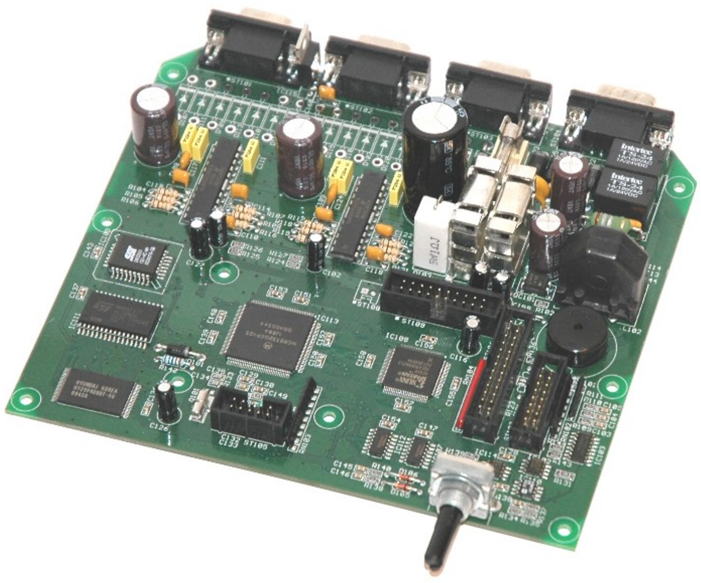
Motorcontroller CO-Series

User's Manual

Version 1.28e



Calantec GmbH Automatisierungstechnik, Industrieelektronik
Aufgang B

Helmholtzstraße 2-9	Telefon: 030 453 01 519	Geschäftsführer:	Hartmut Schäfer
10587 Berlin	Telefax: 030 453 01 531	Handelsregister:	Amtsgericht Charlottenburg HRB 92065
http://www.calantec.de	email: info@calantec.de	USt-ID:	DE232787423

valid for software version 016.031 and higher
Controller CO2200, CO4200, CO4300, CO6100, CO6150,
CO6300, CO6500, CO5400, CO5500

H. Schäfer 2006-12-18

Table of Contents

1. Start Up

The voltage of the power supply must be 24V+/-10% DC. Depending on the connected peripherals the current can be up to 2 A (amplifier current not included).
The motor controller should be connected to a terminal or a computer with a terminal program running. Default serial communication parameters are 19200 baud, no parity, 8 bits, 1 stop bit, no handshake. These values can be altered with the `MODE`-command (see command reference). If the motor controller is equipped with a RS232 compatible interface (see label) you can use a standard cable, if it is equipped with a RS422 interface you have to use the converter IN232/422 or a similar RS422/RS232 converter.

Axis configuration

The motor controller should be configured immediately after power on. The best way is to include the appropriate commands in a file named `CONFIG.SYS`. For configuration, the `MODE` and `SET` commands are used. Each of the three axes is defined either as stepper or servomotor axis and the power amplifier type is determined. Use option `PWMS` for amplifier PWM1660 and `PATTERN` for amplifier SM0435 and SM0860.

Example configuration:
```
MODE 1,PID,INC(1),PWMS(1)
MODE 2,PID,INC(2),PWMS(2)
MODE 3,SM,PATTERN(3)
```

Axis 1 and axis 2 are defined as servo motors, axis 3 as stepper motor. Now use `SET` commands to set necessary parameters like maximum current (`IMAX`, for stepper motors additionally standby current `ISTANDBY`). The PID algorithm for servo motor control needs the parameters `POL`, `DIFF`, `PROP` and `INT`. To determine the value of POL and while testing the reference movements, is it advised to limit the controller's output by setting the `MAXFORCE` parameter. Some microstepping controllers have to be programmed with the desired current profile (`WAVEFORM1`, `WAVEFORM2`).

PID parameters

To achieve maximum velocity and accuracy PID parameter have to be set up carefully. To do this efficiently, create a text file containing all basic configuration commands:

```
MODE 1,PID,INC(1),PWMS(1)
SET 1,IMAX,10
SET 1,PEAK,10
SET 1,PWMPOL,TRUE
SET 1,POL,TRUE
SET 1,MAXFORCE,10
SET 1,PROP,100
SET 1,INT,0
SET 1,DIFF,200
```

For more axes add the appropriate commands. Start with a limited controller output (`SET 1,MAXFORCE,10` or less) and limit, if possible the amplifier current (`SET`

1,MAXFORCE,10). Now switch on the controller with MOTOR 1 ON. If the motor axis is now moved, it will either move back to the former position (which is right) or run continuously in one direction. In the latter case change either SET 1,POL,TRUE to SET 1,POL,FALSE (resp. FALSE to TRUE), change the polarity of the motor cables or exchange incremental inputs A and B. If the control polarity is correct, increase MAXFORCE, IMAX and IPEAK. Now increase step by step the PROP parameter until the motor axis starts vibrating. Provoke the start of vibrating by hand or by using the JUMP command. Parameter PROP should be so high that the vibration stops after a while. Now increase parameter DIFF until the motor axis reaches the desired position with minimum overshoot. The INT parameter can lead to low frequency vibrating and should only be used carefully.

Group configuration

One, two or three axes can be combined in a group by using the GROUP command. Linear and circular interpolation is only possible within a group. All commands for movements refer to a group.

Example:
```
GROUP 1,1,2,3
```

Group 1 consists of axes 1, 2 and 3. The order of the axis numbers in the GROUP command determines which axis is used as X (first position), Y (second position) and Z (third position) axis.
Using the TRAFO command, user coordinates (for example millimeters) can be transformed into machine coordinates (i.e. so-called quadcounts, if you are using servo motors). It is also possible to correct angle errors.

CONFIG.SYS

The commands in the file named CONFIG.SYS on SID0: are executed immediately after power-up (same execution as with EXEC command). This can be prohibited by pressing the **ESC** key during power-up.
To create a CONFIG.SYS file start the editor with EDIT ”SID0:CONFIG.SYS” and type in the commands or transmit them using the ASCII-transfer function of a terminal program. Save file with **MENU/SAVE** and update the FLASH disk with the UPDATE command. Commands without a line number are executed directly during power-up. Commands with a line number are loaded into program memory, but not executed. They can be executed with the RUN command.

Example:
```
NOTAUS INTERN
MODE 1,SM,PATTERN(1)
MODE 2,SM,PATTERN(2)
SET 1,IMAX,30
SET 2,IMAX,30
SET 1,ISTANDBY,20
SET 2,ISTANDBY,20
GROUP 1,1,2
TRAFO 1,534,0,0,534
EXEC "PROGRAMM.SYS"
```

```
RUN
```

Properties of the FLASH and EEPROM disc

Controllers with model names COxxxx have two internal silicon disks. The first is FLASH based and called `SID0:`, the second is EEPROM based and called `SID1:`. `SID0:` is mainly for program storage and needs the `UPDATE` command after modification. Note that all amplifiers are switched off during update. The size of `SID0:` is 256 kbytes. The built-in EEPROM disc (`SID1:`) has a capacity of 256 kbytes. It is possible to save up to 128 different files. Use the `FORMAT` command to initialize. To see the directory, use command `FILES`. Files with extension `.SYS` are not displayed. You can force their appearance with `FILES ″*.*″,1`.

BASIC interpreter

After power-up the controller sends the character `>` (prompt) via device `STD:` (reading: keyboard and `COM1:`, writing: display and `COM1:`), if there is no autostart program as described above. Now you can type in commands which are executed immediately after the end of line (CR or CR/LF). Incoming characters are echoed to `STD:`. Corrections can be made by using the backspace key. These properties can be changed with `PROMPT`, `ECHO` and `SHELL`.
A program is created by typing (or dowloading) commands with line numbers. You can check the current program with `LIST` and save it with `SAVE`. A saved program can be loaded again with `LOAD`.
It is advised to save the basic configuration (baudrate, axes, transformation etc.) in file `CONFIG.SYS` and to save the user program in a second file, for example called `AUTOEXEC.BAS`, which is executed in file `CONFIG.SYS`. Larger programs should be written with a text editor program running on a PC, and then transferred to the motor controller via serial interface (clear memory with `NEW` first). You can use the ASCII transfer function of your terminal program for this purpose and then save the program with `SAVE`.
Files without line numbers (e.g. `CONFIG.SYS`) can be copied into the EEPROM disc by first starting the controller's editor (e.g. `EDIT ″SID0:CONFIG.SYS″`), then transferring the file via ASCII transfer and saving it with **MENU/SAVE**. If the file exists, clear it first with **MENU/NEW**.

Program example

A simple BASIC program can be written, saved and executed as follows: After power-up type in the following lines:

```
>
>100 FOR i=1 to 100
>110   PRINT "HELLO"
>120   SLEEP 200
>130 NEXT
>
```

If no prompt appears, try pressing the **ESC** key during power-up and check the terminal properties and the cable. The program above can be checked with LIST and started with RUN. The program prints 100 times the word `‶HELLO‶` with a delay of 200 milliseconds. Press ctrl-C (ASCII code 3) to abort the program (check terminal

properties if this doesn't work).
Save the program with

```
>SAVE "TEST.BAS"
```

The default device is EEPROM disk `SID1:`. You can get the list of files with

```
>FILES "SID1:"
```

or simply

```
>FILES
```

If you want this program to start automatically after power-up, an appropriate `CONFIG.SYS` has to be created on FLASH disk `SID0:`. Because the commands in the `CONFIG.SYS` should be executed immediately, no line numbers are necessary. For that reason the program cannot be created with the command line processor but with the editor. Call the editor program with

```
>EDIT "SID0:CONFIG.SYS"
```

Now type in the the startup commands

```
EXEC "TEST.BAS"
RUN
```

The first line loads the program, the second starts execution. Use the text file upload function of your terminal program to transfer larger programs. Save program with **MENU/SAVE** and leave the editor with **MENU/EXIT**. Unlike `SID1:` the FLASH disk `SID0:` has to be updated with the `UPDATE` command.

```
>UPDATE "SID0:"
```

This can take several seconds. All amplifiers are switched off during this time. The program starts now automatically at the next power-up.
You can get the directory of SID0: with

```
>FILES "SID0:",1
```

BIOS update

The controllers BIOS can be replaced by new versions. The actual version is shown with

```
>PRINT VERSION$
```

Start BIOS update with

```
>UPDATE 12345
```

The controller acknowledges this with the numbers 1 and 2 (init phase) and waits for the update file. Upload this file with the text file upload function of the terminal program. After the complete transfer the FLASH is cleared and rewritten (phases 3,4 and 5) and the controller resets. Never interrupt the power supply during the last phases. An abort during upload is possible.

2. Command Reference

Variable types:

`FIX`	64 bit fixpoint (32.32 format)
`INTEGER`	32 bit integer
`STRING`	string of arbitrary length
`CHAR`	single 8 bit character

BASIC-Commands and Functions

Conventions:

`a`	numerical expression (integer or fix)
`n`	integer
`nb`	boolean expression
`c`	single character (ASCII)
`s$`	string
`a(n)`	array
`[..]`	optional parameter

ABS

Function

Calculates the absolute value of numerical expression `a`. If `a` is a fixpoint value, the result is fixpoint, if `a` is integer, the result is integer.

Syntax:

```
a=ABS(a)
```

Example:

```
PRINT ABS(x)
PRINT ABS(-100.4)
```

Related commands and functions:

`INT, FRAC`

ACOS

Function

Calculates the arcuscosine of numerical expression `a`.

Syntax:

```
a=ACOS(a)
```

Example:

```
PRINT ACOS(x)
PRINT ACOS(0.4)
```

Related commands and functions:

`SIN, COS, ASIN`

AIN

Function

Returns the input level of analog input `n`. The maximum value is +100.0, minimum value ist -100.0.

Syntax:

```
a=AIN(n)
```

Example:

```
PRINT AIN(2)
```

Related commands and functions:

`CALIBRATE, AOUT`

AOUT

Command
Sets analog output `n` to level `a`. The maximum value is +100.0, minimum value ist -100.0.
Syntax:
`AOUT n,a`
Example:
`AOUT 1,15`
Related commands and functions:
`CALIBRATE, AIN`

ASC

Function
Returns the ASCII code of a character or of the first character of a string.
Syntax:
```
n=ASC(c)
n=ASC(s$)
```
Example:
`PRINT ASC(”Hallo”)`
Related commands and functions:
`CHR$`

ASHIFT

Function
Returns the integer value `n1` arithmetically shifted by `n2` positions.
Syntax:
`n=ASHIFT(n1,n2)`
Related commands and functions:
`ROTATE, SHIFT`

ASIN

Function
Calculates the arcussine of numerical expression `a`.
Syntax:
`a=ASIN(a)`
Example:
```
PRINT ASIN(x)
PRINT ASIN(0.4)
```
Related commands and functions:
`SIN, COS, ACOS`

ASSIGN

Command
Assigns an input or output device to an auxiliary device (`STD:`, `AUX1..4:`). Device `s1$` is assign to auxiliary device `s2$` as input (parameter `INPUT`) or output (parameter `OUTPUT`) device. With option `APPEND` the device is added to existing assignments, without `APPEND` existing assignments are replaced.
Syntax:
`ASSIGN s1$,s2$,INPUT|OUTPUT[,APPEND]`

Example:
```
ASSIGN "DIS:","AUX1:",OUTPUT
ASSIGN "KEY:","AUX1:",INPUT
ECHO "AUX1:",ON
```
Related commands and functions:
`OPEN, CLOSE, PRINT`

BEEP

Command

Variant 1: Switches keyboard signal on or off.
Syntax:
`BEEP ON|OFF`
Variante 2: The beep signal is switched on for `n` milliseconds.
Syntax:
`BEEP n`
Example:
`BEEP 500`

BIN$

Function

Returns a string which is `n2` characters long, containing the binary representation of integer expression `n1`.
Syntax:
`s$=BIN$(n1,n2)`
Example:
`PRINT BIN$(123,8)`
Related commands and functions:
`OCT$, HEX$`

BREAK

Command

Switches the possibility to terminate a running program by pressing Ctrl-C (ACSII code 3) or the **STOP** key on or off.
Syntax:
`BREAK ON|OFF`

CALIBRATE

Command

Calibrates zero of analog output `n1` with value `n2`. If the output is not used as an amplifier control output, the calibration value is updated only at the next `AOUT` command. The calibration values can be written permanently with command `WRITEPREFS` and are then reloaded at power on.
Syntax:
`CALIBRATE n1,n2`
Example:
`CALIBRATE 1,-3`
Related commands and functions:
`AOUT, WRITEPREFS`

CAN

Command (only with CAN hardware)

Switches the event processing for CAN channel `n` (`n=1..4`) on or off.

Syntax:
```
CAN(n) ON|OFF
```
Related commands and functions:
```
ON xx GOTO|GOSUB
```

CANBAUD

Command (only with CAN hardware)
Sets the CAN baudrate of CAN interface `n1` to the value `n2`.
Syntax:
```
CANBAUD n1,n2
```
Example:
```
CANBAUD 1,125000
```
Related commands and functions:
```
CANID, CANERROR, CANREAD, CANWRITE
```

CANERROR

Function (only with CAN hardware)
Returns the last Error Code of CAN interface `n`.
Syntax:
```
n=CANERROR(n)
```
Example:
```
PRINT CANERROR(n)
```
Related commands and functions:
```
CANBAUD, CANID, CANREAD, CANWRITE
```

CANID

Command (only with CAN hardware)
Sets the identifiers for the CAN communication. Channel 0 is used for a serial communication via CAN between one master and multiple slaves. The master→slave direction uses a set of 32 identifiers (base is `IDSource`, identifiers are `IDSource` up to `IDSource+31`). The slave→master direction uses another set of 32 identifiers (base is `IDTarget`, identifiers are `IDTarget` up to `IDTarget+31`). `IDSource` and `IDTarget` must be multiples of 32. `IDSource` of the master and `IDTarget` of the slave must have identical values, as well as `IDTarget` of the master and `IDSource` of the slave.
For channels 1 to 4 the parameter `ID` is the CAN identifier for incoming CAN messages. Channel 1 has an optional identifier mask where zero bits are don't cares while comparing the incoming `ID` with the programmed `ID`.
With parameter `mode=0`, the standard identifier format (10 bit) is used, with `mode=1` the extended format (29 bit).
Syntax:
```
CANID 0,mode,IDSource,IDTarget
CANID n,mode,ID[,mask]
```
Example:
```
CANID 0,0,64,128
CANID 1,0,121,&hfffffff0
CANID 2,0,34
```
Related commands and functions:
```
CANBAUD, CANERROR, CANREAD, CANWRITE
```

CANREAD

Command (only with CAN hardware)

Copies a received CAN message into `array(0)`. `n` is the channel number (1 to 4). The array must be a one-dimensional `array of char` with a size of at least 16. The entries are used as decribed below (`CANWRITE`), they don't have to be initialized. Entry 16 is set to `&hff`, if a new message was received, and set to `0` if not.

Syntax:

```
CANREAD n,array(0)
```

Example:

```
DIM candata(16) AS CHAR
CANREAD 1,candata(0)
```

Related commands and functions:

`CANBAUD, CANERROR, CANID, CANWRITE`

CANWRITE

Command (only with CAN hardware)

Sends a CAN message defined in `array(0)` via CAN interface `n`. The array must be a one-dimensional `array of char` with a size of at least 16. The entries have to be initialized as follows:

`array(1)`	Identifier, most significant byte
`array(2)`	Identifier, second most significant byte
`array(3)`	Identifier, third most significant byte
`array(4)`	Identifier, least significant byte
`array(5)`	data byte 0
`array(6)`	data byte 1
`array(7)`	data byte 2
`array(8)`	data byte 3
`array(9)`	data byte 4
`array(10)`	data byte 5
`array(11)`	data byte 6
`array(12)`	data byte 7
`array(13)`	reserved
`array(14)`	number of data bytes
`array(15)`	mode (0=standard, 1=extended)
`array(16)`	reserved

The standard mode (`mode=0`) uses 10 bit identifiers, the extended mode (`mode=1`) uses 29 bit identifiers.

Syntax:

```
CANWRITE n,array(0)
```

Example:

```
DIM candata(16) AS CHAR
candata(1)=0     :' ID MSB
candata(2)=0     :'
candata(3)=0     :'
candata(4)=20    :' ID LSB
candata(5)="A"   :' CAN Data 1
candata(6)="B"   :' CAN Data 2
candata(7)="C"  :' CAN Data 3
candata(14)=3   :' CAN Length
candata(15)=0   :' CAN Mode
CANWRITE 1,candata(0)
```

Related commands and functions:
`CANBAUD, CANERROR, CANID, CANREAD`

CASE

Command
See `SELECT .. CASE`.

CHR$

Function
Returns a string consisting of one character with ASCII code `n`.
Syntax:
`s$=CHR$(n)`
Example:
`PRINT CHR$(65)`
Related commands and functions:
`ASC`

CINT

Function
Numerical expression `a` is rounded to an integer.
Syntax:
`n=CINT(a)`
Example:
`PRINT CINT(65.6)`
Related commands and functions:
`ABS, FRAC, INT, FIX`

CLEAR

Command
The event processing is switched off (if not prevented earlier with `CONTEVENTS ON`), unused variables are discarded, open files are closed. Used variables are set to default values.
Syntax:
`CLEAR`
Related commands and functions:
`CONTEVENTS, NEW`

CLEARBUF

Command
The input buffer of channel `n` is cleared.
Syntax:
`CLEARBUF n`
Related commands and functions:
`OPEN, INPUT`

CLEARSTACK

Command
Clears BASIC stack.
Syntax:
`CLEARSTACK`
Related commands and functions:

`POPSTACK`

CLOSE

Command
Closes one or more open files.
Syntax:
`CLOSE [[#]n1][,[#]n2]...]`
Example:
```
CLOSE
CLOSE #1,#2
```
Related commands and functions:
`OPEN`

CLS

Command (with display only)
The display is cleared, the cursor is set to the upper left corner and is switched on.
Syntax:
`CLS`
Related commands and functions:
`LOCATE, CURSOR`

CODE(C)

Function
Returns the ASCII code of the last pressed function key.
Syntax:
`a=CODE(C)`
Example:
`PRINT CODE(C)`
Related commands and functions:
`CTRLCODE, ON xx GOTO/GOSUB`

COM

Command
The event processing for serial interface `n` is switched on or off. The option `COM(n) STOP` stops event processing, but all appearing events are processed after switching event processing on.
Syntax:
`COM(n) ON|OFF|STOP`
Related commands and functions:
`ON xx GOTO|GOSUB`

CONNECT

Command
Connects a slave controller to a master controller via the given communication interface. The communication interface is assigned the number `n` which is used to change the slave controller (see `NETMASTER` command). `n` can be between 1 and 31.
Syntax:
`CONNECT n,s$`
Example:
`CONNECT 1,"COM1:"`

`CONNECT 2,"CAN1:3"`

Related commands and functions:

`NETMASTER, NETSLAVE`

CONTEVENTS

Command

Switches event processing after finishing a program on or off. If it is switched on, events like `TIMER`, `COM` and `CAN` are still processed (i.e. the interpreter jumps in the given program line), even if the main program is finished. If the program is altered the system may crash, so it is strongly advised to use the `NEWLIST` command.

Syntax:

`CONTEVENTS ON|OFF`

Related commands and functions:

`CLEAR, NEW, ON xx GOTO|GOSUB`

CONTRAST

Command (with display only)

Adjusts LCD contrast. Values between –10 and 10 change contrast relatively, values larger then 10 set the contrast absolute. Maximum value is 100. The contrast value can be written permanently with command `WRITEPREFS` and is then reloaded at power on.

Syntax:

`CONTRAST n`

Example:

```
CONTRAST -2
CONTRAST 70
```

Related commands and functions:

`WRITEPREFS`

COS

Function

Calculates the cosine of numerical expression `a`.

Syntax:

`a=COS(a)`

Example:

```
PRINT COS(x)
PRINT COS(0.4)
```

Related commands and functions:

`ACOS, ASIN, SIN`

CTRLCODE

Command (with keyboard only)

Switches event processing for function keys on or off. The ASCII code of the last pressed function code can be read with function `CODE(C)`.

Key	ASCII code
STOP	3
HALT	4
EDIT	5

START	6
RUN	7
MENU	14
MOVE	15
STEP	16
REF	18
TEACH	20
ESC	27

Syntax:
`CTRLCODE ON|OFF`
Related commands and functions:
`ON xx GOTO|GOSUB, CODE`

CURSOR

Command (with display only)
Switches cursor on or off.
Syntax:
`CURSOR ON|OFF`
Related commands and functions:
`CLS, LOCATE`

DELETE

Command
Deletes a file named `s$`.
Syntax:
`DELETE s$`
Example:
`DELETE ”TEST1.BAS”`
Related commands and functions:
`OPEN, FORMAT`

DIM

Command
Declares and sets the size of an array. The indices `n1`, `n2` and `n3` are the sizes of the array. Allowed variable types are `INTEGER`, `FIX`, `CHAR` and `STRING`.
Syntax:
```
DIM array(n1[,n2[,n3]]) AS INTEGER
DIM array(n1[,n2[,n3]]) AS FIX
DIM array(n1[,n2[,n3]]) AS CHAR
DIM array(n1[,n2[,n3]]) AS STRING
```
Example:
```
DIM xcoord(100) AS FIX
DIM men1(5) AS STRING
DIM data(64,64,64) AS FIX
```

DO..LOOP

Command
Repeats a block of command as long as a condition is true (`WHILE a`) or until a condition is true (`UNTIL a`). The condition check is performed where `WHILE` und `UNTIL`
are positioned, either at the beginning or at the end of the block

of commands. Expression `a` is `TRUE`, if it is not equal zero, it is `FALSE`, if it is zero.

Syntax:

```
DO [WHILE | UNTIL a]
  [block of commands]
LOOP [WHILE | UNTIL a]
```

Example:

```
x=1
DO WHILE x<10
  x=x+1: PRINT x
LOOP
```

Related commands and functions:

`EXIT, FOR..NEXT`

DRESTORE

Command

Restores the display content previously saved in array `a(0)`.

Syntax:

```
DRESTORE a(0)
```

Related commands and functions:

`CLS, DSAVE`

DSAVE

Command

The actual display content is saved in array `a(0)` (`of char`). With a display size of 20 x 4 characters the size of `a(0)` has to be at least 80.

Syntax:

```
DSAVE a(0)
```

Related commands and functions:

`CLS, DRESTORE`

ECHO

Command

Switches echo mode of a communication channel on or off.

Syntax:

```
ECHO ON|OFF
ECHO s$,ON|OFF
```

Example:

```
ECHO "AUX1:",ON
```

Related commands and functions:

`PROMPT, SHELL`

EDIT

Command (with display only)

Starts the text editor. If a line number `n` is given, the cursor is positioned at the beginning of this line. If file name `s$` is missing, the editor starts with the **FILE/LOAD** menu. If a format string is given instead of a name, only the matching files are listed.

Syntax:

```
EDIT [s$[,n]]
```

Example:

```
EDIT ”TEST1.BAS”
EDIT name$,errorline
EDIT ”*.BAS”
```

Related commands and functions:

END

Command

Terminates a program (`END`) or a block (`END IF`, `END SELECT`). When terminating a program, all open files are closed.

Syntax:

```
END
END IF
END SELECT
```

Related commands and functions:

`IF..THEN..ELSE, SELECT..CASE`

EOF

Function

Returns –1 (`TRUE`), if the end of file `n` is reached, and 0 (`FALSE`), if not.

Syntax:

```
a=EOF(n)
```

Example:

```
OPEN ”TEST.TXT” FOR INPUT AS #1
DO
  PRINT INPUT$(1,1);
LOOP UNTIL EOF(1)
```

Related commands and functions:

`OPEN`

EOLCHAR

Command

Sets the character which is used as the end-of-line character during an `INPUT` or `LINE INPUT` command. Default value is carriage-return (ASCII code 13).

Syntax:

```
EOLCHAR n
EOLCHAR s$
```

Example:

```
EOLCHAR 10
EOLCHAR ”*”
```

Related commands and functions:

`INPUT, LINE INPUT`

ERL

Function

Returns the line number, where the last error occured.

Syntax:

```
a=ERL
```

Related commands and functions:

`ERR, ERR$`

ERR

Function
Returns the error code of the last error.
Syntax:
`a=ERR`
Related commands and functions:
`ERL, ERR$`

ERR$

Function
Returns a string containing the description of error code `n`.
Syntax:
`s$=ERR$(n)`
Example:
`PRINT ERR$(ERR)`
Related commands and functions:
`ERL, ERR`

ERROR

Command
Switches event processing in the case of an error on or off.
Syntax:
`ERROR ON|OFF`
Related commands and functions:
`ON xx GOTO|GOSUB`

EXEC

Command
Executes commands in a file. If commands with line numbers are included, they are added to the current program, but not executed immediately. Execution can be stopped with command `EXIT`.
Syntax:
`EXEC s$`
Example:
`EXEC ”TEST.BAS”`
Related commands and functions:
`LOAD, SAVE, EXIT`

EXECERRORS

Command
If during a file execution (with `EXEC`) an error occurs, the execution is normally terminated. You can prevent this with the command `EXECERRORS OFF`. This is useful, if in a program a file is executed which may contain e.g. syntax errors.
Syntax:
`EXECERRORS ON|OFF`
Related commands and functions:
`EXEC, FERL, FERR`

EXECSTR

Command
Executes commands in a string. If the string contains lines with a line number, these lines are added to the program listing.

This command is useful to execute G-Codes with variables. Only a string variable can be used, a string constant (e.g. `EXECSTR "G00 X0"`) is not possible.

Syntax:

```
EXECSTR s$
```

Example:

```
EXECSTR "G01 X"+2*xofs
```

Related commands and functions:

`EXEC`

EXIT

Command

Terminates a `DO..LOOP` or `FOR..NEXT` or with `EXEC` started execution early. Hint: If you don't terminate a `DO..LOOP` or `FOR..NEXT` execution with `EXIT` but just jump out of the loop, a stack overflow error may occur.

Syntax:

```
EXIT DO
EXIT FOR
EXIT
```

Related commands and functions:

`DO..LOOP, FOR..NEXT, EXEC`

FERL

Function

Returns the file line in which an error during an exectution with `EXEC` occured.

Syntax:

```
a=FERL
```

Example:

```
PRINT FERL
```

Related commands and functions:

`EXEC, EXECERRORS, FERR`

FERR

Function

Returns the error code of last error during a file execution with `EXEC`.

Syntax:

```
a=FERR
```

Example:

```
PRINT FERR
```

Related commands and functions:

`EXEC, EXECERRORS, FERL`

FILES

Command

Shows the directory of a device. If no device name is given, `SID1:` is used as default. If option `,1` is used, all files, even those with `.SYS` as extension are shown. You can use wildcarts (*,?) in the file name.

Syntax:

```
FILES [s$][,1]
```

Example:
```
FILES
FILES ″*.BAS″
FILES ″*.*″,1
```
Related commands and functions:
`FORMAT, LOAD, SAVE, EXEC`

FIX

Function
Converts numerical expression `a` to an integer by setting the fractional part to zero.
Syntax:
```
n=FIX(a)
```
Example:
```
PRINT FIX(65.6)
```
Related commands and functions:
`ABS, CINT, FRAC, INT`

FOR..NEXT

Command
Repeats a block of commands until counter variable `a`, starting at `astart` is equal or greater than teminal value `aend`. The optional increment or decrement value is given by `astep`, default is +1.
Syntax:
```
FOR a = astart TO aend [STEP astep]
  [Commandsblock]
NEXT [a]
```
Example:
```
FOR i = 1 TO 100: PRINT i: NEXT i
FOR x = 100.5 TO 13.2 STEP -0.4
  PRINT x
NEXT
```
Related commands and functions:
`DO..LOOP, EXIT`

FORMAT

Command
Formats a device. A device name must be given, e.g. `″SID1:″`. All data is erased. FLASH devices have to be updated with the `UPDATE` command.
Syntax:
```
FORMAT s$
```
Example:
```
FORMAT ″SID1:″
```
Related commands and functions:
`FILES, LOAD, SAVE, EXEC`

FRAC

Function
Returns the fractional part of a fix expression as integer.
Syntax:
```
n=FRAC(a)
```

Example:
```
PRINT FRAC(65.75)
```
Related commands and functions:
`ABS, CINT, FIX, INT`

FRE

Function

Returns information about the available memory. `FRE(0)` returns the total available memory, `FRE(1)` the largest block, and `FRE(2)` the available stack size.

Syntax:
```
n1=FRE(n2)
```
Related commands and functions:
`MEM, STACKSIZE, WRITEPREFS`

GETFILENAME$

Function (with display only)

Reads a file name. String `title$` is used as a title (e.g. ”LOAD FILE”). String `sel$` is used to select the files, only matching file names are displayed. Flag variable `n` allows to determine, whether the line “”NEW LINE” is added (`n=0`) or not (`n=1`). If this option is used, a custom file name can be edited, whereas string `default$` is used as a default. Files with extension .SYS are not shown.

Syntax:
```
name$=GETFILENAME$(title$,sel$,n[,DEFAULT$])
```
Example:
```
name$=GETFILENAME$(”Load File”,”*.BAS”,0)
n$=GETFILENAME$(”SAVE”,”*.TXT”,1,”NONAME.TXT”)
```
Related commands and functions:
`MENU`

GOSUB .. RETURN

Command

Jumps to the subroutine in line `n`. Command `RETURN` allows to jump back immediately after the `GOSUB` command. A subroutine should always be exited with `RETURN`, otherwise a stack overflow error may occur.

Syntax:
```
GOSUB n
```
Example:
```
100  GOSUB 1000
110  END
1000 PRINT ”Hallo!”
1010 RETURN
```
Related commands and functions:
`GOTO`

GOTO

Command

The program is continued in line `n`.

Syntax:

```
GOTO n
```

Related commands and functions:

`GOSUB, RETURN`

HEX$

Function

Returns a string which is `n2` characters long, containing the hexadecimal representation of integer expression `n1`.

Syntax:

```
s$=HEX$(n1,n2)
```

Example:

```
100 PRINT HEX$(10000,4)
```

Related commands and functions:

`BIN$, OCT$`

IF..THEN..ELSE..END IF

Command

Calculates the value of boolean expression `nb` and executes the command block after `THEN`, if the expression is `TRUE`. If the expression is `FALSE`, the command block after `ELSE` is executed. The `ELSE` branch can be omitted.

Syntax:

```
IF nb THEN
  command block 1
ELSE
  command block 2
END IF
```

Example:

```
IF a+b<>125 THEN PRINT ”1”
  ELSE PRINT ”2”
END IF
```

Related commands and functions:

`GOTO`

INDENT

Command

Switches the automatic formatting of the program listing on or off.

Syntax:

```
INDENT ON|OFF
```

Related commands and functions:

`LIST`

INPUT

Command

Reads a list of variables. The channel number `#n` is optional. If the channel is a communication device (e.g. `STD:`) and the input was wrong, an error message is displayed and the user can start again. If the channel is not a communication device (e.g. `SID1:`), the program terminates with an error in case of a wrong input.

Syntax:

```
INPUT [;]["Text"{;|,}] var1[,var2[,var3..]]
INPUT #n,var1[,var2[,var3..]]
```
Related commands and functions:
`LINE INPUT, INPUT$, TINPUT, TINPUTC, TINPUTL`

INPUT$

Function
Reads `n1` characters from a file or a device. If no channel number `#n2` is given, `STD:` (keyboard and `COM1:`) is used as default.
Syntax:
```
s$=INPUT$(n1[,[#]n2])
```
Related commands and functions:
`INPUT, LINE INPUT, TINPUT, TINPUTC, TINPUTL`

INPUTF

Command
Reads a list of variables. The channel number `#n` is optional. Integer variable `flag` returns an error code, if an input error occured or 0 if the input was successful.
Syntax:
```
INPUTF flag,["Text"{;|,}] var1[,var2..]
INPUTF #n,flag,var1[,var2[,var3..]]
```
Related commands and functions:
`LINE INPUT, INPUT, INPUT$, TINPUT, TINPUTC, TINPUTL`

INSTR

Function
Returns the position of the first occurrence of string `s2$` in string `s1$`. The search is started at position `n2`. `n2` is optional. `s2$` can also be replaced by a ASCII-Code.
Syntax:
```
n1=INSTR([n2,]s1$,s2$)
n1=INSTR([n2,]s1$,n)        ' n=1..255
```
Related commands and functions:
`INPUT, LINE INPUT`

INT

Function
Returns the largest integer, which is greater than or equal to numerical expression `a`.
Syntax:
```
n=INT(a)
```
Related commands and functions:
`ABS, CINT, FIX, FRAC`

KEY

Command
Switches the event processing for keyboard events on or off.
Syntax:
```
KEY ON|OFF
```
Related commands and functions:
`ON xx GOTO|GOSUB`

KEYTIME

Function
Returns the time how long the currently pressed key is already pressed (in 1/1024 seconds).
Syntax:
`n=KEYTIME`
Related commands and functions:
`KEY, KEYREPEAT`

KEYREPEAT

Command
Sets parameters of the repetition function for the keyboard. `a1` is the time (in seconds) until the first repetition. `a2` is the time between the following repetitions. `a3` is the time between the following repetitions in fast mode. After `n` characters the fast mode is switched on. If the **START** or **STOP** key is pressed or the `START` or `NOTAUS` command is used, the relays are switched on respectively switched off for at least `a4` seconds.
These values can be written permanently with command `WRITEPREFS` and are then reloaded at power on.
Syntax:
`KEYREPEAT a1,a2,a3,n,a4`
Related commands and functions:
`KEY, KEYTIME, START, NOTAUS`

LCASE$

Function
Converts all characters in a string to lower case.
Syntax:
`s$=LCASE$(s$)`
Related commands and functions:
`UCASE$`

LED

Command
Switches the LEDs on the keyboard on, off or in blink mode.

n	key
1	**REF**
2	**TEACH**
3	**EDIT**
4	**STEP**

All other LEDs are switched automatically.
Syntax:
`LED [n1][,n2[,n3..]] ON|OFF|BLINK`

LEFT$

Function
Returns a string containing the first `n` characters of string `s$`.
Syntax:
`s$=LEFT$(s$,n)`
Related commands and functions:
`LEN, MID$, RIGHT$`

LEN

Function
Returns the length of string `s$`.
Syntax:
`n=LEN(s$)`
Related commands and functions:
`LEFT$, MID$, RIGHT$`

LET

Command
Assigns value `value` to variabe `varname`. This command is not necessary and it is not displayed in a listing.
Syntax:
`LET varname=value : ' equal to varname=value`

LINE INPUT

Command
Reads a line. Channel number `#n` is optional.
Syntax:
```
LINE INPUT [;]["Text";] s$
LINE INPUT #n,s$
```
Related commands and functions:
`INPUT, INPUT$, TINPUT, TINPUTC, TINPUTL`

LIST

Command
Displays a program listing. The start line `n1` and the end line `n2` are optional.
Syntax:
`LIST [n1][-n2]`
Related commands and functions:
`IDENT`

LOAD

Command
In the current version equal to `EXEC`.
Syntax:
`LOAD s$`
Related commands and functions:
`EXEC`

LOC

Function
Returns the current read/write position within file `n`.
Syntax:
`n=LOC(n)`
Related commands and functions:
`CLOSE, LOF, OPEN, SEEK`

LOCATE

Command (with display only)
Sets cursor at position `n1,n2`.
Syntax:
`LOCATE n1,n2`

Related commands and functions:
`CURSOR`

LOF

Function
Returns the file size of file `n`.
Syntax:
```
n=LOF(n)
```
Related commands and functions:
`CLOSE, LOC, OPEN, SEEK`

LTRIM$

Function
Deletes all spaces, tabs and control characters at the beginning of string `s$`.
Syntax:
```
s$=LTRIM$(s$)
```
Related commands and functions:
`RTRIM$`

MEM

Command
Displays information about the memory usage.
Syntax:
```
MEM
```
Related commands and functions:
`FRE`

MENU

Function (with display only)
Returns the number of a selected menu item in a predefined menu `men(0)`. If the selection is aborted with **ESC**, 0 is returned. `titel$` is used as menu header.
Syntax:
```
n=MENU(titel$,men(0))
```
Example:
```
DIM men1(3) AS STRING
men1(1)="LOAD"
men1(2)="SAVE"
men1(3)="EXIT"
sel=MENU("FILE",men1(0))
```
Related commands and functions:
`DIM, MESSAGE`

MESSAGE

Command (with display only)
Displays a string or an error message and waits for a pressed key.
Syntax 1:
```
MESSAGE n       :´ error code
```
Syntax 2:
```
MESSAGE s$      :´ string
```
Related commands and functions:
`MENU`

MID$

Function

Returns `n2` characters of string `s$` starting at position `n1`. If `n2` is omitted, the rest of string `s$` is returned.

Syntax:

```
s$=MID$(s$,n1[,n2])
```

Related commands and functions:

`LEFT$, LEN, RIGHT$`

MODE

Command

Sets or displays parameters of the serial interface. Maximum Baudrate and exact values depend on the equipment. Small deviations from the desired value can occur, but generally don't affect the data transmission. As parity the modes `NONE`, `EVEN`, `ODD`, `MARK` and `SPACE` are supported. The bit count can be 5 to 8, the stop bits 1 or 2. The handshake mode is `NONE`, `RTSCTS` or `XONXOFF`.

Syntax 1:

```
MODE COMn:baudrate,parity,bits,stop,handshake
```

Syntax 2:

```
MODE COMn:
```

Example:

```
MODE COM1:19200,NONE,8,1,XONXOFF
```

NETMASTER

Command

`NETMASTER ON` puts the controller in the network master mode. In that mode it is possible to switch to the command line of a different controller in the network. The other controllers can be conntected via a serial interface or CAN interface (see `CONNECT` command). The current controller can be recognized by the number or letter preceeding the prompt `>`. The character `M` implies that commands go to the master controller. Communication with slaves can be recognized by the character `S` or the slave number, if the slave controller is already switched to slave mode (see `NETSLAVE` command). The current controller can be selected with a command line beginning with the character @ followed by the controller number. All other characters or commands in this line are ignored. The master controller has number `0`, all slaves the numbers assigned with the `CONNECT` command.
The master mode is switched off using the `NETMASTER OFF` command.

Example:

```
>NETMASTER ON
M>CONNECT 2,"COM1:"
M>@2
S>
```

Syntax:

```
NETMASTER ON|OFF
```

Related commands and functions:
`CONNECT, NETSLAVE`

NETSLAVE

Command

`NETSLAVE ON` puts the controller in the network slave mode. `s$` is the communication interface (serial or CAN), the optional parameter `n` is the slave number. If ommited, only a `S` is displayed preceeding the prompt, else the slave number is shown. The slave number should be the same as the number given in the `CONNECT` command. `NETSLAVE OFF` puts the controller back to the normal mode.

Syntax:
```
NETSLAVE OFF
NETSLAVE s$[,n]
```
Example:
```
NETSLAVE ”COM1:”,12
NETSLAVE ”CAN1:10”,10
```
Related commands and functions:
`CONNECT, NETMASTER`

NEW

Command

The event processing is switched off (if not prevented earlier with `CONTEVENTS ON`), the program and variable memory is cleared, open files are closed.

Syntax:
```
NEW
```
Related commands and functions:
`CLEAR, CONTEVENTS`

NEWLIST

Command

An existing program listing is locked and hidden, a new program can be typed in. The old program is still be used for event processing, if no new `ON xx GOTO|GOSUB` commands are used. Thus it is possible to write a user application (e.g. using G codes), while hardware dependent events are unafflicted.

Syntax:
```
NEWLIST
```
Related commands and functions:
`CLEAR, CONTEVENTS, NEW, ON xx GOTO|GOSUB`

OCT$

Function

Returns a string which is `n2` characters long, containing the octal representation of integer expression `n1`.

Syntax:
```
s$=OCT$(n1,n2)
```
Example:
```
100 PRINT OCT$(10000,8)
```
Related commands and functions:

`BIN$, HEX$`

ON

Command

Sets the program line to which is jumped if the event `event` occurs. `event` can be one of the following:

CAN(n)
A CAN-Frame was received on channel `n`.

COM(n)
A character was received via serial interface `n`.

CTRLCODE
A function key (e.g. **MENU**, **ESC**, **TEACH** ..) was pressed or the corresponding ASCII character was read via the serial interface.

ERROR
A run-time error occurred (e.g. division by zero).

KEY
A key on the front panel was pressed.

PORT(n)
The programmed edge (see command `PORT(n)`) was detected on input `n`.

TIMER(n)
This event occurs every `n` milliseconds.

Syntax:
`ON event GOTO|GOSUB n`

Related commands and functions:
`COM, CTRLCODE, ERROR, KEY, PORT, TIMER`

OPEN

Command

Opens a file. Device and file name are given in string `name$`. If the device name is omitted, `SID1:` is used as default. File names are not supported on devices without a file system (e.g. `COM1:`).
`mode` can be one of the following:

INPUT
File is read-only.

OUTPUT
File is write-only. An existing file with same name is overwritten.

RANDOM
File can be read and written. An existing file with same name is overwritten.

APPEND
File can be read and written. If a file with same name exists, the write pointer is set to the end of the file.

If the optional parameter `FLAG` exists, the program is not terminated is case of an error (e.g. invalid file name), but the error code is stored in variable `n2`.
The following devices are supported (equipment dependent):

`COM1:`	serial interface
`COM2:`	serial interface

`COM3:`	serial interface
`COM4:`	serial interface
`COM5:`	serial interface
`DIS:`	display
`KEY:`	keyboard
`SID1:`	EEPROM disc
`SID0:`	FLASH disc (see `UPDATE` command)
`STD:`	standard input (`COM1:` and `KEY:`) and output (`COM1:` and `DIS:`)
`AUX1:`	auxiliary device (see `ASSIGN` command)
`AUX2:`	auxiliary device (see `ASSIGN` command)
`AUX3:`	auxiliary device (see `ASSIGN` command)
`AUX4:`	auxiliary device (see `ASSIGN` command)
`CAN1:`	CAN-Bus

The CAN bus can be used like a serial connection between a master and multiple slaves. For that purpose the CAN identifier has to be set using the `CANID 0,..` command. The communication can use 31 logical channels, each assigned normally to one slave. The channel number is given like a file name following the colon, e.g. "`CAN1:21`".

Syntax:

```
OPEN name$ FOR mode AS [#]n1 [FLAG n2]
```

Example:

```
OPEN "COM1:" FOR RANDOM AS #1
OPEN "SID0:CONFIG.SYS" FOR OUTPUT AS #2
OPEN "TEST.TXT" FOR INPUT AS #3
OPEN "CAN1:7" FOR RANDOM AS #4
```

Related commands and functions:

`CANID, CLOSE, MODE, OPENSTR, UPDATE`

OPENSTR

Command

Opens string `s$` as a file.

Syntax:

```
OPENSTR s$ AS [#]n
```

Related commands and functions:

`CLOSE, OPEN`

PARITY

Function

Calculates the parity of integer `n`. 0 is returned for even parity, -1 for odd.

Syntax:

```
n=PARITY(n)
```

PI

Function

Returns π as fix value.

Syntax:

```
a=PI
```

PININ

Function

Returns the level of input pin `n` (see appendix). A logical 1 (`TRUE`, resp. -1) corresponds to a 24V input level, a logical 0 (`FALSE`, resp. 0) to 0V.

Syntax:

```
n=PININ(n)
```

Example:

```
PRINT PININ(12)
```

Related commands and functions:

`PINOUT, PORTIN, PORTOUT, PORT`

PINOUT

Command

Sets output pin `n1` to level `n2`. A logical 1 corresponds to a 24V input level, a logical 0 to 0V.

Syntax:

```
PINOUT n1,n2
```

Example:

```
PINOUT 1,1
```

Related commands and functions:

`PININ, PORTIN, PORTOUT, PORT`

POPSTACK

Command

Removes the top element of the basic stack. A stack element is created e.g. by a `DO`, a `FOR` or a `GOSUB` command. If the element is not removed by the corresponding commands (`LOOP`, `NEXT`, `RETURN`) you have to do this using `POPSTACK`, otherwise a stack overflow may occur.

Syntax:

```
POPSTACK
```

Related commands and functions:

`CLEARSTACK, FRE, STACKSIZE`

PORT

Command

Switches the event processing for input edge detection on or off. `PORT(n) RISING` detects a rising edge, `PORT(n) FALLING` a falling edge. The edge detection is only possible with inputs 1 to 8, depending on the controler model up to 16.

Syntax:

```
PORT(n) RISING|FALLING|OFF
```

Related commands and functions:

`ON xx GOTO|GOSUB`

PORTIN

Function

Returns the input levels of 8 bit input port `n`. Depending on the equipment, `n` may be 1 to 4. A logic 1 corresponds to a 24 V input level, a logic 0 to a 0 V level.

Syntax:

```
n=PORTIN(n)
```

Example:

```
PRINT BIN$(PORTIN(1),8)
```

Related commands and functions:

`PORTOUT, PORT`

PORTOUT

Command

Sets the outputs of 8 bit output port to the value `n2`. Depending on the equipment, `n` may be 1 to 3. A logic 1 corresponds to a 24 V output level, a logic 0 to a 0 V level.

Syntax:

```
PORTOUT n1,n2
```

Example:

```
PORTOUT 1,&b10011100
```

Related commands and functions:

`PORTIN, PORT`

PRINT

Command

Prints strings or string representations of numerical expressions to the standard output channel or to a given file `#n`. Expressions separated by semicola are printed without spaces, expressions separated by commata are printed with tabulators (ASCII code 9) between them. The output is terminated with CR/LF (ASCII codes 13 and 10), if the PRINT command doesn’t end with a comma or semicolon.

Syntax:

```
PRINT [#n,][a1|s1$][;|,]..
```

Example:

```
PRINT ”HALLO”;CHR$(33)
PRINT 100.0,20;
PRINT #1,”TEST”
```

Related commands and functions:

`PRINT USING, WRITE`

PRINT USING

Command

Prints formatted strings or formatted string representations of numerical expressions to the standard input and output channel or to a given file `#n`. Expressions separated by semicola are printed without spaces, expressions separated by commata are printed with tabulators (ASCII code 9) between them. The output is terminated with CR/LF (ASCII codes 13 and 10), if the PRINT command doesn’t end with a comma or semicolon. Some characters in the formatting string `format$` have special functions:

`###.##`	Number of figures before and after the decimal point. Leading zeros are replaced by spaces.
`~.#`	Unlimited figures before the decimal point
`$##.##`	Display leading zeros
`+##.##`	Display sign
`&`	Print entire string

`!`	Print first character of a string
`_`	Print following special character (e.g. `#`)
`\  \`	Print n characters of a string n = (number of spaces between `\\`) + 2

Syntax:
```
PRINT [#n,] USING format$; [a1|s1$][;|,]..
```
Example:
```
PRINT USING ”X: ###.##”;SIN(PI/3)
```
Related commands and functions:
`PRINT, WRITE`

PROMPT

Command
Switches the input prompt on or off.
Syntax:
```
PROMPT ON|OFF
```
Related commands and functions:
`ECHO, SHELL`

PULSEOUT

Command
Switches the pulse output `n` on or off. If `period` and `hightime` are given, the output is switched on. `Period` and `hightime` are given in milliseconds (integer or fix). A period value of `0` switches the output off (low level).
Syntax:
```
PULSEOUT n,period,hightime      ' switch on
PULSEOUT n,0                    ' switch off
```
Example:
```
PULSEOUT 1,20.0,1.5
```

RANDOMIZE

Command
Initialises the pseudo random generator with integer `n`.
Syntax:
```
RANDOMIZE n
```
Related commands and functions:
`RND`

REM

Command
The rest of the program line is treated as a remark. You can also use the character `´`.
Syntax:
```
REM
```

RENEW

Command
Clears the program listing entered during the execution of the main program. The main program must first be protected with `NEWLIST`.
Syntax:
```
RENEW
```

Related commands and functions:
`NEW, NEWLIST`

RENUMBER

Command
Renumbers the lines of a program. Start line is `n1`, step is `n2`.
Syntax:
`RENUMBER n1,n2`

RESET

Command
Executes a hardware reset.
Syntax:
`RESET`

RIGHT$

Function
Returns a string containing the last `n` characters of string `s$`.
Syntax:
`s$=RIGHT$(s$,n)`
Related commands and functions:
`LEN, LEFT$, MID$`

RND

Function
Returns a 32 bit pseudo random integer.
Syntax:
`n=RND`
Related commands and functions:
`RANDOMIZE`

ROTATE

Function
Rotates 32 bit integer `n1` by `n2` places.
Syntax:
`n=ROTATE(n1,n2)`
Related commands and functions:
`ASHIFT, SHIFT`

RTRIM$

Function
Deletes all spaces, tabs and control characters at the end of string `s$`.
Syntax:
`s$=RTRIM$(s$)`
Related commands and functions:
`LTRIM$`

RUN

Command
Starts execution of a program. Start line `n` is optional.
Syntax:
`RUN [n]`
Related commands and functions:
`END`

SAVE

Command

Saves a program with name `name$`. If no device name is given, `SID1:` is used as default. FLASH devices must be updated using the `UPDATE` command.

Syntax:

```
SAVE name$
```

Related commands and functions:

`EXEC, FORMAT, LOAD, OPEN, UPDATE`

SEEK

Command

Sets read/write pointer of file `n1` to position `n2`.

Syntax:

```
SEEK [#]n1,n2
```

Related commands and functions:

`OPEN`

SELECT .. CASE

Command

Executes one of several possible command blocks according to the value of a given test expression. As compare operator `cmp` can =, >, <, >=, <= or <> be used.

Syntax:

```
SELECT CASE test
  CASE expression: command block
  CASE IS cmp expression: command block
  CASE expr TO expr: command block
  CASE ELSE command block
END SELECT
```

Example:

```
FOR i=1 TO 5
  SELECT CASE i
    CASE 1: PRINT ”1”
    CASE 2 TO 5: PRINT ”2..5”
    CASE IS >= 6: PRINT ”>=6”
  END SELECT
NEXT i
```

Related commands and functions:

`IF..THEN`

SGN

Function

Returns the sign of a numerical expression.

Syntax:

```
n=SGN(a)
```

Example:

```
PRINT SGN(-3)
PRINT SGN(0.4)
```

Related commands and functions:

`ABS`

SHELL

Command
Switches the command line editor, the echo mode and the prompt on or off.
Syntax:
`SHELL ON|OFF`
Related commands and functions:
`ECHO, PROMPT`

SHIFT

Function
Shifts a 32 bit integer `n1` logically by `n2` places.
Syntax:
`n=SHIFT(n1,n2)`
Related commands and functions:
`ROTATE`

SIN

Function
Calculates the sine of numerical expression `a`.
Syntax:
`a=SIN(a)`
Example:
```
PRINT SIN(x)
PRINT SIN(0.4)
```
Related commands and functions:
`ACOS, ASIN, COS`

SLEEP

Command
Waits `n` milliseconds. The event processing is unaffected.
Syntax:
`SLEEP n`
Related commands and functions:
`TIME`

SPACE$

Function
Create a string containing `n` spaces.
Syntax:
`s$=SPACE$(n)`
Related commands and functions:
`STRING$`

SQR

Function
Calculates the square root of numerical expression `a`.
Syntax:
`a=SQR(a)`

STACKSIZE

Command
Sets the size of the BASIC stack to `n` bytes. This preference has to be written permanently by using the `WRITEPREFS`

command and is used after the next reset or power up.
Syntax:
`STACKSIZE n`
Related commands and functions:
`WRITEPREFS`

STR$

Function
Returns the string representation of numerical expression `a`.
Syntax:
`s$=STR$(a)`
Related commands and functions:
`VAL`

STRING$

Function
Returns a string containing `n1` characters with ASCII code `n2`, repectively the first character of string `s$`.
Syntax:
`s$=STRING$(n1,n2|s$)`
Related commands and functions:
`SPACE$`

SYNC

Command
Synchronizes the program execution with the system timer. The execution is interrupted until the system timer reaches a value which is divisible by n. During this time no events are monitored. For that reason this command should only be used in special cases. Different synchronization delays can occur if the system timer runs over (after about 1 000 000 000 milliseconds). The standard system frequency is 1024 Hz.
Syntax:
`SYNC n`
Related commands and functions:
`SLEEP, SYSTIMER`

SYSTIMER

Command (-m versions only)
Sets the interval of the system timer to `a` milliseconds. This command is only implemented in special BIOS versions. The time resolution is about 200 ns. If the system timer is changed all time-related computations and functions deliver wrong results (except SLEEP and TIME). This command should only be used in special cases. Depending on the controller model and load `a` should not be less than 0.8 ms and especially when using servo motors not greater than 2 ms.
Syntax:
`SYSTIMER a`
Related commands and functions:
`SLEEP, SYNC`

TIME

Function

Returns the time since the last reset or power up in milliseconds. After 1 000 000 000 milliseconds the counter will start at 0 again.

Syntax:

`n=TIME`

Related commands and functions:

`SLEEP`

TIMER

Command

Switches the event processing for timer events on or off.

Syntax:

`TIMER ON|OFF|STOP`

Related commands and functions:

`ON xx GOTO|GOSUB`

TINPUT

Command

Reads a list of variables via channel `n`. If `n=0`, the standard input channel is used. Parameter `timeout` is the maximum time in milliseconds to complete the command. `timeout=0` switches timeout off. `flag` is 0, if the input was successful, -1 if not.

Syntax:

```
TINPUT n,flag,timeout,var1[,var2[,var3..]]
```

Example:

```
TINPUT 2,iflag,2000,a,b
TINPUT kanal,iflag,0,a$
```

Related commands and functions:

`LINE INPUT, INPUT$, INPUT, TINPUTL, TINPUTC`

TINPUTC

Command

Reads a string of length `count` into variable `string$` form channel `n`. If n=0, the standard input channel is used. Parameter `timeout` is the maximum time in milliseconds to complete the command. `timeout=0` switches timeout off. `flag` is 0, if the input was successful, -1 if not.

Syntax:

```
TINPUTC n,flag,timeout,count,string$
```

Example:

```
TINPUTC 2,iflag,2000,10,b$
TINPUTC kanal,iflag,0,laenge,a$
```

Related commands and functions:

`LINE INPUT, INPUT$, INPUT, TINPUTL, TINPUT`

TINPUTL

Command

Reads a line (finished with `CR` respectively the character definded with `EOLCHAR`) into variable `string$` form channel `n`. If n=0, the standard input channel is used. Parameter `timeout` is the maximum time in milliseconds to complete the command. `timeout=0` switches timeout off. `flag` is 0, if the input was successful, -1 if not.

Syntax:

```
TINPUTL n,flag,timeout,a$
```

Example:

```
TINPUTL 2,iflag,2000,a$
TINPUTL kanal,iflag,0,a$
```

Related commands and functions:

`LINE INPUT, INPUT$, INPUT, TINPUTL, TINPUTL`

UCASE$

Function

Converts all characters in a string to upper case.

Syntax:

```
s$=UCASE$(s$)
```

Related commands and functions:

`LCASE$`

UPDATE

Command

Updates a FLASH device (standard `SID0:`) after a `SAVE` or `FORMAT` command or a similar file command.

Syntax:

```
UPDATE s$
```

Example:

```
UPDATE ”SID0:”
```

Related commands and functions:

`FILES, LOAD, SAVE, EXEC, FORMAT`

VAL

Function

Converts a string to a fix or integer value.

Syntax:

```
a=VAL(s$)
```

Related commands and functions:

`STR$`

VERSION$

Function

Returns the software version as string.

Syntax:

```
s$=VERSION$
```

WRITE

Command

`WRITE` prints unlike `PRINT` the list of expressions in a format which can be read by the `INPUT` command. For that purpose the expressions are separated by commata and strings are enclosed in `””`.

Syntax:

`WRITE [#n,] a1|s1$ [,a2|s2$..]`

Related commands and functions:

`PRINT`

`WRITEPREFS`

Command

Writes several preferences permanently.

Syntax:

`WRITEPREFS`

Related commands and functions:

`CALIBRATE, CONTRAST, KEYREPEAT, STACKSIZE`

`XPRINT`

Command

Writes a list of strings or numerical expressions into channel `n`. Expressions separated by semicola are printed without spaces, expressions separated by commata are printed with tabulators (ASCII code 9) between them. The output is terminated with CR/LF (ASCII codes 13 and 10), if the PRINT command doesn’t end with a comma or semicolon. If `n=0` the standard output channel is used.

Syntax:

`XPRINT n[,[a1|s1$][;|,]..`

Example:

```
XPRINT outchan,”HALLO”;CHR$(33)
XPRINT 2,100.0,20;
```

Related commands and functions:

`PRINT, PRINT USING, WRITE`

Motor control commands

Conventions:

`a`	numerical expression (integer or fix)
`ax`	numerical expression, used as X coordinate
`ay`	numerical expression, used as Y coordinate
`az`	numerical expression, used as Z coordinate
`aw`	numerical expression, used as an angle (degrees, zero is the positive x axis, positive values counterclockwise, negative values clockwise)
`n`	integer
`na`	integer, used as axis number (`na`=1..3)
`ng`	Integer, used as group number (`ng`=1..8)
`c`	single ASCII encoded character
`s$`	string
`[..]`	optional parameter
Syntax 1	syntax to be used, if group contains 1 axis
Syntax 2	syntax to be used, if group contains 2 axes
Syntax 3	syntax to be used, if group contains 3 axes

ACC

Command
Sets the acceleration for linear and circular movements (in user units/s²) of group `ng` to value `a`. The command can be executed during a movement and the new value is updated immediately.
Syntax:
`ACC ng,a`
Related commands and functions:
`DEC, HDEC, VEL, TRAFO`

ALENGTH

Function
Returns the current contour position of group `ng`.
Syntax:
`a=ALENGTH(ng)`
Example:
`PRINT ALENGTH(1)`
Related commands and functions:
`CONTOUR, LENGTH`

AMPERROR

Command
Switches the event processing for amplifier errors on or off. `AMPERROR HIGH` causes an event, if the amplifier error line is logical high, `AMPERROR LOW` if it is logical low.
Syntax:
`AMPERROR HIGH|LOW|OFF`
Related commands and functions:
`MAXDIFF, MAXFORCE, MOTOR`

APOS

Function
Returns the actual position of the `n`th axis of group `ng`. If axis `n`

is a servo motor, the back-transformed current position of the incremental encoder is returned, not the calculated desired position (see `TPOS`).

Syntax:

```
a=APOS(ng,n)
```

Example:

```
PRINT APOS(1,1),APOS(1,2),APOS(1,3)
```

Related commands and functions:

```
CPOS, TP, TPOS
```

ARCA

Command

Performs a circular movement with angle `aw`, starting at the current position, around the absolute center point `ax,ay,az`. The mode parameter `nm` determines, whether the circuler movement is in the XY plane (`nm=0`), in the YZ plane (`nm=1`) or in the (`nm=2`) plane.

Syntax 2:

```
ARCA ng,ax,ay,aw
```

Syntax 3:

```
ARCA ng,0,ax,ay,aw
ARCA ng,1,ay,az,aw
ARCA ng,2,ax,az,aw
```

Example:

```
ARCA 1,100.0,50.0,180
```

Related commands and functions:

```
ARCR
```

ARCR

Command

Performs a circular movement with angle `aw`, starting at the current position, around the relative center point `ax,ay,az`. The mode parameter `nm` determines, whether the circuler movement is in the XY plane (`nm=0`), in the YZ plane (`nm=1`) or in the (`nm=2`) plane.

Syntax 2:

```
ARCR ng,ax,ay,aw
```

Syntax 3:

```
ARCR ng,0,ax,ay,aw
ARCR ng,1,ay,az,aw
ARCR ng,2,ax,az,aw
```

Example:

```
ARCR 1,+50.0,-50.0,180
```

Related commands and functions:

```
ARCA
```

AUTOVEL

Command

If the `AUTOVEL` mode is switched on, the speed during circular movements is reduced, so that neither acceleration nor deceleration are exceeded.

Syntax:

`AUTOVEL [ng1[,ng2..] ON|OFF`

Example:

`AUTOVEL ON`

Related commands and functions:

`DEC, ACC, MAXSEGMENTS, RSEGMENTS`

AVEL

Function

Returns the current velocity of group `ng` respectively of axis `n` of group `ng` (`n=1`: x axis, `n=2`: y axis, `n=3`: z axis).

Syntax 1: liner or circular move

`a=AVEL(ng)`

Syntax 2: position move

`a=AVEL(ng,n)`

Related commands and functions:

`VEL`

BORDER

Command

Limits the movement of the axes to a lower (`a1`) and an upper (`a2`) value. If a limit is reached, the movement is stopped immediately without a deceleration profile.

Syntax 1:

`BORDER ng,a1x,a2x`

Syntax 2:

`BORDER ng,a1x,a2x,a1y,a2y`

Syntax 3:

`BORDER ng,a1x,a2x,a1y,a2y,a1z,a2z`

Example:

`BORDER 1,-200,+200,-130,+150`

CODE

Function

Returns the last value of a G code. Code `sel` can have the following values:

fix:	`A, F, I, J, K, R, S, W, X, Y, Z`
integer:	`G, L, M, N`
special:	`C`

Syntax:

`a=CODE(sel)`

Example:

```
F100.5 G02
I+100 J-55.5
PRINT CODE(F),CODE(G),CODE(I),CODE(J)
```

Related commands and functions:

`ON CTRLCODE GOTO|GOSUB`

CONTINUOUS

Command

Switches the continuous mode for groups `ng1`, `ng2`, ... on or off. If the continuous mode is activated, the constant velocity after the acceleration profile is continued as long as movement commands are added (e.g. `LINA`, `LINR`, `ARCA`, `ARCR`). It is

necessary to use the command `NOWAIT ON` first.
Syntax:
`CONTINUOUS ng1[,ng2[,ng3..]] ON|OFF`
Related commands and functions:
`NOWAIT`

CONTOUR

Command
Switches contour acquisition mode for group `ng1`, `ng2`, ... on or off. A contour consists of one or more linear or circular segments. They can be programmed after `CONTOUR ON` with `LINA`, `LINR`, `ARCA` and `ARCR`. Start position is the actual position. The start position can be moved using the `POSA` or `POSR` command. During contour acquisition mode, no movements are performed. After `CONTOUR OFF` the contour can be used with `MOVER` or `MOVEA`.
Syntax:
`CONTOUR ng1[,ng2[,ng3..]] ON|OFF`
Example:
```
CONTOUR ON
LINR 1,0,100
ARCR 1,25,0,-180
CONTOUR OFF
```
Related commands and functions:
`ALENGTH, CTPOS, LENGTH, MOVEA, MOVER, NEWSEG, SETCPOS`

CONTSEG

Command
Used in contour mode to connect two segments with constant velocity, even if segmented mode is selected (`SEGMODE ON`).
Syntax:
`NEWSEG ng`
Related commands and functions:
`CONTOUR, SEGMODE, CONTSEG`

CPOS

Function
Returns the last calculated position of the `n`th axis of group `ng`.
Syntax:
`a=CPOS(ng,n)`
Example:
```
NOWAIT ON
LINA 1,100,200,300
PRINT CPOS(1,1),CPOS(1,2),CPOS(1,3)
```
Related commands and functions:
`APOS, TP, TPOS`

CTPOS

Function
Returns the position of axis `n` of group `ng` at contour position `l`.
Syntax:
`a=CTPOS(ng,n,l)`

Example:
```
CONTOUR ON
LINR 1,100,150
CONTOUR OFF
PRINT CTPOS(1,1,50),CTPOS(1,2,50)
```
Related commands and functions:
`CONTOUR, SETCPOS, LENGTH, ALENGTH`

`CVEL`

Function
Returns the current maximum velocity programmed with `VEL` respectively `POSVEL` of group `ng` respectively axis `n` of group `ng` (`n=1`: x axis, `n=2`: y axis, `n=3`: z axis).
Syntax 1: `VEL`
`a=CVEL(ng)`
Syntax 2: `POSVEL`
`a=CVEL(ng,n)`
Related commands and functions:
`AVEL, VEL, POSVEL`

`DEC`

Command
Sets the deceleration for linear and circular movements (in user units/s²) of group `ng` to value `a`. The command can be executed during a movement and the new value is updated immediately.
Syntax:
`DEC ng,a`
Example:
`DEC 1,200`
Related commands and functions:
`ACC, HDEC, VEL, TRAFO`

`DEFAULTCODE`

Command
If a line contains G code coordinates, but no G code, G01 is automatically executed, if `DEFAULTCODE ON` was used. Otherwise, only the flags are set, but no command is executed.
Syntax:
`DEFAULTCODE ON|OFF`
Related commands and functions:
`G-Codes`

`FMAX`

Command
Sets the maximum value (in user unit/s) which can be set by the `Fa` code.
Syntax:
`FMAX a`
Example:
`FMAX 200`
Related commands and functions:
`G-Codes`

G-Codes

Command (preliminary)

G code are CNC commands, which have a special syntax. They are partly implemented in this controller, but all can be emulated using event processing and BASIC commands.
A G code consists of a character and a number (variables are not allowed). For certain G codes only integers (`n`) are allowed, the other can handle fix (`f`) values.

Syntax:

```
G00 Xf Yf Zf
```
position move

```
G01 Xf Yf Zf
```
linear move

```
G02       [G9n] If Jf
G02 Xf Yf [G9n] If Jf
G02 Xf Zf [G9n] If Kf
G02 Yf Zf [G9n] Jf Kf
```
circular move (clockwise)

```
G03       [G9n] If Jf
G03 Xf Yf [G9n] If Jf
G03 Xf Zf [G9n] If Kf
G03 Yf Zf [G9n] Jf Kf
```
circular move (counter-clockwise)

```
G09 Af Wf
```
polar coordinates

```
G52 Xf Yf Zf
```
move to reference position

```
G53
```
reset coordinate transformation

```
G54 Xf Yf Zf
```
move coordinate system

```
G55 Xf Yf Zf Wf [If Jf]
```
move and rotate coordinate system (relative)

```
G56 Xf Yf Zf Wf If Jf
```
move and rotate coordinate system (absolute)

```
G90
```
absolute coordinates

```
G91
```
relative coordinates

```
M71
```
relative angle

`M72`
absolute angle

`Vn`
wait `n` ms

GETV

Function
Returns the actual value of an axis parameter.
Syntax:
`a=GETV(na,mode)`
Mode:
DEADBAND
Returns the dead band of the PID controller.
DIFF
Returns the differential value of the PID controller.
FACTOR
Returns the scale factor between `input1` and `input2` in the `PIDD`-mode.
FEEDFWD
Not yet supported.
FORCE
Returns the current controller output (value between -100 and 100).
IMAX
Returns the value of the maximum average current of the amplifier.
INT
Returns the integral value of the PID controller.
IPEAK
Returns the value of the peak current of the amplifier, if `na` is a servo axis.
ISTANDBY
Returns the value of the standby current, if `na` is a stepper axis.
ITIME
Returns the value of the peak current time limit.
MAXFORCE
Returns the value of the maximum PID controller output.
MAXDIFF
Returns the maximum difference between current and desired position.
MOTOR
Returns the actual controller state.
OFFSET
Returns the offset of the PID controller output.
POL
Returns the counting direction of the incremental encoder inputs.
PROP
Returns the proportional value of the PID controller.

`PWMOFFSET`
Returns the PWM offset value.
`PWMPOL`
Returns the PWM output polarity.
`RAPOS`
Returns the actual position of the incremental encoder.
`RTPOS`
Returns the desired position of the axis (raw value, in quadcounts).
Related commands and functions:
`SET`

`GROUP`

Command
Combines one or more axes to a group. Only within a group linear and circular movements are possible. The order determines which axes are X, Y and Z.
Syntax:
`GROUP ng,na1,[na2[,na3]]`
Related commands and functions:
`HDEC, POSHDEC, STOP`

`HALT`

Command
The movements of groups `ng1`, `ng2,`... are terminated using the deceleration set with `HDEC` or `POSHDEC` respectively .
Syntax:
`HALT ng1,[ng2[,ng3..]]`
Related commands and functions:
`HDEC, POSHDEC, STOP`

`HDEC`

Command
Sets the deceleration which is used if the HALT command is executed during a linear or circular move, to the value `a` (in user units/s^2). The command can be executed during a movement and the new value is updated immediately.
Syntax:
`HDEC ng,a`
Example:
`HDEC 1,1000`
Related commands and functions:
`ACC, DEC, VEL, TRAFO, HALT`

`INC`

Function
Returns the actual value of incremental decoder `n`.
Syntax:
`a=INC(n)`
Example:
`PRINT INC(1)`
Related commands and functions:
`INCFILTER`

INCFILTER

Command
Sets an input filter for incremental decoder `n1`. The filter parameter `n2` sets the minimum change of position. This filter may increase control loop stability, but reduces accuracy.
Syntax:
```
INCFILTER n1,n2
```
Example:
```
INCFILTER 1,1
```
Related commands and functions:
`INC`

INDEX

Command
Switches the event processing for index input events on or off. `INDEX RISING` detects a rising edge, `INDEX FALLING` detects a falling edge. Index events are available only on some controllers.
Syntax:
```
INDEX RISING|FALLING|OFF
```
Related commands and functions:
`ON xx GOTO|GOSUB`

JUMP

Command
Sets a new command position `n` for axis `na`. This command should only be used in special cases. New positions can be set with less than 512 Hz for stepper motors and less than 1024 Hz for servo motors.
Syntax:
```
JUMP na,n
```
Example:
```
JUMP 1,50
```

LENGTH

Function
Returns the total length of the contour of group `ng`.
Syntax:
```
a=LENGTH(ng)
```
Example:
```
PRINT LENGTH(1)
```
Related commands and functions:
`ALENGTH, CONTOUR`

LINA

Command
Performs a linear movement starting at the current position to the absolute coordinates `ax,ay,az`.
Syntax 1:
```
LINA ng,ax
```
Syntax 2:
```
LINA ng,[ax][,[ay]]
```
Syntax 3:
```
LINA ng,[ax][,[ay]][,az]
```

Example:
`LINA 1,100.0,50.0,100.0`
Related commands and functions:
`LINR`

LINR

Command
Performs a linear movement starting at the current position to the relative coordinates `ax,ay,az`.
Syntax 1:
`LINR ng,ax`
Syntax 2:
`LINR ng,[ax][,[ay]]`
Syntax 3:
`LINR ng,[ax][,[ay]][,az]`
Example:
`LINR 1,-100.0,+50.0,-100.0`
Related commands and functions:
`LINA`

MAXDIFF

Command
Switches the event processing for the supervision of the maximum difference between current and desired position on or off. The maximum value can be set with the command `SET na,MAXDIFF,a`. Controlling and amplifier of axis `na` are automatically switched off, if parameter `AUTO` is used. This automatic mode is switched off with parameter `MANUAL`.
Syntax:
```
MAXDIFF ON|OFF
MAXDIFF AUTO|MANUAL
```
Related commands and functions:
`SET, ON xx GOTO|GOSUB`

MAXFORCE

Command
Switches the event processing for the supervision of the maximum PID controller output on or off. The maximum value can be set with the command `SET na,MAXFORCE,a`.
Syntax:
`MAXFORCE ON|OFF`
Related commands and functions:
`SET, ON xx GOTO|GOSUB`

MAXSEGMENTS

Command
Sets the number of segments which are checked by the `AUTOVEL` function.
Syntax:
`MAXSEGMENTS ng,n`
Related commands and functions:
`AUTOVEL, RSEGMENTS`

MODE

Command

Sets motor and amplifier parameters. There are two syntax versions for stepper and for servo motors.

Syntax for servo motors:

`MODE na,PID,input,output`

`input` can be one of the following

`INC(1|2|3)`	incremental counter 1,2,3
`ADIN(n)`	analog input (depending on equipment)

`output` can be:

`PWMS(1|2|3)`	PWM output with sign
`PWMD(1|2|3)`	differential PWM
`DAOUT(n)`	analog output (depending on equipment)
`UDAOUT(n)`	unipolar analog output + sign (depending on equipment)

Syntax for servo motors with 2 encoders:

`MODE na,PIDD,input1,input2,output`

In this mode proportional and integral part of the PID algorithm are computed with `input2` (e.g. linear encoder), the differential part is computed with `input1` (e.g. rotary encoder). The scale factor between `input1` and `input2` is set with command `SET na,FACTOR,a`.

`input` can be one of the following

`INC(1|2|3)`	incremental counter 1,2,3
`ADIN(n)`	analog input (depending on equipment)

`output` can be:

`PWMS(1|2|3)`	PWM output with sign
`PWMD(1|2|3)`	differential PWM
`DAOUT(n)`	analog output (depending on equipment)
`UDAOUT(n)`	unipolar analog output + sign (depending on equipment)

Syntax for stepper motors:

`MODE na,SM,output`

`output` can be:

`STEPDIR(1|2|3)`	step and direction output 1,2,3
`PATTERN(1|2|3)`	pattern generator 1,2,3

Motor and amplifier parameters can be set only once after reset or power up.

Example:

```
MODE 1,PID,INC(1),DAOUT(1)
MODE 2,PID,INC(2),PWMS(2)
MODE 3,SM,PATTERN(1)
```

Related commands and functions:

`SET`

MOTOR

Command

Switches the amplifiers and the position controlling on or off.

Syntax:
`MOTOR [na1][,na2][,na3..] ON|OFF`
Related commands and functions:
`SET, ON xx GOTO|GOSUB`

MOVEA

Command
Starts a contour move to absolute contour position `l`. First a position move to the actual contour position is performed, then the contour move to the desired position starts.
Syntax:
`MOVEA ng,l`
Example:
`MOVEA 1,120.0`
Related commands and functions:
`CONTOUR, MOVER`

MOVER

Command
Starts a contour move to relative contour position `l`. First a position move to the actual contour position is performed, then the contour move to the desired position starts.
Syntax:
`MOVER ng,l`
Example:
`MOVER 1,120.0`
Related commands and functions:
`CONTOUR, MOVEA`

MOVING

Function
Returns `TRUE`, if group `ng` is moving, `FALSE` if not.
Syntax:
`a=MOVING(ng)`
Related commands and functions:
`NOWAIT, WAIT`

NEWCODE

Function
Returns `TRUE`, if a new value was set with G code `sel`. Code `sel` can be:

fix:	`A, F, I, J, K, R, S, W, X, Y, Z`
integer:	`G, L, M, N`

Syntax:
`a=NEWCODE(sel)`
Related commands and functions:
`CODE, G-Codes, RESETCODEFLAGS`

NEWSEG

Command
Causes a stop after the last contour segment during contour move.
Syntax:

```
NEWSEG ng
```

Related commands and functions:

`CONTOUR, CONTSEG, SEGMODE`

NOTAUS

Command

Switches the event processing in case of a emergency stop (NOTAUS) on or off, sets the mode of control or causes an emergency stop. In the internal mode a power relay is switched on by the `START` command or by pressing the **START** key. It is switched off by the `NOTAUS` command or by pressing the **STOP** key. In the external mode the external relay controller is supervised. The **START** and **STOP** keys can be used if they are connected.

Syntax:

```
NOTAUS INTERN|EXTERN     ; mode
NOTAUS                   ; cause emerg. stop
NOTAUS ON|OFF            ; event proc. on/off
```

Related commands and functions:

`KEYTIME, START`

NOWAIT

Command

Determines, whether the interpreter waits until a movement is finished (`NOWAIT OFF`) or whether following commands can be executed (`NOWAIT ON`).

Syntax:

```
NOWAIT [ng1][,ng2][,ng3..] ON|OFF
```

Related commands and functions:

`MOVING, WAIT`

ON

Command

Sets the program line to which is jumped if the event `event` occurs. Use only `GOSUB` for the events `AMPERROR`, `MAXDIFF` und `MAXFORCE`. If a `GOTO` branch is desired, use `GOSUB` and clear the stack element with `POPSTACK`, after the source of the events has been cleared (e.g. with `MOTOR OFF`).

`event` can be one of the following:

`AMPERROR(na)`

The amplifier error signal has the programmed level (see `AMPERROR(na)`).

`CODE(sel)`

This event occurs, if a G code with character `sel` (`G`, `E` (end of line), `D`, `M`, `T`) is interpreted.

`INDEX(na)`

The programmed edge on index input `na` was detected.

`MAXDIFF(na)`

This event occurs if the difference between desired and current position is greater than the programmed value.

`MAXFORCE(na)`

This event occurs if the PID controller output is greater than the programmed value.

NOTAUS

The NOTAUS detection signal S+ has changed its level from +24 V to 0 V.

START

The NOTAUS detection signal S+ has changed its level from 0 V to +24 V.

TOOLOFF

This event occurs if a `G01`, `G02` or `G03` command is followed by a `G00` command.

TOOLON

This event occurs if a `G00` command is followed by a `G01`, `G02` or `G03` command.

Syntax:

```
ON event GOTO|GOSUB n
```

Related commands and functions:

`AMPERROR, G-Codes, INDEX, MAXDIFF, MAXFORCE, SET, NOTAUS, START, TOOLOFF, TOOLON`

POSA

Command

Performs a position move of group `ng` to the given absolute coordinates.

Syntax:

```
POSA ng,[ax][,[ay][,az]]
```

Example:

```
POSA 1,100.5,,25
```

Related commands and functions:

`POSACC, POSDEC, POSHDEC, POSR, POSVEL`

POSACC

Command

Sets the positioning acceleration (in user units/s²) of the axes of group `ng`. The command can be executed during a movement and the new value is updated immediately.

Syntax 1:

```
POSACC, ng,ax
```

Syntax 2:

```
POSACC ng,[ax][,[ay]
```

Syntax 3:

```
POSACC ng,[ax][,[ay][,az]]
```

Example:

```
POSACC 1,1000,1000,100
```

Related commands and functions:

`POSA, POSDEC, POSHDEC, POSR, POSVEL`

POSDEC

Command

Sets the positioning deceleration (in user units/s²) of the axes of group `ng`. The command can be executed during a movement and the new value is updated immediately.

Syntax 1:

`POSDEC, ng,ax`

Syntax 2:

`POSDEC ng,[ax][,[ay]`

Syntax 3:

`POSDEC ng,[ax][,[ay][,az]]`

Example:

`POSDEC 1,1000,1000,100`

Related commands and functions:

`POSA, POSACC, POSHDEC, POSR, POSVEL`

POSHDEC

Command

Sets the deceleration, which is used if the HALT command is executed during a position move, to the value `a` (in user units/s²). The command can be executed during a movement and the new value is updated immediately.

Syntax 1:

`POSHDEC, ng,ax`

Syntax 2:

`POSHDEC ng,[ax][,[ay]`

Syntax 3:

`POSHDEC ng,[ax][,[ay][,az]]`

Example:

`POSHDEC 1,5000,5000,2000`

Related commands and functions:

`POSA, POSACC, POSDEC, POSR, POSVEL`

POSR

Command

Performs a position move of group `ng` to the given relative coordinates.

Syntax:

`POSA ng,[ax][,[ay][,az]]`

Example:

`POSA 1,100.5,,25`

Related commands and functions:

`POSA, POSACC, POSDEC, POSHDEC, POSVEL`

POSVEL

Command

Sets the positioning velocity (in user units/s) of the axes of group `ng`. The command can be executed during a movement and the new value is updated immediately.

Syntax 1:

`POSVEL ng,ax`

Syntax 2:

`POSVEL ng,[ax][,[ay]`

Syntax 3:

`POSVEL ng,[ax][,[ay][,az]]`

Example:

`POSVEL 1,100,100,20`

Related commands and functions:

`POSACC, POSDEC, POSHDEC`

POWERON

Function

Returns the status of the emergency stop circuit. The return value is `TRUE`, if the **START** LED is green (amplifier power on), and `FALSE`, if it is red (amplifier power off).

Syntax:

```
a=POWERON
```

Example:

```
PRINT POWERON
```

Related commands and functions:

`START, NOTAUS`

RESETCODEFLAGS

Command

Resets all G code flags.

Syntax:

```
RESETCODEFLAGS
```

Related commands and functions:

`G-Codes, CODE, NEWCODE`

RLENGTH

Function

Returns the remaining length while in continuous mode.

Syntax:

```
a=RLENGTH(ng)
```

Related commands and functions:

`CONTINUOUS, RSEGMENTS`

RSEGMENTS

Function

Returns the number of segments while in continuous mode.

Syntax:

```
a=RSEGMENTS(ng)
```

Related commands and functions:

`AUTOVEL, CONTINUOUS, MAXSEGMENTS, RLENGTH`

SCOPE

Command

Puts every 0.977 ms (1024 Hz) the actual position of axis `na` into array `array(0)`, until the array is filled. The actual position is the incremental encoder position as obtained by `GETV(na,RAPOS)`. Array type can be char or integer.

Syntax:

```
SCOPE na,array(0)
```

Example:

```
DIM testarray(1000) AS INTEGER
SCOPE 1,testarray(0): SLEEP 50: JUMP 1,100
SLEEP 950
FOR i=1 to 1000: PRINT testarray(i): NEXT
```

Related commands and functions:

`JUMP`

SEGMODE

Command

Selects the default segmentation mode during contour mode. With `SEGMODE ON` the movement stops between every segment, with `SEGMODE OFF` the movement continues with constant velocity. The mode can changed individually for every segment using the `NEWSEG` or `CONTSEG` command.

Syntax:

`SEGMODE ng1[,ng2[,ng3..]] ON|OFF`

Siehe auch:

`ALENGTH, CONTSEG, CTPOS, LENGTH, MOVEA, MOVER, NEWSEG, SETCPOS`

SET

Command

Sets one of several axis controller parameters.

Syntax:

`SET na,mode,a`

Mode:

DEADBAND

Sets the dead band of the PID controller. This is the difference between desired and current position up to which the PID controller output stays zero.

DIFF

Sets the differential value of the PID controller. The range is 0.0 to 32767.0.

FACTOR

Sets the scale factor between `input1` and `input2` in the `PIDD`-mode. `FACTOR`=Res(`input2`)/Res(`input1`), Res(n) is the encoder resolution in inkrements/length unit.

FEEDFWD

Not yet supported.

IMAX

Sets the value of the maximum average current (depending on the amplifier). The range is 0 to 100.

INT

Sets the integral value of the PID controller. The range is 0.0 to 32767.0.

IPEAK

Sets the peak current with amplifier model PW2000. The function is identical to `ISTANDBY`.

ISTANDBY

Sets the value of the standby current, if `na` is a stepper axis. The range is 0 to 100.

ITIME

Sets the value of the peak current time limit. The range is 0 to 100.

MAXFORCE

Sets the value of the maximum PID controller output. The range is 0 to 100.

MAXDIFF

Sets the maximum difference between current and desired position. The range is 0 to 32767 (in quadcounts).

MOTOR

Switches amplifier of axis `na` on or off. Parameter `a` can be `ON` or `OFF`.

OFFSET

Sets the offset of the PID controller output. The range is -100 to +100. This can e.g. be used for gravity compensation.

PHASEA, PHASEB, PHASEC, PHASED

Sets the 16 bit patterns for the pattern generator. Axis `na` has to be a stepper motor.

POL

Sets the counting direction of the incremental encoder inputs. `a` can be `TRUE` or `FALSE`.

PROP

Sets the proportional value of the PID controller. The range is 0.0 to 32767.0.

PWMOFFSET

Sets PWM offset. This is useful to compensate the offset of PWM amplifiers. The range is -100 to +100.

PWMPOL

Sets the polarity of the PWM output. `a` can be `TRUE` or `FALSE`.

RAPOS

Sets the actual position of the incremental encoder. This should be used for diagnosics only.

RTPOS

Sets the desired position of the PID controller. This should be used for diagnosics only.

SLFACTOR

Sets the scale factor between master and slave axis.
Slave position=Master position * `FACTOR`. This command should only be used after all axes involved are set to the zero position using `SETPOS`.

WAVEFORM1

Sets the current waveform of phase one of motorcontrollers with micro step amplifiers (CO6100 und CO6500). Parameter `a` is an one dimensional array with size 256 of `FIX`. The range of the arrray values is –1 to +1.

WAVEFORM2

Sets the current waveform of phase two of motorcontrollers with micro step amplifiers (CO6100 und CO6500). Parameter `a` is an one dimensional array with size 256 of `FIX`. The range of the arrray values is –1 to +1.

Related commands and functions:

`GETV`

SETPOS

Command

Sets the position (actual and desired) of group `ng`.

Syntax 1:

`SETPOS ng,ax`

Syntax 2:

`SETPOS ng,ax,ay`

Syntax 3:

`SETPOS ng,ax,ay,az`

Related commands and functions:

`APOS, TPOS, CPOS, TP, TRAFO`

SETCPOS

Command

Sets the current contour position of group `ng` to position `l`.

Syntax:

`SETPOS ng,l`

Related commands and functions:

`CONTOUR, CTPOS, MOVEA, MOVER`

SLAVE

Command

Couples one axis to another. Depending on parameter `APOS` or `TPOS` axis `na1` uses the actual position (`APOS`) or target position (`TPOS`) of axis `na2`. If only `na1` is given the coupling is solved.

Syntax:

```
SLAVE na1,na2,APOS|TPOS
SLAVE na1
```

Example:

`SLAVE 3,2,APOS`

START

Command

Without option the EIN relay is switched on for the programmed time (see `KEYREPEAT`). This is the same as pressing the **START** key. The EIN relay must be enabled with `START ENABLE` first. The options `ON` and `OFF` switch the START event processing on or off.

Syntax:

`START [ON|OFF|ENABLE|DISABLE]`

Related commands and functions:

`KEYREPEAT, NOTAUS`

STATUS

Function

Returns the actual state of `AMPERROR`, `MAXDIFF` or `MAXFORCE`. With `MAXDIFF` and `MAXFORCE`, the return value `TRUE` means that the values preset with `SET` are exceeded. Used with parameter `MOTOR` the function returns the actual state of the axis controller (`TRUE` = motor control is on).

Syntax:

`n=STATUS(na,modus)`

Example:

`PRINT STATUS(1,MAXDIFF)`

Related commands and functions:

`SET`

`STOP`

Command
The movement of groups `ng1`, `ng2`,... is stopped immediately without a deceleration profile.
Syntax:
`STOP ng1,[ng2[,ng3..]]`
Related commands and functions:
`HALT`

`TOOL`

Command
Switches the event processing for `TOOLON` and `TOOLOFF` events on or off. See `ON xx GOTO|GOSUB` command.
Syntax:
`TOOL ON|OFF`
Related commands and functions:
`ON .. GOTO|GOSUB`

`TP`

Command
Lists the actual positions of all axes in all groups.
Syntax:
`TP`
Related commands and functions:
`APOS, CPOS, GROUP, TPOS`

`TPOS`

Function
Returns the desired (i.e. calculated) position of the `n`th axis of group `ng`.
Syntax:
`a=TPOS(ng,n)`
Example:
`PRINT TPOS(1,1),TPOS(1,2),TPOS(1,3)`
Related commands and functions:
`APOS, CPOS, TP`

`TRACKING`

Command
Switches the tracking mode for groups `ng1`, `ng2`,... on or off. In tracking mode positioning commands are finished early, if a new position is given. Acceleration and deceleration profiles are used. Tracking mode is not possible during linear or circular movements.
Syntax:
`TRACKING ng1,[ng2[,ng3..]] ON|OFF`
Related commands and functions:
`POSA, POSR`

`TRAFO`

Command
Sets the matrix for the coordinate transformation of group `ng`.
Transformation for a group of one axis:

```
x‘= x*a
```

Transformation for a group of two axes:

```
x‘= x*ax1 + y*ay1
y‘= x*ax2 + y*ay2
```

Transformation for a group of three axes:

```
x‘= x*ax1 + y*ay1
y‘= x*ax2 + y*ay2
z‘= z*az
```

Syntax 1:

`TRAFO ng,a`

Syntax 2:

`TRAFO ng,ax1,ay1,ax2,ay2`

Syntax 3:

`TRAFO ng,ax1,ay1,ax2,ay2,az`

VEL

Command

Sets the deceleration for linear and circular movements (in user units/s²) of group `ng` to value `a`. The command can be executed during a movement and the new value is updated immediately.

Syntax:

`VEL ng,a`

Related commands and functions:

`ACC, DEC, HDEC, TRAFO`

WAIT

Command

Waits until the movements of groups `ng1`, `ng2`,... are finished.

Syntax:

`WAIT ng1[,ng2[,ng3..]]`

Related commands and functions:

`NOWAIT, MOVING`

3. Appendix A: G-Codes

`G-codes` are CNC commands with a special syntax. The basic commands are implemented, extended commands can be emulated with BASIC subroutines. A `G-code` consists of a character immediately followed by a number. Some `G-codes` accept only integer numbers (`n`), some fix numbers (`f`) too.
`G-codes` use group 1 (see `GROUP` command) for movements. Group 1 must consist of 2 or 3 axes.
In each line, only one code of the types `G00`, `G01`, `G02`, `G03` or `G5n` is allowed, another command, even separated by a colon is not possible. Numerical expressions or variables are not allowed as parameters, only simple integer or fix numbers.
Related commands and functions:
`ON CODE(sel) GOTO|GOSUB, FMAX, CODE(sel), RESETCODEFLAGS, NEWCODE(sel), DEFAULTCODE, ON TOOLON GOTO|GOSUB, ON TOOLOFF GOTO|GOSUB, TOOL`

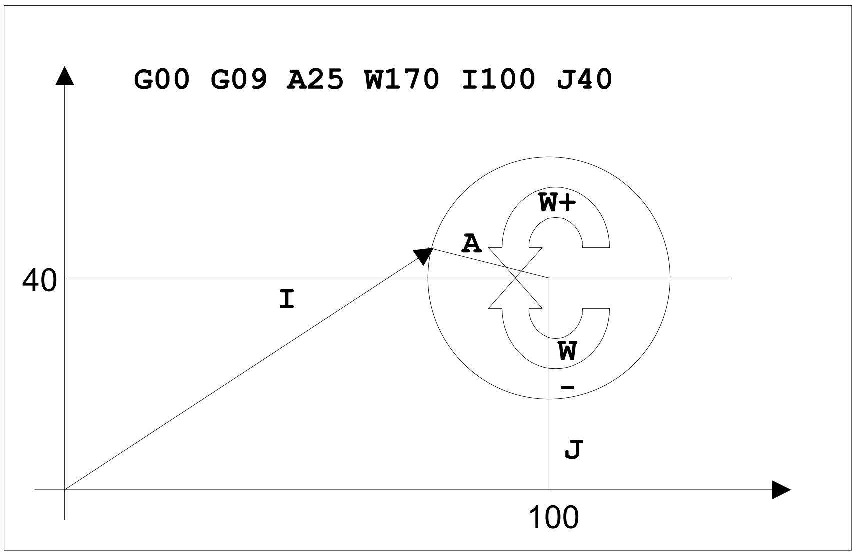
Standard-G-Codes

Ff

Command
Sets the velocity for `G01`, `G02` and `G03`, similar to the `VEL` command. A maximum value can be set with `FMAX`.
Syntax:
```
Ff
```
Example:
```
F50
G02 X20 Y50 F20
```
Similar:
`FMAX, VEL`

G00

Command
Position move to the given coordinates while using the parameters set by `POSVEL`, `POSACC` und `POSDEC`. Relative (`G91`) and absolute (`G90`) coordinates are possible as well as cartesian and polar coordinates (`G09`). Angles are given in degrees and angle=0° is the positive x axis. Positive angles are counterclockwise, negative angles clockwise.
If group 1 consists of 3 axes, the plane for the polar coordinates is defined by the given parameters, that means that only 2 of 3 possible coordinates can be used (e.g. `I` and `J` , `J` and `K`, `K` and `I`)



Definitions:

`Xf`	X coordinate
`Yf`	Y coordinate
`Zf`	Z coordinate
`Af`	radius
`Wf`	angle
`If`	X coordinate of the center using polar coordinates or for circular movements
`Jf`	Y coordinate of the center using polar coordinates or for circular movements
`Kf`	Z coordinate of the center using polar coordinates or for circular movements
`[...]`	optional parameter

Absolute cartesian coordinates:
`X`, `Y` and `Z` are the absolute target coordinates
Syntax:
```
G00 [G90] [Xf] [Yf] [Zf]
```
Example:
```
G00 X100 Z-50
G00 Y-14.5
G00 G90 X10.5
```

Relative cartesian coordinates:
`X`, `Y` and `Z` are the coordinates of the target position relative to the last position.
Syntax:
```
G00 G91 [Xf] [Yf] [Zf]
```
Example:
```
G00 G91 X100 Z-50
G00 G91 Y-14.5
```

Polar coordinates, absolute center, absolute angle:
`I`, `J` and `K` are the absolute center coordinates, `W` is the

absolute angle.

Syntax:

```
G00 G09 [M72] Af Wf [G90] If Jf      ` XY pl.
G00 G09 [M72] Af Wf [G90] Jf Kf      ` YZ pl.
G00 G09 [M72] Af Wf [G90] Kf If      ` ZX pl.
```

Example:

```
G00 G09 A50 W75 I50 J50
G00 G09 M72 A25.5 W33.34 I+20 K-10
```

Polar coordinates, relative center, absolute angle:
`I`, `J` and `K` are the coordinates of the center relative to the last position, `W` is the absolute angle.

Syntax:

```
G00 G09 [M72] Af Wf G91 If Jf        ` XY pl.
G00 G09 [M72] Af Wf G91 Jf Kf        ` YZ pl.
G00 G09 [M72] Af Wf G91 Kf If        ` ZX pl.
```

Example:

```
G00 G09 A50 W75 G91 I10 J10
G00 G09 M72 A25.5 W33.34 G91 I-20.3 J0
```

Polar coordinates, absolute center, relative angle:
`I`, `J` and `K` are the absolute center coordinates, `W` is the angle relative to the last angle.

Syntax:

```
G00 G09 M71 Af Wf [G90] If Jf        ` XY pl.
G00 G09 M71 Af Wf [G90] Jf Kf        ` XZ pl.
G00 G09 M71 Af Wf [G90] Kf If        ` ZX pl.
```

Example:

```
G00 G09 M71 A50 W15 I10 J10
G00 G09 M71 A25.5 W33.34 G90 I-20.3 J0
```

Polar coordinates, relative center, relative angle:
`I`, `J` and `K` are the coordinates of the center relative to the last position, `W` is the angle relative to the last angle.

Syntax:

```
G00 G09 M71 Af Wf G91 If Jf          ` XY pl.
G00 G09 M71 Af Wf G91 Jf Kf          ` YZ pl.
G00 G09 M71 Af Wf G91 Kf If          ` ZX pl.
```

Example:

```
G00 G09 M71 A50 W15 G91 I10 J10
G00 G09 M71 A25.5 W33.34 G91 I-20.3 J0
```

Related commands and functions:
`POSACC, POSDEC, POSVEL, G01`

G01

Command

Linear move to the given coordinates. Acceleration and deceleration are according to the values given with `ACC` and `DEC`. The velocity is set with the `F`-Code. A maximum value must be set with the `FMAX` command. The coordinate options are the same as for the `G00` command.

Related commands and functions:
`ACC, DEC, VEL, FMAX, Ff, G00`

G02

Command
Circular move clockwise to the given target coordinates. Acceleration and deceleration are according to the values given with `ACC` and `DEC`. The velocity is set with the `F`-Code.
If group 1 has 3 axes, the selected coordinates define the plane in which the circular movement is performed, so only 2 of 3 coordinates can be used (e.g. `I` and `J`, `J` and `K`, `K` and `I`, resp. `X` and `Y`, `Y` and `Z`, `Z` and `X`).

Definitions:

`Xf`	X coordinate
`Yf`	Y coordinate
`Zf`	Z coordinate
`Af`	radius
`Wf`	angle
`If`	X coordinate of the center using polar coordinates or for circular movements
`Jf`	Y coordinate of the center using polar coordinates or for circular movements
`Kf`	Z coordinate of the center using polar coordinates or for circular movements
`[...]`	optional parameter

absolute center, full circle:
`I`, `J` and `K` define the center in absolute coordinates.
Syntax:
```
G02 [G90] If Jf                    ' XY plane
G02 [G90] Jf Kf                    ' YZ plane
G02 [G90] Kf If                    ' ZX plane
```
Example:
```
G02 I-50 J50
G02 G90 I22.2 J22.2
```

relative center, full circle:
`I`, `J` and `K` define the center relative to the last position.
Syntax:
```
G02 G91 If Jf                      ' XY plane
G02 G91 Jf Kf                      ' YZ plane
G02 G91 Kf If                      ' ZX plane
```
Example:
```
G02 G91 I-50 J50
G02 G91 I22.2 J22.2
```

cartesian coordinates, absolute center:
`X`, `Y` and `Z` define the target position in absolute coordinates. `I`, `J` and `K` define the center in absolute coordinates.
Syntax:

```
G02 Xf Yf [G90] If Jf          ' XY plane
G02 Yf Zf [G90] Jf Kf          ' YZ plane
G02 Zf Xf [G90] Kf If          ' ZX plane
```

Example:

```
G02 X10 Y10 I50 J50
G02 X-30 Y20 G90 I22.2 J22.2
```

cartesian coordinates, relative center:
`X`, `Y` and `Z` define the target position in absolute coordinates. `I`, `J` and `K` define the center relative to the last position.
Syntax:

```
G02 Xf Yf G91 If Jf            ' XY plane
G02 Yf Zf G91 Jf Kf            ' YZ plane
G02 Zf Xf G91 Kf If            ' ZX plane
```

Example:

```
G02 X10 Y10 G91 I-40 J50
G02 X-30 Y20 G91 I22.2 J22.2
```

Polar coordinates, absolute center, absolute angle:
Angle `W` is the final angle of the circular move, given as an absolute value. `I`, `J` and `K` define the center in absolute coordinates.
Syntax:

```
G02 G09 [M72] Wf [G90] If Jf       ' XY pl.
G02 G09 [M72] Wf [G90] Jf Kf       ' YZ pl.
G02 G09 [M72] Wf [G90] Kf If       ' ZX pl.
```

Example:

```
G02 G09 W100 I50 J50
G02 G09 M72 W-20 G90 I22.2 J22.2
```

Polar coordinates, absolute center, relative angle:
Angle `W` is the final angle of the circular move relative to the last angle. `I`, `J` and `K` define the center in absolute coordinates.
Syntax:

```
G02 G09 M71 Wf [G90] If Jf         ' XY pl.
G02 G09 M71 Wf [G90] Jf Kf         ' YZ pl.
G02 G09 M71 Wf [G90] Kf If         ' ZX pl.
```

Example:

```
G02 G09 M71 W100 I50 J50
G02 G09 M71 W-20 G90 I22.2 J22.2
```

Polar coordinates, relative center, absolute angle:
Angle `W` is the final angle of the circular move, given as an absolute value. `I`, `J` and `K` define the center relative to the last position.
Syntax:

```
G02 G09 [M72] Wf G91 If Jf         ' XY pl.
G02 G09 [M72] Wf G91 Jf Kf         ' YZ pl.
G02 G09 [M72] Wf G91 Kf If         ' ZX pl.
```

Example:

```
G02 G09 W100 G91 I50 J50
```

```
G02 G09 M72 W-20 G91 I22.2 J22.2
```

Polar coordinates, relative center, relative angle:
Angle `W` is the final angle of the circular move relative to the last angle. `I`, `J` and `K` define the center relative to the last position.
Syntax:
```
G02 G09 M71 Wf G91 If Jf              ‵ XY-Eb.
G02 G09 M71 Wf G91 Jf Kf              ‵ YZ-Eb.
G02 G09 M71 Wf G91 Kf If              ‵ ZX-Eb.
```
Example:
```
G02 G09 M71 W100 G91 I50 J50
G02 G09 M71 W-20 G91 I22.2 J22.2
```
Related commands and functions:
`FMAX, VEL, G03`

G03

Command
Circular movement counterclockwise. Coordinates and syntax is like `G02` command.
Related commands and functions:
`FMAX, VEL, G02`

G53

Command
Clear all coordinate shifting and rotating.
Syntax:
```
G53
```
Related commands and functions:
`G54, G55, G56`

G54

Command
Shifts the coordinate system independently form the `SETPOS` command. Valid only for G-codes. The actual position is now the position with coordinates `X`, `Y` and `Z`.
Syntax:
```
G54 Xf Yf Zf
```
Related commands and functions:
`G53, G55, G56`

G55

Command
Shifts and rotates the coordinate system independently from the `SETPOS` and `TRAFO` command. Valid only for G-codes. The coordinate system is rotated relatively by the angle `W` and shifted relatively by `X`, `Y` and `Z`.
Syntax:
```
G55 Xf Yf Zf Wf
```
Related commands and functions:
`G53, G54, G56`

G56

Command
Shifts and rotates the coordinate system independently from

the `SETPOS` and `TRAFO` command. Valid only for G-codes. The coordinate system is rotated absolutely to the angle `W` and shifted absolutely to the coordinates `X`, `Y` and `Z`.
Syntax:
`G56 Xf Yf Zf Wf`
Related commands and functions:
`G53, G54, G55`

M71

Command
The angle in this line is a relative angle.
Syntax:
`M71`
Related commands and functions:
`G00, G01, G02, G03`

M72

Command
The angle in this line is an absolute angle.
Syntax:
`M72`
Related commands and functions:
`G00, G01, G02, G03`

M80

Command
Clears all mirror functions.
Syntax:
`M80`
Related commands and functions:
`G53, G54, G55, G56, M81, M82, M83, M84`

M81

Command
Sets the mirror function for `Y` axis, resp. changes sign of the `X` and `I` coordinates.
Syntax:
`M81`
Related commands and functions:
`G53, G54, G55, G56, M80, M82, M83, M84`

M82

Command
Sets the mirror function for `X` axis, resp. changes sign of the `Y` and `J` coordinates.
Syntax:
`M82`
Related commands and functions:
`G53, G54, G55, G56, M80, M81, M83, M84`

M83

Command
Sets the mirror function for `XY` plane, resp. changes sign of the `Z` and `K` coordinates.

Syntax:
`M83`
Related commands and functions:
`G53, G54, G55, G56, M80, M81, M82, M84`

M84

Command
Changes signs of the `X` and `I` coordinates and the `Y` and `J` coordinates.
Syntax:
`M84`
Related commands and functions:
`G53, G54, G55, G56, M80, M81, M82, M83`

ON CODE

Command
Sets the program line to which is jumped if the event `CODE(sel)` occurs. This event is caused by a G-Code with character `sel` (`G`, `E` (end of line), `D`, `M`, `T`)

CODE(G)
A G-Code is processed.

CODE(E)
The end of the command line is reached.

CODE(D)
A D-Code D is processed.

CODE(M)
A M-Code is processed.

CODE(T)
A T-Code is processed.

Syntax:
`ON CODE(sel) GOTO|GOSUB n`
Example:
```
..    Init
53110 ON CODE(G) GOSUB 72000
53120 ON CODE(E) GOSUB 73000
..
      main program
..
..    Event Subroutines
72000 '----- G-Code -------------------------
72010 uv_sel=CODE(g)
72020 SELECT CASE uv_sel
72030   CASE 40: LED_1 ON: '  direkte Ausf.
72040   CASE 41: G41flag=1: ' indir. Ausf.
72100 END SELECT
72999 RETURN

73000 '----- EOL ----------------------------
73010 IF G41flag=1 THEN GOTO 74000: END IF
```

```
73999 RETURN

74000 '----- G41 ---------------------------
74010 IF NEWCODE(Z) THEN POSA 2,CODE(Z): END IF
74020 RESETCODEFLAGS
74999 RETURN
```

Related commands and functions:

`ON, TOOLOFF, TOOLON`

Vn

Command

Waits `n` seconds.

Syntax:

`Vn`

4. Appendix B: Programming hints for special hardware

Programming of microstepping controllers:

The microstepping amplifiers in the controller models CO6100, CO6500 and CO6150 have to be programmed with the desired current profile. In general a sine and cosine profile is used. Two arrays of fix values and a length of 256 are filled with the appropriate values of a full cycle (4 full steps). A resolution of 1/64 step can be obtained. The current values have to be between -1 and +1 for CO6100 and CO6500 and between 0 and 1 for CO6150.
The profile for phase 1 is programmed with `SET na,WAVEFORM1,sinarray[0]`, for phase two with `SET na,WAVEFORM2,cosarray[0]`. The current polarity is programmable unrestrictedly for CO6100 and CO6500, but predefined for CO6150. With CO6150 `waveform1` is positive from 1 to 128 and negative from 129 to 256 (sine). `waveform2` is positive from 1 to 64 and from 193 to 256, and negative from 65 to 192 (cosine).
The phase current can be calculated from the programmed array value multiplied with the maximum amplifier current multiplied with the percentage given by `SET na,IMAX,a` und `SET na,ISTANDBY,a`. Maximum current for CO6100 and CO6500 is 4 A. Maximum current for CO6150 can be set with a potentiometer from 1.2 A to 4.8 A. For thermal reasons 3 A should not be exceeded in the long run.

Example for CO6100 and CO6500:

```
MODE 1,SM,STEPDIR(1)
DIM sinwave(256) AS FIX
DIM coswave(256) AS FIX
FOR i = 1 TO 256: sinwave(i)=SIN((i-1)*2*PI/256): NEXT
FOR i = 1 TO 256: coswave(i)=COS((i-1)*2*PI/256): NEXT
SET 1,WAVEFORM1,sinwave(0)
SET 1,WAVEFORM2,coswave(0)
SET 1,ISTANDBY,40
SET 1,IMAX,70
```

Example for CO6150:

```
MODE 1,SM,STEPDIR(1)
DIM sinwave(256) AS FIX
DIM coswave(256) AS FIX
FOR i = 1 TO 256: sinwave(i)=ABS(SIN((i-1)*2*PI/256)): NEXT
FOR i = 1 TO 256: coswave(i)=ABS(COS((i-1)*2*PI/256)): NEXT
SET 1,WAVEFORM1,sinwave(0)
SET 1,WAVEFORM2,coswave(0)
SET 1,ISTANDBY,40
SET 1,IMAX,70
```

More axes can share the same array. For precision moves `IMAX` and `ISTANDBY` should be the same.

Programming hints for CO6300

The amplifiers of CO6300 have preprogrammed current profiles (sine and cosine, contact factory for other profiles). An array is not necessary. The resolution is 128 steps per full phase, which corresponds to a resolution of 1/32 step.

Programming example for CO6300:

```
MODE 1,SM,STEPDIR(1)
SET 1,ISTANDBY,40
SET 1,IMAX,70
```

Amplifier PW2000

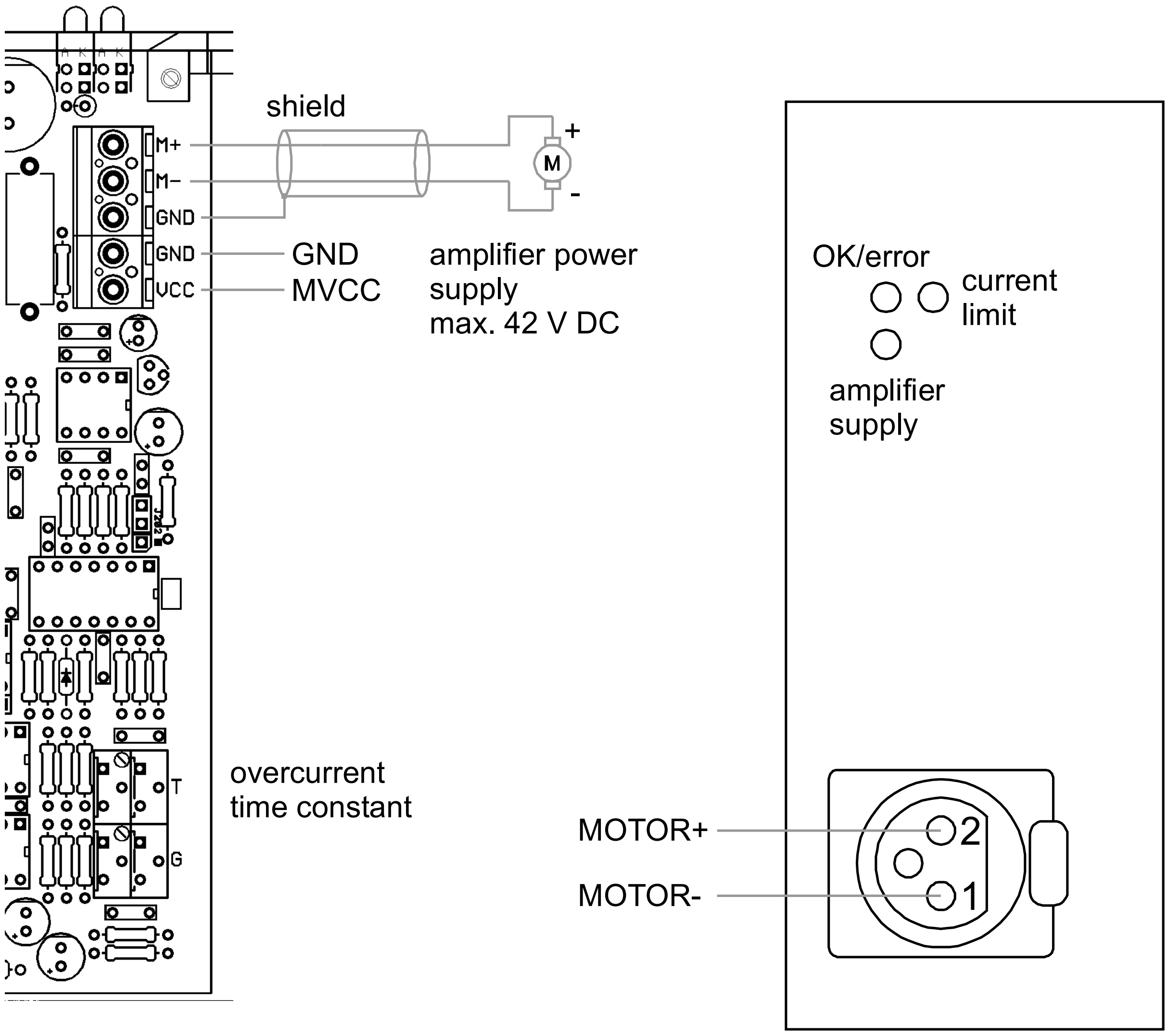
The servo amplifier PW2000 is a PWM amplifier with current monitoring. The average maximum current is limited to the value set with `SET na,IMAX` (% of 10 A). For a certain time (set with trimmer T) the current can be greater than `IMAX`, but is limited to the value set with `SET na,IPEAK` (% of 20 A). Don't use values greater then 90% for `IPEAK`. In case of a short circuit the amplifier is switched off. The error mode can be reset by the `MOTOR na OFF` command followed by `MOTOR na ON`.

PWM frequency:	ca. 20 kHz
max. amplifier voltage:	42 V DC
max. peak motor current:	18 A
max. average motor current:	10 A

Motor supply and controller supply are separated galvanically.

Configuration parameters:

PWM polarity:	`SET na,PWMPOL,TRUE`
max average current:	`SET na,IMAX,a`
	`a`=0..100, 100 corrensponds to 10 A
max. peak current:	`SET na,ISTANDBY,a`
	`a`=0..90, 90 corrensponds to 18 A



Jogdial programming

The jogdial is connected to an incremental decoder. The position is read in during a timer routine and cursor-up and cursor-down character sequences are generated. The character sequence for cursor-up is ESC [A and for cursor-down ESC [B.
The keyboard device can be opened for writing and characters written to the devices can immediately read back form the device, e.g. while using the `GETFILENAME$` function.
The following program generates one cursor-up resp. one cursor-down sequence every four increments.

Initializing:

```
  950 OPEN "KEY:" FOR OUTPUT AS #2
  960 wheelscale=4
  970 wheelpos=INC(1)/wheelscale
  980 lastwheelpos=INC(1)/wheelscale
  990 wheelflag=0
1000 ON TIMER(50) GOSUB 20000: TIMER ON
1010 ...
```

Activate for a menu selection:

```
3000 wheelflag=1
3010 menselect=MENU("CONFIG",menlist(0))
3020 wheelflag=0
```

Timer subroutine:

```
20000 '----------------- TIMER ----------------------------
20010 IF wheelflag=0 THEN GOTO 20200: END IF
20020 wheelpos=INC(1)/wheelscale
20030 wheeldiff=wheelpos-lastwheelpos
20040 IF wheeldiff<0 THEN GOTO 20120: END IF
20050 wheeldiff=CINT(wheeldiff-0.010)
20060 IF wheeldiff=0 THEN GOTO 20200: END IF
20070 lastwheelpos=wheelpos
20080 FOR i=1 TO wheeldiff
20090   PRINT #2,CHR$(27);CHR$(91);CHR$(65);
20100 NEXT i
20110 GOTO 20200
20120 wheeldiff=CINT(ABS(wheeldiff)-0.010)
20130 IF wheeldiff=0 THEN GOTO 20200: END IF
20140 lastwheelpos=wheelpos
20150 FOR i=1 TO wheeldiff
20160   PRINT #2,CHR$(27);CHR$(91);CHR$(66);
20170 NEXT i
20200 ' other timer subroutines
20210 ...
20900 RETURN
```

5. **Appendix C:** Inputs and outputs

Inputs

#	CO2200	CO4200	CO6100	CO6500	CO5400
1	IN1	IN1 (1)	IO1	IO1	IN1 (1)
2	IN2	IN2 (1)	IO2	IO2	IN2 (1)
3	IN3	IN3 (1)	IO3	IO3	IN3 (1)
4	IN4	IN4 (1)	IO4	IO4	IN4 (1)
5	IN5	IN5 (1)	IO5	IO5	IN5 (1)
6	IN6	IN6 (1)	IO6	IO6	IN6 (1)
7	IN7	IN7 (1)	IO7	IO7	IN7 (1)
8	IN8	IN8 (1)	IO8	IO8	IN8 (1)
9	IO1	IN9 (2)	IN9	IN9	REF1 (2)
10	IO2	IN10 (2)	IN10	IN10	REF2 (2)
11	IO3	IN11 (2)	IN11	IN11	REF3 (2)
12	IO4	IN12 (2)	IN12	IN12	REF4 (2)
13	IO5	IN13 (2)	IN13	IN13	S+ (2)
14	IO6	IN14 (2)	IN14	IN14	FAULT1 (2)
15	IO7	IN15 (2)	IN15	IN15	FAULT2 (2)
16	IO8	IN16 (2)	IN16	IN16	EXT (2)
17	PI0	IN17 (3)			INDEX1 (3)
18	PI1	IN18 (3)			INDEX2 (3)
19	PI2	IN19 (3)			INDEX3 (3)
20	PI3	IN20 (3)			
21	ERROR1	ERROR1			
22	ERROR2	ERROR2			
23	ERROR3	ERROR3			
24	S+	ERROR4			
25		INDEX1 (4)			
26		INDEX2 (4)			
27		INDEX3 (4)			
28		INDEX4 (4)			
29		-			
30		-			
31		-			
32		-			
33		PI0 (5)			
34		PI1 (5)			
35		PI2 (5)			
36		PI3 (5)			

#	CO2200	CO4200	CO6100	CO6500	CO5400
37		FAULT (5)			
38		S+ (5)			

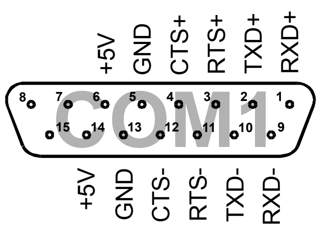
Outputs

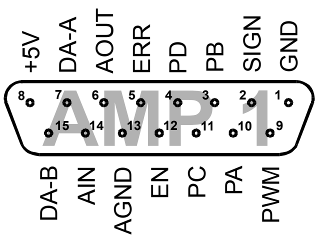
#	CO2200	CO4200	CO6100	CO6500	CO5400
1	IO1	OUT1 (1)	IO1	IO1	OUT1 (1)
2	IO2	OUT2 (1)	IO2	IO2	OUT2 (1)
3	IO3	OUT3 (1)	IO3	IO3	OUT3 (1)
4	IO4	OUT4 (1)	IO4	IO4	OUT4 (1)
5	IO5	OUT5 (1)	IO5	IO5	OUT5 (1)
6	IO6	OUT6 (1)	IO6	IO6	OUT6 (1)
7	IO7	OUT7 (1)	IO7	IO7	OUT7 (1)
8	IO8	OUT8 (1)	IO8	IO8	OUT8 (1)
9		OUT9 (2)			OUT9 (2)
10		OUT10 (2)			OUT10 (2)
11		OUT11 (2)			OUT11 (2)
12		OUT12 (2)			OUT12 (2)
13		OUT13 (2)			
14		OUT14 (2)			
15		OUT15 (2)			
16		OUT16 (2)			
17					
18					
19					
20					
21					
22					
23					
24					

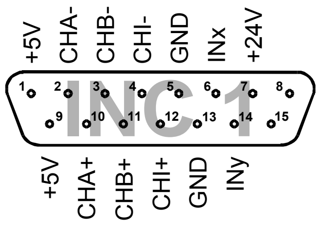
6. Appendix D: ASCII code table

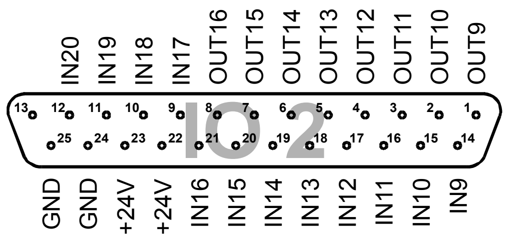
Code	Char	Key	Function	Code	Char	Code	Char	Code	Char
0	Ctrl-A			32	SPACE	64	@	96	`
1	Ctrl-B			33	!	65	A	97	a
2	Ctrl-C			34	“	66	B	98	b
3	Ctrl-D	STOP	BREAK	35	#	67	C	99	c
4	Ctrl-E	HALT		36	$	68	D	100	d
5	Ctrl-F	EDIT		37	%	69	E	101	e
6	Ctrl-G	START		38	&	70	F	102	f
7	Ctrl-H	RUN		39	'	71	G	103	g
8	Ctrl-I	DEL		40	(	72	H	104	h
9	Ctrl-J		TAB	41	)	73	I	105	i
10	Ctrl-J		LF	42	*	74	J	106	j
11	Ctrl-K			43	+	75	K	107	k
12	Ctrl-L		FF	44	,	76	L	108	l
13	Ctrl-M		CR	45	-	77	M	109	m
14	Ctrl-N	MENU		46	.	78	N	110	n
15	Ctrl-O	MOVE		47	/	79	O	111	o
16	Ctrl-P	STEP		48	0	80	P	112	p
17	Ctrl-Q		XON	49	1	81	Q	113	q
18	Ctrl-R	REF		50	2	82	R	114	r
19	Ctrl-S		XOFF	51	3	83	S	115	s
20	Ctrl-T	TEACH		52	4	84	T	116	t
21	Ctrl-U			53	5	85	U	117	u
22	Ctrl-V			54	6	86	V	118	v
23	Ctrl-W			55	7	87	W	119	w
24	Ctrl-X			56	8	88	X	120	x
25	Ctrl-Y			57	9	89	Y	121	y
26	Ctrl-Z			58	:	90	Z	122	z
27		ESC	ESC	59	;	91	[	123	{
28				60	<	92	\	124	|
29				61	=	93	]	125	}
30				62	>	94	^	126	~
31				63	?	95	_	127	

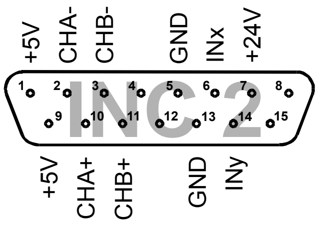
7. Appendix E: Pinouts

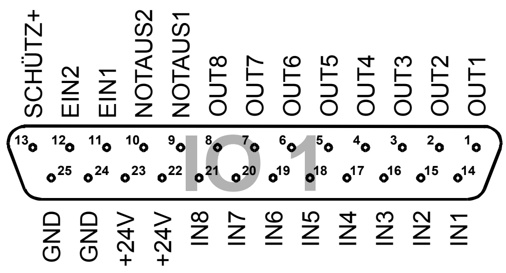


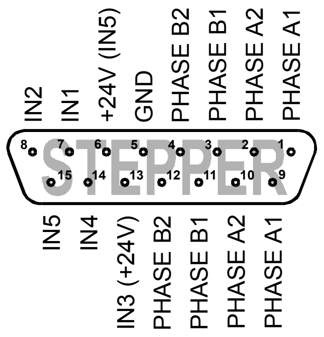


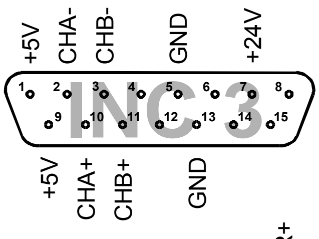


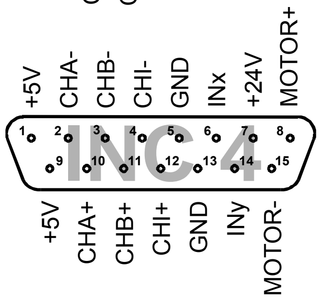


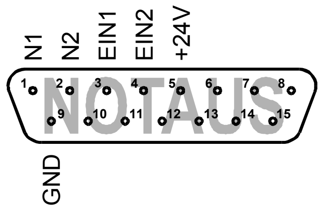


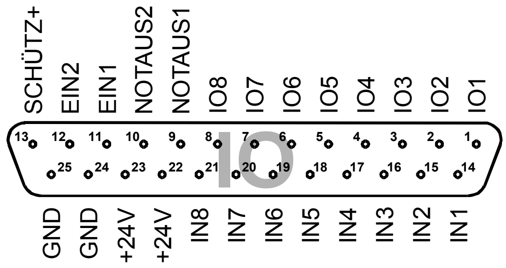


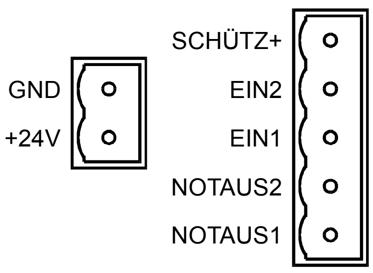


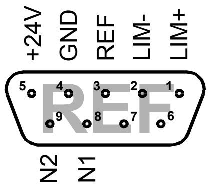


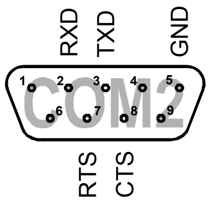


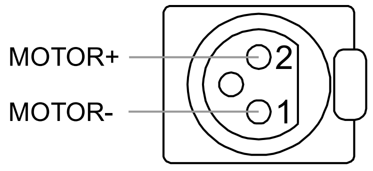


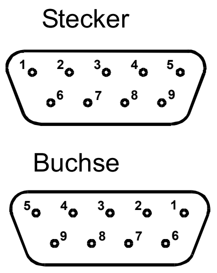


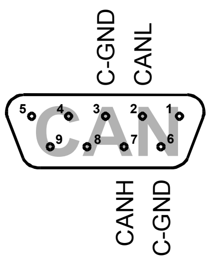




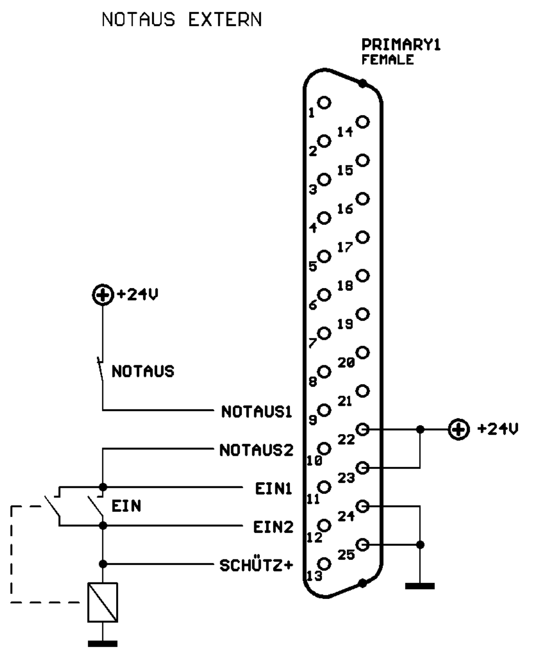


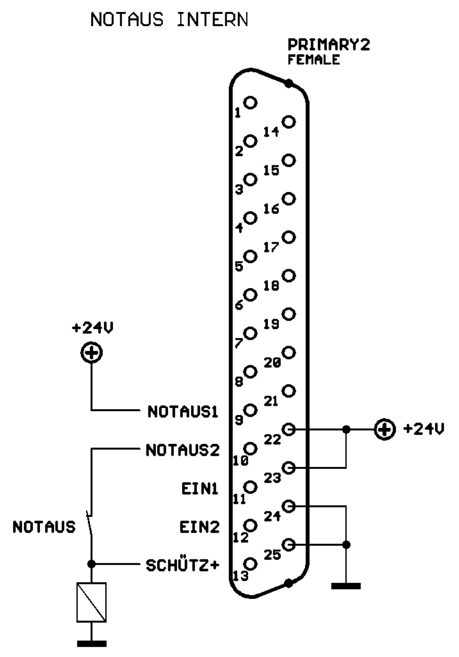


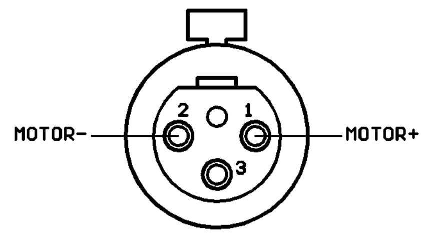




NOTAUSKREIS CO2200, CO4300







8. Appendix F: Pin description

Name	Function
+24V	24 VDC power supply
+5V	+5 VDC power output to supply amplifiers and incremental encoders
AGND	analogue ground
AIN	analogue input +/-10V
AOUT	analogue ouput +/-10V
CANH, CANL	CAN bus, external termination
C-GND	CAN ground
CHA+, CHA-	differential inputs, 120Ω termination, incremental encoder channel A
CHB+, CHB-	differential inputs, 120Ω termination, incremental encoder channel B
CHI+, CHI-	differential inputs, 120Ω termination, incremental encoder channel Index
CTS	RS232 Clear To Send
CTS+, CTS-	RS422 Clear To Send, differential inputs, 120Ω termination
DA-A	analogue output 0..5V, amplifier current control
DA-B	analogue output 0..5V, amplifier current control
EIN1, EIN2	internal EIN relay, emergency stop circuit
EN	amplifier enable output, 0V = Off, +5V = On
ERR	error input, convention: 0V = OK, +5V = ERROR
GND	ground
IN, REF	24 V input, ca. 4 mA input current
IO	24 V input/output,output current max. 350 mA (source driver) if used as input, set pin to 0 (`PINOUT n,0`)
NOTAUS1, NOTAUS2	External emergency stop circuit
OUTn	24 V output,output current max. 350 mA (source driver)
PA	Stepper motor controller phase A, resp. direction
PB	Stepper motor controller phase B, resp. step
PC	Stepper motor controller phase C, resp. standby/max
PD	Stepper motor controller phase D
PWM	PWM output (5V)
RTS	RS232 Ready To Send
RTS+, RTS-	RS422 Ready To Send, differential outputs
RXD	RS232 receive data
RXD+, RXD-	RS422 receive data, differential inputs, 120Ω termination
SCHÜTZ+	Emergency stop circuit, sense input
SIGN	SIGN output (5V)
TXD	RS232 transmit data
TXD+, TXD-	RS422 transmit data, differential outputs