply

/ divide (integer part only, e.g. `@10 = 27 / 5` results in value 5 in variable `@10`)

% remainder of the division („modulo“; e.g. `@11 = 27 % 5` results in value 2 in variable `@11`)

Calculations are executed internally in sign-oriented 32-bit integer arithmetic. Overflows and underflows are not displayed.

Used variables:

- none

Possible axis error codes:

- 6275_{hex} division by zero (in the case of “/ division” only; the result of “Value % 0” is “Value”)

5.19 Jump Dependent on Comparison

Example: `Jump to label 7500 if @51 == 20000`

This block type permits conditional branching of the program execution due to an internal condition, i. e. the current contents of part program variables.

The number after “label” (4 digits) defines the jump destination, i. e. the block where the execution of the part program is continued if the condition is fulfilled. If the condition is not fulfilled, the program continues with the block following the type *jump dependent on comparison* block.

The contents of two variables can be compared or a single variable can be compared with a constant. The following comparisons are available:

== equal to

!= not equal to

<= smaller than or equal to

< smaller than

> greater than

>= greater than or equal to

Used variables:

- none

Possible axis error codes:

- 6220_{hex} label not found

5.20 Select Axis Operating Mode

Example: `Axis operating mode X: Velocity mode`

This block type can be used for selecting a specific axis operating mode.

The first character defines the axis and has to be “X” for TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, MidiDrive D, and MaxiDrive as only one axis exists for these devices. This is followed by the desired axis operating mode.

The operating mode can only be changed in states

- switch on disabled
- ready to switch on
- switched on

of the axis state machine. Prior to a change of the operating mode, a change into one of these states using block type *select axis state* may be required.

A type *position* block always changes the axis operating mode to **profile position mode**; block type *go to reference point* changes the axis operating mode to **homing mode**.

Used variables:

- *axis operating mode*

Possible axis error codes:

- 6257_{hex} change of operating mode not permitted

5.21 Select Axis State

Example: `Axis state X: Operation enabled`

This block type can be used for selecting a specific state of the axis state machine.

The first character defines the axis and has to be “X” for TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, MidiDrive D, and MaxiDrive as only one axis exists for these devices. This is followed by the desired axis state.

Depending on the initial state, the corresponding control commands are sent to the axis state machine on the shortest route until the desired target state is reached. A change to the switch on disabled state is always carried out via control command quick stop. If the axis was in state operation enabled before, the drive is decelerated using a quick stop ramp. For using a braking ramp instead, the state has to be changed to switched on or ready to switch on, before.

Used variables:

- *axis status word, axis control word*

Possible axis error codes:

- none

5.22 Fast Axis Start

Example: `Start positioning axis X if I1.0 equals 1`

This block is intended for applications in which the response times to an enable signal have to be as short as possible. In contrast to block type *position*, the prerequisites for starting the positioning are not created automatically but have to be prepared by the user or the part program.

The first field defines input I1.x supplying the enable signal, the second field states the level to be waited for (“0” or “1”).

Prerequisites:

- axis operating mode **profile position mode**
- sub mode **block mode** (bit 5 in the axis control word = 0)
- axis state machine in operation enabled state
- target velocity set (via the type *feed* block)
- target position set (in variable *target position*)

Part program fragment as an example:

```
Block  10: ...
Block  11: Axis state X: Switch on disabled
Block  12: Axis operating mode X: Profile position mode
Block  13: A 10 [x100 xxxx]
Block  14: Axis state X: Operation enabled
Block  15: X  F +3000.0
Block  16: Write: index 607a  subindex  0  value        @1
Block  17: Start positioning axis X if I1.0 equals 1
Block  18: ...
```

Blocks 11 to 13 serve for switching to operating mode **profile position mode** and state operation enabled. Bits 0 .. 7 of the axis control word are accessed via O 10 (block 14); sub-modes **block mode** and **absolute positioning** are set. Block type *feed* (block 15) sets the target velocity for the positioning to 3,000 VU. Object access in block 16 writes the value of program variable `@1` to the target position variable (index 607A $_{hex}$). In block 17, the part program waits for level 1 at input I1.0 in order to start positioning.

Used variables:

- *axis control word, digital inputs*

Possible axis error codes:

- 6250_{hex} axis access fault
- 6270_{hex} internal I/O access fault

5.23 Logical Operation

Example: `0 15 = _ I 14 & ~ I 2`

Outputs the logical operation of two inputs at one output. The following logical operations are possible:

`&` and

`|` or

`^` exclusive or (either or)

The inputs can be inverted individually with sign ~.

Used variables:

- *digital inputs, digital outputs*

Possible axis error codes:

- 6270_{hex} internal I/O access fault

5.24 Logical Operation with Constant

Example: `0 15 = _ I 2 & 11110000`

Specifies the logical operation of an input with an 8-bit binary constant to an output. The following logical operations are possible:

`&` and

`|` or

`^` exclusive or (either or)

The input can be inverted with sign ~.

Used variables:

- *digital inputs, digital outputs*

Possible axis error codes:

- 6270_{hex} internal I/O access fault

5.25 Velocity Profile Initialize

Example: `Velocity profile axis X initialize`

This block has to be the first one at the beginning of a velocity profile sequence. From firmware version 7.9 on, it should be used instead of block `velocity profile axis X 0 VU ramp = 0 ms duration = 0 ms`.

Used variables:

- none

Possible axis error codes:

- none

5.26 Velocity Profile End

Example: `Velocity profile axis X end`

After initiation of a profile sequence (*velocity profile* block), the part program continues with the next part program block. The duration defined before is not observed unless it is another velocity profile block.

Due to that, further part program blocks can be integrated into the velocity profile without affecting the course of the profile (e. g. digital input and output).

After the last velocity profile block has been terminated, the profile should be ended with a type *velocity profile end* block.

Used variables:

- none

Possible axis error codes:

- none

5.27 Velocity Profile (Block Type)

This block type can be used for adding a new section to the velocity profile. For the specification of the velocity profile data (target velocity, ramp time and duration of each profile section), there are three possibilities described in the following sub-sections.

Before the first section, the velocity profile has to be initialized with a type *velocity profile initialize* block.

The following applies to all type *velocity profile* blocks:

The first character defines the axis and has to be “X” for TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, MidiDrive D, and MaxiDrive as only one axis exists.

- The target velocity is stated in velocity units (VU).
- The ramp time specifies the duration of acceleration or deceleration. For the ramp machine data (acceleration or deceleration time related to the ramps re-

ference velocity), it is converted internally using the velocity profile quantities (ramp time related to the difference between target velocity and actual velocity).

- The specification of the duration of the section refers to the entire section, ramp times included. This time always has to be higher than or equal to the ramp time.

For a description of operating mode **velocity profile**, see section Velocity Profile 32.

Used variables:

- *target velocity, acceleration time, deceleration time, ramps reference velocity*

Possible axis error codes:

- none

5.27.1 Velocity Profile Individual Data

Example: `Velocity profile axis X 3000 GE ramp = 100 ms duration = 100 ms`

Target velocity, ramp time, and duration of each section are specified individually using this block type. The specification is made either in the form of numerical values in the part program block (as in the example above) or via variables containing the individual values.

5.27.2 Velocity Profile Data Group

Example: `Velocity profile axis X from variable 10`

This block type specifies a data group of three successive variables. The variables from the number in the last field on contain the values for target velocity, ramp time and duration (in this order). The variable number in the first variable can either be specified as a numerical value in the part program block (as in the example above) or via a variable.

5.27.3 Velocity Profile Data Group with Autoincrement

Example: `Velocity profile axis X from variable @10 with autoincrement`

With this block type, a data group consisting of three successive variables is specified. The variables include the values for target velocity, ramp time, and duration (in this order).

The variable number of the first variable has to be specified via a variable, then, the value is increased by 3. For above example, this means: variable `@10` contains the number of the first variable of the data group, after execution of the block, variable `@10` contains the number of the first variable of the next data group.

Due to that, subsequent calls of type *velocity profile data group with auto increment* blocks add successive sections to the profile using the same addressing variable. This can be done via several similar blocks or using a program loop. Before using this block type, an initialization with an empty type *velocity profile* block (individual data or data group with auto increment) has to be carried out.

5.28 Velocity Profile Spindle Positioning

Example: `Velocity profile spindle pos. axis X position = 1000 PU`

This command internally triggers command “start spindle positioning”. This corresponds to the 0→1 ramp of input spindle positioning in operating mode **velocity mode direct** or **velocity mode**. This command is terminated when the spindle position has been reached. When the spindle position has been reached, the controller stays in positioning control to maintain the position.

Final status of this command and also initial condition for the subsequent command is: spindle positioning target reached and velocity = 0.

For spindle positioning, the axis has to be operated as circular axis. The corresponding setting is carried out via variable *axis type*.

Used variables:

- *spindle positioning velocity, spindle positioning window, spindle positioning windowtime, deceleration time,* and *ramps reference velocity*

Possible axis error codes:

- none

5.29 Velocity Profile Spindle Positioning End

Example: `Velocity profile spindle pos. axis X end`

This command internally triggers command “spindle positioning stop”. This corresponds to the 0→1 ramp of input spindle positioning in operating mode **target velocity direct** or **target velocity**. This command quits positioning control. After that, the controller is in velocity control mode with target velocity 0 and ready for executing further velocity profiles.

Used variables:

- none

Possible axis error codes:

- none

5.30 Unknown Block Type

Example: `---???--- (unknown block type!!!)`

The SPP Windows editor will always display this line if it has got a part program

in its main memory containing block types of which the editor does not know how to display and edit them. This usually occurs when a part program was created with an object directory version that does know certain block types and the part program is loaded with another version that does not know these block types.

The ability of the servo drive to process certain block types depends on the firmware version in the servo drive, not on the SPP Windows version. In this case, the problem can be solved by updating the data bases of SPP Windows.

6 Access to Digital Inputs and Outputs

6.1 Hardware Inputs/Outputs, Software Inputs/Outputs

The execution of the part program can be controlled and monitored externally using the physical inputs and outputs I 1.x and O 1.x as well as the logical inputs and outputs I 9.x and O 9.x.

Examples:

```
Block 10: ...
Block 11: Wait until I1 equals [xxxx x001]
Block 12: Jump to label 60 if I9 equals [xx11 1100]
Block 13: ...

Block 22: ...
Block 23: O1 [xxxx 1000]
Block 24: X 2500 I  F% 100,0  A 9.1 [±] after pos.
Block 25: ...
```

6.2 Linked Inputs and Outputs

With TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, and MidiDrive D, the internal inputs I 15 to I 20 are linked internally with the corresponding outputs O 15 to O 20. In the part program, they are used as flags.

Examples:

```
Block 10: ...
Block 11: A16 = E1 & 00001111
Block 12: A17 = E9 ^ 00001111
Block 13: A18 = E16 & E17
Block 14: Jump to label 200 if I18 equals [0000 1100]
Block 15: ...

Block 20: ...
Block 21: A 15 [xxxx xxx1]
Block 22: ...
```

Block 21 simultaneously sets I 15.0 with which a drive function may be linked (e. g. “measure position” or “trace start”).

6.3 Access to Axis Control Word and Axis Status Word via Digital Inputs and Outputs

In the part program, the axis control word can be accessed via digital outputs O 10 and O 11, the axis status word can be accessed via digital inputs I 10 and I 11.

Example:

```
Block 10: ...
Block 11: Axis operating mode X: profile position mode
Block 12: O 10 [x100 xxx]
Block 13: Axis state X: operation enabled
Block 14: Write: index 607A subindex 0 value 3400
Block 15: Wait until I1 equals [xxxx xxx1]
Block 16: O 10 [x101 xxxx]
Block 17: ...

Block 25: ...
Block 26: Axis operating mode X: velocity mode
Block 27: Write: index 6081 subindex 0 value 3000
Block 28: Axis state X: operation enabled
Block 29: Wait until E11 equals [xxxx x1xx]
Block 30: ...
```

In this example, sub-mode **block mode** absolute is selected with the axis control word (O10) using block 12. Block 14 specifies the target position. With block 15, the program waits for the start command and with block 16, the target position is accepted by setting the corresponding bit in the axis control word.

Block 27 is used for specifying the target velocity, and the axis is started with block 28. Block 29 waits for the velocity to be reached by querying the corresponding bit in the axis status word (I11).

For further information on the individual bits in axis control word and axis status word, please see Operating Instructions 6710.201 “Functions and Parameters”.

7 Velocity Profile

With operating mode **velocity profile**, (from firmware V 5.95 on), a specified velocity-time-sequence can be run. For each section of the profile, the acceleration or deceleration times as well as the time with constant velocity are stated in milliseconds.

Operating mode **velocity profile** can only be called in program operation using the part program (therefore, option B2 or higher is required for TrioDrive D/xS, MidiDrive D/xS, TrioDrive D, and MidiDrive D). Due to the internal conversion of the velocity profile into a list of micro commands, the execution is accurately in time and independent of the processing periods of the individual part program blocks.

Operating mode **velocity profile** is selected via block type *axis operating mode* and initialized with type *velocity profile initialize* block. The profile is defined section by section (target velocity, ramp time, and duration of the section) using further *velocity profile* blocks. The velocity profile can be terminated with a type *velocity profile end* block.

Velocity profile data (target velocity, ramp time, and duration of each profile section) can be specified in three different ways:

- Profile section as individual data

  In a type *velocity profile individual data* block, the values for target velocity, ramp time, and duration per section can be specified individually, either as numerical values in the part program block or via variables.

- Profile section as data group

  The values for target velocity, ramp time, and duration (in this sequence) can be stored in three successive variables. With a type *velocity profile data group* block, such a variable group can be integrated into the velocity profile. The variable number of the first variable can be specified both either as numerical value in the part program block or via a variable.

- Complete profile

  In the variables, all profile sections are stored in successive variables, for each section target velocity, ramp time, and duration (in this sequence). With a type *velocity profile data group with autoincrement*, a section of the profile is defined. The variable number of the first variable of each group is specified via another variable which is increased automatically by 3. By that, successive calls of type *velocity profile data group with autoincrement* blocks with the same address number add successive sections to the profile.

In axis operating mode **velocity profile**, operating mode-dependent bits are not included in axis control word and axis status word.

In operating mode **velocity profile**, the sequence of motions is influenced by the following parameters:

- ramps machine data with ramps reference velocity, acceleration time, deceleration time, ramps reference velocity

  – define the ramp form (linear or $\sin^2$) and the reference value for converting

the ramp time in part program block *velocity profile*, the ramp times are influenced by part program block *velocity profile*.

7.1 Examples

The following six part program fragments show the use of the different block types using this example of a velocity profile ($\sin^2$ is set as ramp form):

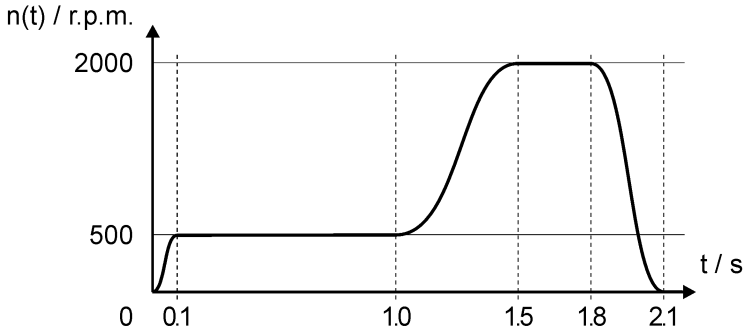

Figure 2: Velocity Profile (Example)

The program variables should contain the following values:

```
Variable 2     13
Variable 3     16
Variable 4     19
Variable 13    500
Variable 14    100
Variable 15    1000
Variable 16    2000
Variable 17    500
Variable 18    800
Variable 19    0
Variable 20    300
Variable 21    300
```

7.2 Velocity Profile as Direct Individual Specification

The values for each section are specified individually in the part program block:

```
Block 9     ...
Block 10    Axis state X: switch on disabled
Block 11    Axis operating mode X: velocity profile
Block 12    Axis state X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Velocity profile axis X 500 VU ramp = 100 ms duration = 1000 ms
Block 15    Velocity profile axis X 2000 VU ramp = 500 ms duration = 800 ms
Block 16    Velocity profile axis X 0 VU ramp = 300 ms duration = 300 ms
Block 17    Velocity profile axis X end
Block 18    ...
```

7.3 Velocity Profile as Indirect Individual Specification

The values for each section are specified individually via variables:

```
Block 9     ...
Block 10    Axis state X: switch on disabled
Block 11    Axis operating mode X: velocity profile
Block 12    Axis state X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Velocity profile axis X @13 VU ramp = @14 ms duration = @15 ms
Block 15    Velocity profile axis X @16 VU ramp = @17 ms duration = @18 ms
Block 16    Velocity profile axis X @19 VU ramp = @20 ms duration = @21 ms
Block 17    Velocity profile axis X end
Block 18    ...
```

7.4 Velocity Profile Via Data Groups

The values for each section are specified via variables. The numbers of these variables are defined in the part program:

```
Block 9     ...
Block 10    Axis status X: switch on disabled
Block 11    Axis operating mode X: velocity profile
Block 12    Axis state X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Velocity profile axis X from variable 13
Block 15    Velocity profile axis X from variable 16
Block 16    Velocity profile axis X from variable 19
Block 17    Velocity profile axis X end
Block 18    ...
```

7.5 Velocity Profile via Data Groups, Addressed Indirectly

The values for each section are specified via variables. The numbers of these variables, in turn, are included in variables:

```
Block 9     ...
Block 10    Axis state X: switch on disabled
Block 11    Axis operating mode X: velocity profile
Block 12    Axis status X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Velocity profile axis X from variable @2
Block 15    Velocity profile axis X from variable @3
Block 16    Velocity profile axis X from variable @4
Block 17    Velocity profile axis X end
Block 18    ...
```

7.6 Velocity Profile Complete

The values for all sections are specified via variables. The first number of these variables, in turn, is included in variables:

```
Block 9     ..
Block 10    Axis state X: switch on disabled
Block 11    Axis operating mode X: velocity profile
```

```
Block 12    Axis status X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Velocity profile axis X from variable @2 with autoincrement
Block 15    Velocity profile axis X from variable @2 with autoincrement
Block 16    Velocity profile axis X from variable @2 with autoincrement
Block 17    Velocity profile axis X end
Block 18    ...
```

Note: at the end of this section, variable `@2` contains value 22 instead of 13!

7.7 Velocity Profile Complete via Loop

The values for all sections are specified via variables. The first number of these variables, in turn, is included in variables:

```
Block 9     ...
Block 10    Axis state X: switch on disabled
Block 11    Axis operating mode X: velocity profile
Block 12    Axis status X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Label 20
Block 15    Velocity profile axis X from variable @2 with autoincrement
Block 16    From label 20 repeat 2 times (counter=0)
Block 17    Velocity profile axis X end
Block 18    ...
```

Note: at the end of this section, variable `@2` contains value 22 instead of 13!

7.8 Various Velocity Profiles

The storage of several velocity profiles in the drive and the selection of them using two variables is possible. For this example, the program variables contain the following values:

```
Variable 1     10      (desired profile)
Variable 2     2       (number of sections minus 1)
Variable 3     0       (auxiliary variable)
Variable 10    500     (profile 1)
Variable 11    100
Variable 12    1000
Variable 13    2000
Variable 14    500
Variable 15    800
Variable 16    0
Variable 17    300
Variable 18    300
Variable 20    800     (profile 2)
Variable 21    200
Variable 22    1000
...
```

The desired profile is selected with variable `@1` (in this case: 10, 20, ...), the num-

ber of sections is specified in variable @2 (subtract 1 from value, i.e. specify 2 for 3 sections, etc.).

```
Block 8     ...
Block 9     @3 = @1 + 0
Block 10    Axis state X: switch on disabled
Block 11    Axis operating mode X: velocity profile
Block 12    Axis status X: operation enabled
Block 13    Velocity profile axis X initialize
Block 14    Label 20
Block 15    Velocity profile axis X from variable @3 with autoincrement
Block 16    From label 20 repeat 2 times (counter=0)
Block 17    Velocity profile axis X end
Block 18    ...
```

This way, more than 80 profile sections can be defined in the available 255 variables, e. g. 10 different profiles with an average of 8 sections each.

7.9 Part Program Example of Spindle Positioning

In this example, the velocity profile described in section Velocity Profile (Block Type) 26 is run. At the end, however, it is decelerated to 50 r.p.m., afterwards, the axis is moved to 0 position using the spindle positioning.

```
Block 9             ...
Block 10      Axis state X: switch on disabled
Block 11      Axis operating mode X: velocity profile
Block 12      Axis state X: operation enabled
Block 13      Velocity profile axis X initialize
Block 14      Velocity profile axis X 500 VU ramp = 100 ms duration = 1000 ms
Block 15      Velocity profile axis X 2000 VU ramp = 500 ms duration = 800 ms
Block 16      Velocity profile axis X 50 VU ramp= 300 ms duration= 300 ms
Block 17      Velocity profile axis X end
Block 18      Velocity profile spindle positioning axis X target = 0 PU
Block 19      Velocity profile spindle positioning axis X end
Block 20      ...
```

8 Variable Descriptions

For information on whether or not a certain variable is valid in operating mode **program operation**, see the corresponding line “Valid: ...” in the respective variable description.

8.1 Parameters of Part Program Operation

The parameters of part program operation include the following variables:

- *Program Variables*
- *Program Start Label*
- *Program Block Number Display*
- *Program Label Display*
- *Part Program*

Program Variables	Index: 5f5e, Short name: PgmVariable
SPP Windows	Variable 1 0
Type	array, 255 type integer32 elements
Var. type	program variables read and write
Unit	–
Standard value	0
Valid	in device operating modes ***program automatic*** and ***program single step***

With part program variables, the value of numerical fields of part program blocks can be specified indirectly. Access to the part program variables is possible from all communication interfaces using the *program variable* object described here.

In a field, a part program variable is displayed by a two-digit number with prefix “@” (e. g. “@23”). 255 part program variables, @1 to @255 are available. The part program editor of the SPP Windows command and commissioning software can be used for specifying whether the value is set directly as a number or indirectly via part program variables.

Part program variables serve for storing values outside the part program. Reading and writing access to the part program variables is possible via the part program and the communication interfaces. They are suitable for influencing motion sequences coded in the part program via the PLC or for parameterizing part program sections from another part program section. The user defines the meaning of a part program variable by its use in the part program.

Application of part program variables in the part program at the example of a type *position* block:

- X -760255 A F% 100,0 A 1.4 [1] to pos.

  In this line, part program variables are not used yet. An absolute positioning of

axis 1 to position –760255 is carried out at a target velocity of 100%. After the position has been reached, output bit 4 of output byte 3 is set to 1.

- `X @6 A F% @33 A 1.4 [1] to pos.`

  With this example, the axis moves to the position stored in part program variable 6 with the relative target velocity stored in part program variable 33.

- `X -760255 A F% 100,0 A @21.@22 [1] nach Pos.`

  In this case, part program variables 21 and 22 define which output (part program variable 22) has to be actuated from which output byte (part program variable 21).

The term part program variables must not be mixed up with the general term variables. The variables provide a general concept for access to all axis and device functions. The part program variables carry out a special sub-function within the part program functions. During that, the general concept of variables is used for accessing the part program variables.

In the SPP Windows command and commissioning software, part program variables can be edited in the Variables tab of the Parameterization window.

Program Start Label	Index: 5f5d, Short name: PgmStartLabel
SPP Windows	`Start label 1000`
Type	simple variable, unsigned16
Var. type	setpoint or parameter for a function read and write
Unit	–
Standard value	0
Valid	in device operating modes ***program automatic*** and ***program single step***

With the *program start label*, the user specifies the label (type *label* block) from which the part program starts. A part program cannot be started unless a type *label* block with the same label as the *program start label* is defined at the beginning. If several of such similar *labels* exist in the part program, processing will start at the *label* with the lowest block number.

If the part program is started via the enable input (check configuration in the enable selection code), the *start label* does not have an effect. In this case, the program will always be started from *label* 9999 which has to be available.

Program Block Number Display	Index: 5f5b, Short name: PgmBlock
SPP Windows	(via pointer in the part program control window)
Type	simple variable, unsigned16
Var. type	actual value read only
Unit	–
Standard value	0
Valid	in device operating modes ***program automatic*** and ***program single step***

In a running part program, the *program block number display* shows the number of the block being processed. With that, monitoring of the part program sequence is possible.

If the part program run is stopped by a type *program halt* block with the program stop option (see description of block type *program halt*) or by the end of a block in program mode single step, this variable shows the number of the block to be processed next.

If the part program run is stopped by a type *program halt* block with the program end option (see description of block type *program halt*) or by an error, this variable shows the number of the block that led to the termination.

Program Label Display	Index: 5f5c, Short name: PgmLabel
SPP Windows	Last label 1000
Type	simple variable, unsigned16
Var. type	actual value read only
Unit	–
Standard value	0
Valid	in device operating modes ***program automatic*** and ***program single step***

Similar to the *program block number display*, the *program label display* informs the user on the part program run. It specifies the label (value of the *label*) processed last.

With this variable, you can check which part program section beginning with a *label* is currently being processed. For example, you can easily check whether a specific iteration or a specific subroutine is being processed as these program sections always start with a *label*.

Part Program	Index: 5f5f, Short name: Teileprogramm
SPP Windows	Block 0 label 0
Type	array, 71 type octet string elements, length 128
Var. type	part program data range, mapped on an array read and write
Unit	–
Standard value	empty, i.e. all blocks are of type *empty*
Valid	in device operating modes ***program automatic*** and ***program single step***

The part program executed in device operating modes program automatic and program single step is included in this array.

Overwriting the *part program* is only permitted when the part program state machine is in idle state.

In the SPP Windows command and commissioning software, the *part program* can be edited in the Part Program tab of the Parameterization window.

9 Appendix

9.1 Appendix A State Machines

State machines describe the behavior of systems. The graphical representation of a state machine is called state diagram.

The elements of a state machine are

- *states*
- *transitions*
- *events*
- *actions*

A representation of the elements of a state machine in the state diagram is shown in the following figure:

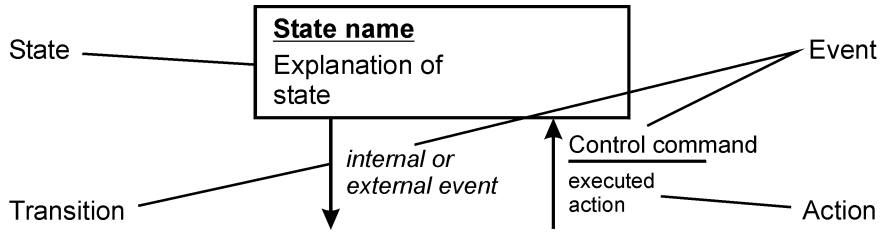

Figure 3: Elements of state machines

States (shown as a rectangle with the name of the state and further explanations, if required) can be changed by **transitions** (displayed as arrows). A transition is performed when an **event** occurs. In the case of a transition, an **action** is carried out (represented below the event, separated from it by a line). Transmissions for which an action is not carried out are also permitted.

In the application of state machines, the following types of events are differentiated:

- Control commands

  These are events the user can trigger by writing the control command into a control word. Each control command is coded as bit pattern.

- Internal or external events

  Internal or external events (represented in italics) are triggered by the servo drive or a component connected to one of its interfaces. This can be e. g. a fault or the fact that a certain velocity has been reached.

9.2 Appendix B Axis Fault Codes and Part Program Errors

In case of errors occurring during the start or the execution of a part program (axis fault codes 62... hex), the part program state machine changes to state idle from where the part program can be restarted after having reset the error.

These operating instructions describe the fault codes of the part program, only. A complete list of all fault codes of the ESR servo drives is included in Operating Instructions 6710.201 “Functions and Parameters”

If an error occurs during the part program execution, variable *program block number display* shows the corresponding block in the part program.

Part Program Errors

6210_{hex} Unknown block type

The type of a block cannot be executed with the existing firmware version of the servo drive. This may be due to the fact that the part program was transferred from a drive with a newer firmware into a drive with an older firmware which knows less block types.

6220_{hex} Label not found

The label searched by a block of the type *jump to label, repeat, jump on input,* or *subroutine call* does not exist anywhere in the part program.

6221_{hex} Start label not found

The label required for starting the part program (*program start label*) does not exist anywhere in the part program.

6231_{hex} Subroutine stack overflow

The maximum nesting depth of 10 subroutine calls (block type *subroutine call*) has been exceeded.

6232_{hex} Subroutine stack underflow

Failed to execute a type *return from subroutine* block because the subroutine was not called before.

6233_{hex} Repeat list overflow

The maximum number of 10 simultaneously active iterations has been exceeded by one block (block type *repeat*).

6235_{hex} Object access fault

Fault with a type *object access* block. This may be due to the following causes:

- The subindex is invalid for the specified index.
- The type of access (read or write) is not permitted for the variable.
- Access is not permitted due to the device status.

6236_{hex} Object does not exist

Fault with a type *object access* block. The variable with the specified in-

dex does not exist in the servo drive.

6237_{hex} Object too long

Fault with a type *object access* block. The variable is larger than 4 bytes and can therefore not be read or written with this block type.

6250_{hex} Axis access fault

A fault occurred while accessing an axis function (block types *position*, *feed*, *go to reference point*). Possible causes: selection of a non-existent axis or the target position is outside the position limit values.

6255_{hex} Fault while going to home position

A fault has occurred while going to home position (block type *go to reference point*).

6257_{hex} Change of operating mode not permitted

You are not permitted to switch operating modes in the following states:

- operation enabled
- quick stop active
- fault reaction active
- fault

6270_{hex} Internal I/O access fault

A fault has occurred while accessing an I/O function in a type *position, M function, jump on input, wait for input, switching point*, or *wait for input value and jump* block.

6275_{hex} Division by zero

Fault during execution of a *mathematical operation*.

6280_{hex} Internal timer fault

An internal fault has occurred in block type *wait*.

9.3 Appendix C Firmware Versions Regarding Part Program Functions

This section summarizes notes on firmware changes. The latest changes are listed first.

If you have worked with a servo drive with an older firmware, before (e. g. V 7.7) and get a new servo drive with a new firmware (e. g. V 8.5.8), please observe all following chapters referring to changes between the two version numbers.

This section states all firmware changes concerning the part program functions described in these operating instructions. For further firmware changes, please see the corresponding appendices of the other operating instructions for the servo drives.

Changes version V 7.9 compared to V 5.95:

- New function spindle positioning for the velocity profile.
  - new block types *velocity profile spindle positioning* and *velocity profile spindle positioning end*.
- Simplified initialization of the velocity profile.
  - new block type *velocity profile initialize*.

Changes version V 5.95 compared to older firmware versions:

- New function velocity profile.
  - new block types *velocity profile individual data, velocity profile data group, velocity profile data group with autoincrement, velocity profile end*.

9.4 Appendix D Versions of the Document

2005-02-28	V 8.0, KS	new on the basis of 6710.201 V 5.7 and the German version of these operating intructions
2009-05-18	V 8.0b, KS	for firmware V 8.0 updated for TrioDrive D/xS and MidiDrive D/xS; technical terms adapted; minor corrections
2014-01-13	V 8.5, KS	for firmware V 8.5 processed for online help function

Index

- T -

- V -

- W -