Communication and ASCII Commands Reference

Chapter IV What you need if you don't use APOSS

Communication Reference

What you need if you don't use APOSS

This chapter is intended for users who do not wish to use the controller with the APOSS-PC software delivered with the device.

Requirements

Programming knowledge, as well as knowledge about serial data exchange is necessary.

The command structure and the method of function of the controller should be known through working with the PC software APOSS.

The ASCII command mode should, as much as possible, be preferred to the communication mode described in this chapter.

What are commands and messages?

- The controller communication with any host computer (as well as with the PC software APOSS) takes place via command and message strings.
- All commands to the controller (except ESC) consist of an "$", followed by a character, or possibly data.
- The first command character (directly following an "$") provides information about the type of message.
- The controller answers every command with a return message which is closed with a Return and a Linefeed.
- Commands and messages consist of ASCII strings.
- Numbers are expected as a series of ASCII numerals which must be closed with a blank, a Return or a Linefeed.

What else is important?

- Blanks are shown in the following description with a _ (Underline).
- Large- and small-case type must be noted.
- The clear text message created by the PC software APOSS for the return messages is given in brackets ().
- Serial data exchange:
  9600 Baud, 8 Bit, no parity, 1 Stop-Bit

Chapter IV All Commands of the Communication Reference

All Commands of the Communication Reference

ESC Correction of error conditions and/or abort program execution. All error messages will be erased and the axis controller will be re-activated.

A program with automatic start identification will again be started immediately after program abort.

Return messages:

B0_#47	(APOSS: Stop) Confirmation of program abort and bearing regulation of all drives in the current position.

$A_*%bu*_ Sets automatic start identification for program *%bu*.

In future, the program concerned will be automatically carried out when the control is switched on or when a program abort is carried out.

By inputting a non-existent program number, a previously defined auto-start-identification can be erased.

%bu = Program number

Return messages:

A0_#37_*%bu*	(APOSS: Program No. %bu, provided with auto-start-identification) The program has been successfully provided with an auto-start-identification number.
A0_#61_*%bu*	(APOSS: Autostart-Program identification has been erased) A previously set auto-start-identification has been erased in that a non-existing program number has been set..
A1_#38_*%bu*	(APOSS: Auto-start-identification for program no. *%bu* not possible) An auto-start-identification for an unavailable program has been attempted, whereby no other previously auto-start-identified program was available.

$C The entire memory, incl. the user parameters, will be erased and the original controller status, incl. the Init parameters will be restored.

Return messages

C0_#45	(APOSS: Memory erased!) The entire memory has been erased and the control unit with the Init set-up re-started.

$D_%u_%bb ..	**Internal Command!** (Not foreseen for the user!)	
	Downloading data into the controller temporary memory.	
	%bu = Number of data bytes which follow *%bb...* = Data bytes (in total *%bu*)	
	Return messages:	
	D0_#39_*%u*	(APOSS: *%bu* Data bytes received) This message appears following a successfully carried out D-command. *%bu* = number of received data bytes
	D1_#41_*%u*	(APOSS: No more *%bu* data bytes available) There was insufficient temporary memory available to store the data bytes. *%bu* = number of data bytes
	D2_#40	(APOSS: Communication error) An error has occurred during data exchange.
$E	Erase <u>all</u> stored programs, and release the corresponding memory space.	
	Return messages	
	E0_#36	(APOSS: All programs have been erased) All programs in the battery buffered memory of the controller have been erased, and the memory has been cleared.
$F_%bu1_%bu2_	List all program names between number *%bn1* and number *%bn2*.	
	%bn1 = Program number (0...254) *%bn2* = Program number (>*%bn1*, 1...254)	
	Return messages	
	M#42_*%bu_%s*	(APOSS: Prog. *%bu* is *%s* ***) The displayed program number contains the given program which is provided with an auto-identification number. *%bu* = Program number *%s* = Program name
	M#43_*%bu_%s*	(APOSS: Prog. *%bu* is *%s*) The displayed program number contains the given program. *%bu* = Program number *%s* = Program name
	F0_	The program list is complete; the last entry of the queried program number area is displayed.
	F1_#44	(APOSS: No more available programs) No programs to be found within the queried program number area.

$G	Execute the program with the internal Tmp-Rg 255 and the current Parameter-Init (Direct or Online mode).

$I Initialize the parameters.

When the battery voltage is too low and a data loss from the battery buffered memory results, it is necessary to re-start the control with the init parameter after switching on.

(A data loss is displayed via a CRC error.)

Return messages:

IO_#30	(APOSS: All parameters have been re-started ! ! !
	All parameters have been re-started with the init value.

$L Delete the EEPROM. EEPROM gets an erase label (1 Byte).

Return message

L0_#69	(APOSS: EEPROM erased).
	EEPROM erased

$M Save the RAM into the EEPROM. There will be save all files programs and parameters.

Return messages

E#62	(APOSS: Error in verifying process)
	error in verifying process
MO_#67	(APOSS: saved in EEPROM)
	saved in EEROM with success
M1_#68	(APOSS: no EEPROM inside / error in verifying process)
	If there is an error in verifying process, you will get the message E#62 pervious.

$N_n_b The command creates the slave-number and sets the baudrate to "baudratenr" for the CAN communication.

During executing the $N-command 2 standard objects will be created:

TX-Object (object to send) with CAN-identification number ID = ‘slavenr’ * 2
RX-Objekt (object to receive) with CAN-identification number ID = ‘slavenr’ * 2 + 1

ID = identification number in the CAN-Controller

Syntax	$N slavenr baudratenr
Parameter	slavenr = Slave-number [1…1000]; two objects are created in the CAN controller. If slavenr = 9999, then no standard objects exist.
	baudratenr = coded CAN-transfer rate

Code The code of the CAN-Baudrate is:

b = 0	corresponds	5 kBaud
b = 1	corresponds	10 kBaud
b = 2	corresponds	20 kBaud
b = 3	corresponds	50 kBaud
b = 4	corresponds	100 kBaud
b = 5	corresponds	125 kBaud
b = 6	corresponds	250 kBaud
b = 7	corresponds	500 kBaud
b = 8	corresponds	1000 kBaud

Return message

N0 #82	if O.K.
N255 #82	if CAN-Controller is not found

Please use a terminal program to set slave numbers.

Syntax-Example $N 5 5 member 5, 125 kB
TX-object ID 10
RX-cbject ID 11

$O_%bu Sets the default interface.

$P Switch over to the ASCII command mode.

This mode permits the entry of all motion and I/O commands in a special, easy to understand ASCII coding. This command interface has been foreseen primarily for simple communication with a host computer.

No syntax checking is carried out in this mode. The user is therefore responsible for the correct use of syntax in the commands.

The ASCII command mode commands are described in more detail in section ASCII Command Reference

This mode can be exited with Q_.

Once the ASCII command mode has been exited, the last entered command series (entries between the last G command and Q) is in the temporary memory.

Return messages

P0_ASCII_ Commandmode	The ASCII command mode has been successfully started. This mode can be exited with Q or ESC.
P0_End_ASCII_ Commandmode	The ASCII command mode has been exited.

$Q_%s Sets the project name to string (8-times)

$R_%bu_%bs_	Save temporary program data, with the names *%bs* under the number *%bn*.

A program that may have already been saved under this number will be overwritten. In case nnn is greater than the previously highest number for a stored program, then empty programs will be created for the numbers between.

%bn = Program number (max. 254)
%bs = Program name (max. 8 ASCII characters)

Return messages

S0_#34_*%u*	(APOSS: Program stored as no. *%bn*) The program has been successfully stored and can be used via number *%bn*. *%bn* = Program number
S1_#35_*%u*	(APOSS: Storage of program no. *%bn* no longer possible) Either there is no valid program available in the temporary memory, or the program number is not valid.

$S_*%bs*_	Saving the temporary program data, with the name *%bs*.

%bs = Program name (max. 8 ASCII characters)

Return messages

S0_#34_*%u*	(APOSS: Program stored as no. *%bn*) The program has been successfully stored and can be used via the no. *%bn*. *%bn* = assigned program number
S1_#35_*%u*	(APOSS: Storage of program no. *%bn* not possible) Either there is no valid program available in the temporary memory, or all program numbers (max. 254) are occupied.

$T	Program carried out in steps.

This instruction can be used when the program contains Break-points (DEBUG commands). In this case, the program will be carried out until the next Break-point, and then the achieved line number will be reported.

Return messages

T0_#31_*%u*	(APOSS: Line *%bl*) Reports that a line has been successfully processed. *%bl* = next line to be processed
T255_	Reports that a line has not been correctly processed.

$U %u1

Reads the array with the no. %u1.
If an array exists with this number, the array elements will be reported.

Return messages

U0_#72_*%u1_%u2*	(APOSS: array no. %u1 with length %u3) This message (72) is reported, if everything is o.k. %u1 shows the number of the desired arrays (starting with 0), %u2 reports the number of elements, to be sent. The elements are divided by CRLF.
U1_#73_*%u1_%u2*	(APOSS: no array nr. %, last array is no. %) This message (73) is only reported, if the desired array nr. %u1 doesn't exist. Same time you will see the number of existing arrays.
U2_#74_*%u1_%u2*	(APOSS: array no. % is empty) This message is reported, if the desired array %u1is empty. This can only happen, if it is the first non existing array. %u2 reports the number of existing arrays.

$V %u1_%u2

Command to create an u1-array with value u2

Return messages

U0_#72_ *%u1_%u2*	(APOSS: array no. %u1 with length %u2) This message (72) is reported, if everything is o.k., that means an array with the %u1 and value %u2 exists or can be created. After that the user has to send the elements (%u2 pieces). The message U0_#72_%u1_%u2 confirms and ends the task..
U1_#73_ *%u1_%u2*	(APOSS: no array no. %, last array is no. %) This message (73) is only reported, if such an array doesn't exist and if it isn't the next free array. (Arrays can only be created one after another, not in any order whatever.) %u2 reports how many arrays exist.
U3_#75_ *%u1_%u2*	(APOSS: Not enough memory, to create an array %u1 with value %u2) This message (75) reports, that such an array doesn't exist and could be created, but there is not enough memory.
U4_#76_ *%u1_%u2_%u3*	(APOSS: array no. %u1 has not value %u2 but %u3) This message (76) is sent, if an array number %u1 exists, but not corresponds to the desired value. %u2 is the desired value of the array; %u3 is the number of defined elements.

$X_%bu_	Carry out of the program with number *%bn*.
	If available, the program will be carried out immediately. During processing, a program abort can be made at any time via ESCAPE (ASCII: 27).
	%bn = Number of the program to be carried out
	%bn = 255 => carry out program in the temporary memory
	Return messages

M#32_*%bu_%s*	(APOSS: Program *%bn %bs* will be carried out)
	The called program with the number *%bn* and the name *%bs* will be immediately carried out.
	%bn = Program number *%bs* = Program name
X0	(APOSS: Program carried out without problems)
	Program has been successfully processed to the end..
X255	An error has occurred during program processing.
X254_#33_*%bu*	(APOSS: Program no. *%bn* is not available)
	An attempt has been made to carry out a program no available in the controller memory.
	%bn = Program number
B0_#47	(APOSS: Stop)
	The user entered an ESC. This causes an immediate program abort, and the drive stops in its current position. The above report confirms this procedure.
E#*%be_%p*	Error message during program processing.
	The individual error messages can be read in Error Messages during Command Execution.
	%be = Error number *%bp* = Error parameter

$Y	Continue last program.

$?	Queries the current error condition.
	%be = Current error number
	Return messages

M#53	(APOSS: no error occurred)
	No current error condition.

$$	Reports SIGN-ON-String.

Command Independent Return messages

K1_#48

(APOSS: V24 Error)

An overflow in the V24 communication buffer has taken place. Too much data has been transmitted.

K3_#66
_%c_%bu

(APOSS: $ must stand at the beginning of the command)

A command without the "$" character at the beginning of the command sequence has been entered.

bc = incorrect command character
%ba = ASCII code of the incorrect character

K2_#46
_%c_%bu_%lu

(APOSS: Incorrect command *%bc %ba* at *%lt*)

An unknown or an un-interpretable command has been entered.

%bc = incorrect command character
%ba = ASCII code of the incorrect character
%lt = internal time - count

ASCII Commands Reference

General Information

This chapter is intended for the user who wishes to use the controller without the accompanying PC software APOSS.

Knowledge of the APOSS language elements, as well as its function, is necessary.

The ASCII command mode makes the operation of the controller possible from any host computer or terminal. (Serial data transmission: 9600 baud, 8 data bit, 1 stop bit, no parity.)

The ASCII command mode can be started via a $P. (also see Communication Reference,)

The special error messages for the ASCII command mode can be read in Error Messages of the ASCII Command Mode.

!!! The APOSS software has been extended constantly especially since version 5. Meantime complex application as synchronizing and CAM functions can be realized with simple commands how DEC, SYNCC, SYNC and much more. But it would be difficult to use these new functions via the ASCII mode. Therefore they are not (yet) described in the following chapter.

Basics to the command structure

All ASCII commands have the same command syntax:
Letter-command identification + possible axis identification + possible parameter

- Large and small case type must be noted!
- For motion commands, the axes can be moved simultaneously or serially.
  When a '&' character precedes the command, the motion is started simultaneously with the following motion command (without '&') of another axis.
- Blanks can be inserted between the command lines.
- All entered commands are stored temporarily until the input of the 'G' command, thus complete command sequence can be transmitted and processed.
- Following the successful, error-free processing of a command sequence (after the 'G' command), an 'X0' message is delivered.
- After exiting the ASCII command mode, sequences not yet processed via the G-command can be stored as a program. (see Communication Reference, Command: $S)
- After exiting the ASCII command mode, all parameters will be restored to their original setting.

Chapter IV Brief Overview of ASCII Commands

Brief Overview of ASCII Commands

Inform you in this brief overview. More details you will find in the enclosed section.

id	command function	command structure
$P	activate ASCII command mode	$P
A	Absolute Positioning	A %bn x *%lp*
B	write output byte	B *%bo* B *%bb*
C	set acceleration	C *%bn* x *%wa*
D	set speed	D *%bn* x *%wv*
E	clears error messages	E
F	wait for target position	F *%bn* x
G	start command execution	G
H	move to machine zero-point	H *%bn* x
I	read input	I *%bi*
J	wait after pos. command on/off	J *%bs*
K	activate speed mode	K *%bn* x
L	Absolute Linear interpolation	&L*%bn* x *%lp*
M	re-sets user parameter	M *%bn* x
N	move to real zero-point	N *%bn* x
O	set output	O *%bo %bs*
P	read position	P *%bn* x
Q	exit ASCII command mode	Q
R	Relative Positioning	R *%bn* x *%lp*
S	set temporary axis parameter	S *%bn* x *%bm* p *%lv*
T	de-activate speed mode	T *%bn* x
U	set real zero-point	U *%bn* x
V	set positioning velocity	V *%bn* x *%wv*
W	delay time	W *%lt*
X	wait for input	X *%bi %bs*
Y	read input byte	Y *%bi*
Z	read system status	Z *%bn* x
a	read axis status	a *%bn* x
c	read command position	c *%bn* x
f	switch off axis controller	f *%bn* x
i	search index position	i *%bn* x
n	activate axis controller	n *%bn* x
o	set temporary zero-point	o *%bn* x *%lp*
p	read temporary axis parameter	p *%bn* x *%bm* p
q	read permanent axis parameter	q *%bn* x *%bm* p
r	erase temporary zero-point	r *%bn* x
s	interrupt motion	s *%bn* x
t	read internal system time	t
w	set permanent axis parameter	w *%bn* x *%bm* p *%lv*

Explanations

%ba	= acceleration	*%bb*	= byte value
%bi	= input number	*%bm*	= parameter number
%bn	= axis number	*%bo*	= output number
%bs	= condition ('t'=true, 'f'=false)	*%lp*	= position
%lt	= time (ms)	*%lv*	= value
%wa	= acceleration	*%wv*	= velocity

Chapter IV All ASCII-Commands from $P to _

All ASCII-Commands from $P to _

$P - activates ASCII command mode

Command	$P
Function	activates ASCII command mode
Return message	P0 ASCII – Command mode

A - Absolute Positioning

Command	A %bn x %lp
Parameter	*%bn* = axis number *%lp* = Absolute position (in UU)
Function	Absolute Positioning in User units.
APOSS reference	POSA
Example	axis 1 -> 500 UU, axis 2 -> 2000 UU: (in serial) A1x500 A2x2000 G axis 1 -> 500 UU, axis 2 -> 2000 UU: (simultaneous movement start) &A1x500 A2x2000 G

B - Write Outputs Byte

Command	B %bo B %bb
Parameter	*%bo* = number of the output byte (0… max. 255, dependent on the used Hardware) *%bb* = output value (byte value: 0...255)
Function	setting and re-setting one output byte
APOSS reference	OUTB
Example	set bit 7 of output byte 0, re-set bit 1 - 6, re-set 8 B0 B64 G

C - Set Acceleration

Command	C %bn x %w
Parameter	*%bn* = axis number *%wa* = acceleration (1...max. speed scaling)
Function	Sets the acceleration values in the speed and positioning mode
APOSS reference	ACC
Example	sets the minimum acceleration for axis 1 C1x1 G

D - Set speed velocity

Command	D %bn x %wv
Parameter	*%bn* = axis number *%wv* = speed
Function	sets the speed values for the speed mode
APOSS reference	CVEL
Example	axis 1: sets minimum velocity D1x1 G

E - Clear error message

Command	E
Parameter	clears the current error message
Function	E#1_%bu %bu = number of the erased message
APOSS reference	ERRCLR, ERRNO
Example	E G -> Return message: E#1 0 (=> no error

F - Wait for target position

Command	F %bn x
Parameter	%bn = Axis number
Function	waits in active NOWAIT mode until target position has been reached
APOSS reference	WAITAX
Example	activate NOWAIT, axis 1 -> 50000 UU, waits until target reached, axis 1 -> 10000 UU Jt A1x50000 F1x A1x10000 G

G - Start Command Execution

Command	G
Function	Start execution of the commands entered since the last G command (or since switch on).
! ! !	All commands will be temporarily stored in a buffer, and only following a G command will the complete sequence be started. After all commands have been processed, the buffer will be erased.
Return message	If an error occurs during the processing of the command sequence, then an error message will appear. If no error occurs during the whole of the command sequence, an 'X0' will appear.
Example	load and start command sequence A1x5000 W 1000 A1x1000 W2000 A1x5000 G

H - Move to Machine zero-point

Command	H %bn x
Parameter	%bn = Axis number
Function	move to machine zero-point (reference switch and index) and set as real zero-point
APOSS reference	HOME
Example	move to machine zero-point axis 1 H1x G

I - Read input

Command	I %bi
Parameter	*%bi* = input number (1...8)
Function	reads signal level at a digital input
Return message	M#22_%bi_%bs %bi = input number (1...8) %bs = input status (0, 1)
APOSS reference	IN
Example	read in input 7 status I7 G -> return message: M#22 7 1 (=> input 7 high)

J - Wait after Pos. Command on/off

Command	J %bs
Parameter	*%bs* = f => waits after pos. commands = t => does not wait after pos. Commands
Function	Activates/de-activates the NOWAIT mode, whereby after pos. command waiting does/does not take place until the target position is reached.
APOSS reference	NOWAIT ON/OFF
Example	from now do not wait after pos. commands Jt G

K - Activate speed mode

Command	K %bn x
Parameter	%bn = Axis number
Function	activation of the speed mode
! ! !	The axis will continue to rotate at the given speed after this command, until a new speed value is set or until the speed mode is de-activated
APOSS reference	CSTART
Example	set speed, activate speed mode D1x-5 K1x G

L - Absolute linear interpolation

Command	&L %bn1 x %lp1 L %bn2 x %lp2
Parameter	%bn1 = Axis number *%lp1* = absolute position of *n1* (in Uu) *%bn2* = axis number (<> *n1*) *%lp2* = absolute position of *n2* (in Uu)
Function	Absolute positioning of two or more axes, so that both axes reach the target position simultaneously.
APOSS reference	LINA
Example	axis 1, axis 2 move synchronously to target position &L1x40000 L2x-10000 G

M - Re-set user parameter

Command	M %bn x
Parameter	%bn = Axis number
Function	re-sets all the parameters of the given axis to the previously valid user parameter before the ASCII mode
Example	re-activate user parameter axis 1 M1x G

N - Move to real zero-point

Command	N %bn x
Parameter	%bn = Axis number
Function	move to the real zero-point.
APOSS reference	ORIGIN or POSA
Example	Axis 1: move to real zero-point N1x G

O - Set output

Command	O %bo %bs
Parameter	*%bo* = output number (1...8) *%bs* = f => re-set output = t => set output
Function	sets / re-sets a digital output
APOSS reference	OUT
Example	set output 1, re-set output 8 O1t O8f G

P - Read position

Command	P %bn x
Parameter	%bn = Axis number
Function	reads the current position in quad counts.
Return message	M#21_%bn_%lp %bn = Axis number %lp = current position (in qc)
APOSS reference	APOS
Example	read axis 1 position P1x G -> Return message: M#21 1 0 (=> Axis 1, Pos.: 0)

Q - Exit ASCII Command Mode

Command	Q
Function	exits the ASCII command mode
Note	All the inputted commands since the last G command are stored in the temporary memory after finishing the ASCII command mode. Via the $S communication command, they can be stored as a program. Parameters altered during the ASCII command mode will be re-set to its original setting.
Return message	P0 End ASCII - Command mode
Example	and store last command sequence A1x5000 O7t W2000 H1x Q $S TEST

R - Relative positioning

Command	R %bn x %lp
Parameter	%bn = Axis number %lp = relative position (in Uu)
Function	relative positioning (in User units) in relation to the current position
APOSS reference	POSR
Example	axis 1 relative + 500 Uu, axis 2 relative + 2000 Uu: (serially) A1x500 A2x2000 G axis 1 relative + 500 Uu, axis 2 relative + 2000 Uu: (simultaneous motion start) &R1x500 R2x2000 G

S - Set temporary axis parameter

Command	S %bn x %bm p %lv
Parameter	%bn = Axis number *%bm* = parameter number *%lv* = parameter value
Function	temporarily re-sets the value of an axis parameter
Note	The altered parameter values remain valid only until the ASCII command mode is exited or until the parameters are again set. These are called program parameters as they are valid only within the current program (in this case, within the ASCII command mode).
APOSS reference	SET
Cross reference	ASCII Command: p, q, w Parameter Reference
Example	temporarily set the ORIGIN velocity S1x16p5 G

T - De-activate speed mode

Command	T %bn x
Parameter	%bn = Axis number
Function	terminates the speed mode, and re-activates the positioning mode
! ! !	This command should only be used when the axis is not in motion.
APOSS reference	CSTOP
Example	axis 1 in speed mode, brake axis 1 end speed mode K1x D1x5 W2000 G D1x0 W500 G T1x G

U - Set real zero-point

Command	U %bn x
Parameter	%bn = Axis number
Function	defines current position as the new real zero-point
APOSS reference	DEF ORIGIN
Example	axis 1 -> 5000 Uu, define as new real zero-point A1x5000 U1x G

V - Set positioning velocity

Command	V %bn x %wv
Parameter	%bn = Axis number *%wv* = velocity
Function	sets the velocity value during positioning mode
APOSS reference	VEL
Example	set velocity, axis 1 -> 50000 V1x10 A1x50000 G

W - Delay time

Command	W %lt
Parameter	*%lt* = delay time in milliseconds
Function	defines program delay in milliseconds
APOSS reference	WAITT
Example	axis 1 -> 50000, wait 10 seconds A1x50000 W10000 G

X - Wait for input

Command	X %bi %bs
Parameter	*%bi* = input number (1...8) *%bs* = f => wait for low level = t => wait for high level
Function	waits until a desired signal level has been reached at a designated input
APOSS reference	WAITI
Example	wait for high on input 1, then set output 7 X1t O7t G

Y - Read input byte

Command	Y %bi
Parameter	*%bi* = number of the input byte (0 … max. 255, depends on the used hardware)
Function	simultaneous reading of eight digital inputs, (one byte)
Return message	M 23_%bi_%bw %bi = number of the input byte %bw = byte value of the eight inputs
APOSS reference	INB
Example	read out the status of the first 8 inputs Y0x G -> Return message: M#23 0 64 (=> only input 7 set)

Z - Read system status

Command	Z %bn x
Parameter	%bn = Axis number
Function	read system internal status.
Return message	M#24_%lv %lv = system status (32 Bit)
APOSS reference	STAT
Example	read system status axis 1 Z1x G -> Return message: M#24 4981280

a - Read axis status

Command	a %bn x
Parameter	%bn = Axis number
Function	read status of an axis.
Return message	M#24_%bu %bu = axis status (8 Bit)
APOSS reference	AXEND
Example	read status axis 1 a1x G -> Return message: M#24 1 (=> axis not in motion)

b - Define an arc

Command	b %lp x
Parameter	%lp1 = position 1 %lp2 = position 2 %lα = angle
Function	Define an arc, start at current position
APOSS reference	ARC
Example	b 1000 1000 10 G

c - Read command position

Command	c %bn x
Parameter	%bn = Axis number
Function	read the current command position in quad counts
Return message	M#21_%bn_%bc %bn = Axis number %bc = command position (in qc)
APOSS reference	CPOS
Example	read command position axis 1 c1x G -> Return message: M#21 1 0 (=> axis 1, Pos.: 0)

d - Read analog input	Command	d %bA
	Parameter	%bA = analog input
	Function	Read analog input. The result is reported via the RS232-interface.
	APOSS reference	INAD
	Example	d2 G
f - Switch off axis control	Command	f %bn x
	Parameter	%bn = Axis number
	Function	switches off the axis controller
	APOSS reference	MOTOR OFF
	Example	switch off axis controller axis 1 f1x G
i - Search index position	Command	i %bn x
	Parameter	%bn = Axis number
	Function	search encoder index position
	APOSS reference	INDEX
	Example	move to index position axis 1 i1x G
l - Define a straight section	Command	l %lp
	Parameter	%lp1 = position 1 %lp2 = position 2
	Function	Define a straight section
	APOSS reference	VEC
	Example	l 10000 10000
m - Save RAM	Command	m
	Function	Save RAM into EEPROM
	APOSS reference	SAVEPROM
	Example	m G

n - Activate axis controller

Command	n %bn x
Parameter	%bn = Axis number
Function	activates axis control and starts positioning to current position
APOSS reference	MOTOR ON
Example	activate axis controller axis 1 n1x G

o - Set temporary zero-point

Command	o %bn x %lp
Parameter	%bn = Axis number *%lp* = absolute position (in UU)
Function	defines absolute position as the temporary zero-point
APOSS reference	SET ORIGIN
Example	absolute position 50000 as temporary NP o1x50000 G

p - Read temporary axis parameter

Command	p %bn x %bm p
Parameter	%bn = Axis number %bm = parameter number
Function	reads the value of a temporary axis parameter
Return message	P%bm_M#26$%bm_%lv %bm = Parameter number %lv = parameter value
APOSS reference	GET
Cross Index	Parameter Reference in chapter II.
Example	read origin speed p1x16p G -> Return message: P16 M#26$16 5

q - Read permanent axis parameter

Command	q %bn x %bm p
Parameter	%bn = Axis number %bm = *%bm* = parameter number
Function	read the value of a permanent axis parameter
Return message	P%bm_M#26$%bm_%lv %bm = parameter number %lv = parameter value
Note	A difference is made between temporary (program) parameters and permanent (user) parameters. User parameters are stored in the battery buffered memory and remain valid even after exiting the ASCII mode, or switching off the controller..
Cross Index	ASCII Command: S, p, w Parameter Reference
Example	read ORIGIN speed q1x16p G -> Return message: P16 M#26$16 5

r - Erase temporary zero-point

Command	r %bn x
Parameter	%bn = Axis number
Function	erases the definition of the temporary zero-point
APOSS reference	RST ORIGIN
Example	set temporary zero-point positioning erase temporary zero-point o1x50000 a1x0 r1x G

s - Interrupt motion

Command	s %bn x
Parameter	%bn = Axis number
Function	interrupts motion with a defined braking ramp
Note	This command stops abrupt the corresponding axis (Version H) or stops with the defined acceleration (Version N)
APOSS reference	MOTOR STOP
Example	positioning, wait 0.5 seconds, interrupt positioning A1x100000 W500 s1x G

t - Read internal system time

Command	t
Function	Reads the internal system time elapsed since switching on the controller
Return message	M#57_0_%lv %lv = system time (in ms)
APOSS reference	TIME
Example	read system time t G -> Return message: M#57 0 532355 (=> 8min, 52s, 355ms)

v - Set analog output

Command	v %bn x %bv
Parameter	%bn = Axis number %bv = Byte value (-127 to 128)
Function	Sets the analog output of the corresponding axis to the corresponding value.
! ! !	Before that the command MOTOR OFF must be executed. (only with HCTL-Version)
APOSS reference	OUTAN

w - Set permanent axis parameter

Command	w %bn x %bm p %lv
Parameter	%bn = Axis number *%bm* = parameter number *%lv* = parameter value
Function	Sets the new value of a permanent axis parameter
Note	The altered parameter values are stored in a battery buffered memory and remain permanent even after switching off the controller. This is termed a User Parameter.
Cross Index	ASCII Command: S, p, q Parameter Reference
Example	set the new permanent ORIGIN speed w1x16p5 G

x - Set desired position to current position

Command	x %bn
Parameter	%bn = Axis number
Function	Set desired position to current position
Example	x1x G

y - Continue interrupted program

Command	y
Function	Continue an interrupted or aborted program
APOSS reference	CONTINUE
Example	y G

[- Define axis for a path sequence

Command	[
Function	Determines which of the two axes participate in path movement. Only two axes can participate, however another axis can be previously started with the NOWAIT command (e.g. spiral path).
APOSS reference	MOVE
Example	& [1x[2x G

\ - Sets velocity for path motion

Command	\
Function	sets velocity for path motion in UU per second
APOSS reference	MVEL
Example	\10000 G

] - Length of vector in UU

Command	]
Function	Determines the length of the following vectors; all vectors must have the same length..
Note	At present the relation length of vector to velocity must be < 3.
Example	] 1000

^ - Start of the vector sequence

Command	^
Function	Shows the start of the vector sequence, which is following by the number (nn) of vectors. The second parameter determines which axis is driven in motor speed mode. The vectors are read in via the serial interface during executing the program. Each vector will be confirmed with V0, if check sum is o.k. In case of error, V255 is reported. The data will be sending binary. The check sum is built with Bytes and sent as negative figure via RS232.
Example	\20000]100 & [1x[2x ^50 3 G V0 14 99 20 –133 V0 … X0 G \20000]100 & [1x[2x ^50 0 G V0 14 99 –133 V0 … X0 G

_ - Start of the vector sequence	Command	_
	Function	Shows the start of the vector sequence, following by the number (nn) of the vectors. After that the vectors follows, in which the vectors get always relative values, at first for the first axis and then for the second, to be separated by a blank. The command is finished with a zero vector (0 0), it means both values 0.
	Note	The single coordinates may not bigger than MAXINT (–32768 to 32767)
	Example	_nn 14 99 20 98 … 00 G
` - Shift zero-point	Command	`
	Function	Shift the zero point with the given value
	Note	only with HCTL-Version
	Example	`1x 30000

Chapter IV ASCII Echo Mode

ASCII Echo Mode

Aim of the Echo Mode

Improvement of the transmission security through exchanging acknowledgement signals between the transmitter and the receiver.

How the Echo Mode Operates

All received signals are immediately returned by the controller to the transmitter.

The transmitter can then compare the originally sent signals with the received information, and after every completed command, confirm the correctness of the transmission, or a transmission error, via an acknowledgement signal.

If a transmission error is signaled, then the characters received since the last acknowledgement signal or the last G-command will be erased, and the indicated command will not be executed.

If no acknowledgement signal is sent, then the command received before the next G-command is regarded as confirmed, and then executed.

The activation of the Echo Mode, the acknowledgement or non-acknowledgement takes place via a special character code, which can be sent after a complete command.

The special character codes are:

Term	ASCII value decimal	PC-Key
SYN	22	Ctrl + V
ACK	06	Ctrl + F
NAK	21	Ctrl + U

Function and Meaning of each Acknowledgement Signal

SYN	The Echo Mode will be alternately switched on / off within the ASCII command mode.
ACK	The commands sent since the last NAK, ACK or G-command are acknowledged as correct. The commands will be stored as valid until executed through a G-command. An ACK signal within a command will be evaluated as an error!
NAK	1. NAK sent to the controller All characters received since the last ACK signal or G-command will be erased from the controller memory, and the indicated commands will not be executed. 2. NAK sent from the controller The controller has not understood one of the received characters. All further commands will be acknowledged with a NAK until the sender clears the block with its own NAK signal.

Chapter III Error Messages in ASCII-Command Mode

Error Messages in ASCII-Command Mode	The following error messages, which are a completion of those treated in the previous chapter, can only appear in the ASCII command mode.	
P1_no_free_memory	Cause	Too many programs are stored. In the temporary memory there is no more room to accept further commands.
	Tip	clear memory
P2_number_expected	Cause	The command syntax is not correct. In one part of the command, an unexpected character appears instead of a numeral.
	Tip	check command syntax
P3_axes_order _expected	Cause	The command syntax is not correct. No valid axis identification such as 1x, 1X or 2x, etc. has been entered.
	Tip	check command syntax
P4_internal_error, _illeg._param	Cause	internal error
	Tip	If this error appears, the controller should be switched off and the technical service department notified.
P5_illegal_Parameter, _command_ignored	Cause	The command syntax is not correct. The given parameter did not correspond to the expected type (number, axis identification, etc.). The corresponding command has been ignored.
	Tip	check command syntax
P6_illegal _command_*%c*	Variable	*%c* = unknown command identification
	Cause	The entered command identification (first letter) could not be allocated to a valid ASCII command.
P7_number_too_big	Cause	The entered number has more numerals than permitted for this command section.
P8_%nnn%c_expected	Variable	*%c* = command identification *%nnn* = expected command section
	Meaning	This message indicates which command section has been expected.