
POSITIONING- AND CONTOURING CONTROL SYSTEM PA8000 PROGRAMMING AND REFERENCE MANUAL / PM



1 Introduction	9
2 Internal details of the <i>rw_TOS</i> operating system software.....	10
2.1 The PA8000 position controller	10
2.1.1 Control loop opened/closed	10
2.1.2 PIDF filter	10
2.1.2.1 The filter parameters K_D , K_I , K_P	10
2.1.2.2 Additional phase element	11
2.1.2.3 Scan time.....	11
2.2 The PA8000 profile generator	11
2.2.1 Profile generation for JOG commands	11
2.2.2 Profile generation for MOVE commands	12
2.2.3 Acceleration	13
2.2.4 Maximum velocity	13
2.2.5 Target velocity.....	13
2.2.6 Velocity correction.....	13
2.2.7 Target position / Traverse distance	13
2.2.8 Operating modes for command processing	14
2.2.8.1 Direct mode	14
2.2.8.2 Spool mode	14
2.3 Interpolation with the PA8000	15
2.3.1 Linear interpolation	15
2.3.1.1 Formal linear interpolation	15
2.3.2 Circular interpolation.....	15
2.3.3 Helical interpolation.....	15
2.3.4 Synchronous and asynchronous interpolations	15
2.4 PA8000 limit switch handling	16
2.4.1 TOM limit switch function(Turn-Off-Motor)	16
2.4.2 SMA limit switch function (Stop-Motor-Abruptly)	16
2.4.3 SMD limit switch function (Stop-Motor-Decelerate)	16
3 The PA8000 programming methods.....	17
3.1 PC application programming (PCAP programming, or direct programming).....	17
3.2 Stand-alone application programming (SAP programming)	17
3.2.1 SAP-Multitasking	18
4 PC application programming	19
4.1 Introduction	19
4.2 Example programs for using the function libraries.....	19
4.3 Definitions, structures and records	20
4.3.1 Definitions	20
4.3.2 Structures and records	20
4.3.2.1 Structure/record type AS	20
4.3.2.2 Structure/record type TSRP	21

4.3.2.3 Structure/record type TRU (Trajectory Units)	22
4.3.2.4 Structure/record type LMP (Linear Motion Parameters)	22
4.3.2.5 Structure/record type CMP (Circular Motion Parameters)	22
4.3.2.6 Structure/record type HMP (Helical Motion Parameters)	23
4.3.2.7 Structure/record type TOSI (Transputer Operating System Information)	23
4.3.2.8 Structure/record type CBCNT (Common Buffer CNC-Task)	24
4.3.2.9 Structure/record type CNCTS (Computerized Numerical Control Task Status)	24
4.4 PCAP high-level language function reference list	25
4.4.1 Structure of the reference list	25
4.4.2 General information	25
4.4.3 azo, activate zero offsets	26
4.4.4 cl, close loop	26
4.4.5 contcnct, continue numeric controller task	26
4.4.6 ctru, change trajectory units	27
4.4.7 dummy, dummy function call	28
4.4.8 InitMcuSystem, initialise mcu system	28
4.4.9 ja, jog absolute	29
4.4.10 jhi, jog home index	30
4.4.11 jhl, jog home left	30
4.4.12 jhr, jog home right	31
4.4.13 jr, jog relative	31
4.4.14 js, jog stop	31
4.4.15 lps, latch position synchronous	32
4.4.16 mca, move circular absolute smca, spool motion circular absolute	32
4.4.17 mcr, move circular relative smcr, spool motion circular relative	33
4.4.18 mcuinit, motion control unit initialisation	33
4.4.19 mha, move helical absolute smha, spool motion helical absolute	34
4.4.20 mhr, move helical relative smhr, spool motion helical relative	34
4.4.21 mla, move linear absolute smla, spool motion linear absolute	35
4.4.22 mlr, move linear relative smlr, spool motion linear relative	35
4.4.23 ms, motion stop	36
4.4.24 ol, open loop	36
4.4.25 ra, reset axis	36
4.4.26 rdap, read axis parameters	37
4.4.27 rdaxst, read axis status	37
4.4.28 rdaxstb, read axis status bit	39
4.4.29 rdcbcnct, read common buffer CNC-Task	39
4.4.30 rdcd, read common double	40
4.4.31 rdci, read common integer	40
4.4.32 rdcncts, read computerized numeric controller task status	40
4.4.33 rdigi, reset digital inputs	41
4.4.34 rddigi, read digital inputs	41
4.4.34.1 Axis-qualifier digi	42
4.4.35 rddigib, read digital input bit	43
4.4.36 rddigo, read digital outputs	43
4.4.37 rddigob, read digital output bit	44
4.4.38 rddp, read desired position	44
4.4.39 rddv, read desired velocity	44
4.4.40 rdepc, read EEPROM programming cycle	45
4.4.41 rdf, read filter	45
4.4.42 rdgf, read gear factor	45
4.4.43 rdhac, read home acceleration	46
4.4.44 rdhvl, read home velocity	46
4.4.45 rdifs, read interface status	47
4.4.45.1 Axis qualifier ifs	47
4.4.46 rdifsb, read interface status bit	47
4.4.47 rdipw, read in position window	48

4.4.48 rdirqpc, read interrupt request PC	48
4.4.49 rdjac, read jog acceleration.....	48
4.4.50 rdjtv, read jog target velocity	49
4.4.51 rdjvl, read jog velocity	49
4.4.52 rdledgn, read led green.....	49
4.4.53 rdledrd, read led red	50
4.4.54 rdledyl, read led yellow.....	50
4.4.55 rdldp, read latched position.....	50
4.4.56 rdldpndx, read latched position index	51
4.4.57 rdlsml, read left spool memory	51
4.4.58 rdmcp, read motor command port	52
4.4.59 rdmpe, read maximum position error	52
4.4.60 rdrp, read real position.....	53
4.4.61 rdsdec, read stop deceleration	53
4.4.62 rdsll, read software limit left	53
4.4.63 rdsrl, read software limit right.....	54
4.4.64 rdtp, read target position.....	54
4.4.65 rdtrov, read trajectory override	54
4.4.66 rifs, reset interface status register	55
4.4.67 rs, reset system	55
4.4.68 sdels, spooler delete synchronous	55
4.4.69 shp, set home position.....	56
4.4.70 ssms, start spooled motions synchronous.....	56
4.4.71 sstps, spooler stop synchronous	56
4.4.72 startcnct, start numeric controller task.....	57
4.4.73 stepcnct, step numeric controller task	57
4.4.74 stopcnct, stop numeric controller task	57
4.4.75 txbf, transmit binary file.....	58
4.4.76 uf, update filter	59
4.4.77 utrov, update trajectory override.....	59
4.4.78 wrbcnct, write common buffer CNC-Task.....	59
4.4.79 wrcd, write common double	60
4.4.80 wrcl, write common integer	60
4.4.81 wrdigob, write digital outputs.....	61
4.4.82 wrdigob, write digital output bit.....	61
4.4.83 wrdp, write desired position	62
4.4.84 wrgf, write gear factor	62
4.4.85 wrhac, write home acceleration	63
4.4.86 wrhvl, write home velocity	63
4.4.87 wrpw, write in position window	63
4.4.88 wrjac, write jog acceleration.....	64
4.4.89 wrjovr, write jog override.....	64
4.4.90 wrjtv, write jog target velocity	64
4.4.91 wrjvl, write jog velocity	65
4.4.92 wrledgn, write led green.....	65
4.4.93 wrledrd, write led red.....	65
4.4.94 wrledyl, write led yellow.....	65
4.4.95 wrldp, write latched position.....	66
4.4.96 wrldpndx, write latched position index	66
4.4.97 wrmcp, write motor command port	67
4.4.98 wrmpe, write maximum position error.....	68
4.4.99 wrrp, write real position	68
4.4.100 wrsdec, write stop deceleration	68
4.4.101 wrsll, write software limit left	69
4.4.102 wrsrl, write software limit right.....	69
4.4.103 wrtrovr, write trajectory override.....	69
4.5 Accessing the PA8000 over the I/O address area	70

4.5.1 Function libraries for PA8000 I/O programming	70
4.5.2 DLL library for the PA8000 I/O programming function.....	70
4.5.3 Interface library for the Borland DELPHI programming language	70

5 The rw_SymPas programming language for stand-alone application programming 71

5.1 Introduction	71
5.2 Lexical grammar	71
5.2.1 Whitespace	71
5.2.2 Comments	71
5.2.3 Symbole	72
5.2.3.1 Keywords	72
5.2.3.2 Designators	72
5.2.3.2.1 Name and length restrictions	73
5.2.3.2.2 Designator upper and lower case.....	73
5.2.3.2.3 Unambiguity and validity of designators	73
5.2.3.3 Standard designators	73
5.2.3.4 Axis designators	73
5.2.3.5 Qualified designators.....	74
5.2.3.6 Labels	74
5.2.3.7 Constants	75
5.2.3.7.1 Integer constants.....	76
5.2.3.7.1.1 Decimal constants.....	76
5.2.3.7.1.2 Hexadecimal constants.....	76
5.2.3.7.2 Floating-point constants	76
5.2.3.7.2.1 The type of floating-point constants	76
5.2.3.7.2.2 Declaration of constants	76
5.2.3.7.3 Punctuation characters	76
5.2.3.7.3.1 Parentheses	77
5.2.3.7.3.2 Comma	77
5.2.3.7.3.3 Semi-colon	77
5.2.3.7.3.4 Equals sign.....	77
5.3 Semantic grammar.....	78
5.3.1 Declarations.....	78
5.3.1.1 Objects	78
5.3.1.2 Typens.....	78
5.3.1.2.1 Boolean type.....	78
5.3.1.2.2 Integer type	79
5.3.1.2.3 Floating-point types (real types)	79
5.3.1.2.4 Assignment compatibility of types	79
5.3.1.3 Variables.....	80
5.3.1.3.1 Automatic type conversion	80
5.3.2 Blocks, locality and range of application.....	80
5.3.2.1 Syntax.....	80
5.3.2.1.1 Declaration section I.....	80
5.3.2.1.1.1 Label declaration section	81
5.3.2.1.1.2 Constant declaration section.....	81
5.3.2.1.1.3 Variable declaration section.....	81
5.3.2.1.2 Command section	81
5.3.2.2 Range of application.....	82
5.3.2.2.1 Redclaration in a subordinate block	82
5.3.2.2.2 The location of a declaration in a block.....	82
5.3.2.2.3 Redclarations inside a block	82
5.3.2.2.4 Standard designators	82
5.3.3 Variables	83
5.3.3.1 The declaration of variables	83
5.3.3.1.1 Timer declaration	84

5.3.3.2 Conversion of variable types	84
5.3.4 Expressions	85
5.3.4.1 Syntax of expressions	86
5.3.4.2 Operators.....	86
5.3.4.3 Arithmetical operators.....	86
5.3.4.4 Logic operators.....	87
5.3.4.5 Boolean operators	87
5.3.4.6 Relational operators	87
5.3.5 Statements.....	88
5.3.5.1 Assignments.....	88
5.3.5.2 Procedure calls.....	88
5.3.5.3 The goto statement	88
5.3.5.4 Structured instructions.....	89
5.3.5.5 Compound statements	89
5.3.5.6 Conditional statements.....	90
5.3.5.6.1 The if statement	90
5.3.5.7 Loops.....	90
5.3.5.7.1 The while statement	90
5.3.5.7.2 The repeat statement.....	91
5.3.5.7.3 The for statement.....	91
5.3.6 Procedures and functions	92
5.3.6.1 Procedure declarations.....	92
5.3.6.2 Function declarations	92
5.3.7 The syntax of an <i>rw_SymPas</i> program	92
5.3.7.1 The program descriptor	93
5.3.7.2 The program block	93

6 Stand-alone application programming 94

6.1 Introduction	94
6.2 <i>rw_SymPas</i> example programs.....	94
6.3 Abbreviations, system parameters, axis specifiers and axis qualifiers	94
6.3.1 System parameters.....	95
6.3.1.1 PC interrupt generation	95
6.3.1.2 System parameters for unit processing.....	96
6.3.2 Axis specifiers.....	96
6.3.3 Axis qualifiers.....	97
6.3.4 Structured axis qualifiers	99
6.3.5 Abbreviations	99
6.4 Reserved procedure names with event function.....	100
6.4.1 Event procedure EVEO	100
6.4.2 Event-Prozedur EVDNR	100
6.4.3 Event procedure EVLSH.....	100
6.4.4 Event procedure EVLSS.....	101
6.4.5 Event procedure EVMPE	101
6.4.6 Event procedure EVUI	101
6.4.7 Priority and processing sequence for the event procedures.....	101
6.5 SAP block commands.....	102
6.6 <i>rw_SymPas</i> SAP command reference list	103
6.6.1 Structure of the reference list.....	103
6.6.2 ABORT, abort	103
6.6.3 ABS, absolute function	104
6.6.4 ACOS, arc cosine function.....	104
6.6.5 ASIN, arc sine function	104
6.6.6 ATAN, arc tangent function.....	104

6.6.7 AZO, activate zero offsets	104
6.6.8 CL, close loop	105
6.6.9 CONTCNCT, continue CNC-Task	105
6.6.10 COS, cosine function	105
6.6.11 COSH, hyperbolic cosine function	105
6.6.12 DISEV, disable event	106
6.6.13 ENEV, enable event	106
6.6.14 EXP, exponential function	106
6.6.15 JA, jog absolute	106
6.6.16 JAW, jog absolute waiting	107
6.6.17 JHI, jog home index	107
6.6.18 JHIW, jog home index waiting	107
6.6.19 JHL, jog home left	108
6.6.20 JHLW, jog home left waiting	108
6.6.21 JHR, jog home right	108
6.6.22 JHRW, jog home right waiting	109
6.6.23 JR, jog relative	109
6.6.24 JRW, jog relative waiting	109
6.6.25 JS, jog stop	109
6.6.26 JSW, jog stop waiting	110
6.6.27 LN, natural logarithm function	110
6.6.28 MCA, move circular absolute SMCA, spool motion circular absolute	110
6.6.29 MCAW, move circular absolute waiting	110
6.6.30 MCR, move circular relative SMCR, spool motion circular relative	111
6.6.31 MCRW, move circular relative waiting	111
6.6.32 MHA, move helical absolute SMHA, spool motion helical absolute	111
6.6.33 MHAW, move helical absolute waiting	111
6.6.34 MHR, move helical relative SMHR, spool motion helical relative	111
6.6.35 MHRW, move helical relative waiting	112
6.6.36 MLA, move linear absolute SMLA, spool motion linear absolute	112
6.6.37 MLAW, move linear absolute waiting	112
6.6.38 MLR, move linear relative SMLR, spool motion linear relative	112
6.6.39 MLRW, move linear relative waiting	113
6.6.40 MS, motion stop	113
6.6.41 MSW, motion stop waiting	113
6.6.42 OL, open loop	113
6.6.43 RA, reset axis	114
6.6.44 RDCBD, read COMMON BUFFER double function	114
6.6.45 RDCBI, read COMMON BUFFER integer function	114
6.6.46 RDCBS, read COMMON BUFFER single function	115
6.6.47 RS, reset system	115
6.6.48 SHP, set home position	115
6.6.49 SIN, sine function	116
6.6.50 SINH, hyperbolic sine function	116
6.6.51 SQRT, square root function	116
6.6.52 SSMS, start spooled motions synchronous	117
6.6.53 SSMSW, start spooled motions synchronous waiting	117
6.6.54 STARTCNCT, start CNC-Task	117
6.6.55 STOP, stop	118
6.6.56 STOPCNCT, stop CNC-Task	118
6.6.57 TAN, tangent function	118
6.6.58 TANH, hyperbolic tangent function	119
6.6.59 UF, update filter	119
6.6.60 UTROVR, update trajectory override	119
6.6.61 WRCBI, write COMMON BUFFER integer procedure	120
6.6.62 WRCBS, write COMMON BUFFER single procedure	120
6.6.63 WRCBD, write COMMON BUFFER double procedure	120

6.6.64 WT, wait timer.....	121
6.7 Compiler commands.....	122
6.7.1 Include file.....	122
6.7.2 Compiler commands, task selection.....	122

1 Introduction

This manual contains all the details you will need for programming the PA8000. Before the various programming methods and operating modes can be presented, we must first describe various functions provided by the *rw_TOS* operating system software.

Please note: you will find further information on *rw_TOS* in the Operating Manual [OM / chapter 4.1].

2 Internal details of the *rw_TOS* operating system software

As already mentioned in the Operating Manual, one of the main factors in the PA8000's performance capabilities is the *rw_TOS* operating system software. The following chapters will describe the functions implemented in *rw_TOS*, like profile generation or limit switch handling.

2.1 The PA8000 position controller

The PA8000's basic operating mode is the position control mode. In this operating mode, the PA8000 attempts to keep the motor position in the setpoint position. The control loop usually consists of the following components: digital controller - digital/analog converter - power section - motor - encoder - pulse acquisition. The encoder is in most cases attached directly to the motor, i.e. rigidly connected to the motor axis.

If this is not the case, the transmission elements between motor axis and encoder axis are also incorporated in the control loop. The load is also connected to the motor axis. The response of the control system is determined by all the elements contained in the control loop, and by the load. In any given system, the control response can be influenced only by the filter parameters of the digital filter. Remember that all possible operating cases (e.g. changes in load) have to be allowed for.

2.1.1 Control loop opened/closed

After power-up, the control loop is at first open. The value 0 is outputted on the manipulated variable output (Motor-Command-Port). The axis connected can be traversed in uncontrolled mode by outputting a value. The PCAP command *cl()* (close loop) is used to close the control loop. Note that the current position is accepted as the setpoint position, in order to prevent the motor axis being traversed unintentionally. Traversing profiles cannot be carried out until the position control has been activated. This also applies for stepping motors.

2.1.2 PIDF filter

The digital filter has the structure of a real PIDF filter. Almost all controlled systems encountered in practice can be stably adjusted with this type of controller.

2.1.2.1 The filter parameters K_D , K_I , K_P

The setting procedure utilizes the filter parameters K_D , K_I and K_P . The significance of these parameters can be very simply understood in terms of the common parameters encountered in the literature: proportional amplification K_P , derivative-action time K_D , and integral-action time K_I .

- K_P - Proportional amplification
- K_I - Integral-action coefficient
- K_D - Derivative-action coefficient
- T_V - Derivative-action time
- T_N - Integral-action time

$$\begin{aligned} K_I &= K_P / T_N \\ K_D &= K_P * T_V \end{aligned}$$

If a controller with a different structure is to be implemented, the individual components involved can be simply de-activated by setting them to zero.

2.1.2.2 Additional phase element

The digital PIDF filter provided is in the standard version cascaded with a first-order time-delay element with a time constant of $T_A/2$ (half the scan time). This is why it is referred to as a real PIDF filter. The filter parameter K_{PL} can now be used to reduce this time-delay still further, thus making a harder controller setting possible. The K_{PL} parameter may in theory assume any value between 0 and 1. In practice, however, a value greater than approx. 0.95 is no longer expedient.

The connection between K_{PL} and the time delay can be simply represented as:

K_{PL}	- Filter parameter
T_{DELAY}	- real time-delay of the PIDF filter
T_A	- Scan time

$$T_{DELAY} = (1 - K_{PL}) * T_A / 2$$

2.1.2.3 Scan time

In the paragraph above, the scan time T_A was used: this is a characteristic variable for the digital controller. The scan time is the time after which setpoint and actual values are each scanned, and the command value is computed using the control algorithm. If the scan time is small compared to the system time constants involved, the controller can be dimensioned like a continuous controller. This means that no special knowledge of digital control engineering is required for adjustment purposes.

Note: In the standard version of the PA8000, the scan time has been set to **1.28 ms**.

2.2 The PA8000 profile generator

When traversing with the individual axes, the specified paths are approached with a trapezoidal speed profile. For a trapezoidal speed profile of this kind, the determinant variables are initial velocity, initial position, acceleration, maximum velocity, target position and target velocity. The profile generation feature under discussion here generates the appropriate setpoint values for the position controller [chapter 2.1] synchronously with the scans, so that starting from the current position the axis accelerates from the current velocity up to the maximum velocity. The initial velocity and initial position are instantaneous values, and are not specified as parameters for a motion profile. Before the target position is reached, the profile generator decelerates in good time with the specified deceleration, so that the target velocity is reached in the specified target point.

2.2.1 Profile generation for JOG commands

There are certain special cases possible when running a trapezoidal speed profile (single-axis movements):

- The initial velocity is negative in terms of the traversing direction. This means that the axis is initially traversing in the wrong direction, but decelerates, reverses, and now accelerates in the right direction.
- The final velocity is negative in terms of the traversing direction. The axis initially moves beyond the target point, decelerates, reverses its direction, and has the target velocity when it reaches the target point again.
- The initial velocity is equal to the maximum velocity.
- The initial velocity is higher than the maximum velocity. In this case, the axis is automatically decelerated to the maximum velocity.
- The final velocity is equal to the maximum velocity.
- The maximum velocity is not reached, because the axis has to be decelerated beforehand in order to reach the target velocity by the time it gets to the target position. In this case, a triangular speed profile is run.

All these cases will be correctly handled if the distance to be travelled is sufficient. In addition, a positive maximum velocity and acceleration must always be specified, and the final velocity must be smaller than or equal to the maximum velocity. When a negative acceleration is stated, this will be utilized for the profile's braking ramp.

With JOG traversing commands, it is thus possible to program acceleration ramps and braking ramps with differing degrees of steepness.

In the cases listed below, velocity jumps occur (undesirably high accelerations). If these cannot be implemented by the system, a position error will occur, which will, however, generally be corrected after a limited time period. When stepping motors are used, these cases cannot usually be permitted.

- The traverse distance specified is not sufficient for deceleration.
- The target velocity is higher than the maximum velocity. In this case, the traversing velocity is set to target velocity at the end of the profile.

2.2.2 Profile generation for MOVE commands

When running a trapezoidal speed profile with MOVE traversing commands (multiple-axis movements with interpolation) with one or more than one axis, the following special cases are possible:

- The final velocity is negative in relation to the traversing direction. The system first traverses beyond the target point, decelerates, reverses direction, and when it reaches the target point again possesses the target velocity.
- The initial velocity is equal to the maximum velocity.
- The initial velocity is higher than the maximum velocity. With direct MOVE commands, the system automatically decelerates down to maximum velocity in this case. With spooler commands, the initial velocity is set to maximum velocity. This corresponds to a velocity jump.
- The final velocity is equal to the maximum velocity.
- The maximum velocity is not reached, since the system must decelerate beforehand in order to reach the target velocity by the time the target position is reached. In this case, a triangular speed profile is run.

All these cases are handled correctly if the traverse distance is sufficient in each case. Furthermore, a positive maximum velocity and acceleration must always be stated, and the final velocity must be smaller than or equal to the maximum velocity. If a negative acceleration or negative maximum velocity is stated, the profile will be discarded. With MOVE traversing commands, it is not possible to program acceleration ramps and braking ramps with differing degrees of steepness in one traversing command. Should this be required, you can, however, program several MOVE commands consecutively.

In the cases listed below, velocity jumps are involved, i.e. unwantedly high accelerations. If these cannot be implemented by the system, a position error will occur, which will, however, generally be corrected again after a limited time period. These cases must not as a rule be permitted in conjunction with stepping motors.

- The traverse distance stated is not sufficient for accelerating up to target velocity. In this case, the target velocity is set to a value which can actually be reached within the profile stated. In this case, there will however be no velocity jump.
- The traverse distance stated is not sufficient for deceleration. In this case, the profile's initial velocity is set to a value which permits deceleration down to final velocity within the profile stated.
- The target velocity is higher than the maximum velocity. In this case, the traversing velocity is set to target velocity at the end of the profile.
- The traversing profile's direction is altered. In this case, the amount of the velocity vector is taken from the previous direction and placed in the direction now to be traversed. In this case, there will be velocity jumps of varying magnitude at the axes involved. Special caution is required here when stepping motor systems are used.

This type of profile generation is not only executed when linear MOVE commands are being run. This pattern is also used for generating the trajectory velocity when running circular movements with two axes.

2.2.3 Acceleration

If an acceleration smaller than zero is stated, then the data record is discarded with MOVE commands. With JOG commands, a negative acceleration specifies the steepness of the braking ramp. As a default, the braking ramp and the acceleration ramp are of identical steepness, or an endless profile is run into the wrong direction (JOG). The units for the acceleration can be axis-specifically stated in the *mcf.exe* utility program. For the interpolation commands (MOVE commands) there are various options for selecting the units. The value for acceleration is specified as a floating-point number, meaning that the value range is almost unlimited. If you specify an acceleration higher than the system can implement, an enlarged position error will be produced during the acceleration phase.

2.2.4 Maximum velocity

The maximum velocity must always be specified as greater than zero, otherwise the data record will be rejected (MOVE commands) or an endless profile will be run in the wrong direction (JOG). The units for the maximum velocity can be axis-specifically specified in the *mcf.exe* utility program. For the interpolation commands there are various options for selecting the units. The value for the maximum velocity is specified as a floating-point number, meaning that the value range is almost unlimited. If you specify a velocity higher than the system can implement, an enlarged position error will be produced during traversing. If the maximum velocity specified is smaller than the initial velocity, the conditions mentioned above shall apply, depending on the command type involved.

2.2.5 Target velocity

The target velocity can be specified as positive, negative or natural with 0. The direction of the target velocity is always referenced to the direction of traversing. If traversing is in a negative direction, and the target velocity is positive, this means the system will continue to move in a negative direction. The target velocity has the same unit as the maximum velocity. The value is specified as a floating-point number, meaning that the value range is almost unlimited. If you specify a velocity higher than the system can implement, an enlarged position error will be produced during traversing. If the target velocity specified is greater than the maximum velocity, the traversing profile will be concluded with a velocity jump. The current velocity will in this case be set to the target velocity at the end of the profile.

2.2.6 Velocity correction

In certain cases, you may want to alter the axis or trajectory velocity during execution of a trapezoidal speed profile. A typical example of this is manual velocity correction (override). You have various SAP and PCAP commands available for this purpose.

The velocity correction factor, whose default value is 1.0, acts on velocities and accelerations alike.

2.2.7 Target position / Traverse distance

The target can be specified as a relative or absolute value. If you specify a relative value traversing will be by the distance specified, i.e. you have programmed a traverse distance. If you specify an absolute value, the system will traverse to the position specified, i.e. you have programmed a target position. The reference point for absolute target positions is the machine zero.

2.2.8 Operating modes for command processing

Traversing commands and other commands can be executed in two different operating modes, the "direct mode" and the "spool mode". The operating mode being implemented at any one time is automatically specified by the syntax of the command involved.

Note: The command abbreviations for the *spool* commands are distinguished from the direct commands by the character 's' as the first letter in the command word. There are identical *spool* commands available for both programming methods, SAP and PCAP programming alike.

2.2.8.1 Direct mode

Direct mode is activated automatically by calling special *move* and *jog* commands. When you program a traversing command in direct mode, the program begins to execute the specified command after a system-entailed time-delay (approx. 2 - 3 scan intervals). A profile which is already running will not be run till its end: the instantaneous values for velocity and position will be accepted as initial values for the current traversing command. If the profile data and the initial values are consistent, i.e. comply with the above requirements, a currently running profile will be seamlessly continued. It is thus possible, for example, to alter the target point of a running profile, to increase the velocity again, or to subsequently alter the deceleration of the braking ramp, or even alter the acceleration during an acceleration ramp. If different profiles are to be run in succession, you have to wait for the end of the profile concerned in each case.

Note: Any data present in the spooler will be rejected when commands are executed in direct mode.

2.2.8.2 Spool mode

In spool mode, a large number of traverse or other commands can be entered in a queue (spooler). Processing of the commands entered in the spooler is started by the PCAP command *ssms()*, for example. During processing, you can write further commands into the spooler. Commands from the spooler are processed one after the other without any time-delay. The free spooler area becomes smaller each time a command is entered, but becomes larger again every time a command is executed. When all commands in the spooler have been processed, the system automatically switches back to direct mode, i.e. after more *spool* commands have been entered, their processing has to be started anew.

Note: For the spooler entries to be processed correctly, the following preconditions must be satisfied::

- All axes for which commands are to be spooled must be in position control at the first spooler entry.
- The velocity of these axes must be zero before the first *spool* command is executed, which is why the Start Spooled Motions Synchronized *ssms()* command may be executed only when all axes involved are at rest.

2.3 Interpolation with the PA8000

Individual axes are moved with the PA8000 using the *jog* commands. The *move* commands are available for moving more than one axis in interpolated mode. The PA8000 enables you to perform circular, linear and helical interpolations. It can process several interpolation profiles simultaneously, with any initial and final velocities you want. All interpolation computations are synchronized with the scan function (1.28 ms).

2.3.1 Linear interpolation

With linear interpolation, any desired number of axes are moved on a line of space (n-dimensional) from the starting point to the target point (absolute positioning) or by a space vector (relative positioning). Parameters used in linear interpolation are the axes involved, the traverse distance or the target position, the trajectory acceleration, the maximum trajectory velocity, and the trajectory target velocity. When interpolating with an initial velocity, you should make sure that the direction vectors for the initial velocity and for the interpolation profile coincide. Otherwise the direction of the velocity vector will be altered, and this may lead to velocity jumps at the axes involved. If the interpolation direction has to be altered from one profile to the next, an intermediate stop should be made. For direction reversal, there is an option for ending the first profile with negative target velocity.

2.3.1.1 Formal linear interpolation

When running contours, there is an option for having one axis retain the instantaneous motor position, while the other axes are run in interpolated mode. This stationary axis can, however, participate formally in this interpolation for the other axes, and thus remains synchronized with them. This formal interpolation is particularly important in the spool operating mode, and is selected automatically for all axes at which a traverse distance of 0 is programmed.

2.3.2 Circular interpolation

Circular interpolation is performed with any two axes. Parameters used for circular interpolation are the axes involved, the coordinates of the circle's centre, the traverse angle (positive or negative), the trajectory acceleration, the trajectory maximum velocity, and the trajectory target velocity. The coordinates of the circle's centre can be specified in absolute or relative coordinates.

When interpolating a circle with an initial velocity, you must always make sure that the initial velocity has the direction you want, i.e. the direction of the tangent in the circle's starting point. Otherwise the direction of the velocity vector will be altered, and this may lead to velocity jumps at the axes involved. If the interpolation direction has to be altered from one profile to the next, an intermediate stop should be made. For direction reversal, there is an option for ending the first profile with negative target velocity.

2.3.3 Helical interpolation

Helical interpolation is executed for any two axes as a circular interpolation, and with any third axis as a linear interpolation.

2.3.4 Synchronous and asynchronous interpolations

One of the options provided by the PA8000 is to process several different interpolations at the same time. It is possible, for example, to execute two circular interpolations with two different axis channels each. The interpolations concerned can be executed synchronously or asynchronously with each other. The synchronous operating mode is supported particularly well by the spooler mechanism. Of course, besides an interpolation, any other axis you want that is not used in an interpolation context can be run independently.

2.4 PA8000 limit switch handling

The PA8000 offers a wide range of options for limit switch handling and traversing range limitation. You have options, for example, for configuring any one or more digital inputs as left or right hardware limit switches. During configuration, a TOM, SMA or SMD function is additionally assigned to the limit switch input. What's more, you can additionally define a software limit switch (left and right) for each axis channel. You can select any limit switch positions you want. Here, too, you can choose between the TOM, SMA and SMD functions. The state of the limit switches can be taken from the *axst* status flag.

A particular limit switch state is erased if the setpoint position is below the limit switch position.

Note: All limit switch states are erased when the control loop is closed [chapter 4.4.4 - *cl()*].

2.4.1 TOM limit switch function(Turn-Off-Motor)

With this limit switch function, the motor is de-energized in the limit switch direction, i.e. the axis comes to rest in uncontrolled mode when the limit switch is tripped, and cannot be moved further into the limit switch zone, only against the limit switch direction. The setpoint position can, however, continue to run into the limit switch zone, e.g. due to a profile currently being run. When it exits from the limit switch zone, uncontrolled velocity jumps may occur.

2.4.2 SMA limit switch function (Stop-Motor-Abruptly)

With this limit switch function, the setpoint position is retained when the limit switch position is exceeded. The position controller halts the axis in this position. The setpoint position computed by the profile generator will, however, be correctly continued internally. When the setpoint value position and leaves the limit switch zone, uncontrolled velocity jumps may occur.

2.4.3 SMD limit switch function (Stop-Motor-Decelerate)

With this limit switch function, the axis concerned is decelerated with the stop deceleration *{sdec}* specified down to zero velocity. The axis is switched to direct mode, and any spooler entries are discarded. It is no longer possible to perform further controlled traversing into the limit switch area. The axis can be moved out of the limit switch area with all traversing commands. This is the multi-purpose limit switch function.

3 The PA8000 programming methods

One of the important features of the PA8000 positioning and contouring control system is the realtime multi-task operating system *rw_TOS* (Transputer Operating System). This is contained in the *rwtos.btl* file, and is loaded, once per PC boot, into the main memory of the TPU-6002 transputer board within a few seconds, using the *mcbt.exe* boot program. *rw_TOS* utilizes the hardware characteristics implemented in the transputer, like task scheduler and multiprocessor parallel processing with other transputer systems over what are called transputer link channels.

The *rw_TOS* operating system software is divided up into various tasks, which basically provide for two different kinds of user programming.

Note: *rwtos.btl* and *mcbt.exe* form part of the PA8000 TOOLSET software. You will find further information in the Operating Manual.

3.1 PC application programming (PCAP programming, or direct programming)

The PA8000 application programming (PCAP) is handled with a user program running on the PC. Programs are written using a higher-level programming language like *Turbo C*, *Microsoft C* or *Turbo Pascal*. By using the function libraries included in the scope of delivery for these programming languages, you can draw on a powerful reservoir of commands, enabling you to create your programs quickly and effectively. The commands available include traversing commands, for example, with and without interpolation, input/output commands, interrogation commands, *spool* commands, etc.

A typical application program transmits one or more of these commands to the PA8000 and then waits for these orders to be processed. After the commands concerned have been autonomously executed by the *PC-Task* in the *rw_TOS* operating system, new command orders can be transferred to the *PC-Task*. The time between command order and command processing can be utilized by the application program to perform other application-specific tasks.

Since programming is performed by directly accessing a PC application program, this programming method is also referred to as "PC direct programming".

Note: In the following chapters, you will occasionally find the term "PCAP command". This type of command is based on the programming method outlined above.

3.2 Stand-alone application programming (SAP programming)

In contrast to PC application programming, stand-alone application programming permits a program to be processed entirely without the aid of a PC application program. An application program written in the *rw_SymPas* programming language is compiled using the *NCC* compiler integrated in the development environment *mcf.exe*, and generates an operating program which the PA8000 can understand.

This operating program can be loaded onto the PA8000, and is executed autonomously using the *CNC-Task* (CNC = Computerized Numerical Control) in *rw_TOS*. If synchronization is required between a PC application program and the PA8000 stand-alone program, this can be carried out using predefined system variables, which both system partners (PC and PA8000) can access.

Note: In the following chapters, you will often encounter the term "SAP command". This type of command is based on the programming method outlined above.

3.2.1 SAP-Multitasking

The operating system software *rw_TOS* can process up to 4 SAP programs simultaneously. All tasks executed simultaneously have the same priority. The different tasks are addressed by means of numbers. The smallest task number has the value of 0, and the largest thus the value of 3.

This multi-tasking programming option enables a complex task to be divided up into small, easy-to-handle subtasks. For example, one task could be used for reference travel, another for monitoring the drive with appropriate EVENT handlers, and yet another for PLC control pure and simple, with appropriate accessing of digital I/O or PC communication with predefined registers.

The various SAP programs can be automatically stopped, started or continued by means of various task control commands.

The CNC tasks are synchronized with each other, synchronization with any parallel-running PCAP application program, and exchange of data between these, can be carried out using predefined registers, what are referred to as COMMON variables. 100 common integer and 100 common floating-point registers are available to all CNC tasks for this purpose.

Each CNC task can also utilize a local memory area of 1000 bytes (COMMON BUFFER), which the PC and the CNC task involved can access in both read and write modes. This can be used to build up a user-specific command set, for example.

4 PC application programming

4.1 Introduction

The PA8000 TOOLSET Software includes library functions for the *Turbo Pascal* (from Version 5.0), *Turbo C* and *Microsoft C* programming languages. The individual functions are executed using the TSR driver *mcutsr.exe*. The significance of the individual function parameters and their data types is identical for both programming languages.

Integration of the function libraries into the programming language involved is explained below:

Turbo Pascal:

The name of the function library is *mcutsr.pas*. These functions are used to establish the link between PC application program and the TSR driver *mcutsr.exe*. This file is declared as a *unit*, and is linked to the application program by means of the *uses* statement.

Important: Various system parameters possess the data type *double*. This means that the user program has to be compiled with the *{\$N+}* option!

C (Turbo C, Microsoft C):

The function library's name is *mcutsr.c*. These functions are used to establish a link between PC application program and the TSR driver *mcutsr.exe*. This file can also be incorporated into the application program using the *#include* statement. An object file could also be generated from this file, and linked to the application program. In this case, the header file *mcutsr.h*, which contains the prototype definition of the individual functions, should be incorporated in the application program.

4.2 Example programs for using the function libraries

The example programs included in the PA8000 TOOLSET software show simple applications for the functions described below. The source texts for the example programs are provided with comments to render them self-explanatory. So there is no need for a detailed description of these example programs at this point. The individual example programs for the two programming languages can be found in the subdirectories specified here, and have the following names:

Turbo Pascal:

ld.pas, mcutsr.pas, move.pas etc.

Directory: TP

C:

ld.c, mcutsr.c, mcutsr.h, move.c etc.

Directory: C

4.3 Definitions, structures and records

Before the individual functions are explained, certain definitions, structures and records will be described, some of which are required as parameters for these functions. The structure/record data fields required are always declared in the application program. The advantage of this is that the TSR driver does not take up too much PC RAM memory.

All the structure/record types and system constants listed below have been defined in the *mcutsr.h* or *mcutsr.pas* files using the programming languages mentioned above.

Note: The system constant *REALAXIS* (*mcutsr.h*) must be set to the actual number of axes present in the system concerned.

4.3.1 Definitions

Table 1: System constants

Name	Type	Function
softint	integer	This parameter is necessary for all function calls. It specifies the software interrupt number under which the TSR driver <i>mcutsr.exe</i> is called. The default value for <i>softint</i> is 60 hex. This value can be altered with the <i>mcfgr.exe</i> configuration program.
MAXAXIS	integer	Maximum number of possible axes. Currently, the TOOLSET software supports up to 18 axes.
LONGINT	long/longint	Data type long in the C programming language, or longint in the <i>Turbo Pascal</i> programming language.

4.3.2 Structures and records

Depending on the programming language involved, we speak either of structures (C) or records (Pascal). The composition and the functioning of these data types is identical in both programming languages. For easier comprehension, all structure or record types are written in capitals, and their components in lower-case characters.

4.3.2.1 Structure/record type AS

Tabelle 2: Structure/record type AS

Element	Type	(Abbr. meaning), Function
unoa	LONGINT	(used number of axis) Number of axes to be selected at various function calls.
san	Field with MAXAXIS LONGINT	(selected axis number) Field of the axes to be selected. This field must be initialized beginning with Index 0, depending on the number of axes used.

Note: Counting for axis channels begins with the value 0.

Example: Selecting the first and third axes

```
as.unoa = 2;      // number of axes
as.san[0] = 0;    // first axis
as.san[1] = 2;    // third axis
```

4.3.2.2 Structure/record type TSRP

To enable you to work with the individual axis systems, a structure/record type *TSRP* has to be declared for each axis. Using the structure/record elements contained in *TSRP*, data are exchanged with the PA8000 at various PCAP commands. For example, axis-specific system variables like accelerations, velocities and positions can be interrogated or set using special read and write commands.

Important: The individual elements of the *TSRP* structure are not initialized automatically, i.e. you have to update them by setting them directly, and reading them in beforehand.

Note: You must be careful to make sure that when more than one axis channel is being used the *TSRP* structures/records are located directly behind each other in memory, since the TSR driver *mcutsr.exe* sometimes accesses the various axis parameters using address computations. Correct arrangement in the PC's main memory is reliably achieved by declaring *TSRP* as a field variable. The size of the field here depends on the number of axes in the system.

Tabelle 3: Structure/record type *TSRP* (axis-specific parameters)

Element	Type	(Abbr. meaning), Function
an	LONGINT	(axis number)
kp	double	(PIDF filter parameter kp)
ki	double	(PIDF filter parameter ki)
kd	double	(PIDF filter parameter kd)
kpl	double	(PIDF filter parameter kpl)
kfca	double	(PIDF forward compensation acceleration)
kfcv	double	(PIDF forward compensation velocity)
jac	double	(jog acceleration)
jvl	double	(jog velocity)
jtv	double	(jog target velocity)
jovr	double	(jog override)
hac	double	(home acceleration)
hvl	double	(home velocity)
rp	double	(real position)
dp	double	(desired position)
tp	double	(target position)
sll	double	(software limit left)
slr	double	(software limit right)
ipw	double	(in position window)
mpe	double	(maximum position error)
gf	double	(gear factor)
mcp	LONGINT	(motor command port)
axst	LONGINT	(axis status)
lsm	LONGINT	(left spool memory)
epc	LONGINT	(eeprom programming cycle)
digi	LONGINT	(digital inputs)
digo	LONGINT	(digital outputs)
ifs	LONGINT	(interface status)
scratch	Field with 4 times LONGINT	(scratch field) wildcard for next <i>TSRP</i> record

4.3.2.3 Structure/record type TRU (Trajectory Units)

This structure or record type is a parameter for the PCAP command *ctru()*.

Table 4: Structure/record type TRU

Element	Type	(Abbr. meaning), Function
pu	LONGINT	position unit
tu	LONGINT	time unit

4.3.2.4 Structure/record type LMP (Linear Motion Parameters)

This structure or record type is a parameter with all linear interpolation commands.

Table 5: Structure/record type LMP

Element	Type	(Abbr. meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tv	double	(target velocity) trajectory target velocity
dtm	Field with MAXAXIS double	(distance to move) This field must be initialized in accordance with the index of the axes used. Index counting begins with 0. The traverse distances desired are entered into the individual elements to suit the positioning mode involved (absolute or relative). Assignment of these specified traverse distances to the desired axis channels must agree with the AS structure/record type.

4.3.2.5 Structure/record type CMP (Circular Motion Parameters)

This structure or record type is a parameter with all circular interpolation commands.

Table 6: Structure/record type CMP

Element	Type	(Abbr. meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tv	double	(target velocity) trajectory target velocity
phi	double	traverse angle in degrees
dtca1	double	(distance to center x-axis)
dtca2	double	(distance to center y-axis)
The assignment of dtca1 and dtca2 to the desired axis channels is established with the structure/record type AS. The axis channel entered there in Field 0 is the x-axis. The y-axis is correspondingly entered in Field 1.		

4.3.2.6 Structure/record type HMP (Helical Motion Parameters)

This structure or record type is a parameter with all helical interpolation commands.

Table 7: Structure/record type HMP

Element	Type	(Abbr. meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tvI	double	(target velocity) trajectory target velocity
phi	double	traverse angle in degrees
dtca1	double	(distance to center x-axis)
dtca2	double	(distance to center y-axis)
dtma3	double	(distance to move z-axis)
The assignment of <i>dtca1</i> , <i>dtca2</i> and <i>dtma3</i> to the desired axis channels is established with the structure/record type AS. The axis channel entered there in Field 0 is the x-axis. The y-axis is entered correspondingly in Field 1, and the z-axis in Field 2.		

4.3.2.7 Structure/record type TOSI (Transputer Operating System Information)

This structure or record type is a parameter for the PCAP initialization command *mcuinit()*. After successful initialization of the PA8000, the following *rw_TOS* data (*rwtos.btl*) are entered in the TOSI structure:

Table 8: Structure/record type TOSI

Element	Type	(Abbr. meaning), Function
revision	Field with SIZE_STRREV characters	Current software revision of the <i>rw_TOS</i> operating system software. This must agree with REVISION system constant.
number_axis	LONGINT	Number of axis channels present
sysfile_loaded	LONGINT	This status variable indicates with the value 1 whether the system file has already been transferred to the PA8000.

Note: You can use the PCAP load command *txbf()* to transfer the system.dat system file (which is altered mainly by means of the TOOLSET program *mcfg.exe*) to the PA8000, where it will trigger initialization of intra-system parameters like accelerations, velocities, filter coefficients, limit values, etc. This load operation must be run once per system boot.

4.3.2.8 Structure/record type CBCNCT (Common Buffer CNC-Task)

Each CNC task is provided with a local memory area with a size of 1000 bytes (COMMON BUFFER), which both the PC and the CNC task involved can access in both read and write modes. This buffer can be used, for example, to build up a user-specific command set.

The structure/record type *CBCNCT* is a parameter for the PCAP commands *rdcbcncct()* and *wrcbcncct()*, which can be used to read or write the COMMON BUFFERS.

Table 10: Structure/record type CBCNCT

Element	Type	(Abbr. meaning), Function
TaskNr	LONGINT	Task Number (0..3)
size	LONGINT	Size of buffer[Bytes]
BuffPtr	Pointer	Pointer to a buffer which you want transferred to the PA8000, or read in from the PA8000. The buffer must be at least <i>size</i> bytes in size!

4.3.2.9 Structure/record type CNCTS (Computerized Numerical Control Task Status)

This structure/record type is a parameter for the PCAP status interrogation command *rdcncts()*.

Table 11: Structure/record type CNCTS

Element	Type	(Abbr. meaning), Function
errnum	LONGINT	Internal CNC task error number. If no error has occurred, then <i>errnum</i> has the value 0.
errline	LONGINT	In connection with <i>errnum</i> , this element is used to display the error-causing source text line of the stand-alone application program.
stackfree	LONGINT	Currently free stack areas [bytes] for the CNC task.
running	LONGINT	This status flag indicates with the value 1 whether the CNC task is currently processing a program.

4.4 PCAP high-level language function reference list

4.4.1 Structure of the reference list

The function and command reference list is sorted alphabetically. The descriptions for the individual commands and functions are structured as follows:

FUNCTION NAME:	This is the name which is used to call the function subsequently described.
ABBR. MEANING:	Here you will find a detailed description of the function name concerned.
TURBO PASCAL:	Here you will find the prototype definitions for the <i>Turbo Pascal</i> programming language. You will see which parameters are required for the function call involved.
C:	Prototype definition for the <i>C</i> programming language, otherwise as for <i>Turbo Pascal</i> .
TSRP COMPONENTS:	Various functions require as parameters components of the structure/record type TSRP. These are listed here.
DESCRIPTION:	Plaintext description of the command.
RETURN VALUE:	If the function returns a value, you will find here a description.
NOTE:	For recurrent notes and explanations, you will find a cross-reference to the appropriate chapters here.
EXAMPLE:	Occasionally, examples are given for the function calls involved.

4.4.2 General information

All commands and functions, except the *spool* commands, are executed immediately after being called. For all *move* and *jog* commands, you must make sure before they are executed that the axes involved have been switched into position control beforehand (PCAP command *cl()*). In addition, some of the motion functions require differentiation between absolute and relative traversing commands. The absolute traversing commands are executed in the absolute measurement system, i.e. are referenced to the machine zero. The relative traversing commands are executed incrementally, i.e. starting from the current motor position.

The end of profile processing is indicated both in direct mode and in spool mode by the *pe* flag in the *axst* register of the structure/record TSRP [chapter 4.4.27 - *rdaxst()*].

In the case of the axis-specific motion commands, (*jog* commands), all system parameters like positions, traverse distances, accelerations and velocities are specified in the axis-specific units stated in the TOOLSET program *mcf.exe*. For the interpolation commands (*move* commands), the units selected in the TRU structure (record) are utilized.

Conversion between application-specific and intra-system units is made automatically, using the factors specified in *mcf.exe*. Conversion is determined by the encoder resolution or step number, the gear factor, and the distance and time units selected.

4.4.3 azo, activate zero offsets

TURBO PASCAL: `procedure azo(set_:integer; softint:integer);`

C: `void azo(int set, int softint);`

DESCRIPTION: Each axis channel can be assigned five different zero offsets. You can use the `azo()` command to activate the axis-specific offset parameters you want. In the `set` (or `set_`) parameter, you specify which set of zero offsets you want to have activated. This variable, with the value 0 .. 4, is used to select the set of zero offsets you want. But if the variable has a value greater than 4, no zero offsets will be taken into account any more.

NOTE: Zero offsets are used to specify a new system of coordinates, without having to influence (new setting) the actual machine zero.

4.4.4 cl, close loop

TURBO PASCAL: `procedure cl(var as:AS; softint:integer);`

C: `void cl(struct AS far *as, int softint);`

DESCRIPTION: All axis channels specified in `AS` are brought into position control with this command. Note that the actual positions of the axes involved are accepted as setpoint positions, in order to avoid large system deviations. In addition, all digital outputs planned with PAE are set. These outputs can, for example, be used for controlling relays, which in turn can be used to enable power amplifier units. Depending on what axis channel is selected, the relays K2 (axis channel 1), K3 (axis channel 2) and K4 (axis channel 3) are switched on [OM / chapter 4.1.2.8].

NOTE: The position control causes the PIDF filter to be processed with the appropriately set filter coefficients.
When the position control loop is closed, all spooler data for the axis channels specified will be rejected!

4.4.5 contcnct, continue numeric controller task

TURBO PASCAL: `procedure contcnct(TaskNr:integer; softint:integer);`

C: `void contcnct(int TaskNr, int softint);`

DESCRIPTION: You can use this command to continue a SAP program which has previously been halted with the SAP command `STOP`, `STOPCNCT()` or with the PCAP command `stopcnct()`. The task selected in `TaskNr` (values 0..3) will be continued.

NOTE: A SAP program which has been halted with the SAP command `ABORT`, can only be restarted (i.e. not continued) with the SAP command `STARTCNCT()` or the PCAP command `startcnct()`.

4.4.6 ctru, change trajectory units

TURBO PASCAL: `procedure ctru(var tru:TRU; softint:integer);`

C: `void ctru(struct TRU far *tru, int softint);`

DESCRIPTION: This command can be used to switch over the units for the velocity, acceleration and position parameters of all interpolation commands (*move* commands). The parameters are specified in the units selected.

The following values are permitted for the TRU structure component *pu* (position unit):

Index	Unit	Description
0	mm	Millimeter
1	inch	Inch
2	m	Meter
3	rev	Revolution
4	deg	Degree
5	rad	Radian
6	counts	Counts
7	steps	Steps

The following values are permitted for the TRU structure component *tu* (time unit):

Index	Unit	Description
0	sec	Seconds
1	min	Minutes
2	tsample	Sampling Time

NOTE: The default value for *pu* and *tu* is 0. This means that for all distance particulars the unit [mm] is assumed, for velocities the unit [mm/s], and for accelerations the unit [mm/s²]. The units selected are utilized only for interpolation commands (all *move* commands)! If the commands involved are axis-specific motion commands (all *jog* commands), the axis units specified in *mcfg.exe* are taken into account. The units selected are also determinant for any SAP program running in parallel.

4.4.7 dummy, dummy function call

TURBO PASCAL: function dummy(softint:integer):integer;

C: int dummy(int softint);

DESCRIPTION: This command has no effect on the PA8000, but returns the value -1. If this value is returned, you can assume that the right TSR driver (*mcutsr.exe*) has been loaded.

4.4.8 InitMcuSystem, initialise mcu system

TURBO PASCAL: function InitMcuSystem(var tsrp:TSRP; softint:integer):integer;

C: int InitMcuSystem(var TSRP far *tsrp, int softint);

DESCRIPTION: This function performs the complete software initialization routine for the drive system. The function call should be executed at the beginning of every PCAP application program. Inside this function, various PCAP basic functions are called. This includes initialization of the axis numbers {an} in the *tsrp* structure. You must make sure that the number of axes actually present in the system has been appropriately set in the *REALAXIS* system constant. The function also checks whether the TSR driver *mcutsr.exe* has been loaded resident into PC main memory, and whether the driver loaded is in fact the correct one. If the *system.dat* system file has not yet been transferred onto the PA8000, this will be done here. At the end of the function, the axis parameters of all axes are read into the *tsrp* structure.

NOTE: PCAP commands *txbf()*, *mcuinit()*, structure/record type TOSI

RETURN VALUE: The function can return the following values:

Return value	Error description
0	No error
30	TSR driver <i>mcutsr.exe</i> has not been loaded into resident into PC main memory at <i>softint</i>
31	Wrong TSR driver
32	The PA8000 cannot be accessed. This error may have the following causes: - the <i>rw_TOS</i> operating system software has not been loaded (<i>mcbt.exe</i>) - the PA8000 base address has been incorrectly set [OM / chapter 4.2] - the PA8000 has not been installed in the PC
33	Wrong <i>rw_TOS</i> operating system software
<i>lderr</i>	Error return value from PCAP command <i>txbf()</i>

4.4.9 ja, jog absolute

TURBO PASCAL: procedure ja(var as:AS; var tsrp:TSRP; softint:integer);

C: void ja(struct AS far *as, struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].tp
 n = 0 .. Number of axes present-1

DESCRIPTION: The axis channels selected in *AS* are moved absolutely to the target positions specified in *TSRP[n].tp*, using a trapezoidal speed profile. The profile is generated using the axis-specific system parameters *jac* (*jog* acceleration), *jvl* (*jog* velocity) and *jtv* (*jog* target velocity). You can set and interrogate these parameters at any time using write and read commands. The default values are specified in the *mcfg.exe* utility program. The trajectory parameters are stated in the axis-specific units (distance, time) specified in *mcfg.exe*.

NOTE: If this command is executed simultaneously for more than one axis, these may (due to the axis-specific system parameters) reach the target positions at different points in time [chapter 2.2.8.1].
 You can set and interrogate the axis-specific parameters like accelerations and velocities at any time using write and read commands.

4.4.10 jhi, jog home index

TURBO PASCAL: procedure jhi(var as:AS; var tsrp:TSRP; softint:integer);

C: void jhi(struct AS far *as, struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].tp
n = 0 .. Number of axes present -1

DESCRIPTION: You use this command to start the index search run for all the axis channels selected in AS. The search run is terminated either when the index (zero track) signal of the incremental encoder is activated or after the distance or angle particular specified in *tp* has been exceeded. The search run is carried out using a trapezoidal speed profile. The parameters for the profile generator are the system data *hac* and *hvl*, which can be set using *mcfg.exe* or the appropriate write commands. When the index signal (zero track) is detected, the motor is decelerated with the deceleration *hac* to velocity 0. The *tp* parameter is stated as a relative traverse distance in the axis-specific position unit. The search direction is determined by the sign of *tp*. Generally, the axis system is first run in relation to a reference switch (cam). To eliminate the mechanical inaccuracy of this cam, the obvious solution is to perform the index search run afterwards. The command can be executed with the aid of the profile flag in the *axst* register and the state of the index signal interrogated with the *digi* register [chapter 4.4.34]. The profile flag remains set to 1 until the end of the search run.

NOTE: To maximize the accuracy of index positioning, the search run should be executed with as small a traversing velocity as possible. You do, however, also have an option for performing the search run in two steps. In the first of these steps, the search run can be started in a positive traversing direction, for example, at a relatively high search speed. In the second step, the search run is then concluded in the negative direction at a low search speed. The search speed can be read and written with the PCAP commands *rdhvl()* and *wrhvl()*.

4.4.11 jhl, jog home left

TURBO PASCAL: procedure jhl(var as:AS; softint:integer);

C: void jhl(struct AS far *as, int softint);

DESCRIPTION: This command starts the reference search run for all axis channels specified in AS, in a negative traversing direction. The search run is executed with the aid of an endless trapezoidal speed profile. The axis-specific system data *hac* and *hvl* here serve as parameters for profile generation. If a digital input of the PA8000 planned with REF function is activated at the axis channel selected, the search run will be terminated by decelerating (with *hac*) the axis to a velocity of 0. This state can be interrogated in the *axst* register with the aid of the *pe* profile flag. The profile flag remains set to 0 until the end of the search run.

4.4.12 jhr, jog home right

TURBO PASCAL: procedure jhr(var as:AS; softint:integer);

C: void jhr(struct AS far *as, int softint);

DESCRIPTION: This command functions in an identical way to the PCAP command *jhl()*, but the search run is started in the positive traversing direction.

4.4.13 jr, jog relative

TURBO PASCAL: procedure jr(var as:AS; var tsrp:TSRP; softint:integer);

C: void jr(struct AS far *as, struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].tp
 n = 0 .. number of existing axes-1

DESCRIPTION: This command is identical to the PCAP command *ja()*, except that the distance particular *tp* is a relative (incremental) traverse distance. Starting from the instantaneous position, the motor is moved by the specified distance (or angle) to the left (negative values) or the right (positive values).

4.4.14 js, jog stop

TURBO PASCAL: procedure js(var as:AS; softint:integer);

C: void js(struct AS far *as, int softint);

DESCRIPTION: The axis channels - selected in AS - are decelerated with the axis-specific time-delay *sdec* to velocity 0, and are hold in position control. Until the end of deceleration the *pe* flag is reset in the *axst* register. You can set and interrogate the time-delay *sdec* at any time using write and read commands. The default values are specified in the *mcfq.exe* utility program.

NOTE: If this command is executed simultaneously for more than one axis, these may (due to the axis-specific system parameters) reach the target positions at different points in time [chapter 2.2.8.1].

4.4.15 **lps, latch position synchronous**

TURBO PASCAL: procedure lps(an:integer; mst:integer; softint:integer);

C: void lps(int an, int mst, int softint);

DESCRIPTION: This command can be used to initiate a latch routine synchronized with the scan cycle of the axis channel selected in *an*. After call-up, the actual position $\{rp\}$ is put into intermediate storage after every *mst* scan intervals. If a latch procedure has taken place, this will be displayed in the *axst* register in the *lpsf* flag (Bit No. 16). The PCAP read command *rdlp()* or the *lp* SAP axis qualifier can be used to read out the position from intermediate storage. Readout will also erase the *lpsf* flag in the *axst* register.

NOTE: The command is primarily used when recording contours and teach-in applications, since it enables position data in real time to be recorded from one or more axes. Typical values for *mst* are 10 ... 100 scan intervals (-> 12.8 ms ... 128.0 ms). The precise value will, however, depend on the processing speed of the application concerned.

4.4.16 **mca, move circular absolute smca, spool motion circular absolute**

TURBO PASCAL: procedure mca(var as:AS; var cmp:CMP; softint:integer);
 procedure smca(var as:AS; var cmp:CMP; softint:integer);

C: void mca(struct AS far *as, struct CMP far *cmp, int softint);
 void smca(struct AS far *as, struct CMP far *cmp, int softint);

DESCRIPTION: This command causes circular interpolation of the first two axis channels specified in AS. There are no restrictions regarding axis selection. Circular interpolation is carried out on the basis of a trapezoidal speed profile, i.e. taking into account maximum acceleration and maximum velocity. The structure/record components specified in CMP are utilized as interpolation parameters. These are the trajectory acceleration *ac*, the trajectory velocity *vl* and the trajectory target velocity *tvI*. The coordinates entered in *dtca1* and *dtca2* specify the centre of the circle in an absolute system of units. Note that *dtca1* is assigned to the first axis programmed in AS, and *dtca2* to the second axis specified in AS. The units for the trajectory parameters are selected with the PCAP command *ctru()*.
The angle *phi* specifies the traverse angle to be run with the unit degrees. The sense of rotation is specified by the sign of the angle variable. Positive values signify clockwise rotation, and negative values signify anti-clockwise rotation. The traverse angle range is not fixed to defined limits, i.e. part or multiple circles can be run as well.

NOTE: Chapter 2.3 Interpolation with the PA8000.

4.4.17 mcr, move circular relative smcr, spool motion circular relative

TURBO PASCAL:	<pre>procedure mcr(var as:AS; var cmp:CMP; softint:integer); procedure smcr(var as:AS; var cmp:CMP; softint:integer);</pre>
C:	<pre>void mcr(struct AS far *as, struct CMP far *cmp, int softint); void smcr(struct AS far *as, struct CMP far *cmp, int softint);</pre>
DESCRIPTION:	<p>This command is identical to the PCAP command <i>mca()</i>, except that the coordinates specified in <i>dtca1</i> und <i>dtca2</i> are incrementally (or relatively) referenced to the current motor position.</p>
NOTE:	<p>Chapter 2.3 Interpolation with the PA8000.</p>

4.4.18 mcuinit, motion control unit initialisation

TURBO PASCAL:	<pre>procedure mcuinit(var tosi:TOSI; softint:integer);</pre>
C:	<pre>void mcuinit(struct TOSI far *tosi, int softint);</pre>
DESCRIPTION:	<p>This function is used to carry out various initialization routines inside the TSR driver <i>mcutsr.exe</i>. It checks whether communication is possible between PC and PA8000. If this is the case, the <i>rw_TOS</i>-specific system data returned by the PA8000 are entered in the structure/record TOSI, which can then be used to check the <i>rw_TOS</i>-specific system information for validity.</p> <p>If it has not proved possible to establish communication to the PA8000, the entire TOSI structure will have the value 0.</p>
NOTE:	<p>This command does not trigger a reset on the PA8000. This must be carried out with the PCAP commands <i>ra()</i> or <i>rs()</i>.</p> <p>You can use the TOSI.sysfile_loaded return value to ascertain whether the <i>system.dat</i> system file has already been transferred to the PA8000 with the aid of the PCAP load command <i>txbf()</i>. If this value is 0, then after a successful <i>mcuinit()</i> PCAP command the PCAP command <i>txbf()</i> must be executed, so that you can work with the PA8000 .</p> <p>The PCAP example programs provided include this command in the <i>InitMcuSystem()</i> function, where the monitoring mechanism for system initialization is once more illustrated.</p>

4.4.19 mha, move helical absolute smha, spool motion helical absolute

TURBO PASCAL: procedure mha(var as:AS; var hmp:HMP; softint:integer);
 procedure smha(var as:AS; var hmp:HMP; softint:integer);

C: void mha(struct AS far *as, struct HMP far *hmp, int softint);
 void smha(struct AS far *as, struct HMP far *hmp, int softint);

DESCRIPTION: This command is used to perform a helical interpolation; it is an extension of circular interpolation. This is why the particulars given for the PCAP command *mca()* also apply to this command, except that the trajectory parameters are entered in the structure/record HMP. For the third axis specified in AS, the *dtma3* parameter can be programmed as well. This is the absolute traverse distance for the third axis. While the first two axes perform a circular interpolation, the third executes a linear movement. All three axes reach their target positions at the same moment.

NOTE: This command has not yet been implemented at present!

4.4.20 mhr, move helical relative smhr, spool motion helical relative

TURBO PASCAL: procedure mhr(var as:AS; var hmp:HMP; softint:integer);
 procedure smhr(var as:AS; var hmp:HMP; softint:integer);

C: void mhr(struct AS far *as, struct HMP far *hmp, int softint);
 void smhr(struct AS far *as, struct HMP far *hmp, int softint);

DESCRIPTION: This command is identical to the PCAP command *mha()*, except that the distance particulars programmed in *dtca1*, *dtca2* and *dtma3* are referenced to the instantaneous motor position incrementally (or relatively).

NOTE: This command has not yet been implemented at present!

4.4.21 mla, move linear absolute smla, spool motion linear absolute

TURBO PASCAL: `procedure mla(var as:AS; var lmp:LMP; softint:integer);`
 `procedure smla(var as:AS; var lmp:LMP; softint:integer);`

C: `void mla(struct AS far *as, struct LMP far *lmp, int softint);`
 `void smla(struct AS far *as, struct LMP far *lmp, int softint);`

DESCRIPTION: This command is used to carry out a linear interpolation with absolute target particulars. All axes in n-dimensional space are permitted for interpolation. You specify in AS which axes you want to participate in interpolation. You use LMP to specify the trajectory acceleration *ac*, the trajectory velocity *vl* and the trajectory target velocity *tv* for linear interpolation. The units for the trajectory parameters are selected with the *ctru()* command.

Depending on the number of axes involved (*unoa*), you enter the axes you want in the *san* field and the corresponding traverse distances in the *dtm* field. Note that the traverse distance in the *dtm[n]* field is assigned to the axis number *n + 1*. The interpolation is referenced to the axes entered in AS. The traverse distances are interpreted as absolute distance or angle information, i.e. referenced to the machine zero.

NOTE: Chapter 2.3 Interpolation with the PA8000.

4.4.22 mlr, move linear relative smlr, spool motion linear relative

TURBO PASCAL: `procedure mlr(var as:AS; var lmp:LMP; softint:integer);`
 `procedure smlr(var as:AS; var lmp:LMP; softint:integer);`

C: `void mlr(struct AS far *as, struct LMP far *lmp, int softint);`
 `void smlr(struct AS far *as, struct LMP far *lmp, int softint);`

DESCRIPTION: This command is identical to the PCAP command *m/a()*, except that the traverse distances specified in the *dtm* field are interpreted incrementally or relatively to the instantaneous motor position.

NOTE: Chapter 2.3 Interpolation with the PA8000.

4.4.23 ms, motion stop

TURBO PASCAL: `procedure ms(var as:AS; softint:integer);`

C: `void ms(struct AS far *as, int softint);`

DESCRIPTION: The axis channels selected in AS are decelerated with the trajectory acceleration or axis deceleration currently valid down to zero velocity and kept in position control mode. The *pe* flag in the *axst* register is reset by the time the deceleration procedure has been completed. The direction vector of a perhaps currently ongoing interpolation function is not altered by this command. If the axes selected are currently running a circle, deceleration will be performed on the circular trajectory with the trajectory acceleration specified..
 Axes which traverse with one final velocity are decelerated down to zero velocity with the axis-specific deceleration *sdec*.

NOTE: Axes which are not interpolating jointly may reach the target point at different points in time.

4.4.24 ol, open loop

TURBO PASCAL: `procedure ol(var as:AS; softint:integer);`

C: `void ol(struct AS far *as, int softint);`

DESCRIPTION: This command opens the position control loop of all axes selected in AS. On each of the Motor-Command-Ports, 0 V output voltage is outputted in the case of servo axes, and 0 Hz stepping frequency in the case of stepping motor axes. All PA8000 digital outputs planned with PAE function are de-activated for the axis channels programmed.
 Depending on the axis channels selected, the relais K2 (axis channel 1), K3 (axis channel 2) and K4 (axis channel 3) [OM / chapter 4.1.2.8] are switched off.

NOTE: This command is used mainly in exceptional situations, like limit switch limitation, position error violation, etc.

4.4.25 ra, reset axis

TURBO PASCAL: `procedure ra(var as:AS; softint:integer);`

C: `void ra(struct AS far *as, int softint);`

DESCRIPTION: This command can be used to carry out an axis-specific reset operation. This means that any profile running will be aborted, the position control loop will be opened, the setpoint value will be switched off, any spooler data will be rejected, and the position registers set to zero. The digital outputs are set to the default values planned. The axis-specific override factors (PCAP commands *wrjovr()* and *wrtrovrr()*) are set to the value 1.0. Any software limits planned will no longer be monitored for the axis channels selected in *ra()*.

NOTE: All system data, like accelerations, velocities, filter parameters, etc. remain stored in memory, and therefore need not be loaded anew. This command is mainly used at system initialization or in exceptional situations.

4.4.26 rdap, read axis parameters

TURBO PASCAL:	procedure rdap(var tsrp:TSRP; softint:integer);
C:	void rdap(struct Tsrp far *tsrp, int softint);
TSRP COMPONENTS:	all, i.e. Tsrp[n].an ... Tsrp[n].ifs
DESCRIPTION:	This command can be used to read in all axis-specific input and output variables of the structure and/or the Tsrp record with one read command.
RETURN VALUE:	Once the command has been executed, the input and output variables will be located in the structure or record components concerned in each case, or in the Tsrp record.
NOTE:	The individual structure or record components can also be interrogated, using special read commands. Normally, these read commands are preferred due to the shorter access time involved.

4.4.27 rdaxst, read axis status

TURBO PASCAL:	procedure rdaxst(var tsrp:TSRP; softint:integer);
C:	void rdaxst(struct Tsrp far *tsrp, int softint);
TSRP-KOMPONENTEN:	TSrp[n].axst
DESCRIPTION:	This command can be used to interrogate various axis-specific status and error flags of the ramp and interpolation task. Normally this command is repeated cyclically in the PCAP program, in order to check by means of the <i>pe</i> flag described below whether the traversing commands of the axes involved have been completely processed. In addition, this command causes a series of error flags in the <i>axst</i> register to be updated. These should likewise be evaluated cyclically, to guarantee reliable operating behaviour of the PCAP program.
RETURN VALUE:	After this command has been executed, the bit-coded return value is located in the structure/record component <i>axst</i> , with the structure described in the table below.
NOTE:	[OM / chapter 4.4.8.1] and [OM / chapter 4.4.8.3].

Table 15: Bit-decoded structure of the axst word

Bit No.	Name	Function
0		Not assigned, this flag always has the value 0.
1	<i>eo</i>	Emergency-Out error-flag: has the value 1, when a digital input (as EO-planned) is active.
2	<i>dnr</i>	Drive-Not-Ready error-flag: has the value 1, when a digital input (as DR-planned) is inactive.
3	<i>lslh</i>	Limit-Switch Left Hardware error-flag: has the value 1, when a digital input (as LSL_TOM or LSL_SMA planned) is active.
4	<i>lsrh</i>	Limit-Switch Right Hardware error-flag: has the value 1, when a digital input (as LSR_TOM or LSR_SMA planned) is active.
5	<i>lsls</i>	Limit-Switch left software error-flag: has the value 1, when the left software limit is exceeded. The left software limit is filed in the axis-specific system parameter {sll}. For this flag to become active, two additional conditions must be satisfied: the software limit must be planned with one of the functions TOM, SMA or SMD, and the shp() command must have been executed beforehand.
6	<i>lsrs</i>	Limit-Switch right software error-flag: has the value 1, when the right software limit is exceeded. The right software limit is filed in the axis-specific system parameter {slr}. For this flag to become active, two additional conditions must be satisfied: the software limit must be planned with one of the functions TOM, SMA or SMD, and the shp() command must have been executed beforehand.
7	<i>mpe</i>	Maximum Position error-flag: has the value 1, when the permissible position error has been exceeded. The maximum permitted position error is specified in system parameter {mpe}. The PCAP commands <i>wrmpe()</i> and <i>rdmpe()</i> can be used to alter the parameter even during run time.
8	<i>dhef</i>	Data Handling error-flag: has the value 1, when a data error (e.g. inconsistent profile data) is detected by the <i>rw_TOS</i> operating system. If this error occurs, the control loops of the axis concerned in each case are opened. Closing of the control loop is possible only after a system restart (<i>mcbt.exe</i>) or after the <i>ra()</i> [chapter ra, reset axis 4.4.25] or <i>rs()</i> [chapter 4.4.67] has been executed.
9	<i>cef</i>	Data Configuration error-flag. The cef flag is set when the information for operating modes, signal processing or CPU number on the PA8000 do not agree with the system data (system.dat). The configuration-check is carried out automatically after the following events: <ul style="list-style-type: none"> • after every reset statement (e.g. PCAP-Befehl <i>rs()</i>) • after every transfer of the <i>system.dat</i> system file with the PCAP command <i>txbf()</i>. The cause of the error can be eliminated by saving the system data in the [Save Changes] menu.
10..11		Not assigned, these flags always have the value 0.
12	<i>pe</i>	Profile-End status-flag: has the value 1, when the end of the profile has been reached.
13	<i>cl</i>	Closed-Loop status-flag: has the value 1, when the axis channel is in position control.
14	<i>ip</i>	In-Position Status-flag: has the value 1, when the profile end has been reached and in addition the difference of setpoint and actual position of the axis channel is smaller than the position differential contained in the axis-specific system parameter {ipw}.
15	<i>ui</i>	User Input status-flag: has the value 1, when a digital input (as UI-planned) is active.
16	<i>lpsf</i>	The Latch Position Synchronous Flag indicates that latching has occurred synchronously to the sampling cycle [chapter 4.4.15], or that a digital input (planned with the LP function) has been activated [OM / chapter 4.4.3.1].
17..31		Not assigned, these flags always have the value 0.

4.4.28 rdaxstb, read axis status bit

TURBO PASCAL:	function rdaxstb(an:integer; bitnr:integer; softint:integer):boolean;
C:	int rdaxstb(int an, int bitnr, int softint);
DESCRIPTION:	This function can be used to interrogate <u>one</u> piece of the axis status-information of the PA8000. The axis number must be specified in the <i>an</i> parameter (0, 1, ... <i>REALAXIS</i>).
RETURN VALUE:	The function returns the value 1 or TRUE, if the relevant input of <i>bitnr</i> is active. Assignment of <i>bitnr</i> to the axis status information involved is described in Table 15, but in the case of <i>bitnr</i> counting starts with the value 1, so that to interrogate <i>pe</i> , for example, <i>bitnr</i> must have the value 13!
NOTE:	[OM / chapter 4.4.8.1], [OM / chapter 4.4.8.3] and the PCAP command <i>rdaxst()</i>

4.4.29 rdcbcnct, read common buffer CNC-Task

TURBO PASCAL:	function rdcbcnct(var cbcnct:CBNCT; softint:integer):integer;
C:	int rdcbcnct(struct CBNCT far *cbcnct, int softint);
DESCRIPTION:	Each CNC task has a local memory area, referred to as the "Common Buffer", which can be read and written both by the CNC task involved, and by a PCAP program. This function can be used to read in the complete CNC-task-specific buffer (or part of it). The function parameter <i>cbcnct</i> is used to select the CNC task buffer, the read-in size in bytes, and the memory address where this block is to be read in.
RETURN VALUE:	The <i>rdcbcnct()</i> function has the following bit-coded return value: Bit 0: 1 when invalid Task Number Bit 1: 1 when maximum permitted buffer size exceeded This means that the function normally returns the value 0.
NOTE:	The CNC-task-specific buffer size is <u>1000 bytes</u> . The record structure of CBNCT is described in chapter 4.3.2.8. PCAP command <i>wrcbcnct()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>

4.4.30 rdc, read common double

TURBO PASCAL: procedure rdc(ndx: integer; var cdbuf:CDBUF; softint:integer);

C: void rdc(int ndx, struct CDBUF far *cdbuf, int softint);

DESCRIPTION: This function can be used to read in predefined variables of the CNC task. The variables concerned are the *rw_SymPas* variables CD0 .. CD99. The first parameter here specifies the number *-index-* of the variable you want to have read in. The value range of *index* here is 0 to 99. The second parameter is a pointer to a field with 100 double variables.

RETURN VALUE: The *rdc()* command enters the current value of the relevant *CD* variable in the field specified with *index*.

NOTE: The content of all common variables remains stored in memory even after a system reset operation, executed by the *rs()* command, for example. If you do not want this, you should set the variables concerned to the value you want when starting the program.

4.4.31 rdc, read common integer

TURBO PASCAL: procedure rdc(ndx: integer; var cibuf:CIBUF; softint:integer);

C: void rdc(int ndx, struct CIBUF far *cibuf, int softint);

DESCRIPTION: This command is identical to the PCAP command *rdc()*, except that here it is not values of the double type that are read in, but of the LONGINT type. The values concerned are the *rw_SymPas* variables CI0 .. CI99.

4.4.32 rdcncts, read computerized numeric controller task status

TURBO PASCAL: procedure rdcncts(TaskNr:integer; var cncts:CNCTS; softint:integer):integer;

C: void rdcncts(int TaskNr, struct CNCTS far *cncts, int softint);

DESCRIPTION: This command can be used to interrogate the current status of the CNC task selected in *TaskNr* (values 0..3). After this command has been executed, the results can be found in the structure/record *CNCTS*.

RETURN VALUE: The return values obtained in CNCTS after *rdcncts()* has been executed are described in chapter 4.3.2.9.

4.4.33 rdigi, reset digital inputs

TURBO PASCAL: procedure rdigi(var tsrp:TSRP; softint:integer);

C: void rdigi(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].digi
 n = 0 .. number of axes -1

DESCRIPTION: This function can be used to reset axis-specific status information(s) filed in *digi*.

NOTE: *rddigi()* [chapter 4.4.34]

4.4.34 rddigi, read digital inputs

TURBO PASCAL: procedure rddigi(var tsrp:TSRP; softint:integer);

C: void rddigi(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].digi
 n = 0 .. Number of axes -1

DESCRIPTION: This function you can be used to interrogate the following signal states:

- The current status of the 16 digital inputs of the PA8000
- The current status of the zero-track (index) signal from the incremental coder
- An error of the measured-value-acquisition system put into intermediate storage
- An edge of the zero-track (index) signal from the incremental coder put into intermediate storage
- An edge of the hardware latch signal (strobe) put into intermediate storage

If an input is active, this will be indicated by the bit concerned having the value 1. As an optional extra, all digital inputs in the *mcfg.exe* TOOLSET program can be planned with inversion. It is likewise possible to plan the polarity you want when an incremental coder with index signal is used.

RETURN VALUE: The bit-encoded return value is located in the *digi* structure or record component and is structured as described in the table printed below.

NOTE: There is no specified axis assignment for the digital inputs. Bits 16 ... 19 can be reset by means of the *rdigi()* command [chapter 4.4.33]. [OM / chapter 4.4.3.1] and [OM / chapter 4.4.3.2].

4.4.34.1 Axis-qualifier *digi*

The register *digi* can be used to check the state of the PA8000's digital inputs. Active inputs have the value 1 at the concerned bit position.

Table 12: Bit-coded structure of the *digi* word

Bit No.	Function	X22/Pin
0	Input 1	9
1	Input 2	10
2	Input 3	11
3	Input 4	12
4	Input 5	13
5	Input 6	14
6	Input 7	15
7	Input 8	16
8	Input 9	42
9	Input 10	43
10	Input 11	44
11	Input 12	45
12	Input 13	46
13	Input 14 and hardware strobe signal for latching the actual position (axis channel 1)	47
14	Input 15 and hardware strobe signal for latching the actual position (axis channel 2)	48
15	Input 16 and hardware strobe signal for latching the actual position (axis channel 3)	49
16	Zero track of incremental encoder, axis-specific	--
17	Error of the encoder data acquisition system, axis-specific	--
18	Value of the zero-track signal from the incremental coder (axis-specific) put into intermediate storage	--
19	Value of the latch signal (hardware strobe) (axis-specific) put into intermediate storage	--
20..31	Not assigned, these flags always have the value 0	--

4.4.35 rddigib, read digital input bit

- TURBO PASCAL: function rddigib(an:integer; bitnr:integer; softint:integer):boolean;
- C: int rddigib(int an, int bitnr, int softint);
- DESCRIPTION: This function can be used to interrogate the current state of one PA8000 digital input and other logic signals. The axis number must be specified in the *an* parameter (0, 1, ... *REALAXIS*).
- RETRUN VALUE: The function returns the value 1 or TRUE, if the corresponding input of *bitnr* is active.
- NOTE: Bit numbers 17..20 can be reset via the *rdigi()* command [chapter 4.4.33]. [OM / chapter 4.4.3.1], [OM / chapter 4.4.3.2] and PCAP command *rddigi()*

Table 13: Assignment of *bitnr* to the various PA8000 digital inputs

'bitnr'	Function	X22/Pin
1	Input 1	9
2	Input 2	10
3	Input 3	11
4	Input 4	12
5	Input 5	13
6	Input 6	14
7	Input 7	15
8	Input 8	16
9	Input 9	42
10	Input 10	43
11	Input 11	44
12	Input 12	45
13	Input 13	46
14	Input 14	47
15	Input 15	48
16	Input 16	49
17	Zero track of incremental encoder, axis-specific	--
18	Error of the encoder data acquisition system, axis-specific	--
19	Value of the zero-track signal from the incremental coder (axis-specific) put into intermediate storage	--
20	Value of the latch signal (hardware strobe) (axis-specific) put into intermediate storage	--
	Strobe), achsspezifisch	
21..32	Not assigned, these flags always have the value 0	--

4.4.36 rddigo, read digital outputs

- TURBO PASCAL: procedure rddigo(var tsrp:TSRP; softint:integer);
- C: void rddigo(struct TSRP far *tsrp, int softint);
- TSRP COMPONENTS: TSRP[n].digo
- DESCRIPTION: This command is used to read the current output status of the PA8000 digital outputs into the axis-specific structure/record component *digo*. The bits set there represent outputs set.
- RETURN VALUE: After this command has been executed the bit-coded return values are located in the structure/record component *digo*. This component has the structure/record defined in the PCAP-command *wrdigo()*.

4.4.37 rddigob, read digital output bit

TURBO PASCAL:	function rddigob(an:integer; bitnr:integer; softint:integer):boolean;
C:	int rddigob(int an, int bitnr, int softint);
DESCRIPTION:	This function can be used to interrogate the current state of <u>one</u> PA8000 digital output. The axis number must be specified in parameter <i>an</i> (0, 1, ... <i>REALAXIS</i>).
RETRUN VALUE:	This function returns the value 1 or TRUE, if the corresponding output of <i>bitnr</i> is active. Assignment of <i>bitnr</i> to the outputs involved is shown in the PCAP command <i>wrdigob()</i> .

4.4.38 rddp, read desired position

TURBO PASCAL:	procedure rddp(var tsrp:TSRP; softint:integer);
C:	void rddp(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].dp
DESCRIPTION:	The PA8000 profile generator computes an internal reference variable, referred to as the "setpoint position" (= desired position). This can be read in with this command. Normally, in the position control operating mode, the actual position [chapter 4.4.60 - <i>rdrp()</i>] and this setpoint position must be identical, apart from tolerable deviations.
RETURN VALUE:	After the command has been executed, the setpoint position is available in the <i>dp</i> field. The value is returned in the axis-specific position unit.
NOTE:	This setpoint position is also utilized for setpoint/actual-differential formation, for the automatic position error monitoring function.

4.4.39 rddv, read desired velocity

TURBO PASCAL:	procedure rddv(var tsrp:TSRP; softint:integer);
C:	void rddv(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].dv
DESCRIPTION:	This function returns the axis-specific setpoint velocity of the PA8000 profile generator. In best case the value read in corresponds to the real axis velocity (actual velocity).
RETRUN VALUE:	After the command has been executed, the setpoint velocity is available in the <i>dv</i> register with the axis-specific velocity unit.
NOTE:	The setpoint velocity can only be influenced by corresponding traversing commands.

4.4.40 rdepc, read EEPROM programming cycle

TURBO PASCAL:	procedure rdepc(var tsrp:TSRP; softint:integer);
C:	void rdepc(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].epc
DESCRIPTION:	This function can be used to read the instantaneous number of PA8000 programming cycles. The cycle number is increased by one in the EEPROM for every save operation in the TOOLSET program <i>mcf</i> .exe. The EEPROM can be written at least 10000 times.
RETURN VALUE:	After this command has been executed, the current programming cycle number is in the structure/record component <i>epc</i> .

4.4.41 rdf, read filter

TURBO PASCAL:	procedure rdf(var tsrp:TSRP; softint:integer);
C:	void rdf(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv n = 0 .. number of axes present-1
DESCRIPTION:	This command can be used to read in the current axis-specific PIDF filter coefficients of the PA8000. The default values of these coefficients are specified using the TOOLSET program <i>mcf</i> .exe.
RETURN VALUE:	After the command has been executed, the return values are in the TSPR structure/record components listed above.
NOTE:	You will find further details on the PIDF filter in chapter 2.1.2, [OM / chapter 4.1.1] and [CM / chapter 5.2]. PCAP command <i>uf()</i>

4.4.42 rdgf, read gear factor

TURBO PASCAL:	procedure rdgf(var tsrp:TSRP; softint:integer);
C:	void rdgf(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].gf
DESCRIPTION:	This function returns the axis-specific gear factor {gf}. The default value is specified using the TOOLSET program <i>mcf</i> .exe.
RETURN VALUE:	After the command has been executed, the factor is available in the <i>gf</i> field with the axis-specific unit.
NOTE:	The gear factor can be set at any time with the PCAP command <i>wrgf()</i> .

4.4.43 rdhac, read home acceleration

TURBO PASCAL:	procedure rdhac(var tsrp:TSRP; softint:integer);
C:	void rdhac(struct TSPR far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].hac
DESCRIPTION:	This command can be used to read in the axis-specific reference travel acceleration <i>hac</i> . The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
RETURN VALUE:	After the command has been executed, the reference travel acceleration is available in the <i>hac</i> field. The value is returned in the axis-specific acceleration unit.
NOTE:	The reference travel acceleration can be set at any time with the PCAP command <i>wrhac()</i> .

4.4.44 rdhvl, read home velocity

TURBO PASCAL:	procedure rdhvl(var tsrp:TSRP; softint:integer);
C:	void rdhvl(struct TSPR far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].hvl
DESCRIPTION:	This command can be used to read in the axis-specific reference travel velocity <i>hvl</i> . The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
RETURN VALUE:	After the command has been executed, the reference travel velocity is available in the <i>hvl</i> field. The value is returned in the axis-specific velocity unit.
NOTE:	The reference travel velocity can be set at any time with the PCAP command <i>wrhvl()</i> .

4.4.45 rdifs, read interface status

TURBO PASCAL:	procedure rdifs(var tsrp:TSRP; softint:integer);
C:	void rdifs(struct Tsrp far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].ifs
DESCRIPTION:	This command can be used to read in status information of the PA8000.
RETURN VALUE:	The bit-coded return value is located in the structure/record component <i>ifs</i> and has the structure described in the table below.
NOTE:	[OM / chapter 4.4.8.5]

4.4.45.1 Axis qualifier ifs

This register can be used to interrogate various pieces of status information for the PA8000. If the status information concerned is valid, this is indicated by the value 1 at the bit position involved.

Table 14: Bit-coded structure of the ifs word

Bit No.	Function
0	<i>edv</i> : the system information and data filed in the EPROM are valid.
16	<i>pfe</i> : The Power Fail Error flag is set to "1" whenever the operating voltage at the PA8000 falls below a threshold voltage of 4.75 V. After the module is switched on, the flag is likewise set to "1".
17	<i>wdog</i> : The Watchdog flag is set to "1" if the watchdog logic on the PA8000 has been tripped..
18	<i>iae</i> : The Invalid Access Error flag is set to "1" if an invalid access operation has taken place within the <i>rw_TOS</i> operating system software.
19..31	Not assigned, these flags always have the value 0

Note: In an initialization routine of the *rw_TOS* firmware, error flags 16 ... 18 are copied from an internal logic register into the *ifs* register. The logic register is then erased, i.e. the flags are no longer available after a second booting routine (*mcbt.exe*). The flags can also be reset by the *rifs()* command [chapter 4.4.66].

4.4.46 rdifsb, read interface status bit

TURBO PASCAL:	function rdifsb(an:integer; bitnr:integer; softint:integer):boolean;
C:	int rdifsb(int an, int bitnr, int softint);
DESCRIPTION:	This function can be used to interrogate <u>one</u> piece of PA8000 interface status information. The axis number must be specified in the <i>an</i> parameter (0, 1, ... <i>REALAXIS</i>).
RETURN VALUE:	This function returns the value 1 or TRUE, if the corresponding input of <i>bitnr</i> is active. Assignment of <i>bitnr</i> to the status information concerned is described in Table 14, but in the case of <i>bitnr</i> counting starts with the value 1, i.e. to interrogate <i>edv</i> , for example, <i>bitnr</i> has to have the value 1!
NOTE:	[OM / chapter 4.4.8.5] and PCAP command <i>rdifs()</i>

4.4.47 rdipw, read in position window

TURBO PASCAL: procedure rdipw(var tsrp:TSRP; softint:integer);

C: void rdipw(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].ipw

DESCRIPTION: This function returns the axis-specific In-Position Window.

NOTE: After the comand has been executed, the In-Position Window is available in the *ipw* register in the axis-specific position unit.
PCAP command *wripw()*

4.4.48 rdirqpc, read interrupt request PC

TURBO PASCAL: function rdirqpc(softint:integer):integer;

C: int rdirqpc(int softint);

DESCRIPTION: This command can be used to interrogate the instantaneous status of the interrupt source generated on the PA8000. If the interrupt is active, the function returns the value 1, otherwise the value 0.

NOTE: The interrupt can be set or reset by the system variable *IRQPC* using an SAP program [chapter 6.3.1.1 - PC interrupt generation].

4.4.49 rdjac, read jog accleration

TURBO PASCAL: procedure rdjac(var tsrp:TSRP; softint:integer);

C: void rdjac(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].jac

DESCRIPTION: This command can be used to read in the axis-specific *jog* acceleration *jac*. The default value is specified using the TOOLSET program *mcfg.exe*.

RETURN VALUE: After the command has been executed, the *jog* acceleration is available in the *jac* field. The value is returned in the axis-specific acceleration unit.

NOTE: The *jog* acceleration can be set at any time with the PCAP command *wrjac()*.

4.4.50 rdjtvI, read jog target velocity

TURBO PASCAL:	procedure rdjtvI(var tsrp:TSRP; softint:integer);
C:	void rdjtvI(struct Tsrp far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].jtvI
DESCRIPTION:	This command can be used to read in the axis-specific <i>jog</i> target velocity <i>jtvI</i> . The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
RETURN VALUE:	After the command has been executed, the <i>jog</i> target velocity is available in the <i>jtvI</i> field. The value is returned in the axis-specific velocity unit.
NOTE:	The jog target velocity can be set at any time using the PCAP command <i>wrjtvI()</i> .

4.4.51 rdjvI, read jog velocity

TURBO PASCAL:	procedure rdjvI(var tsrp:TSRP; softint:integer);
C:	void rdjvI(struct Tsrp far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].jvI
DESCRIPTION:	This command can be used to read in the axis-specific <i>jog velocity</i> <i>jvI</i> . The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
RETURN VALUE:	After the command has been executed, the <i>jog</i> velocity is available in the <i>jvI</i> field. The value is returned in the axis-specific velocity unit.
NOTE:	The jog velocity can also be set at any time using the PCAP command <i>wrjvI()</i> .

4.4.52 rdledgn, read led green

TURBO PASCAL:	function rdledgn(softint:integer):integer;
C:	int rdledgn(int softint);
DESCRIPTION:	This function can be used to read in the current state of LED D4 (PA8000, motherboard, green).
RETURN VALUE:	The function's return value is 1, provided the LED is switched on, otherwise it is 0.
NOTE:	PCAP command <i>wrledgn()</i> , system variable <i>LEDGN</i>

4.4.53 rdledrd, read led red

TURBO PASCAL:	function rdledrd(softint:integer):integer;
C:	int rdledrd(int softint);
DESCRIPTION:	This function can be used to read in the current state of LED D2 (PA8000, masterboard, red).
NOTE:	PCAP command <i>wrledrd()</i> , system variable <i>LEDRD</i>

4.4.54 rdledyl, read led yellow

TURBO PASCAL:	function rdledyl(softint:integer):integer;
C:	int rdledyl(int softint);
DESCRIPTION:	This function can be used to read in the current state of LED D3 (PA8000, masterboard, yellow).
NOTE:	PCAP command <i>wrledyl()</i> , system variable <i>LEDYL</i>

4.4.55 rdlp, read latched position

TURBO PASCAL:	procedure rdlp(var tsrp:TSRP; softint:integer);
C:	void rdlp(struct TSRP far *tsrp, int softint);
TSRP-KOMPONENTEN:	TSRP[n].lp
DESCRIPTION:	<p>This function returns the axis-specific latch position. The latching procedure can be triggered by various mechanisms:</p> <ol style="list-style-type: none"> 1. When an input planned with LP function is activated. Here, the maximum time delay is two scan intervals (2.56 ms). A new latching procedure will only be enabled after the latching input has been de-activated. 2. If an <i>lps()</i> PCAP command [chapter 4.4.15] has previously been executed and the time delay specified there in the <i>mst</i> parameter has elapsed. 3. In real time (max. 1 μs time delay) by means of default-setting digital inputs of the PA8000. A new latching procedure will only be enabled after the latching input has been de-activated. <p>In all these methods, the actual position <i>{rp}</i> of the motor axis is put into intermediate storage.</p>
RETURN VALUE:	<p>After the function has been executed, the latch position is available in the <i>lp</i> register in the axis-specific position unit.</p> <p>The priority of the three methods is the same as the order of their listing, i.e. realtime latching has top priority.</p>
REMARK:	PCAP command <i>wrlp()</i>

4.4.56 rdlnpdx, read latched position index

TURBO PASCAL:	procedure rdlnpdx(var tsrp:TSRP; softint:integer);
C:	void rdlnpdx(struct TSRP far *tsrp, int softint);
TS COMPONENTS:	TSRP[n].lp
DESCRIPTION:	This function returns the axis-specific latch position of the index signal (zero track). When the incremental coder's zero track is activated, the actual position { <i>rp</i> } of the motor axis in real time is put into intermediate storage.
RETURN VALUE:	After the function has been executed, the latch position is available in the <i>lp</i> register in the axis-specific position unit.
REMARK:	Latching of the incremental coder's zero track is helpful in the coder verification routine and for reference travel programming. PCAP coommand <i>wrlpndx()</i>

4.4.57 rdlsn, read left spool memory

TURBO PASCAL:	procedure rdlsn(var tsrp:TSRP; softint:integer);
C:	void rdlsn(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].lsn
DESCRIPTION:	This command returns the free spool area in bytes. By means of a PCAP or SAP-application program, the freely available spool area can be interrogated at any time you want, and reloaded if necessary. This enables you to load very large traversing profiles without interrupting profile generation. The spool area is loaded with <i>spool</i> commands, using both programming methods (PCAP and SAP). All <i>spool</i> commands cause the freely available spool area to decrease, and all commands executed from the spool area cause it to grow again.
NOTE:	The spooler size is axis-specific, i.e. the free spool area of the individual axis channels may vary significantly. Approx. 145 kByte of spool area are available for each axis channel.

4.4.58 rdmcp, read motor command port

TURBO PASCAL:	procedure rdmcp(var tsrp:TSRP; softint:integer);
C:	void rdmcp(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].mcp
DESCRIPTION:	This command can be used to read in the current command values of the Motor-Command-Ports.
RETURN VALUE:	The return value is available in the <i>mcp</i> field after the command has been executed.

In the case of servo axes, a value in the range -32767 .. 32767 is returned. This corresponds to a setpoint output voltage of approx. -10V .. +10V.

In the case of stepping motor axes, this value is a time-delay, which is determinant for the stepping frequency outputted. The time-delay can be converted into the unit [s] as follows:

$$tver = (mcp+1) / CLOCK;$$

*Example: with mcp = 12499 and CLOCK = 25mhz
tver = 0.5ms and f = 1khz*

Each time the time-delay *tver* elapses, the pulse signal is switched over, i.e. after $2 \cdot tver$ a stepping signal with $f = 1 / (2 \cdot tver)$ [Hz] is outputted. The value returned in *mcp* lies within the range of -1048575 .. +1048575. The sign determines the current sense of rotation, i.e. for computing *tver* only the absolute value of *mcp* must be utilized. If the value 0 is returned in *mcp*, this means that no stepping signal is being outputted, i.e. the motor is at a standstill.

4.4.59 rdmpe, read maximum position error

TURBO PASCAL:	procedure rdmpe(var tsrp:TSRP; softint:integer);
C:	void rdmpe(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].mpe
DESCRIPTION:	This function returns the axis-specific position error limit value.
NOTE:	After the function has been executed, the maximum permitted position error is available in the <i>mpe</i> register in the axis-specific position unit. PCAP command <i>wrmpe()</i>

4.4.60 rdrp, read real position

TURBO PASCAL: procedure rdrp(var tsrp:TSRP; softint:integer);

C: void rdrp(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].rp

DESCRIPTION: This function returns the axis-specific current position (= actual position or real position). The position can be read out at any time you want, even while the axis is being moved. A new actual value is available in each scan cycle (1.28 ms).

NOTE: After the function has been executed, the current position is available in the *rp* register in the axis-specific position unit

4.4.61 rdsdec, read stop deceleration

TURBO PASCAL: procedure rdsdec(var tsrp:TSRP; softint:integer);

C: void rdsdec(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].sdec

DESCRIPTION: This command returns the axis-specific stop deceleration *sdec*. The default value is specified using the TOOLSET program *mcfg.exe*.

RETURN VALUE: After the command has been executed the stop deceleration is available in the *sdec* field. The value is returned in the axis-specific acceleration unit.

NOTE: The stop deceleration can be set at any time using the PCAP-command *wrsdec()*.

4.4.62 rdsll, read software limit left

TURBO PASCAL: procedure rdsll(var tsrp:TSRP; softint:integer);

C: void rdsll(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].sll

DESCRIPTION: This function returns the axis-specific left software limit position.

NOTE: After the function has been executed, the left software limit position is available in the *sll* register in the axis-specific position unit.
PCAP command *wrsll()*

4.4.63 rdslr, read software limit right

TURBO PASCAL: procedure rdslr(var tsrp:TSRP; softint:integer);

C: void rdslr(struct Tsrp far *tsrp, int softint);

TSRP COMPONENTS: Tsrp[n].slr

DESCRIPTION: This function returns the axis-specific right software limit position.

NOTE: After the function has been executed, the right software limit position is available in the *slr* register in the axis-specific position unit.
PCAP command *wrslr()*

4.4.64 rdtpr, read target position

TURBO PASCAL: procedure rdtpr(var tsrp:TSRP; softint:integer);

C: void rdtpr(struct Tsrp far *tsrp, int softint);

TSRP COMPONENTS: Tsrp[n].tp

DESCRIPTION: This function can be used to interrogate the axis-specific target position. The target position is always returned as an absolute distance or angle quantity.

NOTE: After the function has been executed, the target position of the last traversing command is available in the *tp* register in the axis-specific position unit. This command is used for monitoring purposes only.

4.4.65 rdtrovr, read trajectory override

TURBO PASCAL: procedure rdtrovr(var value:double; softint:integer);

C: void rdtrovr(double *value, int softint);

DESCRIPTION: This command reads a state variable of the currently set trajectory velocity correction value, which is taken into account for all interpolation commands (*move* commands) and the correspondingly selected axes (PCAP command *utrovr()*).

RETURN VALUE: After the command has been executed, the trajectory velocity correction value will be in the *value* variable.

NOTE: PCAP commands *utrovr()*, *wrtrovr()*, *wrtrovr()*, *rdtrovr()* and *rdtrovr()*

4.4.66 rifs, reset interface status register

TURBO PASCAL: procedure rifs(var tsrp:TSRP; softint:integer);

C: void rifs(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].ifs

DESCRIPTION: This command causes various error flags in the PA8000 interface status register *ifs* (error bits 16, 17, 18 - *pfe*, *wdog*, and *iae*) to be reset. Resetting should be performed only in exceptional situations, e.g. in an error monitoring routine.

NOTE: [chapter 4.4.45- *rdifs()*]

4.4.67 rs, reset system

TURBO PASCAL: procedure rs(softint:integer);

C: void rs(int softint);

DESCRIPTION: This command causes the complete axis system to be reset. The digital outputs are set to the default values planned with the aid of the TOOLSET program *mcfg.exe*. On the setpoint value channels 0 V output voltage is outputted in the case of servo axes, and 0 Hz stepping frequency in the case of stepping motor axes. The position control loop is opened for all axes. The spooler data are rejected in their entirety. All CNC task are halted. All software limits planned will no longer be monitored. All override factors (PCAP commands *wrjovr()* and *wrtrov()*) are set to the value 1.0.

NOTE: All system data, like accelerations, velocities, filter parameters, etc. remain stored in memory, and therefore need not be loaded again.
The status flags in register *ifs* are not influenced by this command.
The contents of all common integers and double variables are retained.

4.4.68 sdels, spooler delete synchronous

TURBO PASCAL: procedure sdels(var as:AS; softint:integer);

C: void sdels(struct AS far *as, int softint);

DESCRIPTION: All commands entered in the spooler will be rejected. The entire spooler area is again freely available. Spooler data rejection takes place for the axes specified in AS.

NOTE: The ongoing operation, like a traversing command, will be concluded.

4.4.69 shp, set home position

TURBO PASCAL:	procedure shp(var tsrp:TSRP; softint:integer);
C:	void shp(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].tp n = 0 .. number of axes present-1
DESCRIPTION:	This command can be used to set the axis-specific zero (home position). The <i>tp</i> parameter is stated in the axis-specific position unit. The command is generally used after a reference search run for setting the machine zero. It can be executed in both operating modes: control loop open and control loop closed. In order to prevent jerky motor movements, however, it should not be used while the selected axis channel is being moved.
NOTE:	Until the first time this command is executed, the software limits planned are not being monitored. This means that before execution of the <i>shp()</i> command a reference travel can be carried out using all <i>move</i> and <i>jog</i> commands. After the <i>shp()</i> command has been executed, the software limits are monitored until the next <i>ra()</i> or <i>RA()</i> or <i>rs()</i> or <i>RS</i> command.

4.4.70 ssms, start spooled motions synchronous

TURBO PASCAL:	procedure ssms(var as:AS; softint:integer);
C:	void ssms(struct AS far *as, int softint);
DESCRIPTION:	<i>Spool</i> commands can be used to transfer commands to the PA8000 individual axis channels: they are entered in a queue. The PCAP command <i>ssms()</i> causes a synchronous start for spooler command processing for all axes specified in AS.
NOTE:	chapter 2.2.8.2 - Spool-Mode

4.4.71 sstps, spooler stop synchronous

TURBO PASCAL:	procedure sstps(var as:AS; softint:integer);
C:	void sstps(struct AS far *as, int softint);
DESCRIPTION:	This command is used to interrupt command processing from the spooler for all axis channels selected in AS.
NOTE:	The current command will be processed completely.

4.4.72 startcnct, start numeric controller task

TURBO PASCAL: procedure startcnct(TaskNr:integer; softint:integer);

C: void startcnct(int TaskNr, int softint);

DESCRIPTION: This command can be used to start a previously loaded SAP program. The CNC task selected in *TaskNr* (values 0..3) processes the SAP program right from its beginning. The PCAP command *txbf()* can be used for loading.

NOTE: A currently running SAP program will be stopped automatically before this command is executed.
PCAP command *txbf()*

4.4.73 stepcnct, step numeric controller task

TURBO PASCAL: procedure stepcnct(TaskNr:integer; softint:integer);

C: void stepcnct(int TaskNr, int softint);

DESCRIPTION: This command is used for executing an SAP program line by line.

NOTE: The PCAP command *stepcnct()* has not yet been implemented at present!

4.4.74 stopcnct, stop numeric controller task

TURBO PASCAL: procedure stopcnct(TaskNr:integer; softint:integer);

C: void stopcnct(int TaskNr, int softint);

DESCRIPTION: This command causes the SAP program currently being run to stop in the CNC task selected with *TaskNr* (values 0..3), and de-activates this CNC task. The SAP program can be continued with the SAP command *CONTCNCT()* or the PCAP command *contcnct()*.

NOTE: Any EVENT handlers enabled in the SAP program will no longer be processed after the *stopcnct()* command has been executed. Before this command is executed, the drive should be put into a safe operating state.

4.4.75 txbf, transmit binary file

TURBO PASCAL: function txbf(var filename:string; softint:integer):integer;

C: int txbf(char far *filename, int softint);

DESCRIPTION: This function is used to transfer the file specified in the string or character parameter to the PA8000 . Note, however, that only two special file types are permitted. Firstly, the *system.dat* system file (or files with a compatible structure) and secondly the autocode files (CNC files) with the file extension name ".CNC" generated from the IDE or using the *ncc.exe* command line compiler.

Transferring the *system.dat* system file has the following results:

All axis channels will be initialized with the axis-specific system data. The filter coefficients of the PIDF filter will be recomputed, as with the PCAP command *uf()*. These system data can also be edited in the TOOLSET program *mcf.exe*. Any system variables previously altered (e.g. axis-specific velocities, accelerations, etc.) are overwritten again by this command.

Important! Transferring CNC files has the following result: the current program main memory of a CNC task is overwritten with the contents of the specified autocode file. This is why the task concerned is automatically halted before the load operation. The CNC file also contains the information on which task it has to be loaded into (Task 0..3). After the CNC file has been successfully transferred, it can be started with the PCAP command *startcnct()* or the PCAP command *STARTCNCT()*.

NOTE: Normally, the *system.dat* system file need be loaded only once per system start. Please see the particulars given for the PCAP command *mcuinit()* in this context. If you want, you can specify drive and path names in the *filename* parameter.

RETURN VALUE: The function can return the following values:

Return value	Error description
0	No error
1	Invalid function code
2	File not found
3	Path not found
4	Too many files opened
5	Access refused
6	Invalid file handle
12	Invalid access
20	File too large for CNC task main memory
21	Invalid file type! (Not an SAP file nor a system file)

4.4.76 uf, update filter

TURBO PASCAL:	procedure uf(var tsrp:TSRP; softint:integer);
C:	void uf(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv n = 0 .. number of axes present-1
DESCRIPTION:	You can use this command to set the PA8000 PIDF filter for specific axes. Before the command is executed, you must make sure that <u>all</u> the structure components listed above have been initialized. This command can be executed at any time, even during profile generation. This characteristic enables the system to be matched to different load conditions in real time.
NOTE:	You will find more details on the PIDF filter in Chapter 2.1.2, [OM / chapter 4.1.1] and [CM / chapter 5.2] PCAP command <i>rdf()</i>

4.4.77 utrovr, update trajectory override

TURBO PASCAL:	procedure utrovr(var as:AS; softint:integer);
C:	void utrovr(struct AS far *as, int softint);
DESCRIPTION:	The velocity override currently set for all axis channels selected in AS is taken into account.
NOTE:	You will find further information under the PCAP command <i>wrtrovr()</i> .

4.4.78 wrbcnct, write common buffer CNC-Task

TURBO PASCAL:	function wrbcnct(var cbcnct:CBNCT; softint:integer):integer;
C:	int wrbcnct(struct CBNCT far *cbcnct, int softint);
DESCRIPTION:	Each CNC task has a local memory area (referred to as the "Common Buffer"), which can be read and written both by the CNC task concerned and by a PCAP program. This function can be used to write the complete CNC-task-specific buffer (or only a part of it). The <i>cbcnct</i> function parameter is used to select the CNC task buffer, the number of bytes to be written, and the start address of the block which is to be transferred to the PA8000.
RETURN VALUE:	The <i>wrbcnct()</i> function has the following bit-coded return value: Bit 0: 1 if task number invalid Bit 1: 1 if maximum permitted buffer size exceeded This means that the function in normal circumstances returns the value 0.
NOTE:	The CNC-task-specific buffer size is <u>1000</u> bytes. The record structure for CBNCT is shown in Chapter 4.3.2.8. PCAP command <i>rdcbcnct()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>

4.4.79 wrcd, write common double

TURBO PASCAL: procedure wrcd(ndx: integer; var cdbuf:CDBUF; softint:integer);

C: void wrcd(int ndx, struct CDBUF far *cdbuf, int softint);

DESCRIPTION: This function can be used for write access operations to the common variables, which are predefined variables of the CNC task. The variables concerned are the *rw_SymPas* system variables CD0 .. CD99. The first parameter here specifies the number *ndx* of the double variable to be written. The value range of *ndx* here is 0 to 99. The second parameter is a pointer to the CDBUF structure with 100 double variables. Before the command is executed, the variable to be written must be initialized with the appropriate value you want.

NOTE: The content of all common variables remains stored in memory even after a system reset operation, which is executed by the *rs()* command, for example. If you do not want this, you should set the variables concerned to the values you want when you start the program.

4.4.80 wrci, write common integer

TURBO PASCAL: procedure wrci(ndx: integer; var cibuf:CIBUF; softint:integer);

C: void wrci(int ndx, struct CIBUF far *cibuf, int softint);

DESCRIPTION: This command is identical to the PCAP command *wrcd()*, except that here the variables concerned are the *rw_SymPas* system variables CI0 .. CI99 of the LONGINT type.

NOTE: PCAP command *wrcd()*

4.4.81 wrdigo, write digital outputs

TURBO PASCAL: procedure wrdigo(var tsrp:TSRP; softint:integer);

C: void wrdigo(struct Tsrp far *tsrp, int softint);

TSR COMPONENTS: Tsrp[n].digo

DESCRIPTION: This register can be used to set the digital outputs of the PA8000

Attention! The digital outputs of the PA8000 are not grouped axis-specifically. If you want to set an output, you do this by setting the bit concerned. The bit-coded structure of the *digo* status word can be found in the table below:

Table 16: Bit-coded structure of the status word *digo*

Bit No.	Function	Connector X22 / PIN
0	Output 1	26
1	Output 2	27
2	Output 3	28
3	Output 4	29
4	Output 5	30
5	Output 6	31
6	Output 7	32
7	Output 8	33
8..31	Not assigned	--

4.4.82 wrdigob, write digital output bit

TURBO PASCAL: procedure wrdigob(an:integer; bitnr:integer; value: boolean; softint:integer);

C: wrdigob(int an, int bitnr, int value, int softint);

DESCRIPTION: This function can be used to set or reset one PA8000 digital output. The axis number must be specified in the *an* parameter (0, 1, ... *REALAXIS*). The output is reset with the value 0 or FALSE.

NOTE: PCAP command *wrdigo()*

Table 17: Assignment of *bitnr* to the PA8000 digital outputs involved

'bitnr'	Function	Connector X22 / PIN
1	Output 1	26
2	Output 2	27
3	Output 3	28
4	Output 4	29
5	Output 5	30
6	Output 6	31
7	Output 7	32
8	Output 8	33
9..32	Not assigned	--

4.4.83 wrdp, write desired position

TURBO PASCAL: procedure wrdp(var tsrp:TSRP; softint:integer);

C: void wrdp(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].dp

DESCRIPTION: You can use this command to write the axis-specific setpoint position (*dp*). This command is never normally needed, and should be used only in quite exceptional cases, like testing or commissioning jobs. Alteration of the setpoint position is operative only in the position control operating mode. If there are significant differences between this setpoint position (*dp*) and the current position (*rp*), you must anticipate that the motor will be corrected to this position at maximum system acceleration.

NOTE: Writing the setpoint position (*dp*) during execution of motion commands may lead to uncontrolled process behaviour, and should therefore be avoided.
PCAP command *rd dp()*

4.4.84 wrgf, write gear factor

TURBO PASCAL: procedure wrgf(var tsrp:TSRP; softint:integer);

C: void wrgf(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].gf

DESCRIPTION: You can use this command for resetting the axis-specific gear factor in the appropriate unit. This is necessary, for example, with indexing mechanisms or runtime-entailed alterations to system variables, like workpiece or tool dimensions or other correction factors.

NOTE: Remember that (particularly if there are large alterations in the gear factor) the current axis-specific acceleration and velocity parameters have to be matched to this new factor, since this is utilized for converting these system parameters. The value currently set for *gf* can be read with the PCAP command *rd gf()*.

4.4.85 wrhac, write home acceleration

TURBO PASCAL: procedure wrhac(var tsrp:TSRP; softint:integer);

C: void wrhac(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].hac

DESCRIPTION: You use this command to set the axis-specific maximum acceleration *hac* for all reference travel commands (*home* commands). If this command is not executed, the system will work with the system parameter specified in the TOOLSET program *mcfg.exe*. The system parameter can be overwritten any time you want.

NOTE: The value currently set for *hac* can be read with the PCAP command *rdhac()*.

4.4.86 wrhvl, write home velocity

TURBO PASCAL: procedure wrhvl(var tsrp:TSRP; softint:integer);

C: void wrhvl(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].hvl

DESCRIPTION: You use this command to set the axis-specific maximum velocity with the aid of the *hvl* variable for all reference travel commands (*home* commands). If this command is not executed, the system will work with the system parameter specified in the TOOLSET program *mcfg.exe*. The system parameter can be overwritten any time you want.

NOTE: The value currently set for *hac* can be read with the PCAP command *rdhvl()*.

4.4.87 wripw, write in position window

TURBO PASCAL: procedure wripw(var tsrp:TSRP; softint:integer);

C: void wripw(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].ipw

DESCRIPTION: This command can be used to alter (during the run time) the In-Position Window {ipw} specified using the TSW program *mcfg.exe*. The window is re-specified to the value set in *ipw*. The value is stated in the axis-specific position unit.

NOTE: The "In-Position-Window" is monitored only when a value greater than 0.0 has been specified.
[OM / chapter 4.4.2.12]
PCAP command *rdipw()*

4.4.88 wrjac, write jog acceleration

TURBO PASCAL:	procedure wrjac(var tsrp:TSRP; softint:integer);
C:	void wrjac(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].jac
DESCRIPTION:	This command is identical to the PCAP command <i>wrhac()</i> , except that here the maximum system acceleration is specified for all <i>jog</i> commands using the <i>jac</i> variable.
NOTE:	The value currently set for <i>jac</i> can be read with the PCAP command <i>rdjac()</i> .

4.4.89 wrjovr, write jog override

TURBO PASCAL:	procedure wrjovr(var trsp:TSRP; softint:integer);
C:	void wrjovr(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].jovr
DESCRIPTION:	This command sets the axis-specific velocity correction value. This correction value is taken into account in all <i>jog</i> commands. The <i>jovr</i> parameter must have a value greater than 0.0. All values smaller than 1.0 will result in a reduction in axis velocity. If <i>value</i> has a value greater than 1.0, this will be manifested in an increased velocity.
NOTE:	Remember that the specified correction value acts equally on the current axis acceleration. If the correction value is increased or reduced too rapidly, this may be manifested in an acceleration jump (jerk) of the axis. The correction factor should therefore be incremented or decremented in linear mode over time-delay loops, until the final value you want has been reached. For execution of the PCAP commands <i>ra()</i> , <i>rs()</i> or SAP commands <i>RA()</i> , <i>RS</i> , the override factor is initialized to the default value of 1.0. PCAP command <i>rdjovr()</i>

4.4.90 wrjtvI, write jog target velocity

TURBO PASCAL:	procedure wrjtvI(var tsrp:TSRP; softint:integer);
C:	void wrjtvI(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].jtvI
DESCRIPTION:	This command is used to set the axis-specific target velocity (<i>jog</i>) with the aid of the <i>jtvI</i> variable for the <i>jog</i> commands <i>ja()</i> and <i>jr()</i> . If this command is not executed, the system will work with the system parameter specified in the TOOLSET program <i>mcfg.exe</i> . The system parameter can be overwritten any time you want.
NOTE:	The value currently set for <i>jtvI</i> can be read with the PCAP command <i>rdjtvI()</i> .

4.4.91 wrjvl, write jog velocity

TURBO PASCAL: procedure wrjvl(var tsrp:TSRP; softint:integer);

C: void wrjvl(struct Tsrp far *tsrp, int softint);

TSRP COMPONENTS: Tsrp[n].jvl

DESCRIPTION: This command is identical to the PCAP command *wrhvl()*, except that here the maximum traversing velocity is specified using the *jvl* variable for all *jog* commands.

NOTE: The value currently set for *jvl* can be read with the PCAP command *rdjvl()*.

4.4.92 wrledgn, write led green

TURBO PASCAL: procedure wrledgn(value:integer; softint:integer);

C: void wrledgn(int value, int softint);

DESCRIPTION: This command can be used to switch the green LED D4 (PA8000 masterboard) on and off. It is switched on with the value 1, and switched off with the value 0.

NOTE: This command is primarily used as a testing and diagnostic tool.

4.4.93 wrledrd, write led red

TURBO PASCAL: procedure wrledrd(value:integer; softint:integer);

C: void wrledrd(int value, int softint);

DESCRIPTION: as for PCAP command *wrledgn()*, but for the red LED D2

4.4.94 wrledyl, write led yellow

TURBO PASCAL: procedure wrledyl(value:integer; softint:integer);

C: void wrledyl(int value, int softint);

DESCRIPTION: as for PCAP command *wrledgn()*, but for the yellow LED D3

4.4.95 wrlp, write latched position

TURBO PASCAL: procedure wrlp(var tsrp:TSRP; softint:integer);

C: void wrlp(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].lp

DESCRIPTION: This command is used to set the axis-specific latch position to the value set in *lp*. The value is specified in the axis-specific position unit.

NOTE: PCAP-Befehl *rdlp()*

4.4.96 wrlpndx, write latched position index

TURBO PASCAL: procedure wrlpndx(var tsrp:TSRP; softint:integer);

C: void wrlpndx(struct TSRP far *tsrp, int softint);

TSRP COMPONENTS: TSRP[n].lp

DESCRIPTION: This command is used to set the axis-specific latch position of the zero track (index) to the value set in *lp*. The value is specified in the axis-specific position unit.

NOTE: PCAP command *rdlpndx()*

4.4.97 wrmcp, write motor command port

TURBO PASCAL: procedure wrmcp(var tsrp:TSRP; softint:integer);

C: void wrmcp(struct Tsrp far *tsrp, int softint);

TSRP COMPONENTS: Tsrp[n].mcp

DESCRIPTION: This command is used to write the Motor-Command-Port to the value set in the *mcp* field. You will find this particularly helpful during commissioning work, if you want to check the drive system's setpoint value channel, for example. In idle mode (no position control), the motor axis can be moved with this command in uncontrolled form. This means, for example, that you can check the drive's sense of rotation, or check the pulse acquisition feature and limit switches for correct functioning, and so on, before commissioning work is continued in the position control mode.

In the case of servo axes, *mcp* can be set to a value between -32767 and +32767. This value range corresponds to the analog output voltage range of -10 V to +10 V. It may be necessary to allow for a planned inversion of the analog output signal.

In the case of stepping motor axes, *mcp* can be used to specify a time-delay, with the aid of which a stepping signal for stepping motor power output stages is generated. The frequency of this stepping signal can be computed as follows:

$$f_{Pulse} = CLOCK / 2 / (mcp + 1)$$

Example: with *mcp* = 100 and *CLOCK* = 25mhz

$$f_{Pulse} = 12376 [Hz]$$

The value range of *mcp* lies between -1048574 and +1048574. The sign selects the desired traversing direction, and influences the axis-specific directional signal. For the stepping signal f_{Pulse} , only the absolute value of *mcp* is determinant. Remember that the value 0 in *mcp* causes a stepping signal of 0 Hz, i.e. the motor halts.

NOTE: If the axis system is in position control, this command will be effective at most for the duration of a scan interval, since the Motor-Command-Ports are set to new values after the PIDF filter has been processed.
PCAP command *rdmcp()*

4.4.98 wrmpe, write maximum position error

TURBO PASCAL:	procedure wrmpe(var tsrp:TSRP; softint:integer);
C:	void wrmpe(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].mpe
DESCRIPTION:	This command can be used to alter, during the run time, the position error limit {mpe} specified with the aid of the TSW program <i>mcfg.exe</i> . The axis-specific maximum permitted position error is reset to the value set in <i>mpe</i> . The value is specified in the axis-specific position unit.
NOTE:	Position error monitoring is performed only if a value greater than 0.0 has been specified and the control loop is closed. [OM / chapter 4.4.2.10] PCAP command <i>rdmpe()</i>

4.4.99 wrrp, write real position

TURBO PASCAL:	procedure wrrp(var tsrp:TSRP; softint:integer);
C:	void wrrp(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].rp
DESCRIPTION:	This command sets the axis-specific current position register to the value set in <i>rp</i> , and is operative only in open-loop mode (no position control). The value is specified in the axis-specific position unit.
NOTE:	This command will cause the machine zero to be shifted automatically!

4.4.100 wrsdec, write stop deceleration

TURBO PASCAL:	procedure wrsdec(var tsrp:TSRP; softint:integer);
C:	void wrsdec(struct TSRP far *tsrp, int softint);
TSRP COMPONENTS:	TSRP[n].sdec
DESCRIPTION:	This command is used to set the axis-specific stop deceleration <i>sdec</i> for the following: the PCAP command <i>js()</i> [chapter 4.4.14], SAP command <i>JS()</i> [chapter 6.6.25], the software end positions planned with SMD [OM / chapter 4.4.2.11] and the digital inputs planned with LSL_SMD or LSR_SMD projektierten Digital-Eingänge [OM / chapter 4.4.3.1]. If <i>wrsdec()</i> is not executed, the system will work with the system parameter specified in the TOOLSET programm <i>mcfg.exe</i> . The system parameter can be overwritten any time you want.
NOTE:	The value currently set for <i>sdec</i> kann can be read with the PCAP command <i>rdsdec()</i> . [OM / chapter 4.4.2.11] and [OM / chapter 4.4.3.1]

4.4.101 wrsll, write software limit left

TURBO PASCAL: procedure wrsll(var tsrp:TSRP; softint:integer);

C: void wrsll(struct Tsrp far *tsrp, int softint);

TSRP COMPONENTS: Tsrp[n].sll

DESCRIPTION: This command can be used to alter, during the run time, the axis-specific left software limit position {sll} defined with the aid of the TSW program *mcf.exe*. The left software limit is reset to the value set in *sll*. The value is specified in the axis-specific position unit.

NOTE: The software limit set is taken into account only if the home position of the axis channel involved has already been defined or is set after execution of this command.
[OM / chapter 4.4.2.11], PCAP commands *rdsl()*, *shp()*, SAP command *SHP()*

4.4.102 wrslr, write software limit right

TURBO PASCAL: procedure wrslr(var tsrp:TSRP; softint:integer);

C: void wrslr(struct Tsrp far *tsrp, int softint);

TSRP COMPONENTS: Tsrp[n].slr

DESCRIPTION: This command is identical to the PCAP command *wrsll()*, but the right software limit is redefined with the value set in the *slr* parameter.

4.4.103 wrtrovr, write trajectory override

TURBO PASCAL: procedure wrtrovr(var value:double; softint:integer);

C: void wrtrovr(double *value, int softint);

DESCRIPTION: This command sets the trajectory velocity correction value for all interpolation commands (*move* commands). The *value* parameter must have a value greater than 0.0. All values smaller than 1.0 result in a reduction in the trajectory velocity. If *value* has a value greater than 1.0, this will be manifested in an increase in trajectory velocity.
The correction value specified in *value* is placed in intermediate storage on the PA8000 in a system variable, and does not become operative until after execution of the PCAP command *utrovr()*, or the SAP command *UTROVR()*. The axis channels selected there will be decelerated or accelerated even during trajectory travel, depending on the *value* correction factor.

NOTE: Remember that the specified correction value acts equally on the current trajectory acceleration. If the correction value is increased or decreased too quickly, this may be manifested in an abrupt acceleration (jerk) of the axes. The correction factor should therefore be incremented or decremented over time-delay loops in linear mode until the final value you want has been reached. When the PCAP command *rs()* or the SAP command *RS* is executed, the override factor is initialized to the default value of 1.0.
PCAP commands *wrtrovr()*, *wrjovr()*, *rdtrovr()* and *rdjovr()*

4.5 Accessing the PA8000 over the I/O address area

The TOOLSET software includes library functions for the *Turbo C* and *Microsoft C* programming languages, enabling you to access the PA8000 directly. In this case, you can create user programs which can do without the TSR driver *mcutsr.exe*. Direct access is necessary, for example, if you want to use an operating system other than DOS, such as Unix, RMX, QNX, LINUX or others. These program files may also be helpful when using WINDOWS or WINDOWS-NT.

Note: The TOOLSET utility programs like *mcfg.exe* require the *mcutsr.exe* device driver.

4.5.1 Function libraries for PA8000 I/O programming

In the C:\SRVR directory on the PA8000 TSW floppy, you will find the function libraries *mcusrvr.c* and *tpulink.c*, plus the header files *mcusrvr.h* and *tpulink.h*.

The functions and definitions of the *mcusrvr.c* and *mcusrvr.h* files are absolutely identical to the *mcutsr.c* and *mcutsr.h* files described in chapter 4.1, except that here the *softint* parameter is dispensed with for all functions.

The *tpulink.c* and *tpulink.h* files contain library functions with which various data types and data blocks can be transferred from and to the PA8000. The source text is properly documented, so that you should experience no difficulty in customizing it for the hardware and software environment required.

Important: Read the information provided for the system constants *TPUBASEADDRESS* (*tpulink.h*) *DOSCALLALLOWED* (*tpulink.h*) and *REALAXIS* (*mcusrvr.h*)!

The *move.c* program file is the same example program as described in chapter 4.2.

4.5.2 DLL library for the PA8000 I/O programming function

The DLL library *MCUDLL.DLL* can be found in the DLL directory of the PA8000's TSW floppy disk. PA8000 programming can be performed with the aid of this additional library in the commonly used high-level languages for WINDOWS programming, such as Microsoft Visual C++, Microsoft Visual Basic, Borland C++ or Borland Delphi. The DLL file runs under the WINDOWS operating systems 3.x and under WINDOWS 95.

4.5.3 Interface library for the Borland DELPHI programming language

The *mcusrvr.pas* unit file can be found in the DELPHI directory. This function reference library can be used, together with the *mcudll.dll* DLL file described above, to write programs for the WINDOWS platforms mentioned above.

5 The *rw_SymPas* programming language for stand-alone application programming

5.1 Introduction

rw_SymPas is a programming language for creating autonomously running CNC programs (stand-alone application programs) for the PA8000 positioning control system. The lexical and semantic grammar of *rw_SymPas* is very similar to that of the *Pascal* programming language.

5.2 Lexical grammar

This chapter contains a formal definition of the lexical grammar used in *rw_SymPas*. This deals with the word-like units of a language, referred to as »symbols« or »tokens«. The semantic grammar determines the rules by which symbols can be combined to form expressions, statements or other units.

In *rw_SymPas*, the symbols are obtained as a result of the operations performed by the NCC compiler with the user program. An *rw_SymPas* program is a sequence of ASCII characters representing the source code, and written with a text editor (e.g. CNC-Edit). The basic program unit in *rw_SymPas* is the file, which corresponds to a named DOS file in the memory or on the disk, and has the extension ".SRC".

5.2.1 Whitespace

In the lexical analytical phase of compiling, the source code file is parsed (broken down) into symbols and »white space«. White space is the collective term for characters categorized as separators: blanks, tabs, line breaks and comments. White space is used for marking the beginning and end of a symbol, but apart from this white space is ignored.

5.2.2 Comments

Comments are text lines containing explanations on the program. They are removed from the source text prior to parsing.

An *rw_SymPas* comment is a character string located after the character "{". The comment ends at the first occurrence of the "}" character following the start symbol "{". Comments cannot be nested.

There is also an option for creating a one-line comment with two slashes "//". The comment can begin at any point, and extends up to the next line.

5.2.3 Symbole

rw_SymPas recognizes the following kinds of symbol

Symbol:

Keyword
Designator
Qualified designator
Labels
Constant
Operator
Punctuation character (including separators)

5.2.3.1 Keywords

Keywords are words reserved for special purposes, which may not be used as normal designator names. The table below lists all the *rw_SymPas* keywords.

Table 18: All *rw_SymPas* keywords

and	begin	boolean	const
do	double	downto	else
end	for	goto	if
integer	label	mod	module
not	or	procedure	repeat
shl	shr	single	then
timer	to	until	var
while	xor		

5.2.3.2 Designators

Designators can consist of the following elements:

Designator

Non-figure
Designator non-figure
Designator figure

Non-figure: one of the following characters

a b c d e f g h i j k l m n o p q r s t u v w x y z _
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Figure: one of the following characters

0 1 2 3 4 5 6 7 8 9

Examples:

A, AA, AB, A1, A2, _A	// valid
1A, ?B	// invalid

5.2.3.2.1 Name and length restrictions

Designators can be any names of any length for variables, procedures, label names, etc. Designators may contain the letters A to Z, a to z, the underscore, and the figures 0 to 9. However, the following restrictions apply:

- The first character must be a letter or an underscore.
- Only the first 32 characters are significant. If the designator contains more than 32 characters, the remaining characters are ignored. In the case of large `rw_SymPas` programs, you should keep to short names, so as not to overload the PC's main memory.

5.2.3.2.2 Designator upper and lower case

`rw_SymPas` distinguishes between upper and lower-case letters, so that *Position*, *position* and *positioN* are different designators.

5.2.3.2.3 Unambiguity and validity of designators

Designators can be any names which conform to the applicable rules. Errors may, however, occur if the same name is used inside the same range of application for several different designators having the same name range. Identical names are permissible for different name ranges, irrespective of the range of application involved. The definition of a designator range of application is explained in chapter 5.3.2.2.

5.2.3.3 Standard designators

`rw_SymPas` already has a series of predefined designators, which are accordingly referred to as "standard designators". All `rw_SymPas` standard designators are listed in the table below.

Table 19: All standard designators predefined `rw_SymPas`

abort	cl	contcnct	disev
enev	ja	jaw	jhi
jhiw	jhl	jhlw	jhr
jhrw	jr	jrw	js
jsw	mca	mcaw	mcr
mcrw	mha	mhaw	mhr
mhrw	mla	mlaw	mlr
mlrw	ms	msw	ol
ra	rs	shp	smca
smcr	smha	smhr	smla
smlr	ssms	ssmsw	startcnct
stop	stopcnct	uf	wt
utrovr			

5.2.3.4 Axis designators

Each axis channel is referenced using a symbolic name. This name can be freely chosen by the user, with up to 8 characters. These axis designators are likewise incorporated in the standard designator list by `rw_SymPas`.

Note: Automatic declaration of the axis designators deviates from Standard Pascal.

5.2.3.5 Qualified designators

Referencing to designators of the same name which have been declared for different axis systems (by *rw_SymPas*) is handled in a qualifying routine by prefixing the axis designator.

Examples:

```
A1.digo := 0;           // Reset all outputs of the PA8000
A2.digo := $FFFFFFFF;   // Set all outputs of the PA8000
```

Note: Variable referencing to qualified designators deviates from Standard Pascal.

5.2.3.6 Labels

The same rules apply for the structure of a label as for the designators. Labels are used solely in connection with the *goto* statement.

5.2.3.7 Constants

Constants are symbols which stand for fixed numerical values. *rw_SyMPas* knows two classes of constants: floating-point and integer. A constant's data type is derived by the *NCC* compiler on the basis of its numerical value and its format in the source text. Table 20 shows the formal definition of a constant

Table 20: Formal definition of a constant.

Constant:
Floating-point constant
Integer constant
Floating-point constant:
Fractional constant<exponent>
Digit string exponent
Fractional constant:
<Digit string>.Digit string
Digit string.
Exponent:
e<sign>Digit string
E<sign>Digit string
Sign: one of the following characters
+ -
Digit string:
Digit
Digit string digit
Integer constant:
<sign>Decimal constant
Hexadecimal constant
Decimal constant:
Digit
Decimal constant digit
Hexadecimal constant:
\$ Hex digit
Hexadecimal constant hex digit
Digit:
0 1 2 3 4 5 6 7 8 9
Hex digit:
0 1 2 3 4 5 6 7 8 9
a b c d e f
A B C D E F

5.2.3.7.1 Integer constants

Integer constants can be decimal (base 10) or hexadecimal (base 16) numbers. Remember that different rules apply for decimal and non-decimal constants.

5.2.3.7.1.1 Decimal constants

Decimal constants of -2147483648 to 2147483647 are permitted. Constants outside this range will automatically be limited to the appropriate minimum or maximum value.

5.2.3.7.1.2 Hexadecimal constants

All constants which begin with the dollar sign (\$) are interpreted as hexadecimal constants. Hexadecimal constants of \$80000000 to \$7FFFFFFF are permitted. Constants outside this range will be limited to the appropriate minimum or maximum value.

5.2.3.7.2 Floating-point constants

A floating-point constant is made up of 4 constituents:

- Places before the decimal point
- Decimal point
- Decimal places
- e or E and a signed integer exponent (optional)

You can omit either the places before the point or after it (but not both). The decimal point or the letter e (E) can be omitted (but not both). These rules enable you to use both the conventional and the scientific notation (with exponents).

5.2.3.7.2.1 The type of floating-point constants

Floating-point constants are always handled as double values. They are filed in a double word (8 bytes) in accordance with IEEE. The range is 1.7×10^{-308} to 1.7×10^{308} .

5.2.3.7.2.2 Declaration of constants

A constant declaration agrees a designator, which inside the block concerned stands for a constant value. An example of a constant declaration is:

```
Const           one = 1;
```

Signed constants stand for an integer or floating-point value. Computation of constants is not possible.

5.2.3.7.3 Punctuation characters

Punctuation characters (also referred to as separators) are defined in *rw_SymPas* as follows:

Punctuation characters: one of the following symbols
 () , ; : =

5.2.3.7.3.1 Parentheses

Parentheses `()` group expressions together, isolate conditional expressions, and represent procedure calls and procedure parameters:

```
d := c * (a + b);           // Alter the normal sequence
if (d = z) then ...        // Required with a conditional
                               // statement
proc()                     // Procedure call without arguments
```

5.2.3.7.3.2 Comma

The comma `(,)` separates the elements in a procedure argument list:

```
mlr (A1, A2);
```

5.2.3.7.3.3 Semi-colon

The semi-colon `(;)` is used as the end criterion for a statement. Every valid `rw_SymPas` expression (including an empty expression) with a semi-colon at its end will be interpreted as a statement (expression statement).

5.2.3.7.3.4 Equals sign

The equals sign `(=)` separates constant declarations from the initialization values:

```
Const one = 1.0;
```

5.3 Semantic grammar

This chapter will explain the formal definition of the *rw_SymPas* language structure. This semantic grammar determines the rules by which symbols can be combined to form expressions, statements or other meaningful units.

5.3.1 Declarations

The following section provides a brief summary of subjects involving declarations: objects, types, blocks, locality and range of application. Locality and range of application define those parts of the program from which the object linked to the designator can permissibly be accessed.

5.3.1.1 Objects

An object is an identifiable memory area in which a fixed or variable value (or a quantity of values) is located. Each object has a name and a type (referred to as the "data type"). An object is accessed over its name. This name can be a simple designator or a complex expression which unambiguously indicates an object. The type is used in order to:

- specify the correct memory reservation required at the beginning
- check the types so as to ensure that correct assignments are made

The predefined types of *rw_SymPas* include the Boolean data type, integer numbers with sign, and floating-point numbers with differing accuracy.

Declarations establish the link between designators and objects. Each declaration links a designator to a data type. In addition, most declarations (referred to as the "definition declarations") also determine the generation of the object (where and when), and handle assignment of the memory location.

5.3.1.2 Typens

Every declaration of a variable has to specify the type of this variable. The type specifies the value range of the variable concerned, and determines the operations which can be performed with it. Thus a type definition agrees a designator, which in turn stands for a particular type.

Type declaration:

Designator = Typ;

Type:

Boolean type
Integer type
Floating-point type

5.3.1.2.1 Boolean type

The *Boolean* data type can assume only one of the predefined values *FALSE* or *TRUE*. Note that the following relations apply:

- *FALSE* < *TRUE*
- Ordinal number of *FALSE* = 0
- Ordinal number of *TRUE* = 1

5.3.1.2.2 Integer type

`rw_SymPas` provides the integer types *Integer* and *Timer*.

Table 21: The integer type and its value range

Type	Range	Format
Integer	-2147483648 .. 2147483647	32 bits with sign
Timer	0 .. 4294967295	32 bits without sign

5.3.1.2.3 Floating-point types (real types)

`rw_SymPas` knows two different kinds of floating-point types: *Single* and *Double*. These types differ from each other both in their value ranges and in the accuracy of operations performed with them.

Note: Occasionally the term "real type" is also used for "floating-point type".

Table 22: The floating-point types and their accuracy

Type	Range	Format
Single	$-1.2e^{-38} .. 3.4e^{38}$	7 to 8 places
Double	$-2.2e^{-308} .. 1.8e^{308}$	15 to 16 places

5.3.1.2.4 Assignment compatibility of types

Assignment compatibility is essential if a value is to be assigned. The value of a type T_2 can be assigned to a value T_1 (i.e. $T_1 := T_2$), if one of the following conditions is satisfied:

- T_1 and T_2 are of the same type.
- T_1 has the type double, T_2 the value integer or single.
- T_1 has the type single, T_2 the value integer.

If none of these conditions is satisfied, but assignment compatibility is required, the *NCC* compiler will report an error.

5.3.1.3 Variables

5.3.1.3.1 Automatic type conversion

rw_SymPas executes an automatic type conversion function if there are different types in one expression. Conversion is performed as follows: integer to single or integer and single to double. Example:

```

...
Var
    i : Integer;
    s : Single;
    d : Double;
...

d := s * i;      // s and i are automatically converted to double

s := i;          // i is automatically converted to single

```

5.3.2 Blocks, locality and range of application

A block consists of declarations and statements arranged at will. Each block is part of a procedure declaration or of a program. All designators and labels in the block's declaration section are restricted in their effect to this block - they are local to this block.

5.3.2.1 Syntax

The syntactic structure of each block can be represented as follows:

Block:

- Declaration section
- Command section

5.3.2.1.1 Declaration section I

Declaration section:

- Label declaration section
- Constant declaration section
- Variable declaration section
- Declaration section label declaration section
- Declaration section constant declaration section
- Declaration section variable declaration section

5.3.2.1.1.1 Label declaration section

In the *Label declaration section*, all labels are agreed which are to represent *goto* jump destinations in the command section of the block involved. Each label may be defined once only inside the command section (i.e. each *goto* must have an unambiguous destination).

Structure of the label declaration section:

label Labels;

Labels:

LabelName

Labels, LabelName

5.3.2.1.1.2 Constant declaration section

The declaration section for constants contains all agreements for constants which are local to the block involved.

Structure of the constant declaration section:

const constant declarations

Constant declarations:

Constant declaration

Constant declarations constant declaration

5.3.2.1.1.3 Variable declaration section

The declaration section for variables contains all variable declarations which are local to the block involved.

Structure of the variable declaration section:

var variable declarations

Variable declarations:

Variable declaration

Variable declarations variable declaration

5.3.2.1.2 Command section

It is in the command section that all operations are defined which are executed at block activation.

Command section:

Compound statement

Permissible compound statements are explained in chapter 5.3.5.5.

The command section of the main program block is structured as follows:

begin

Statement list;

end.

5.3.2.2 Range of application

Each designator and each label of a declaration agrees precisely one object or jump destination. This is why a designator, like a label, must always be in its declaration's range of application when it appears in the program. The range of application for designators and labels lies between the actual declaration as such and the end of the block involved, with all those blocks being included which this block encloses. There are, however, a few exceptions to this, which are explained in the paragraphs below.

5.3.2.2.1 Redeclaration in a subordinate block

With the assumption that a block »outside« encloses a block i.e. is of a higher order, every redeclaration of a designator from »outside« in the block »inside« restricts this designator's range of application to the »inside« block. Or to put it another way: if a variable *x* is declared »outside«, and a variable of the same name is declared »inside«, then statements in the block »inside« cannot access the variable *x* declared »outside«.

5.3.2.2.2 The location of a declaration in a block

Designators and labels must be declared before they can be used in a block. The *NCC* compiler will react to access attempts before such declaration with Error Number 3.

5.3.2.2.3 Redeclarations inside a block

Designators and labels can each be declared only once on the topmost level of a block, unless they are redeclared inside a subordinate block.

5.3.2.2.4 Standard designators

rw_SymPas offers a whole series of predefined constants, types and procedures, which work as if they had been declared inside a block covering the whole program. Consequently their range of application also covers the entire program.

5.3.3 Variables

5.3.3.1 The declaration of variables

The variable declaration contains a list of designators, which in their turn stand for new variables and their types.

Variable declaration:

Designator list: type;

Designator list:

Variable names

Variable names, variable name

Type:

BOOLEAN

INTEGER

SINGLE

TIMER

DOUBLE

Examples of valid variable declarations are:

```
var
    on, off:    BOOLEAN;
    one:        INTEGER;
    dvalue:     DOUBLE;
    ticks:      TIMER;
```

When a designator is located in the designator list of a declaration section, it applies inside the entire block for which it has been declared. Reference can be made to this variable throughout the block, provided the same designator is not being used for a different variable in a subordinate block ("redeclaration"). A redeclared variable uses the name of an already-existing designator, but otherwise represents an autonomous unit. The value of the original variable is not affected by the redeclaration. Variables declared outside procedures are referred to as *global*. Variables declared inside procedures are *local*.

5.3.3.1.1 Timer declaration

An entry with the aid of the predefined system variable *CLOCK* supplies a value of the *timer* type, which represents the time. This value is supplied by the transputer's internal clock, which alters its value at regular intervals. This value continues cyclically, i.e. after the largest positive value the next value supplied is the smallest negative value. The time interval in which this internal clock is incremented is 64 μ s.

CLOCK can be used at any time to assign the counter reading of this clock to an *integer* or *timer* variable. If different times have to be compared with each other, this comparison should be carried out only by means of *timer* variables, since here a timer overflow will automatically be taken into account at the comparison operator *>*. Likewise, addition and subtraction with *timer* variables is performed in modulo technique, i.e. without signs. With *integer* variables, conversely, there may be an overflow or underflow, which in turn will cause the transputer error flag to be set, and in certain situations will trigger an abort of the *rw_TOS* operating system.

A practical timer application might look like this:

```

Const
    s := 15625;                // 15625 ticks = 1s
Var
    t: timer

    t := CLOCK + 5*s;          // Compute time-delay of 5 s from now
    ...
    repeat
        ...
    until CLOCK > t;           // wait until 5 s have passed
    ...

```

In this example, you can see that the addition of *CLOCK* and the time-delay results in an overflow at large values for *CLOCK*. We therefore recommend using the *timer* instead of the *integer* type for declaration of the variable *t*. Another reason is the interrogation of whether the computed time-delay has been reached. In the event of an overflow in computing the time-delay, you see, the content of *t* is smaller than *CLOCK*. This circumstance is likewise handled properly by the declaration as a *timer* variable.

The value range of a timer variable lies between 0 and 4294967295. Time-delays of up to 38h can be implemented.

5.3.3.2 Conversion of variable types

The reference to a variable of a particular type can be converted into a reference to a variable of a different type.

Type conversion:

Type designator (variable reference)

Type designator:

BOOLEAN
INTEGER
SINGLE
DOUBLE

A few examples for the conversion of variable types:

```
var
  B : BOOLEAN;
  I : INTEGER;
  D : DOUBLE;

  B := BOOLEAN ( I );
  B := BOOLEAN ( D );
  D := B;
  I := INTEGER ( D );
  I := B;
```

5.3.4 Expressions

Expressions consist of operators and operands. Most operators of *rw_SymPas* link two operands, and are therefore referred to as binary. The remaining operators work with only one operand, and are therefore referred to as unary. Binary operators utilize the conventional algebraic form like $a+b$. A unary operator is always positioned immediately before its operand, as with $-b$. In the case of extensive expressions, the order of *precedence* shown in Table 23 governs the sequence of computation. Three basic rules apply:

- An operand between two operators of different precedence rankings is always linked to the higher-ranking operator.
- An operand between equal-ranking operators is always linked to the operator located to the left of it.
- Expressions in brackets are regarded as a single operand, and always evaluated first.

Table 23: Operator precedence

Operators	Precedence	Category
-, +, not	1 (highest)	unary
*, /, mod, shl, shr, and	2	multiplying
+, -, or, xor	3	adding
=, <>, <, >, <=, >=	4	relational

Operations of the same precedence ranking are normally performed from left to right.

5.3.4.1 Syntax of expressions

The order of precedence for operators follows the syntax for expressions composed of factors, terms and simple expressions. Factors can be represented by the following syntax:

Factor:

- variable reference
- unsigned constant
- (expression)
- not* factor
- type conversion (values)

unsigned constant:

- unsigned numerical value
- character string
- constant designator

The following particulars represent valid factors:

- Dummy* variable reference
- 15 unsigned constant

5.3.4.2 Operators

We distinguish between four groups of operators: arithmetical, logic, boolean and relational operators.

5.3.4.3 Arithmetical operators

The tables below show the types of operand and result involved in binary and unary arithmetical operations.

Table 24: Binary arithmetical operators

Operator	Operation	Operand type	Result type
+	Addition	Integer, Real	Integer, Real
-	Subtraction	Integer, Real	Integer, Real
*	Multiplication	Integer, Real	Integer, Real
/	Division	Integer, Real	Integer, Real
mod	Modulo	Integer	Integer

Note: If one of the operands is of the *Timer* type, addition and subtraction are performed using the modulo technique. No overflow check is made, since the *Timer* values are cyclical. You will find more details on the *Timer* type in Chapter 5.3.3.1.1.

Table 25: Unary arithmetical operators

Operator	Operation	Operand type	Result type
+	Identity	Integer, Real	Integer, Real
-	Negation	Integer, Real	Integer, Real

If both of an operator's operands +, -, *, /, or *mod* have an integer type, the result will likewise be of the integer type. If one of an operator's operands is +, -, *, or / is of the *Real* type, then the result will likewise be of the *Real* type.

The *mod* operator returns the rest of the division of its operands as follows:

$$i \text{ mod } j = i - (i/j)*j;$$

5.3.4.4 Logic operators

Table 26 shows the types of operand involved, and the results of logic operations.

Table 26: Logic operations

Operator	Operation	Operand type	Result type
not	bitwise negation	Integer	Integer
and	bitwise AND	Integer	Integer
or	bitwise OR	Integer	Integer
xor	bitwise exclusive OR	Integer	Integer
shl	Shift left	Integer	Integer
shr	Shift right	Integer	Integer

Note: *not* is a unary operator.

The *shl j* and *shr j* operations shift the value of *i* by *j* bit positions to the left or the right, and thus correspond to a multiplication or division by 2^j .

5.3.4.5 Boolean operators

Table 27 shows the types of operand involved, and the results of Boolean operations.

Table 27: Boolean operators

Operator	Operation	Operand type	Result type
not	logic negation	Boolean	Boolean
and	logic AND	Boolean	Boolean
or	logic OR	Boolean	Boolean
xor	logic exclusive OR	Boolean	Boolean

Note: the operator **not** is unary here as well.

In the case of operands of the *Boolean* type, normal Boolean logic determines the result of these operations. For example, *a and b* will only give *TRUE* when *a* and *b* are both true

5.3.4.6 Relational operators

Table 28 shows the operand types involved, and the results of relational operations.

Table 28: Relational operators

Operator	Operation	Operand type	Result type
=	equal	Integer, Real	Boolean
<>	unequal	Integer, Real	Boolean
<	smaller than	Integer, Real	Boolean
>	greater than	Integer, Real	Boolean
<=	smaller than/equal	Integer, Real	Boolean
>=	greater than/equal	Integer, Real	Boolean

Note: If one of the operands is of the *Timer* type, the *greater than* (>) operation is performed using the modulo technique. No overflow check is made, since the *Timer* values are cyclical. You will find more details on the *Timer* type in chapter 5.3.3.1.1.

5.3.5 Statements

This term stands for all constructs which agree an action which can be executed by the PA8000. In this manual, the term »statement« is used as a generic term for statements (like *begin*, *end* or *for*) and *commands* (like *goto*, assignments, procedure calls, etc.).

Each statement (i.e. each agreement for an executable action) can be preceded by a label, which in its turn can be referenced with *goto*: a *goto* this label causes a direct jump to this statement and its execution.

Structure of a statement:

Label: statement

Statement:

Assignment

Procedure statement

Goto statement

5.3.5.1 Assignments

Assignments replace the instantaneous value of a variable with a new value, which is specified by mean of an expression.

Structure of an assignment:

Variable reference := expression

5.3.5.2 Procedure calls

A procedure is called by specifying a procedure designator with which the procedure concerned has been declared. Parameter transfer to the procedure is not supported.

5.3.5.3 The goto statement

executes a jump to the label specified: the program is continued at a point immediately following the label concerned. The syntax of *goto* is:

goto Label

When *goto* is used, the following rules must be observed:

- The label to which *goto* is referenced must be located in the same block as the *goto* statement itself. It is not possible to jump back and forth at will between procedures with *goto*.
- Referencing to a structured statement block from a program section outside this block (i.e. a jump to a deeper nesting level) may have unforeseeable consequences. *rw_SymPas* cannot detect errors of this sort.

5.3.5.4 Structured instructions

consist of several interrelated levels, which in their turn contain statements. They are executed either in the order of their appearance (compound statements), conditionally (conditional statements) or repeatedly (repeat statements or loops).

Structured statement:

- block command
- conditional statement
- repeat statement

5.3.5.5 Compound statements

Compound statements specify that the individual components they contain are to be executed in the order in which they appear in the source text concerned. All statements contained in the compound are handled as one single block, and thus satisfy the requirements at points where the syntax of *rw_SyMPas* permits only a single statement. Beginning and end of a compound are indicated by *begin* and *end*, with the individual components separated from each other by semi-colons.

A compound statement can be represented as follows:

begin statement list **end;**

Statement list:

- statement;
- statement list statement;

Example:

```
// ...
var
    i: Integer;
    j: Integer;
    temp: Integer;

// ...
begin
    if (i > 0) then i := 0;
    else begin
        // interchange j and i
        temp := i;
        i := j;
        j := temp;
    end;
end.
```

5.3.5.6 Conditional statements

Conditional statements offer one or more options and select one of their components (or none) for an instruction.

5.3.5.6.1 The if statement

can be represented as follows:

if (conditional expression) **then** w-statement **<else f-statement>**

The brackets around *Conditional expression* are not absolutely necessary. The result of *Conditional expression* must be of the standard *Boolean* type. If *Conditional expression* is TRUE, then w-statement will be executed; otherwise w-statement will be ignored.

If the optional *else f-statement* is present, and *Conditional expression* is true, then *w-statement* will be executed; otherwise *w-statement* will be ignored and *f-statement* executed.

The statements *f-statement* and *w-statement* may themselves be *if-statements*, thus enabling a nested conditional test to be implemented in almost any depth you want. You have to be very cautious in using nested *if..else* constructs - make absolutely sure that the correct statements are chosen. *Else* ambiguities are resolved by assigning an *else* to the last *if-without-else* occurring on the same nesting depth. Compound statements are also permissible for *w-statement* and *f-statement*.

5.3.5.7 Loops

Loops (or repeat statements) specify the repeated execution of defined program sections.

Loop:

while statement
repeat statement
for statement

5.3.5.7.1 The while statement

The format for a **while** statement is:

while (conditional expression) **do** w-statement;

The brackets around *Conditional expression* are not absolutely necessary. The loop statement *w-statement* will be executed as long as the *Conditional expression* gives the value FALSE. The *Conditional expression* is evaluated and tested beforehand. If the value obtained is TRUE, then *w-statement* will be executed. If the program does not encounter any jump statements, causing it to leave the loop, the *Conditional expression* will be evaluated anew. This operation is repeated until *Conditional expression* gives the value FALSE. If there are no jump statements, then *w-statement* must influence the value of *Conditional expression*, or *Conditional expression* itself must alter during evaluation, so as to avoid endless loops. Compound statements are also permissible for *w-statement*.

5.3.5.7.2 The repeat statement

The format for a *repeat* statement reads:

repeat *r-statement* **until** (conditional expression);

The brackets around *Conditional expression* are not absolutely necessary. The *r-statement* is executed as long as *Conditional expression* has the value FALSE. In contrast to the *while* statement, *Conditional expression* is tested not before, but after every execution of the loop statement. *r-statement* will accordingly be executed at least once.

Compound statements are also permissible for *r-statement*.

5.3.5.7.3 The for statement

The format for a *for* statement reads:

for controlled variable := Start value **to/downto** final value **do** *f-statement*;

The controlled variable must be the designator of an integer-type variable, which has been declared either inside the same block locally like the *for* statement, or globally for the entire program. The definition of a loop with *for* includes the specification of a *start* and *final value* as well. Both these values must likewise be of the integer type, which is assignment-compatible to that of the controlled variable.

When the loop is started, the controlled variable is set to the *start value*, and increased or reduced by one each time the loop is run - until the *final value* is reached. In each run, the *f-statement* or compound statement contained in the rump of the loop is executed once. If the final condition of the loop is already given before the first run (i.e. *final value* < *start value* or *final value* > *start value* when *downto* is being used), then the loop and its rump will be skipped completely.

5.3.6 Procedures and functions

In formal terms, procedures and functions represent additional levels inside the main program block, i.e. a nesting feature. A procedure is activated by a procedure call (i.e. specification of a designator), and does not return a direct value. A function is activated during the computation of an expression in which its designator appears, and normally has a result which can for this call be equated with the function designator.

5.3.6.1 Procedure declarations

A declaration initiated with the reserved word *procedure* links a designator and a block of statements for a procedure. Procedures declared in this manner can be activated (i.e. called) by specifying their designator. A procedure declaration has the following formal structure:

Procedure header; procedure block;

The procedure header names the procedure (i.e. assigns a designator to it). Under *Pascal* standard, the declaration of formal parameters would be permitted at this point. This is not supported in *rw_SymPas*. However, data can be exchanged between the main program and a procedure can be performed over global variables. A procedure is activated by specifying its designator: the actions defined in the command section of the procedure declaration involved are executed.

A procedure which contains its own statement as part of its command section is executed recursively, i.e. it calls itself repeatedly. In this context, a suitable criterion must be found for aborting the recursion before the internal CNC task stack overflows.

It is not possible to nest procedures in *rw_SymPas*.

5.3.6.2 Function declarations

Note: The function declaration implemented to *Pascal* standard is not possible in *rw_SymPas*, but there are various predefined system functions.

These functions are activated during the computation of expressions in which their designator appears, and stand there for the value they return. A function designator can be inserted anywhere in an expression in place of an operand, provided the type of the function result concerned is compatible with that of the operand replaced.

Assignments to a function designator are not permitted.

A function is called by specifying its designator, followed by a list of current parameters, which in type and sequence must conform to the formal parameters of the correspondingly predefined function.

5.3.7 The syntax of an *rw_SymPas* program

An *rw_SymPas* program is similar in form to a procedure declaration. The differences are merely in the program descriptor.

rw_SymPas program:

Program descriptor; program block

5.3.7.1 The program descriptor

The program descriptor specifies the name of a program, but has no special significance of its own.

Program descriptor:

program designator

Example:

```
program Test ;
```

5.3.7.2 The program block

Program block:

Implementation section

Procedure command section

Initialization section

Implementation section:

Constant declaration

Variable declaration

Implementation section constant declaration

Implementation section variable declaration

Initialization section:

begin

Command section

end

The initialization section is the final constituent part of an *rw_SymPas* program, and represents the main program. It consists of a block initiated with **begin**, which contains statements and is concluded by a terminating **end**. The entire program block is concluded with the (.) character.

6 Stand-alone application programming

6.1 Introduction

The *rw_SymPas* programming language incorporates a comprehensive set of commands, which you can use for flexible, efficient program creation. The procedure calls are performed in accordance with *Pascal* convention, apart from a few exceptions.

Since the procedure names and also the functioning of the individual procedures are identical for the two programming methods involved - stand-alone application programming [SAP] and PC application programming [PCAP], a detailed description is provided here only for the commands involved in PCAP programming.

The individual commands are listed in alphabetical order.

6.2 *rw_SymPas* example programs

The *rw_SymPas* example programs included in the PA8000 TOOLSET software show how simple it is to use the functions described below. The source texts for the example programs incorporate comments to make them self-explanatory. So there is no need to go into a detailed description of these example programs here. They all have the file extension .SRC and can be found in the SAP subdirectory of the PA8000 TOOLSET software floppy.

6.3 Abbreviations, system parameters, axis specifiers and axis qualifiers

For the SAP function reference list printed below, we will start off by explaining the various abbreviations and types involved, some of which are used as parameters for the different functions in question.

6.3.1 System parameters

The system parameters predefined by the *rw_SymPas* programming language are listed in tabular form, with an explanation of how they function. Remember that the *NCC* compiler distinguishes between upper and lower case for these parameters.

Table 29: *rw_SymPas* predefined system parameters

Name	Type	Abbr. meaning	Function
CI0..CI99	integer	Common Integer 0..99	100 predefined integer variables for data exchange or for synchronization with a PC application program running in parallel. Further information at the PCAP commands <i>rdci()</i> and <i>wrci()</i> .
CD0.. CD99	double	Common Double 0..99	100 predefined double variables. Otherwise as for CI0..CI99. Further information at the PCAP commands <i>rdcd()</i> and <i>wrcd()</i> .
LEDGN	boolean	Led green	Green LED (D4) on PA8000, switched on when TRUE
LEDRD	boolean	Led red	Red LED (D2), otherwise as for LEDGN
LEDYL	boolean	Led yellow	Yellow LED (D3), otherwise as for LEDGN
IRQPC	boolean	Interrupt Request PC	PC interrupt request, active when TRUE
PHI	double		Traverse angle for circular and helical profiles
PU	integer	Position Unit	Index for position unit
TRAC	double	Trajectory Acceleration	Trajectory acceleration for linear, circular, and helical profiles
TROVR	double	Trajectory Override	Trajectory velocity correction value
TRTVL	double	Trajectory Target Velocity	Trajectory target velocity for linear, circular, and helical profiles
TRVL	double	Trajectory Velocity	Trajectory velocity for linear, circular, and helical profiles
TU	integer	Time Unit	Index for time unit

6.3.1.1 PC interrupt generation

You can use the *IRQPC* system parameter to trigger a hardware interrupt on the PC. This option offers an efficient approach for using the two programming methods: PC application and stand-alone application programming. A stand-alone program can be used for largely autonomous process sequence, which needs to interrupt the parallel-running PC program only if necessary, or in the event of an error. The program is then interrupted with the aid of this interrupt generation feature. After the PC program has detected the hardware interrupt, the common variables listed above can be used for exchanging data between the two parallel-running programs.

Note: The hardware configuration for PC interrupt generation is described in the Commissioning Manual. You must also remember that in the PC application program the hardware interrupt selected is enabled in the Interrupt Mask Register (I/O Address 21 hex), and an interrupt handler is set up for the corresponding interrupt code number.

6.3.1.2 System parameters for unit processing

All *move commands* of the *rw_SymPas* programming language require specification of the acceleration (*TRAC*), velocity (*TRTVL*, *TRVL*) and position parameters, each in selected distance and time units. You can use the two system parameters listed below to switch over the path unit (PU) and time unit (TU) parameters any time you want.

Table 30: System parameter PU

Value	Unit	Abbr. meaning
0	mm	Millimeter
1	inch	Inch
2	m	Meter
3	rev	Revolution
4	deg	Degree
5	rad	Radian
6	counts	Counts
7	steps	Steps

Table 31: System-Parameter TU

Value	Unit	Abbr. meaning
0	sec	Seconds
1	min	Minutes
2	tsample	Sampling Time

Note: The default values for TU and PU are specified in the [Setup][Set CNC-specific parameters] menu in the CNC Editor environment.

The units selected are utilized only for interpolation commands (all *move* commands)! If the commands concerned are axis-specific motion ones (all *jog* commands), the axis units specified in *mcfg.exe* are taken into account. There is no option here for switching over during the run time.

6.3.2 Axis specifiers

The various axis channels are referenced with a symbolic name. You can choose these names quite freely in the *mcfg.exe* program. In the *rw_SymPas* programming language, these names are predefined automatically, and serve in the user program as parameters for various commands. Remember that the *NCC* compiler distinguishes between upper and lower case for the axis specifiers.

6.3.3 Axis qualifiers

The system parameters listed below are used as axis qualifiers, and are therefore available for all the axis channels in the system, and thus for all axis specifiers. You can use these parameters to interrogate or set various axis-specific data. Remember that the *NCC* compiler distinguishes between upper and lower case for these parameters. An axis qualifier is referenced by stating an axis specifier, the character "." and the axis qualifier. The example below illustrates this.

```
...  
var      input: integer;  
...  
input := A2.digi;           // Read in digital inputs from  
                           // Axis Channel 2  
...
```

Table 32: Axis qualifiers

Name	Typ	Abbr. meaning	Function
epc	integer	EEPROM programming cycles	Number of programming cycles
digi	integer	digital inputs	Digital inputs of the PA8000 (wordwise) Various flags of this register can be erased by assigning any desired value to this register [chapter 4.4.33].
digo	integer	digital outputs	Digital outputs of the PA8000 (wordwise)
ifs	integer	interface status	Interface status flags of the PA8000 (wordwise) Various flags of this register can be erased by assigning any desired value to this register [chapter 4.4.66].
axst	integer	axis status	Error, status and profil flags (wordwise)
dp	double	desired position	Setpoint position of axis channel
dv	double	desired velocity	Setpoint velocity of axis channel
hac	double	home acceleration	Acceleration for <i>home</i> commands
hvl	double	home velocity	Velocity for <i>home</i> commands
ipw	double	in position window	Position-dependent target window
jac	double	jog acceleration	Acceleration for <i>jog</i> commands
jovr	double	jog override	Velocity factor
jtv	double	jog target velocity	Target velocity for <i>jog</i> commands
jvl	double	jog velocity	Velocity for <i>jog</i> commands
kd	double		PIDF filter coefficient for differentiation
ki	double		PIDF filter coefficient for integration
kp	double		PIDF filter coefficient for amplification
kpl	double		PIDF filter coefficient for add. phase lead
kfca	double		PIDF filter coefficient for forward compensation for acceleration
kfcv	double		PIDF filter coefficient for forward compensation for velocity
lp	double	latched position	latched position value
lpndx	double	latched position index	latched position value with index signal (zero track)
lsm	integer	left spool memory	free spool area [Bytes]
mcp	integer	Motor Command Port	Servo motors: setpoint value for analog port, value range -32767 .. +32767 (-10V .. +10V) Stepping motors: stepping signal for stepping motor power output stages, value range -1048575 .. +1048575
mpe	double	maximum position error	Maximum permitted position error
rp	double	real position	Actual position of the axis channel
sdec	double	stop deceleration	Stop deceleration of the axis channel
sll	double	software limit left	Left software limit
slr	double	software limit right	Right software limit
tp	double	target position	Target position of axis channel

The function of these qualifiers can be found at the relevant *rdxxxx()* and *wrxxxx()* commands in the function reference list for PCAP programming. The significance of the qualifier *digo*, for example, is explained under the *wrdigo()* command.

Exception: The PIDF filter coefficients become operative together with the SAP command *UF()*. These coefficients are read and written on PCAP level using the *rdf()* and *uf()* commands.

6.3.4 Structured axis qualifiers

The system parameters listed below are used as structured axis qualifiers, and are therefore available for all the axis channels in the system, and thus for all axis specifiers. You can use these parameters for bitwise interrogation and setting of various axis-specific data. Remember that the *NCC* compiler distinguishes between upper and lower case for these parameters. Referencing to a structured axis qualifier is illustrated by the example below:

```
...
const
    enable = 1;
var
    input: boolean;
...
input := A2.digib.enable;    // read digital input 1 of axis
                             // channel 2 (I1)
A1.digob.7 := TRUE;         // Set digital output 7 (O7)
...
```

Table 33: Structured axis qualifiers

Name	Type	Abbr. meaning	Function
digib	boolean	digital-input-bit	Digital inputs of the PA8000 (bitwise)
digob	boolean	digital-output-bit	Digital outputs of the PA8000 (bitwise)
ifsb	boolean	interface-status-bit	Status flags of the PA8000 (bitwise)
axstb	boolean	axis status-bit	Error, status and profile flags (bitwise)

The function of these qualifiers can be found at the relevant *rdxxxxb()* and *wrxxxxb()* commands in the function reference list for PCAP programming. The significance of the qualifier *digib*, for example, is explained in the *rd digib()* command.

Note: Bit counting for the structured axis qualifiers begins at 1!

6.3.5 Abbreviations

Some of the abbreviations used in the function reference list will be explained to start with:

Table 34: Abbreviations

A1	Symbolic name for the first axis channel. This name can be freely selected in mcfg.exe. Is mainly used for examples
A2	Symbolic name for the second axis channel. Otherwise as for A1.
Spec	Axis specifier, such as A1 or A2
Qual	Axis qualifier, such as digi, digib, digo, digob, axst etc.
Pos	Position setpoint value (data type: double)
Event	Procedure with function as event handler

6.4 Reserved procedure names with event function

rw_SymPas incorporates a series of predefined procedure names with event function. If there are procedure definitions with these procedure names in the user program, the CNC task can be made by means of an enable command to call these procedures automatically if a procedure-specific event occurs. These procedures are accordingly also referred to as "event handlers".

Note: The events are checked after every execution of an *rw_SymPas* statement.

6.4.1 Event procedure EVEO

The EVEO event procedure is processed automatically after the definition of the procedure EVEO and the release of the corresponding event. The EO (Emergency Out) event occurs when a digital input planned with EO function is activated. If the system includes more than one EO inputs, the *axst* status register can be used to check which EO input is causing the error concerned. A simple example program for implementing an EO-handler is listed below:

```

...
procedure EVEO;          // predefined name for
                          // Timeout EVENT hHandler
begin
    CI0 := 999;           // Common Variable
                          // signals program abort
    abort;                // Abort application program
end;

...
begin
    ...
    CI0 := 0;             // Delete common
                          // Variable
    enev(EVEO);           // Enable timeout handler
    ...
end.

```

6.4.2 Event-Prozedur EVDNR

The EVDNR event procedure also operates like EVEO, except that this procedure is processed automatically when the Drive Not Ready event occurs. The DNR event occurs when a digital input planned with DR function becomes inactive [OM / chapter 4.4.3.1].

6.4.3 Event procedure EVLSH

The EVLSH event procedure also operates like EO, except that this procedure is processed automatically when the Limit Switch Hardware event occurs. The LSH event occurs when a digital input planned with LSL_TOM, LSL_SMA, LSL_SMD, LSR_TOM, LSR_SMA or LSR_SMD function is activated [OM / chapter 4.4.3.1].

6.4.4 Event procedure EVLSS

The EVLSS event procedure also operates like EVEO, except that this procedure is processed automatically when the Limit Switch Software (software limit) event occurs. The LSS event occurs when the current position of an axis system exceeds a limit value specified in the TOOLSET program *mcf.exe* and the limit value concerned has been planned with the TOM, SMA or SMD function [OM / chapter 4.4.3.1].

6.4.5 Event procedure EVMPE

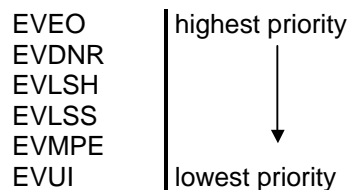
The EVMPE event procedure also operates like EVEO, except that this procedure is processed automatically when the Maximum Position Error event occurs. The MPE event occurs when the control loop is closed and the difference between setpoint and actual positions of an axis system exceeds the limit value specified in the TOOLSET program *mcf.exe* [OM / chapter 4.4.2.10].

6.4.6 Event procedure EVUI

The EVUI event procedure also operates like EVEO, except that this procedure is processed automatically when the User Input event occurs. The UI event occurs when a digital input planned with UI is activated [OM / chapter 4.4.3.1]. You have an option for building up user-specific special functions with UI-planned digital inputs in the SAP program. Alternative cyclical polling can be dispensed with.

6.4.7 Priority and processing sequence for the event procedures

It is possible that different events will occur at the same point in time. In this case, the following priorities apply:



If one event procedure (Event 1) is currently being processed, the occurrence of another event (Event 2) with lower or higher priority will be ignored; this event will not be executed until the current event handler (Event 1) has been processed. But Event 2 must still be active then!

Note: After the *STOP* and *ABORT* SAP commands, and during execution of the *WT()* SAP command, no event handlers will be processed!

6.5 SAP block commands

The command reference list provided below contains a series of commands which can be used to achieve a block-oriented program structure. All these commands have names which end with the character "W". Examples include the SAP commands *MLAW()*, *JAW()* or *SSMSW()*. These commands automatically wait for the profile end of all axes involved, i.e. the next statement will not be processed until the target positions of the selected axes have been reached. For this purpose, the CNC task polls the profile end flags of these axes, and continues the program at the next statement when appropriate. This check routine takes the above-enabled EVENT handlers into account, and processes them automatically when and as required.

Note: Another option for profile end checking is to evaluate the *axst* axis qualifier.

6.6 *rw_SymPas* SAP command reference list

6.6.1 Structure of the reference list

The reference list is structured as follows:

FUNCTION NAME:	This is the name which is used to call the function subsequently described.
ABBR. MEANING:	Here you will find a detailed description of the function name concerned.
FUNCTION PARAMETERS:	If the function demands a parameter transfer, these are listed here.
SYSTEM PARAMETERS:	Various functions are executed by taking various system parameters into account. These are listed here.
SIMULTANEOUS FUNCTION:	With various functions, it is permitted to specify one or more axes for which the function concerned is to be executed.
REFERENCES:	Refers to other functions and chapters.
DECLARATION:	The formal declaration of predefined system functions; user-defined elements are shown in italics.
RESULT TYPE:	The type of the value returned (with system functions only).
DESCRIPTION:	Plaintext description of the command concerned.
NOTE:	Recurrent notes and explanations here indicate the chapters you should consult.
EXAMPLE:	An example of the function involved.

6.6.2 ABORT, abort

DESCRIPTION:	This command causes a running SAP program to be aborted. In contrast to the STOP statement, the program cannot be continued with the PCAP command <i>contcnct()</i> or the SAP command <i>CONTCNCT()</i> . This is possible only with the PCAP command <i>startcnct()</i> or the PCAP command <i>STARTCNCT()</i> .
NOTE:	After the command has been executed, the enabled EVENT handler procedures will no longer be processed.
EXAMPLE:	<i>ABORT;</i>

6.6.3 ABS, absolute function

DECLARATION: `abs(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the absolute value of *value*.

EXAMPLE:

```
...
var
    d1, d2: double;
...
d1 := -5.0;
d2 := ABS(d1);      // d2 := 5.0
```

6.6.4 ACOS, arc cosine function

DECLARATION: `acos(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the arc cosine of *value*. The argument *Value* must lie within the range $[-1..+1]$. The return value has the unit rad, and lies within the limits $[0..pi]$.

6.6.5 ASIN, arc sine function

DECLARATION: `asin(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the arc sine of *value*. The argument *Value* must lie within the range $[-1..+1]$. The return value has the unit rad, and lies within the limits $[-pi/2..+pi/2]$.

6.6.6 ATAN, arc tangent function

DECLARATION: `atan(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the arc tangent of *value*. The return value has the unit rad, and lies within the limits $[-pi/2..+pi/2]$.

6.6.7 AZO, activate zero offsets

FUNCTION PARAMETERS: Integer constant in the value range of 0..4

DESCRIPTION: PCAP command *azo()*

EXAMPLE:

```
const Offsets1 = 1;

azo(Offsets1);      // Activate zero offsets Set 1
```


6.6.8 CL, close loop

FUNCTION PARAMETERS: *Spec*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *cl()* [chapter 4.4.4]

EXAMPLE: *CL(A1, A2); // Bring Axis Channels 1 and 2 into position control*

6.6.9 CONTCNCT, continue CNC-Task

FUNCTION PARAMETERS: Integer constant in the range of 0..3

DESCRIPTION: This command continues the CNC task transferred in the parameter.

NOTE: The command can be used to continue a stopped SAP program. An SAP program which has been stopped with the SAP command *ABORT* can only be restarted (i.e. not continued) with the SAP command *STARTCNCT()* or the PCAP command *startcnct()*. Automatic continuation of stopped tasks is not possible either.

EXAMPLE:

```

...
const
    TASK0 = 0;
...
CONTCNCT(TASK0);      // continue Task 0
CONTCNCT(1);          // continue Task 1

```

6.6.10 COS, cosine function

DECLARATION: *cos(value:double)*

RESULT TYPE: double

DESCRIPTION: The function returns the cosine of *value*. The argument *Value* is interpreted as an angle in the unit rad ($0..2\pi = 0..360$) degrees.

NOTE: *Sin()*, *Tan()* -function

EXAMPLE:

```

...
var
    d1, d2: double;
...
d1 := 3.1415;
d2 := COS(d1);      // d2 := -1.0 (rounded)

```

6.6.11 COSH, hyperbolic cosine function

DECLARATION: *cos(value:double)*

RESULT TYPE: double

DESCRIPTION: The function returns the hyperbolic cosine of *value*.

6.6.12 DISEV, disable event

FUNCTION PARAMETERS: *Event*

REFERENCES: Chapter 6.4 and SAP command *ENEV()*

DESCRIPTION: disables the event handler specified

EXAMPLE: *DISEV(EVEO); // ignore emergency out handler*

6.6.13 ENEV, enable event

FUNCTION PARAMETERS: *Event*

REFERENCES: Chapter 6.4 and SAP command *DISEV()*

DESCRIPTION: enables the event handler specified

EXAMPLE: *ENEV(EVEO); // enable emergency out handler*

6.6.14 EXP, exponential function

DECLARATION: *exp(value:double)*

RESULT TYPE: double

DESCRIPTION: The function returns the value e^{value} , where e is the base of the natural logarithm (2.718281...).

NOTE: Function *Ln()*

6.6.15 JA, jog absolute

FUNCTION PARAMETERS: *Spec* and *Pos*

SYSTEM PARAMETERS: Qualifiers: *jac*, *jvl* and *jtl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: PCAP command *ja()*, SAP command *JAW()*

DESCRIPTION: The axis channel(s) selected is/are moved absolutely to the position setpoints specified. For this purpose, the motor is accelerated with the axis-specific acceleration *jac* to the velocity *jvl*, and moved to the specified target position *Pos*. In addition, you can use the *jtl* parameter to specify a target velocity. The trajectory parameters are specified in the axis-specific units.

NOTE: PCAP command *ja()*

EXAMPLE: *JA(A1:=100.0); // Move Axis 1 absolutely to Position 100*
JA(A1:=100.0, A2:=100.0);

6.6.16 JAW, jog absolute waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: Qualifiers: *jac, jvl* and *jtl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: *JA*

DESCRIPTION: This command is identical to the SAP command *JA()* and PCAP command *ja()*, except that the system also waits for the profile end of all axes involved. The use of this command gives the SAP program a block-like form of the kind found in commercially available CNC controls.

NOTE: You should use EVENT handlers to ensure that the drive is operated properly even in exceptional situations, since the CNC program dwells concomitantly long on this command, particularly when very time-consuming positioning operations are involved.

EXAMPLE: *JAW(A2:=-1000.0); // Move Axis 1 absolutely to Position -1000.0 and*
JAW(A1:=1e3, A2 := 1.3e4); // wait until the profile end is reached

6.6.17 JHI, jog home index

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: Qualifiers: *hac* and *hvl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: PCAP command *jhi()*, SAP command *JHIW()*

DESCRIPTION: The reference search run for the zero track (index) of the rotary transducer or the linear scale for all selected axis channels is started. The search run will be aborted if the traverse distance or angle specified in *Pos* is exceeded.

EXAMPLE: *JHI(A1 := 1.0, A2 := 1.5); // Start reference search run for axes 1 and 2.*

6.6.18 JHIW, jog home index waiting

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *hac* and *hvl*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: This command is identical to PCAP command *jhi()* and SAP command *JHI()*. In addition, the system waits for the profile end of the axes involved.

NOTE: SAP command *JA()*

EXAMPLE: *JHIW(A1 := 5.0);*

6.6.19 JHL, jog home left

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *hac* and *hvl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: PCAP command *jhl()*, SAP command *JHLW()*

DESCRIPTION: The reference search run on a digital input planned with REF for all selected axis channels is started towards the left traversing direction.

EXAMPLE: *JHL(A1);*

6.6.20 JHLW, jog home left waiting

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *hac* and *hvl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: PCAP command *jhl()*, SAP command *JHL()*

DESCRIPTION: This command is identical to the PCAP command *jhl()* and SAP command *JHL()*. In addition, the system waits for the profile end of the axes involved.

NOTE: SAP command *JA()*

EXAMPLE: *JHLW(A2);*

6.6.21 JHR, jog home right

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *hac* and *hvl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: PCAP command *jhr()*, SAP command *JHRW()*

DESCRIPTION: The reference search run on a digital input planned with REF for all selected axis channels is started towards the right traversing direction.

EXAMPLE: *JHR(A2);*

6.6.22 JHRW, jog home right waiting

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *hac* and *hvl*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: This command is identical to the PCAP command *jhr()* and SAP command *JHR()*. In addition, the system waits for the profile end of the axes involved.

NOTE: SAP command *JA()*

EXAMPLE: *JHRW(A1);*

6.6.23 JR, jog relative

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: Qualifiers: *jac, jvl* and *jtl*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: For description, please consult PCAP command *jr()*.

EXAMPLE: *JR(A1);*

6.6.24 JRW, jog relative waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: Qualifiers: *jac, jvl* and *jtl*

SIMULTANEOUS FUNCTION: yes

REFERENCES: JR

DESCRIPTION: This command is identical to the PCAP command *jr()* and the SAP command *JR()*. In addition, the system waits for the profile end of the axes involved.

6.6.25 JS, jog stop

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *sdec*

SIMULTANFUNKTION: yes

DESCRIPTION: For description, please consult PCAP command *js()*.

EXAMPLE: *JS(A1);*

6.6.26 JSW, jog stop waiting

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *sdec*

SIMULTANFUNKTION: *yes*

DESCRIPTION: This command is identical to the PCAP command *js()* and the SAP command *JS()*. In addition, the system waits for the profile end of the axes involved.

EXAMPLE: *JSW(A1);*

6.6.27 LN, natural logarithm function

DECLARATION: *ln(value:double)*

RESULT TYPE: *double*

DESCRIPTION: The function returns the natural logarithm of *value*, i.e. the power by which the constant 2.71828... must be raised to obtain *value*.

NOTE: Values smaller than/equal to 0.0 for *value* are not defined mathematically. In this case the function has no valid return value.
Function *Exp()*

6.6.28 MCA, move circular absolute SMCA, spool motion circular absolute

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: PCAP command *mca()*, *smca()*

EXAMPLE: *MCA(A1 := 50.0, A2 := 0.0, PHI := 720.0);*
SMCA(A1 := 0.0, A2 := 10.0, PHI := 0.1);

6.6.29 MCAW, move circular absolute waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

REFERENCES: PCAP command *mca()*

DESCRIPTION: This command is identical to the SAP command *MCA()*, except that here the system also waits for the profile end of the two axes involved.

6.6.30 MCR, move circular relative SMCR, spool motion circular relative

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: PCAP command *mcr()*, *smcr()*

EXAMPLE: *MCR(A1 := 50.0, A2 := 0.0, PHI := 360.0);*
SMCR(A1 := 0.0, A2 := 10.0, PHI := 45.0);

6.6.31 MCRW, move circular relative waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: This command is identical to the SAP command *MCR()*, except that here the system also waits for the profile end of the two axes involved.

6.6.32 MHA, move helical absolute SMHA, spool motion helical absolute

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: PCAP command *mha()*, *smha()*

NOTE: This command has not yet been implemented at present.

6.6.33 MHAW, move helical absolute waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: This command is identical to the SAP command *MHA()*, except that here the system also waits for the profile end of the three axes involved.

NOTE: This command has not yet been implemented at present.

6.6.34 MHR, move helical relative SMHR, spool motion helical relative

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: PCAP command *mhr()*, *smhr()*

NOTE: This command has not yet been implemented at present.

6.6.35 MHRW, move helical relative waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL, PHI*

DESCRIPTION: This command is identical to the SAP command *MHR()*, except that here the system also waits for the profile end of the three axes involved.

NOTE: This command has not yet been implemented at present.

6.6.36 MLA, move linear absolute SMLA, spool motion linear absolute

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: The description is provided at the PCAP commands *mlla()* and *smla()*.

EXAMPLE: *MLA(A1:=1000.0, A2:=3.2e2);*
SMLA(A1:=100.0, A2:=-335.0);

6.6.37 MLAW, move linear absolute waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: This command is identical to the SAP command *MLA()*, except that here the system also waits for the profile end of the axes involved.

EXAMPLE: *MLAW(A1:=-0.3e3, A2:=100.4);*

6.6.38 MLR, move linear relative SMLR, spool motion linear relative

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: The description is provided at the PCAP commands *mlr()* and *smlr()*.

EXAMPLE: *MLR(A1:=2000.0, A2:=3.2e2);*
SMLR(A1:=300.0, A2:=-35.3);

6.6.39 MLRW, move linear relative waiting

FUNCTION PARAMETERS: *Spec, Pos*

SYSTEM PARAMETERS: *TRAC, TRVL, TRTVL*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: This command is identical to the SAP command *MLRW()*, except that here the system also waits for the profile end of the axes involved.

EXAMPLE: *MLRW(A1:=-3.45e3, A2:=100.4e-1);*

6.6.40 MS, motion stop

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: *None*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *ms()* [chapter 4.4.23].

EXAMPLE: *MS(A1, A2);*

6.6.41 MSW, motion stop waiting

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: *None*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: This command is identical to the PCAP command *ms()* and the SAP command *MS()*. In addition, the system waits for the profile end of the axes involved.

EXAMPLE: *MSW(A1, A2);*

6.6.42 OL, open loop

FUNCTION PARAMETERS: *Spec*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *ol()* [chapter 4.4.24]

EXAMPLE: *OL(A1, A2); // Open position control loop of A1 and A2*

6.6.43 RA, reset axis

FUNCTION PARAMETERS: *Spec*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *ra()*

EXAMPLE: *RA(A1, A2); // Reset Axes A1 and A2*

6.6.44 RDCBD, read COMMON BUFFER double function

DECLARATION: *RDCBS(offset:integer)*

RESULT TYPE: double

DESCRIPTION: The function returns a floating-point value with double accuracy from the CNC-task-specific COMMON BUFFER. The *offset* parameter is a byte offset referenced to the first element (Element 0) of the COMMON BUFFER.

The double data type occupies 8 bytes in the COMMON BUFFER. To enable the PA8000 CPU system to access this correctly, *offset* must always be word-oriented, i.e. have a value which is divisible by 4.

NOTE: The CNC-task-specific buffer size is 1000 bytes. PCAP commands *rdcbcnct()* and *wrcbcnct()*, SAP commands *RDCBx()* and *WRCBx()*

EXAMPLE:

```
...
var
    cbd: double;
...
cbd := RDCBI(500);           // Read in double variable from offset 500
```

6.6.45 RDCBI, read COMMON BUFFER integer function

DECLARATION: *RDCBI(offset:integer)*

RESULT TYPE: integer

DESCRIPTION: The function returns an integer value from the CNC-task-specific COMMON BUFFER. The *offset* parameter is a byte offset referenced to the first element (Element 0) of the COMMON BUFFER.

The integer data type occupies 4 bytes in the COMMON BUFFER. To enable the PA8000 CPU system to access this correctly, *offset* must always be word-oriented, i.e. have a value which is divisible by 4.

NOTE: The CNC-task-specific buffer size is 1000 bytes. PCAP commands *rdcbcnct()* and *wrcbcnct()*, SAP commands *RDCBx()* and *WRCBx()*

EXAMPLE:

```
...
var
    cbi: integer;
...
cbi := RDCBI(500);           // Read in integer variable from offset 500
```

6.6.46 RDCBS, read COMMON BUFFER single function

DECLARATION: RDCBS(*offset*:integer)

RESULT TYPE: single

DESCRIPTION: The function returns a floating-point value with single accuracy from the CNC-task-specific COMMON BUFFER. The *offset* parameter is a byte offset referenced to the first element (Element 0) of the COMMON BUFFER. The single data type occupies 4 bytes in the COMMON BUFFER. To enable the PA8000 CPU system to access this correctly, *offset* must always be word-oriented, i.e. have a value which is divisible by 4.

NOTE: The CNC-task-specific buffer size is 1000 Bytes. PCAP commands *rdcbcnct()* and *wrcbcnct()*, SAP commands *RDCBx()* and *WRCBx()*

EXAMPLE:

```
...
var
    cbs: single;
...
cbs := RDCBI(500);           // Read in single variable from offset 500
```

6.6.47 RS, reset system

DESCRIPTION: PCAP command *rs()*

NOTE: Once this command has been executed, no more monitoring can be performed by the stand-alone application program, since the CNC task is halted by this command.

EXAMPLE: *RS; // reset complete axis system*

6.6.48 SHP, set home position

FUNCTION PARAMETERS: *Spec, Pos*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *shp()*

EXAMPLE: *SHP(A2:=1000.0);*

6.6.49 SIN, sine function

DECLARATION: `sin(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the sine of *value*. The argument *Value* is interpreted as an angle in the unit rad ($0..2\text{Pi} = 0..360$) degrees.

NOTE: *Cos()*, *Tan()* function

EXAMPLE:

```
...
var
    d1, d2: double;
...
d1 := 3.1415;
d2 := SIN(d1);      // d2 := 0.0 (rounded)
```

6.6.50 SINH, hyperbolic sine function

DECLARATION: `cos(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the hyperbolic sine of *value*.

6.6.51 SQRT, square root function

DECLARATION: `sqrt(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the square root of *value*.

NOTE: Negative values of *value* are not defined mathematically. In this case, the function does not have a valid return value.

EXAMPLE:

```
...
var
    d1, d2: double;
...
d1 := 9.0;
d2 := SQRT(d1);    // d2 := 3.0
```

6.6.52 SSMS, start spooled motions synchronousFUNCTION PARAMETERS: *Spec*

SIMULTANEOUS FUNCTION: yes

REFERENCES: PCAP command *ssms()*, SAP command *SSMSW()*

DESCRIPTION: *Spool* commands can be used to transfer commands to the individual axis channels of the PA8000; they are entered in a queue. The *SSMS()* command causes a synchronized start for spooler command processing at all the axes specified in *AS*.

EXAMPLE:

```

...
SMLA(A1:=1000.0, A2:= 1000.0);    // Spool traversing command
SMLR(A1:=200.0, A2:=500.0);      // Spool traversing command
...
SSMS(A1, A2);                     // Start spooler

```

6.6.53 SSMSW, start spooled motions synchronous waitingFUNCTION PARAMETERS: *Spec*

SIMULTANEOUS FUNCTION: yes

REFERENCES: SAP command *SSMS()*

DESCRIPTION: Synchronized start of all axes selected, and wait until all spooled motion profiles of these axes have been run completely and the profile end of all axes involved has been reached.

NOTE: SPOOL mode

EXAMPLE:

```

...
SMLR(A1:=1000.0, A2:= 1000.0);    // Spool traversing command
SMLR(A1:=200.0, A2:=500.0);      // Spool traversing command
...
SSMSW(A1, A2);                    // Start spooler

```

6.6.54 STARTCNCT, start CNC-Task

FUNCTION PARAMETERS: Integer constant in the range of 0..3

DESCRIPTION: This command starts the CNC task transferred in the parameter, and executes the SAP program stored there from its beginning.

NOTE: An SAP program can also start itself automatically from the beginning with this command.

EXAMPLE:

```

...
const
    Task1 = 1;
...
STARTCNCT(Task1);

```

6.6.55 STOP, stop

DESCRIPTION: This command causes the currently running stand-alone application program to stop. In addition, the corresponding CNC task (Task 0, 1, 2, or 3) is put into idle state.
The application program can be resumed by means of the *contcnct()*-PCAP command, the *CONTCNCT()*-SAP command or in the TOOLSET program *mcfg.exe*.

NOTE: Any EVENT handling procedures enabled will no longer be processed after execution of the Stop command. The drive should therefore be put into a safe operating state before this command is executed.

EXAMPLE: `STOP; // stops the SAP program`

6.6.56 STOPCNCT, stop CNC-Task

FUNCTION PARAMETERS: Integer constant in the range of 0..3

DESCRIPTION: This command halts the CNC task transferred in the parameter, and thus halts the SAP program stored in it as well.

NOTE: Any EVENT handling procedures enabled will no longer be processed by the correspondingly selected task after executing *STOPCNCT()*.

EXAMPLE:

```
...
const
    Task3 = 3;
...

STOPCNCT(Task3);
```

6.6.57 TAN, tangent function

DECLARATION: `tan(value:double)`

RESULT TYPE: double

DESCRIPTION: The function returns the tangent of *value*. The argument *Value* is interpreted as an angle in the unit rad ($0..2\pi = 0..360$) degrees.

NOTE: *Sin()*, *Cos()* function

EXAMPLE:

```
...
var
    d1, d2: double;
...
d1 := 0.5;
d2 := TAN(d1); // d2 := 0.5463 (rounded)
```

6.6.58 TANH, hyperbolic tangent function

DECLARATION: `tan(value:double)`

RESULT TYPE: `double`

DESCRIPTION: The function returns the hyperbolic tangent of *value*.

6.6.59 UF, update filter

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETERS: Qualifiers: *kp, ki, kd, kpl, kfca, kfcv*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *uf()*

NOTE: For updating the PIDF filter coefficients, all the qualifiers listed above must be initialized before executing the command.

EXAMPLE:

```

...
A1.kp := 5.0; // Alter proportional amplification
A1.ki := 0.0;
A1.kd := 0.0;
A1.kpl := 0.0;
A1.kfca := 0.0;
A1.kfcv := 0.0;
UF(A1);
...

```

6.6.60 UTROVR, update trajectory override

FUNCTION PARAMETERS: *Spec*

SYSTEM PARAMETER: *TROVR*

SIMULTANEOUS FUNCTION: yes

DESCRIPTION: PCAP command *utrovr()*

EXAMPLE:

```

...
TROVR := 0.9; // Trajectory velocity override = -10%
UTROVR(A1, A2); // Reduced trajectory velocity for axes A1 and A2
...

```


6.6.64 WT, wait timer

FUNCTION PARAMETERS: Integer values with a unit of 64 μ s

DESCRIPTION: Wait for the wait time transferred as a parameter before continuing the SAP program again. This command de-activates the CNC task, and therefore does not need any CPU time. To reduce the workload on the master CPU system, this command may be used in queues, etc.

NOTE: The EVENT handling procedures are not processed while this command is being executed. But if you want these to be monitored, this can, for example, be achieved by means of several *WT()* calls with shorter wait times (perhaps in a loop).

EXAMPLE:

```
...  
CONST sec = 15625;  
...  
WT(5*sec);    // wait 5 s  
...
```

6.7 Compiler commands

As the name implies, a compiler command instructs the compiler, while it is compiling a source text, to execute (or not to execute) certain operations. In *rw_SymPas*, a compiler command is activated as follows:

Inside the SAP source text program, a special syntax is formulated inside a comment: The opening bracket ({} is followed directly by a dollar sign (\$) and the name of the command, which consists of one or more letters. These "comments" can (apart from a few exceptions) appear at any position in the source text at which a normal comment would also be permissible.

6.7.1 Include file

SYNTAX: {\$I Filename}

This compiler command instructs the compiler to read in the file designated by *filename*. Basically, the compiler behaves as if the text read is in place of the {\$I} command. *rw_SymPas* permits include files to be nested up to 15 levels. A file inserted by means of {\$I} can thus itself insert further files, which in turn contain {\$I} commands.

Note: If in the *mcfg.exe* NCC editor environment an include file has already been opened in one of the three editor windows, the SAP source text of this editor will be incorporated, and not the content of the file concerned.

6.7.2 Compiler commands, task selection

SYNTAX: {\$TASK TaskNr}

DESCRIPTION: You can use this compiler command to specify the task (*TaskNr*, values 0..3) in which the SAP program involved is to be run. The information is stored in the autocode file "*filename.cnc*". The PCAP command *txbf()* is used to transfer this file automatically into the right task.

NOTE: If the SAP program concerned does not contain this statement, the task number currently selected will be utilized for compiling. But if the (\$TASK) command is given, the correspondingly selected task number also becomes the default task number for all subsequent display, start and stop commands.
chapter 3.2

EXAMPLE: ...
 const
 Task1 = 1;
 ...

 {\$TASK Task1}; // or
 {\$TASK 1};

