

---

# **POSITIONING AND CONTOURING CONTROL SYSTEM**

## **APCI-8001, APCI-8008 AND CPCI-8004**

## **PROGRAMMING AND REFERENCE MANUAL / PM**



<b>1</b>	<b>Introduction .....</b>	<b>11</b>
<b>2</b>	<b>Internal details of the <i>rw_MOS</i> operating system software .....</b>	<b>12</b>
2.1	The xPCI-800x position controller .....	12
2.1.1	Control loop opened/closed .....	12
2.1.1.1	PIDF filter.....	12
2.1.1.2	The filter parameters $K_D$ , $K_I$ , $K_P$ .....	12
2.1.1.3	Additional phase element .....	13
2.1.1.4	Scan time.....	13
2.2	The xPCI-800x profile generator .....	13
2.2.1	Profile generation for JOG commands.....	14
2.2.2	Profile generation for MOVE commands.....	14
2.2.3	Acceleration.....	15
2.2.4	Maximum velocity.....	15
2.2.5	Target velocity .....	16
2.2.6	Velocity correction .....	16
2.2.7	Target position / Traverse distance .....	16
2.2.8	Operating modes for command processing .....	16
2.2.8.1	Direct mode .....	16
2.2.8.2	Spool mode .....	17
2.2.8.3	Additional notes on spooler operation .....	17
2.3	Interpolation with xPCI-800x .....	17
2.3.1	Linear interpolation.....	18
2.3.1.1	Formal linear interpolation.....	18
2.3.2	Circular interpolation .....	18
2.3.3	Helical interpolation .....	18
2.3.4	Surface area processing .....	18
2.3.5	Synchronous and asynchronous interpolations .....	19
2.4	xPCI-800x limit switch handling .....	19
2.4.1	TOM limit switch function (Turn-Off-Motor).....	19
2.4.2	SMA limit switch function (Stop-Motor-Abruptly).....	20
2.4.3	SMD limit switch function (Stop-Motor-Decelerate) .....	20
<b>3</b>	<b>xPCI-800x Programming methods .....</b>	<b>21</b>
3.1	PC application programming (PCAP programming, or direct programming).....	21
3.2	Stand-alone application programming (SAP programming) .....	21
3.2.1	SAP-Multitasking .....	22
<b>4</b>	<b>PC application programming.....</b>	<b>23</b>
4.1	Introduction .....	23
4.2	Example programs for using the function libraries.....	23
4.3	Definitions, structures and records .....	24
4.3.1	Definitions.....	24
4.3.2	Structures, records and types .....	24
4.3.2.1	Structure/record type AS .....	24
4.3.2.2	Structure/record type TSRP .....	25
4.3.2.3	Structure/record type TRU (Trajectory Units).....	26
4.3.2.4	Structure/record type LMP (Linear Motion Parameters) .....	26
4.3.2.5	Structure/record type CMP (Circular Motion Parameters) .....	26
4.3.2.6	Structure/record type HMP (Helical Motion Parameters).....	27
4.3.2.7	Structure/record type HMP 3D (Helical Motion Parameters 3Dimensional) .....	27
4.3.2.8	Structure/record type ROSI (Risc Operating System Information) .....	28
4.3.2.9	Structure/record type CBCNT (Common Buffer CNC-Task).....	28
4.3.2.10	Structure/record type CNCTS (Computerized Numerical Control Task Status) .....	29

4.4	PCAP high-level language function reference list.....	29
4.4.1	Structure of the reference list .....	29
4.4.2	General information .....	30
4.4.3	azo, activate zero offsets.....	30
4.4.4	BootErrorReport, initialisation error report .....	30
4.4.5	BootFile, boot operating system file .....	31
4.4.6	CardSelect.....	31
4.4.7	ClearCI99 .....	31
4.4.8	cl, close loop.....	32
4.4.9	clv, close loop velocity.....	32
4.4.10	contcnct, continue numeric controller task .....	32
4.4.11	ctru, change trajectory units .....	33
4.4.12	getEnvStr, get Environment String.....	33
4.4.13	gettskinfo, Get Task Informations .....	34
4.4.14	gettskstr, Get Task Message String .....	34
4.4.15	InitMcuErrorReport, initialisation error report .....	35
4.4.16	InitMcuSystem, initialise mcu system.....	35
4.4.17	InitMcuSystem2, initialise mcu system (2 <sup>nd</sup> method) .....	36
4.4.18	InitMcuSystem3, initialise mcu system (3 <sup>rd</sup> method).....	36
4.4.19	ja, jog absolute .....	37
4.4.20	jhi, jog home index .....	37
4.4.21	jhl, jog home left .....	38
4.4.22	jhr, jog home right.....	38
4.4.23	jr, jog relative .....	38
4.4.24	js, jog stop .....	38
4.4.25	lpr – Latch Position Registers .....	39
4.4.26	lprs – Latch Position Registers Synchronous .....	39
4.4.27	lps, latch position synchronous .....	39
4.4.28	mca, move circular absolute - smca, spool motion circular absolute.....	40
4.4.29	mcr, move circular relative - smcr, spool motion circular relative .....	40
4.4.30	mca3d, move circular absolute three dimensional - smca3d, spool motion circular absolute three dimensional .....	40
4.4.31	mcr3d, move circular relative three dimensional - smcr3d, spool motion circular relative three dimensional .....	41
4.4.32	mcuinit, motion control unit initialisation.....	41
4.4.33	MCUG3_SetBoardIntRoutine .....	42
4.4.34	MCUG3_ResetBoardIntRoutine.....	42
4.4.35	mha, move helical absolute - smha, spool motion helical absolute .....	42
4.4.36	mhr, move helical relative - smhr, spool motion helical relative .....	43
4.4.37	mla, move linear absolute - smla, spool motion linear absolute .....	43
4.4.38	mlr, move linear relative - smlr, spool motion linear relative .....	43
4.4.39	ms, motion stop .....	44
4.4.40	MsgToScreen, message to screen .....	44
4.4.41	ol, open loop.....	44
4.4.42	ra, reset axis.....	45
4.4.43	rdap, read axis parameters .....	45
4.4.44	rdaux, read auxiliary register .....	45
4.4.45	rdaxst, read axis status .....	46
4.4.46	rdaxstb, read axis status bit .....	47
4.4.47	rdcbcnct, read common buffer CNC-Task.....	48
4.4.48	rdcd, read common double .....	48
4.4.49	rdci, read common integer .....	49
4.4.50	rdcncts, read computerized numeric controller task status.....	49
4.4.51	rddigi, read digital inputs .....	49
	4.4.51.1 Axis-qualifier digi .....	50
4.4.52	rddigib, read digital input bit .....	51
4.4.53	rddigo, read digital outputs.....	52

4.4.54	rd digob, read digital output bit.....	52
4.4.55	rd dp, read desired position.....	52
4.4.56	rd dpoffset, read desired position offset.....	53
4.4.57	rd dpd – read desired position in display unit.....	53
4.4.58	rd dv, read desired velocity .....	53
4.4.59	rd dvoffset, read desired velocity offset.....	54
4.4.60	rd EffRadius – Read Effective Radius.....	54
4.4.61	rd epc, read EEPROM programming cycle.....	54
4.4.62	rd ErrorReg, read Error Register.....	54
4.4.62.1	Register ErrorReg.....	55
4.4.63	rd f, read filter .....	56
4.4.64	rd GCR, read gear configuration register.....	56
4.4.65	rd gf, read gear factor.....	56
4.4.66	rd gfaux, read gear factor auxiliary channel.....	56
4.4.67	rd hac, read home acceleration.....	57
4.4.68	rd hvl, read home velocity .....	57
4.4.69	rd ifs, read interface status.....	57
4.4.69.1	Axis qualifier ifs.....	57
4.4.70	rd ifsb, read interface status bit.....	58
4.4.71	rd igi, reset digital inputs.....	58
4.4.72	rd ipw, read in position window .....	58
4.4.73	rd irqpc, read interrupt request PC.....	59
4.4.74	rd jac, read jog acceleration .....	59
4.4.75	rd JerkRel, read jerkrel.....	59
4.4.75.1	Axis qualifier <i>jerkrel</i> .....	59
4.4.76	rd jtv, read jog target velocity .....	60
4.4.77	rd jvl, read jog velocity.....	60
4.4.78	rd ledgn, read led green .....	60
4.4.79	rd ledrd, read led red .....	61
4.4.80	rd ledyl, read led yellow.....	61
4.4.81	rd lp, read latched position .....	61
4.4.82	rd lpndx, read latched position index.....	62
4.4.83	rd lsm, read left spool memory.....	62
4.4.84	rd MaxAcc – Read Maximum Acceleration Check.....	62
4.4.85	rd MaxVel – Read Maximum Velocity Check.....	63
4.4.86	rd MCiS – Read Move Commands in Spooler .....	63
4.4.87	rd mcp, read motor command port.....	63
4.4.88	rd MDVel – Read Maximum Velocity Skip .....	64
4.4.89	rd ModeReg – Read MODEREG .....	64
4.4.90	rd mpe, read maximum position error .....	64
4.4.91	rd nfrax – read No-Feed-Rate-Axis.....	64
4.4.92	rd PosErr, read Position Error .....	65
4.4.93	rd rp, read real position .....	65
4.4.94	rd rpd – read real position in display unit .....	65
4.4.95	rd rv, read real velocity.....	65
4.4.96	rd SampleTime – Read Sample Time .....	66
4.4.97	rd sdec, read stop deceleration.....	66
4.4.98	rd sll, read software limit left.....	66
4.4.99	rd slr, read software limit right .....	66
4.4.100	rd slsp, read Slits / Stepperpulses.....	66
4.4.101	rd tp, read target position .....	67
4.4.102	rd tpd – read target position in display unit .....	67
4.4.103	rd trovr, read trajectory override.....	67
4.4.104	rd trovrst, read trajectory override settling time.....	67
4.4.105	rd zeroOffset, read zero offset .....	68
4.4.106	rd ifs, reset interface status register .....	68
4.4.107	RPtoDP, Real-Position to Desired-Position .....	68

4.4.108 rs, reset system .....	69
4.4.109 scp – set controller params .....	69
4.4.110 sdels, spooler delete synchronous .....	69
4.4.111 shp, set home position .....	69
4.4.112 ssms, start spooled motions synchronous .....	70
4.4.113 sstps, spooler stop synchronous .....	70
4.4.114 ssf, Spool-Special-Function .....	70
4.4.114.1 Notes on SSF wait commands .....	72
4.4.115 startcnct, start numeric controller task .....	72
4.4.116 stepcnct, step numeric controller task .....	72
4.4.117 stopcnct, stop numeric controller task .....	72
4.4.118 szpa, set zero position absolute .....	73
4.4.119 szpr, set zero position relative .....	73
4.4.120 txbf, transmit binary file .....	73
4.4.121 txbfErrorReport, initialisation error report .....	74
4.4.122 uf, update filter .....	74
4.4.123 utrov, update trajectory override .....	75
4.4.124 wraux, write auxiliary register .....	75
4.4.125 wrbcnct, write common buffer CNC-Task .....	75
4.4.126 wrcd, write common double .....	76
4.4.127 wrci, write common integer .....	76
4.4.128 wrdigo, write digital outputs .....	76
4.4.129 wrdigob, write digital output bit .....	77
4.4.130 wrdp, write desired position .....	78
4.4.131 wrdp offset, write desired position offset .....	78
4.4.132 wrdoffset, write desired velocity offset .....	79
4.4.133 wrEffRadius – Write Effective Radius .....	79
4.4.134 wrGCR, write gear configuration register .....	79
4.4.135 wrgf, write gear factor .....	79
4.4.136 wrgfau, write gear factor auxiliary channel .....	80
4.4.137 wrhac, write home acceleration .....	80
4.4.138 wrhvl, write home velocity .....	80
4.4.139 wripw, write in position window .....	80
4.4.140 wrjac, write jog acceleration .....	81
4.4.141 wrJerkRel, write jerkrel .....	81
4.4.142 wrjovr, write jog override .....	81
4.4.143 wrjtv, write jog target velocity .....	82
4.4.144 wrjvl, write jog velocity .....	82
4.4.145 wrledgn, write led green .....	82
4.4.146 wrledrd, write led red .....	82
4.4.147 wrledyl, write led yellow .....	83
4.4.148 wrlp, write latched position .....	83
4.4.149 wrlpndx, write latched position index .....	83
4.4.150 wrMaxAcc – Write Maximum Acceleration Check .....	83
4.4.151 wrMaxVel – Write Maximum Velocity Check .....	84
4.4.152 wrmcp, write motor command port .....	84
4.4.153 wrMDVel – Write Maximum Velocity Skip .....	85
4.4.154 wrModeReg – Write MODEREG .....	85
4.4.155 wrmpe, write maximum position error .....	85
4.4.156 wrnfax, write No-Feed-Rate-Axis .....	86
4.4.157 wrrp, write real position .....	86
4.4.158 wrsdec, write stop deceleration .....	86
4.4.159 wrsll, write software limit left .....	86
4.4.160 wrslr, write software limit right .....	87
4.4.161 wrslsp, write Slits / Stepperpulses .....	87
4.4.162 wrtp – write target position .....	87
4.4.163 wrtrovr, write trajectory override .....	87

4.4.164 wrtrovrst, write trajectory override settling time .....	88
--	----

## 5 The rw\_SymPas programming language for stand-alone application programming.....89

5.1	Introduction .....	89
5.2	Lexical grammar.....	89
5.2.1	White space.....	89
5.2.2	Comments .....	89
5.2.3	Symbols.....	90
5.2.3.1	Keywords.....	90
5.2.3.2	Designators .....	90
a)	Name and length restrictions.....	91
b)	Designator upper and lower case.....	91
c)	Unambiguity and validity of designators.....	91
5.2.3.3	Standard designators .....	91
5.2.3.4	Axis designators .....	91
5.2.3.5	Qualified designators.....	92
5.2.3.6	Labels .....	92
5.2.3.7	Constants .....	92
a)	Integer constants.....	93
b)	Decimal constants .....	93
c)	Hexadecimal constants .....	93
d)	Floating-point constants .....	93
e)	The type of floating-point constants .....	93
f)	Declaration of constants.....	93
g)	Punctuation characters.....	93
h)	Parentheses .....	94
i)	Comma .....	94
j)	Semi-colon.....	94
k)	Equals sign .....	94
5.3	Semantic grammar.....	94
5.3.1	Declarations .....	94
5.3.1.1	Objects .....	94
5.3.1.2	Types.....	95
a)	Boolean type.....	95
b)	Integer type.....	95
c)	Floating-point types (real types).....	95
d)	Assignment compatibility of types .....	96
5.3.1.3	Variables.....	96
a)	Automatic type conversion .....	96
5.3.2	Blocks, locality and range of application .....	96
5.3.2.1	Syntax.....	96
a)	Declaration section.....	97
♦	Label declaration section.....	97
♦	Constant declaration section.....	97
♦	Variable declaration section .....	97
b)	Command section.....	97
5.3.2.2	Range of application.....	98
a)	Redeclaration in a subordinate block .....	98
b)	The location of a declaration in a block .....	98
c)	Redeclarations inside a block.....	98
d)	Standard designators .....	98
5.3.3	Variables .....	98
5.3.3.1	The declaration of variables .....	98
a)	Axis-type declaration .....	99
b)	Timer declaration.....	99
5.3.3.2	Conversion of variable types .....	100
5.3.4	Expressions.....	100

5.3.4.1	Syntax of expressions .....	101
5.3.4.2	Operators.....	101
5.3.4.3	Arithmetical operators .....	101
5.3.4.4	Logic operators.....	102
5.3.4.5	Boolean operators .....	102
5.3.4.6	Relational operators .....	102
5.3.5	Statements .....	103
5.3.5.1	Assignments .....	103
5.3.5.2	Procedure calls.....	103
5.3.5.3	The goto statement.....	103
5.3.5.4	Structured instructions.....	104
5.3.5.5	Compound statements .....	104
5.3.5.6	Conditional statements .....	104
a)	The if statement.....	105
5.3.5.7	Loops.....	105
a)	The while statement .....	105
b)	The repeat statement .....	105
c)	The for statement .....	106
5.3.6	Procedures and functions .....	106
5.3.6.1	Procedure declarations .....	106
5.3.6.2	Function declarations .....	106
5.3.7	The syntax of an <i>rw_SymPas</i> program.....	107
5.3.7.1	The program descriptor.....	107
5.3.7.2	The program block.....	107

## 6 Stand-alone application programming .....108

6.1	Introduction .....	108
6.2	<i>rw_SymPas</i> example programs .....	108
6.3	Abbreviations, system parameters, axis specifiers and axis qualifiers.....	108
6.3.1	System parameters .....	108
6.3.1.1	PC interrupt generation .....	110
6.3.1.2	System parameters for unit processing.....	110
6.3.1.3	ERRORREG.....	111
6.3.1.4	MODEREG .....	111
6.3.2	Axis specifiers .....	113
6.3.3	Axis qualifiers .....	113
6.3.4	Structured axis qualifiers.....	116
6.3.5	Abbreviations.....	116
6.4	Reserved procedure names with event function.....	117
6.4.1	Event procedure EVEO.....	117
6.4.2	Event procedure EVDNR .....	117
6.4.3	Event procedure EVLSH .....	117
6.4.4	Event procedure EVLSS .....	118
6.4.5	Event procedure EVMPE .....	118
6.4.6	Event procedure EVUI.....	118
6.4.7	Priority and processing sequence for the event procedures.....	118
6.5	SAP block commands.....	119
6.6	<i>rw_SymPas</i> SAP command reference list .....	119
6.6.1	Structure of the reference list .....	119
6.6.2	ABORT, abort.....	119
6.6.3	ABS, absolute function .....	120
6.6.4	ACOS, arc cosine function .....	120
6.6.5	ASIN, arc sine function.....	120
6.6.6	ATAN, arc tangent function .....	120
6.6.7	AZO, activate zero offsets.....	120
6.6.8	CL, close loop.....	121



6.6.9	CLV.....	121
6.6.10	CONTCNCT, continue CNC-Task.....	121
6.6.11	COS, cosine function.....	121
6.6.12	COSH, hyperbolic cosine function .....	122
6.6.13	DISEV, disable event .....	122
6.6.14	ENEV, enable event.....	122
6.6.15	EXP, exponential function .....	122
6.6.16	JA, jog absolute.....	122
6.6.17	JAW, jog absolute waiting .....	123
6.6.18	JHI, jog home index.....	123
6.6.19	JHIW, jog home index waiting .....	123
6.6.20	JHL, jog home left .....	123
6.6.21	JHLW, jog home left waiting .....	124
6.6.22	JHR, jog home right.....	124
6.6.23	JHRW, jog home right waiting .....	124
6.6.24	JR, jog relative.....	124
6.6.25	JRW, jog relative waiting .....	124
6.6.26	JS, jog stop.....	125
6.6.27	JSW, jog stop waiting .....	125
6.6.28	LN, natural logarithm function .....	125
6.6.29	LPR, latch position registers .....	125
6.6.30	LPRS, latch position registers synchronous .....	125
6.6.31	MCA, move circular absolute - SMCA, spool motion circular absolute.....	126
6.6.32	MCAW, move circular absolute waiting.....	126
6.6.33	MCA3D, move circular absolute three-dimensional SMCA3D, spool move circular absolute three-dimensional .....	126
6.6.34	MCA3DW, move circular absolute three-dimensional waiting .....	126
6.6.35	MCR3D, move circular relative three-dimensional SMCR3D, spool move circular relative three-dimensional.....	126
6.6.36	MCR3DW, move circular relative three-dimensional waiting .....	127
6.6.37	MCR, move circular relative - SMCR, spool motion circular relative .....	127
6.6.38	MCRW, move circular relative waiting .....	127
6.6.39	MHA, move helical absolute - SMHA, spool motion helical absolute .....	127
6.6.40	MHAW, move helical absolute waiting .....	127
6.6.41	MHR, move helical relative - SMHR, spool motion helical relative .....	128
6.6.42	MHRW, move helical relative waiting.....	128
6.6.43	MLA, move linear absolute - SMLA, spool motion linear absolute .....	128
6.6.44	MLAW, move linear absolute waiting .....	128
6.6.45	MLR, move linear relative - SMLR, spool motion linear relative .....	128
6.6.46	MLRW, move linear relative waiting.....	129
6.6.47	MS, motion stop .....	129
6.6.48	MSW, motion stop waiting.....	129
6.6.49	OL, open loop.....	129
6.6.50	POWER.....	129
6.6.51	RA, reset axis .....	130
6.6.52	RDCBD, read COMMON BUFFER double function .....	130
6.6.53	RDCBI, read COMMON BUFFER integer function.....	130
6.6.54	RDCBS, read COMMON BUFFER single function .....	131
6.6.55	RPTODP, Real-Position to Desired-Position .....	131
6.6.56	RS, reset system .....	131
6.6.57	SHP, set home position.....	131
6.6.58	SIN, sine function .....	132
6.6.59	SINH, hyperbolic sine function .....	132
6.6.60	SQR, square function .....	132
6.6.61	SQRT, square root function .....	132
6.6.62	SSF, Spool-Special-Function .....	133
6.6.63	SSMS, start spooled motions synchronous .....	133

6.6.64	SSMSW, start spooled motions synchronous waiting.....	133
6.6.65	STARTCNCT, start CNC-Task.....	134
6.6.66	STOP, stop.....	134
6.6.67	STEPCNCT, stop CNC-Task .....	134
6.6.68	STOPCNCT, stop CNC-Task.....	135
6.6.69	STOPTOSS,.....	135
6.6.70	SZPA – Set Zero Position Absolut .....	135
6.6.71	SZPR – Set Zero Position Relativ .....	136
6.6.72	TAN, tangent function.....	136
6.6.73	TANH, hyperbolic tangent function .....	136
6.6.74	UF, update filter .....	136
6.6.75	UTROVR, update trajectory override .....	137
6.6.76	WRCBI, write COMMON BUFFER integer procedure .....	137
6.6.77	WRCBS, write COMMON BUFFER single procedure .....	137
6.6.78	WRCBD, write COMMON BUFFER double procedure.....	138
6.6.79	WRITE 138	
6.6.80	WRITELN .....	138
6.6.81	WT, wait timer .....	139
6.7	Compiler commands .....	139
6.7.1	Include file .....	139
6.7.2	Task selection .....	140
6.7.3	Full system compiling .....	140
6.8	SAP run time errors.....	141

# 1 Introduction

*What is the content of this manual?*

This manual contains all the details you will need for programming the xPCI-800x controllers. The complete documentation is divided into 3 parts: OM (Operating Manual), PM (Programming and Reference Manual) and CM (Commissioning Manual).

*Which boards belong to the xPCI-800x family?*

The xPCI-800x family includes positioning and contouring control systems of the third generation, that is the boards APCI-8001, APCI-8008 and CPCI-8401. Other devices are being developed to round the family off.

**Further remarks**

If the functions described in this manual do not apply to all devices of the xPCI-800x family, they are specially marked. In this case, the respective function only applies to the marked device!

Before the various programming methods and operating modes can be presented, we must first describe various functions provided by the *rw\_MOS* operating system software. You will find further information on *rw\_MOS* in the Operating Manual, Chapter 4.1.

## 2 Internal details of the *rw\_MOS* operating system software

As already mentioned in the Operating Manual, one of the main factors in the performance capabilities of the xPCI-800x controllers is the *rw\_MOS* operating system software. The following chapters will describe the functions implemented in *rw\_MOS*, like profile generation or limit switch handling.

### 2.1 The xPCI-800x position controller

The basic operating mode of the xPCI-800x controllers is the position control mode. In this operating mode, the board attempts to keep the motor position in the setpoint position. The control loop usually consists of the following components: digital controller - digital/analogue 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 connected axis can be traversed in uncontrolled mode by outputting a value. The PCAP command *c()* (close loop) is used to close the control loop. Note that the current position is adopted 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.

##### 2.1.1.1 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.

##### 2.1.1.2 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.

### 2.1.1.3 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

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

### 2.1.1.4 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 xPCI-800x standard version of the controller boards, the scan time has been set to 1.28 ms.

## 2.2 The xPCI-800x 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 with JOG traversing commands (single-axis movements):

- The initial velocity is negative in relation to 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 relation to 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 traversed 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. The units for the acceleration can be axis-specifically stated in the *mcfg.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 *mcfg.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 set to 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.

According to the operating mode it is important to differentiate the way the override is used by programming: For one-axis traversing commands (JOG commands) the JOG override can be separately programmed for each axis. Yet it must not be made by interpolation travel. For interpolation commands (MOV commands) the trajectory override is to be set and taken over with the command `utrovr` synchronously for all axes, which take part to the interpolation travel. The synchronisation of the axes to be interpolated can only be ensured this way. When the trajectory override is taken over, the value is automatically accepted as a JOG override by the working axes. (can be switched off). To avoid velocity jumps during the programming of the override, an adjustment time can be programmed for the trajectory-override.

### 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 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, 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). Each axis has its own spooler. Once an interpolation command has occurred, the respective spoolers are synchronously loaded and processed. 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 another 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 conditions are to 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.
- Traverse profiles in the spooler need an execution time that is higher than the scan time of the control (default: 1.28 ms). The execution time of a traverse profile is calculated (approx.) by way / velocity. Shorter traverse profiles must be suppressed by the application program.

#### 2.2.8.3 Additional notes on spooler operation

In order that a contour programmed with spooler commands can be run on an accurate path, within a command sequence, all axes must always be programmed and started synchronously. Furthermore, any override value must always be taken over synchronously for all interpolation axes (*utrovr* command). As soon as the spooler operation is interrupted by an asynchronous operation, it can be expected that the programmed contour is not complied with. Automatic spooler synchronisation monitoring can be used to detect this type of error. This is available from RWMOS V2.5.3.88.

If an asynchronous spooler operation is detected by the operating system, the SAF (#19) bit is set. If the JSatSAF (#28) bit is set in the MODEREG register, in this case, all axes are stopped using Jog-Stop with the programmed stop deceleration.

## 2.3 Interpolation with xPCI-800x

Individual axes are moved with the board xPCI-800x using the *jog* commands. The *move* commands are available for moving more than one axis in interpolated mode. The xPCI-800x boards enable 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, one axis can remain in 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 Surface area processing

For interpolation commands the trajectory parameters speed and acceleration are defined with the system parameters PositionUnit (PU) and TimeUnit (TU). The condition is that axes of the same type (translatory or rotatory axes) always take part to the interpolation travel. This is to ensure that the PositionUnit is processed appropriately. When only rotatory axes are involved in translatory interpolation travel as for example by processing cylinder surface, the effective radius must be defined. It will then enable the conversion of the rotatory axes values in translatory interpolation values.

For this, the axis-specific value *effradius* is available. The trajectory speed and acceleration can be set correctly. The radius is given in the unit set in the linear interpolation. This setting is possible for linear, circular as well as helical interpolation.

Example:

The surface of a pipe must be welded. This pipe has a diameter of 200mm and is turned with an axis C defined as rotatory.

```
PU := 0;           // Position Unit = mm
C.effradius := 100; // Enter radius
```

The axis C kann nows be used in the linear interpolation:

```
mlr (X := 25, C := 60);
```

The traverse distance of the rotatory axis is given in the translatory position unit (here mm). In case the traverse distance of the rotary axis must be given in the axis-specific rotatory unit (e.g. deg), the Bit 10 must set in the register MODEREG (see chapter 6.3.1.4). The conversion can be made simultaneously for all axes.

### 2.3.5 Synchronous and asynchronous interpolations

One of the options provided by the xPCI-800x board 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 xPCI-800x limit switch handling

The xPCI-800x board 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 4.4.6 - *cl()*].

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

With this limit switch function, the motor is turned off 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 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 xPCI-800x Programming methods

One of the important features of the xPCI-800x positioning and contouring control system is the real-time multi-task operating system *rw\_MOS* (Mips Operating System).

This is contained in the *rwmos.elf* file and is loaded, once per PC boot, into the local main memory of the xPCI-800x board within a few seconds, using the *mcfg.exe* boot menu or a user program.

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

**Note:** *rwmos.elf* and *mcfg.exe* form part of the xPCI-800x TOOLSET software. You will find further information in the Operating Manual.

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

The xPCI-800x application programming (PCAP) is handled with a user program running on the PC. Programs are written using a higher-level programming language like *Borland C*, *Microsoft C*, *Borland Delphi* or *Microsoft Visual Basic*. 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 xPCI-800x board and then waits for these orders to be processed. After the commands concerned have been autonomously executed by the *PC-Task* in the *rw\_MOS* 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 *mcfg.exe* or the command line compiler *ncc.exe* and generates an operating program which the xPCI-800x board can understand.

This operating program can be loaded onto the xPCI-800x board and is executed autonomously using the *CNC-Task* (CNC = Computerized Numerical Control) in *rw\_MOS*. If synchronization is required between a PC application program and the xPCI-800x board stand-alone program, this can be carried out using predefined system variables, which both system partners (PC and xPCI-800x board) 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\_MOS* 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.

Multitasking programming 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 autonomously stop, start or continue 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. 1,000 common integer and 1,000 common floating-point registers are available to all CNC tasks for this purpose.

Each CNC task can also utilize a local memory area of 1,000 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 xPCI-800x TOOLSET Software includes library functions for the programming languages *Borland Delphi*, *C* (e.g. *Borland C++Builder*, *Microsoft Visual C++*) and *Microsoft Visual Basic*. These are programming tools for the Windows platforms Windows 95, 98, Me, Windows NT 4.0, Windows NT Embedded 4.0, Windows 2000, XP, Vista and Windows 7. The individual functions of the high-level language libraries are executed by using the system driver *mcug3.dll*. The meaning of the individual function parameters and their data types is identical for all programming languages listed above.

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

Programming language	Use description
<b>Borland Delphi</b>	The name of the function library is <i>mcug3.pas</i> . These functions are used to establish the link between the PC application program and the system driver <i>mcug3.dll</i> . This file is declared as a <i>unit</i> and is linked to the application program by means of the <i>uses</i> statement. <u>Important:</u> Various system parameters possess the data type <i>double</i> . This means that the user program has to be compiled with the <i>{\$N+}</i> option!
<b>C (Borland C, Microsoft C or others)</b>	The function library's name is <i>mcug3.lib</i> . These functions are used to establish the link between the PC application program and the system driver <i>mcug3.dll</i> . The Lib files are available for various C programming tools and are to be linked with the application program. The file <i>mcug3.h</i> contains the function declarations. It should be incorporated in the application program by using the <i>#include</i> -instruction.
<b>Microsoft Visual Basic</b>	The name of the function library is <i>mcug3.bas</i> . The link between PC application program and the system driver <i>mcug3.dll</i> is created by the functions declared in <i>mcug3.bas</i> . This file is available as a basic module and can be inserted in the project environment of the application program.

### 4.2 Example programs for using the function libraries

The example programs included in the xPCI-800x 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:

Programming language	Sub-directory	Files
<b>Borland Delphi</b>	<b>Delphi</b>	<i>mcug3.pas</i> , <i>ld.pas</i> , <i>move.pas</i> etc.
<b>Borland C++ Builder</b>	<b>C</b> <b>C/Borland</b>	<i>mcug3.h</i> , <i>ld.c</i> , <i>move.c</i> etc. <i>mcug3.lib</i>
<b>Microsoft Visual C++</b>	<b>C</b> <b>C/mvc</b>	<i>mcug3.h</i> , <i>ld.c</i> , <i>move.c</i> etc. <i>mcug3.lib</i>
<b>Microsoft Visual Basic</b>	<b>Vb</b>	<i>mcug3.bas</i> , <i>ld.bas</i> , <i>move.bas</i> etc.

## 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 system driver does not take up too much PC RAM memory and that several PC applications can access the xPCI-800x controllers at the same time.

All the structure/record types and system constants listed below have been defined in the *mcug3.h*, *mcug3.pas* or *mcug3.bas* files using the programming languages mentioned above.

### 4.3.1 Definitions

Table 1: System constants

Name	Type	Function
MAXAXIS	integer	Maximum number of possible axes. Currently, the TOOLSET software supports up to 18 axes. <b>Warning:</b> This value must not be modified!
LONGINT	integer int long	Synonym for the data type <u>int</u> or <u>integer</u> in the <i>C</i> or <i>DELPHI Pascal</i> programming language and <u>longint</u> in the <i>Microsoft Visual Basic</i> programming language.

### 4.3.2 Structures, records and types

Depending on the programming language involved, we speak either of structures (*C*), records (*Pascal*) or types (*Visual Basic*). The composition and the functioning of these data types is identical in all programming languages. In the description below the term structure or record type is used. 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

Table 2: Structure/record type AS

Element	Type	(Abbreviation 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

A structure/record type *TSRP* has to be declared for each axis to work with the individual axis systems. Using the structure/record elements contained in *TSRP*, data are exchanged with the xPCI-800x board 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 have to make sure that, when more than one axis channel are used, the *TSRP* structures/records are located directly behind each other in memory, since the system driver *mcug3.dll* sometimes accesses the various axis parameters using address computations. Therefore the data alignment has to be defined on 4 bytes if necessary. Correct arrangement in the PC's main memory is reliably achieved by declaring *TSRP* as a field variable.

The size of the field is to be defined for the MAXAXIS axes.

Before use, this data structure must have been initialised. The initialisation is done, e.g. with the commands *InitMcuSystem*, *InitMcuSystem2* or *InitMcuSystem3*. In the most cases an instance of this data structure for each control in the system is defined globally and is initialised when calling the program or after booting of the control. A use of locally declared instances without previous initialisation is not allowed and can lead to unexpected error functions.

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

Element	Type	(Abbreviation meaning), Function
<b>an</b>	LONGINT	(axis number)
<b>kp</b>	double	(PIDF filter parameter kp)
<b>ki</b>	double	(PIDF filter parameter ki)
<b>kd</b>	double	(PIDF filter parameter kd)
<b>kpl</b>	double	(PIDF filter parameter kpl)
<b>kfca</b>	double	(PIDF forward compensation acceleration)
<b>kfcv</b>	double	(PIDF forward compensation velocity)
<b>jac</b>	double	(jog acceleration)
<b>jvl</b>	double	(jog velocity)
<b>jtv</b>	double	(jog target velocity)
<b>jovr</b>	double	(jog override)
<b>hac</b>	double	(home acceleration)
<b>hvl</b>	double	(home velocity)
<b>rp</b>	double	(real position)
<b>dp</b>	double	(desired position)
<b>tp</b>	double	(target position)
<b>sll</b>	double	(software limit left)
<b>slr</b>	double	(software limit right)
<b>ipw</b>	double	(in position window)
<b>mpe</b>	double	(maximum position error)
<b>gf</b>	double	(gear factor)
<b>mcp</b>	LONGINT	(motor command port)
<b>axst</b>	LONGINT	(axis status)
<b>lsm</b>	LONGINT	(left spool memory)
<b>epc</b>	LONGINT	(eeprom programming cycle)
<b>digi</b>	LONGINT	(digital inputs)
<b>digo</b>	LONGINT	(digital outputs)
<b>ifs</b>	LONGINT	(interface status)
<b>scratch</b>	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	Abbreviation 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	(Abbreviation meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tvI	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 from 0. The traverse distances desired are entered into the individual elements to suit the positioning mode involved (absolute or relative). The entries in this data field must correspond with the selected axes in the AS structure/record type. E.g. the traverse distance of the 5 <sup>th</sup> axis (axis index 4) always must be entered in the element dtm[4]

#### 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	(Abbreviation 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) 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	(Abbreviation meaning), Function
<b>ac</b>	double	(acceleration) trajectory acceleration
<b>vl</b>	double	(velocity) trajectory velocity
<b>tvI</b>	double	(target velocity) trajectory target velocity
<b>phi</b>	double	traverse angle in degrees The sign determines the circular direction. If the traverse angle $\leq 1e-100$ a circle is run according to information of the target point.
<b>dtca1</b>	double	(distance to center x-axis)
<b>dtca2</b>	double	(distance to center y-axis)
<b>dtm</b>	Field with MAXAXIS double	(distance to move z-axis and higher) This field must be initialized in accordance with the index of the axes used. Index counting begins from 0. (see also LMP). The traverse distances desired are entered into the individual elements to suit the positioning mode involved (absolute or relative). By running a circle specified by the traverse angle, the traverse direction and target points of the axes to be linearly interpolated are entered from Index 2. By running a circle specified by the target point, the target points of the circular axes are entered as well.

#### 4.3.2.7 Structure/record type HMP 3D (Helical Motion Parameters 3Dimensional)

This structure or record type is a parameter with all 3D interpolation commands.

**Table 8: Structure/record type HMP3D**

Element	Type	(Abbreviation meaning), Function
<b>ac</b>	double	(acceleration) trajectory acceleration
<b>vl</b>	double	(velocity) trajectory velocity
<b>tvI</b>	double	(target velocity) trajectory target velocity
<b>phi</b>	double	traverse angle in degrees The sign determines the circular direction. Running with target point instructions is not possible here.
<b>dtca1</b>	double	(distance to center x-axis)
<b>dtca2</b>	double	(distance to center y-axis)
<b>dtca3</b>	double	(distance to center z-axis)
<b>pn1</b>	double	Surface normal X-vector
<b>pn2</b>	double	Surface normal Y-vector
<b>pn3</b>	double	Surface normal Z-vector
<b>dtm</b>	Field with MAXAXIS double	reserved for future program extensions

#### 4.3.2.8 Structure/record type ROSI (Risc Operating System Information)

This structure or record type is a parameter for the PCAP initialization command *mcuinit()*. After successful initialization of the xPCI-800x board, the following *rw\_MOS* data (*rwmos.elf*) are entered in the ROSI structure:

**Table 9: Structure/record type ROSI**

Element	Type	(Abbreviation meaning), Function
revision	Field with SIZE_STRREV characters	Current software revision of the <i>rw_MOS</i> operating system software.
number_axis	LONGINT	Number of axis channels available
sysfile_loaded	LONGINT	This status variable indicates with the value 1 whether the system file has already been transferred to the xPCI-800x board.

**Note:** You can use the PCAP load command *InitMcuSystem2()* or *InitMcuSystem3()* to transfer the system.dat system file (which is altered mainly by means of the TOOLSET program mcfg.exe) to the xPCI-800x board, 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.

#### 4.3.2.9 Structure/record type CBCNT (Common Buffer CNC-Task)

Each CNC task is provided with a local memory area with a size of 1,000 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 *rdcbcnct()* and *wrcbcnct()*, which can be used to read or write the COMMON BUFFERS.

**Table 10: Structure/record type CBCNCT**

Element	Type	(Abbreviation meaning), Function
Task number	LONGINT	Task number (0..3)
Size	LONGINT	Size of buffer [Bytes]
Buffer	Pointer	Pointer to a buffer which is to be transferred to the xPCI-800x board, or read in from the xPCI-800x board. The buffer must be at least <i>size</i> bytes in size!

#### 4.3.2.10 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	(Abbreviation meaning), Function
<b>errnum</b>	LONGINT	Internal CNC task error number. If no error has occurred, then <i>errnum</i> has the value 0. Information about runtime errors can be found in section 6.8
<b>errline</b>	LONGINT	In connection with <i>errnum</i> , this element is used to display the error-causing source text line of the CNC stand-alone application program.
<b>stackfree</b>	LONGINT	Currently free stack areas [bytes] for the CNC task.
<b>running</b>	LONGINT	This status word shows in Bit 0 whether the CNC task is currently processing a program. Bit 1 shows that the task is in single step operating mode, the system waits for a step ( <i>stepcnct</i> ) or continuation command ( <i>contcnct</i> ). If in the Halt-mode Bit 2 is set, it is indicated that the Task-stopp was caused by the SAP-command <i>writelin</i> . (see also notes to the register <i>MODEREG</i> Bit26 in section 6.3.1.4).

## 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:

Element	Description
<b>FUNCTION NAME:</b>	This is the name which is used to call the function subsequently described.
<b>ABBREVIATION MEANING:</b>	Here you will find a detailed description of the function name concerned.
<b>BORLAND DELPHI :</b>	Here you will find the prototype definitions for the <i>Borland Delphi</i> programming language ( <i>Pascal programming language</i> ). The parameters necessary to call up the function are listed.
<b>C:</b>	Prototype definition for the <i>C</i> programming language, e.g. <i>Microsoft Visual C++</i> or <i>Borland C++Builder</i> otherwise for <i>Borland Delphi</i> .
<b>VISUAL BASIC:</b>	Prototype definition for the <i>Microsoft Visual Basic</i> or <i>Borland Delphi</i> programming language.
<b>TSRP COMPONENTS:</b>	Various functions require components of the structure or record <i>TSRP</i> as parameters. They are listed here.
<b>DESCRIPTION:</b>	Plaintext description of the command.
<b>RETURN VALUE:</b>	If the function returns a value, you will find here a description.
<b>NOTE:</b>	For recurrent notes and explanations, you will find a cross-reference to the corresponding chapters here.
<b>EXAMPLE:</b>	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 4.4.45 - *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 *mcfg.exe*. For the interpolation commands (*move* commands), the units selected in the TRU structure (record) are utilized. This means that a PCAP function is to be called up before executing the *move* commands.

Conversion between application-specific and intra-system units is made automatically, using the factors specified in *mcfg.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

<b>DESCRIPTION:</b>	Each axis channel can be assigned five different zero offsets. You can use the <i>azo()</i> command to activate the axis-specific offset parameters you want. In the <i>set</i> (or <i>set_</i> ) parameter, you specify which set of zero offsets is to be 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.
<b>BORLAND DELPHI:</b>	procedure <i>azo</i> ( <i>set_</i> : integer);
<b>C:</b>	void <i>azo</i> (int <i>set</i> );
<b>VISUAL BASIC:</b>	Sub <i>azo</i> (ByVal <i>set_</i> As Long)
<b>NOTE:</b>	Zero offsets are used to specify a new system of coordinates, without having to influence (new setting) the actual machine zero. The currently set position value of the zero offset can be read with the command <i>rdZeroOffset</i> (Chapter 4.4.105).

### 4.4.4 BootErrorReport, initialisation error report

<b>DESCRIPTION:</b>	This function explains in plaintext the error return values of the function <i>BootFile()</i> described below. A message box displays it on the screen. The user has then to close it.
<b>BORLAND DELPHI:</b>	procedure <i>BootErrorReport</i> ( <i>filename</i> :PChar; <i>error</i> :integer);
<b>C:</b>	void <i>BootErrorReport</i> (char * <i>filename</i> , int <i>error</i> );
<b>VISUAL BASIC:</b>	Sub <i>BootErrorReport</i> (ByVal <i>filename</i> As String, ByVal <i>error</i> As Long)
<b>NOTE:</b>	PCAP command <i>BootFile()</i>
<b>EXAMPLE:</b>	<i>booterror</i> = <i>BootFile</i> ( ... );                   // execute boot sequence <i>BootErrorReport</i> (..., <i>booterror</i> );           // In case of error, display error return value

#### 4.4.5 BootFile, boot operating system file

<b>DESCRIPTION:</b>	The function transfers the operating system software (rwmos.elf) to the control process. The system is reset first. Afterwards the file specified in <i>BootFileName</i> (usually <i>rwmos.elf</i> ) is loaded for the control.																		
<b>BORLAND DELPHI:</b>	function BootFile(var BootFileName:string; TpuBaseAddress: integer):integer;																		
<b>C:</b>	int BootFile(char* BootFileName, int TpuBaseAddress);																		
<b>VISUAL BASIC:</b>	Function BootFile(ByVal filename As String, ByVal TpuBaseAddress As Long) As Long																		
<b>NOTE:</b>	After successful booting the function InitMcuSystem2() or InitMcuSystem3() <u>is to be called up</u> in order to initialise the control completely. <i>TpuBaseAddress</i> is available to be compliant with the PA 8000 controller and is to be initialised with the value 0.																		
<b>RETURN VALUE:</b>	The function delivers return values as follows: <table border="1"> <thead> <tr> <th>Return value</th><th>Error description</th></tr> </thead> <tbody> <tr> <td>0</td><td>No error, boot process is completed successfully</td></tr> <tr> <td>10</td><td>The file name specified in <i>BootFileName</i> is not correct.</td></tr> <tr> <td>11</td><td>The file specified in <i>BootFileName</i> cannot be opened.</td></tr> <tr> <td>12</td><td>Unknown file format. At the moment only files with ELF file format are allowed</td></tr> <tr> <td>13</td><td>Incorrect ELF file format or transfer error.</td></tr> <tr> <td>14</td><td>An incorrect start address in RWMOS.ELF has been detected. RWMOS.ELF may be incorrect.</td></tr> <tr> <td>15</td><td>Incorrect platform for RWMOS.ELF The used RWMOS.ELF is not suited for the available hardware platform.</td></tr> <tr> <td>16</td><td>Verify has failed while transferring the boot file, the file has been incorrectly transferred.</td></tr> </tbody> </table>	Return value	Error description	0	No error, boot process is completed successfully	10	The file name specified in <i>BootFileName</i> is not correct.	11	The file specified in <i>BootFileName</i> cannot be opened.	12	Unknown file format. At the moment only files with ELF file format are allowed	13	Incorrect ELF file format or transfer error.	14	An incorrect start address in RWMOS.ELF has been detected. RWMOS.ELF may be incorrect.	15	Incorrect platform for RWMOS.ELF The used RWMOS.ELF is not suited for the available hardware platform.	16	Verify has failed while transferring the boot file, the file has been incorrectly transferred.
Return value	Error description																		
0	No error, boot process is completed successfully																		
10	The file name specified in <i>BootFileName</i> is not correct.																		
11	The file specified in <i>BootFileName</i> cannot be opened.																		
12	Unknown file format. At the moment only files with ELF file format are allowed																		
13	Incorrect ELF file format or transfer error.																		
14	An incorrect start address in RWMOS.ELF has been detected. RWMOS.ELF may be incorrect.																		
15	Incorrect platform for RWMOS.ELF The used RWMOS.ELF is not suited for the available hardware platform.																		
16	Verify has failed while transferring the boot file, the file has been incorrectly transferred.																		

#### 4.4.6 CardSelect

<b>DESCRIPTION:</b>	With this function you can select an xPCI-800x controller if several should be installed in the PC. The selection is active until the function CardSelect is called for another device or until the application is terminated. After the selection all commands of the mcug3.dll, which are called within the application, refer to to selected device.
<b>BORLAND DELPHI:</b>	function CardSelect (CardNum: integer): integer;
<b>C:</b>	int CardSelect (int CardNumber);
<b>VISUAL BASIC:</b>	Function CardSelect (ByVal CardNr As Long) As Long
<b>PARAMETER:</b>	Index of the board in the PC (0, 1, ...)
<b>RETURN VALUE:</b>	Index of the board that has been selected successfully. -1 if the selected device is not in the PC, in this case the device is selected with Index 0).
<b>NOTE:</b>	See also CM, Chapter 5.3

#### 4.4.7 ClearCI99

<b>DESCRIPTION:</b>	With this function, the common integer variable CI99 is reset synchronously to the operating system software RWMOS.ELF.
<b>BORLAND DELPHI:</b>	procedure ClearCI99 ();
<b>C:</b>	void ClearCI99 (void);
<b>VISUAL BASIC:</b>	Sub ClearCI99 ()

<b>PARAMETER:</b>	none
<b>RETURN VALUE:</b>	none, the variable CI99 is set to 0
<b>NOTE:</b>	This function has to be used when the ssf functions 1005 – 1025 for the synchronisation of spooler commands are used.

#### 4.4.8 cl, close loop

<b>DESCRIPTION:</b>	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 the selected axis channel, the release relays of the assigned axis channel are switched on (CM / Chapter 5.2.9).
<b>BORLAND DELPHI:</b>	procedure cl(var as:AS);
<b>C:</b>	void cl(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub cl(DASEL As ASEL) 'close loop
<b>NOTE:</b>	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! See also PCAP command clv().

#### 4.4.9 clv, close loop velocity

<b>DESCRIPTION:</b>	All axis channels specified in AS are brought in position control with the command. The actual positions of the axes involved are accepted as setpoint positions and the actual speeds as setpoint speeds in order to avoid large system deviations. In addition, all digital outputs planned with PAE are set. This command is to be used when the axes are moving before the control loop is closed. The corresponding axes obtain the current speed when the control loop is closed and are running further with this command. They can now be decelerated e.g. through js() to prevent a hard stop of the axes when the control loop is closed. Depending on the selected axis channel, the release relays of the assigned axis channel are switched on (CM / Chapter 5.2.9).
<b>BORLAND DELPHI:</b>	procedure clv(var as:AS);
<b>C:</b>	void clv(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub clv(DASEL As ASEL) 'close loop velocity
<b>NOTE:</b>	See also PCAP command cl()

#### 4.4.10 contcnct, continue numeric controller task

<b>DESCRIPTION:</b>	You can use this command to continue a SAP program which has previously been halted with the SAP command <i>STOP</i> , <i>STOPCNCT()</i> or with the PCAP command <i>stopcnct()</i> . The task selected in <i>TaskNr</i> (values 0..3) will be continued.
<b>BORLAND DELPHI:</b>	procedure contcnct(TaskNr:integer);
<b>C:</b>	void contcnct(int TaskNr);
<b>VISUAL BASIC:</b>	Sub contcnct(ByVal TaskNr As Long)
<b>NOTE:</b>	A SAP program which has been halted with the SAP command <i>ABORT</i> , can only be restarted (i.e. not continued) with the SAP command <i>STARTCNCT()</i> or the PCAP command <i>startcnct()</i> .



#### 4.4.11 ctru, change trajectory units

DESCRIPTION:	This command can be used to switch over the units for the velocity, acceleration and position parameters of all interpolation commands ( <i>move</i> commands). The parameters are specified in the units selected. The following values are permitted for the TRU structure component <i>pu</i> (position unit).																																							
BORLAND DELPHI:	procedure ctru(var tru:TRU);																																							
C:	void ctru(struct TRU far *tru);																																							
VISUAL BASIC:	Sub ctru(DTRU As tru)																																							
ALL LANGUAGES:	<p>The following values are permitted for the TRU structure component <i>pu</i> (position unit):</p> <table><tr><th>Index</th><th>Unit</th><th>Description</th></tr><tr><td>0</td><td>mm</td><td>Millimeter</td></tr><tr><td>1</td><td>inch</td><td>Inch</td></tr><tr><td>2</td><td>m</td><td>Meter</td></tr><tr><td>3</td><td>rev</td><td>Revolution</td></tr><tr><td>4</td><td>deg</td><td>Degree</td></tr><tr><td>5</td><td>rad</td><td>Radian</td></tr><tr><td>6</td><td>counts</td><td>Counts</td></tr><tr><td>7</td><td>steps</td><td>Steps</td></tr></table> <p>The following values are permitted for the TRU structure component <i>tu</i> (time unit):</p> <table><tr><th>Index</th><th>Unit</th><th>Description</th></tr><tr><td>0</td><td>sec</td><td>Seconds</td></tr><tr><td>1</td><td>min</td><td>Minutes</td></tr><tr><td>2</td><td>tsample</td><td>Sampling time</td></tr></table>	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	Index	Unit	Description	0	sec	Seconds	1	min	Minutes	2	tsample	Sampling time
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																																						
Index	Unit	Description																																						
0	sec	Seconds																																						
1	min	Minutes																																						
2	tsample	Sampling time																																						
NOTE:	<p>The default value for <i>pu</i> and <i>tu</i> 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 <i>move</i> commands)! If the commands involved are axis-specific motion commands (all <i>jog</i> commands), the axis units specified in <i>mcfg.exe</i> are taken into account.</p> <p>The units selected are also determinant for any SAP program running in parallel.</p>																																							

#### 4.4.12 getEnvStr, get Environment String

<b>DESCRIPTION:</b>	With this command the environment variable, which is specified in the string or sign parameter, is read out from the control and the value is entered into the calling parameter.								
<b>BORLAND DELPHI:</b>	function getEnvStr (var EnvStr:string):integer;								
<b>C:</b>	int getEnvStr (char far * EnvStr);								
<b>VISUAL BASIC:</b>	Function getEnvStr (ByVal EnvStr As String) As Long								
<b>RETURN VALUE:</b>	<p>The function can return the following values:</p> <table border="1"> <thead> <tr> <th>Return value</th><th>Error description</th></tr> </thead> <tbody> <tr><td>-1</td><td>Error: E.g. RWMOS does not supply the function</td></tr> <tr><td>0</td><td>The parameter was not found or is an empty string</td></tr> <tr><td>&gt; 0</td><td>Indicates the string length of the found string (without concluding zero byte).</td></tr> </tbody> </table>	Return value	Error description	-1	Error: E.g. RWMOS does not supply the function	0	The parameter was not found or is an empty string	> 0	Indicates the string length of the found string (without concluding zero byte).
Return value	Error description								
-1	Error: E.g. RWMOS does not supply the function								
0	The parameter was not found or is an empty string								
> 0	Indicates the string length of the found string (without concluding zero byte).								

<b>NOTE:</b>	<p>With this function an application program can check the availability of environment variables that are significantly important for the application. In this way an application can react even then controlled, if for example because of a hardware change important characteristics of the control are not available anymore.</p> <p>The writing of environment variables is only possible with unbooted system in fwsetup</p> <p>This function firstly is available in RWMOS.ELF from V2.5.3.37 on and in mcug3.dll from V2.5.3.25 on.</p>
--------------	---

#### 4.4.13 gettskinfo, Get Task Informations

<b>DESCRIPTION:</b>	With this command a task can be asked if there is still a string that is not already read out.
<b>BORLAND DELPHI:</b>	function gettskinfo (TaskNr: integer; var tsinfo: integer): integer;
<b>C:</b>	int gettskinfo (int TaskNr, int *tsinfo);
<b>VISUAL BASIC:</b>	Function gettskinfo (ByVal tasknr As Long, tsinfo As Long) As Long
<b>Parameter:</b>	<p>TaskNr: Tasknummer (0..3)</p> <p>tsinfo: In this variable the function value is returned.</p>
<b>Return value:</b>	<p>&lt; 0: Command is not supported by RWMOS.ELF.</p> <p>= 0: Command has been executed successfully.</p> <p>&gt; 0: Time excess. Command not executed.</p>
<b>NOTE:</b>	<p>This function is returned in tsinfo. Bit 0 indicates that there is a not already completed string (write). Bit 1 indicates that there is a completed string (writeln). The respecting bits are reset automatically by reading the string by gettskstr().</p> <p>Task message strings can be generated in the programming environment of the Stand-Alone-Tasks by WRITE or WRITELN (chapter 6.6.78 and 6.6.79).</p>

#### 4.4.14 gettskstr, Get Task Message String

<b>DESCRIPTION:</b>	With this command the task specific output string can be read.
<b>BORLAND DELPHI:</b>	function gettskstr (TaskNr: integer; buffer: PChar, szbuffer: integer): integer;
<b>C:</b>	int gettskstr (int TaskNr, char * buffer, int szbuffer);
<b>VISUAL BASIC:</b>	Function gettskstr (ByVal tasknr As Long, ByVal buffer As String, ByVal szbuffer) As Long
<b>Parameter:</b>	<p>TaskNr: Task number (0..3)</p> <p>buffer: In this variable the read string is returned.</p> <p>szbuffer: Max. size of the string to be read.</p>
<b>Return value:</b>	Number of the read signs
<b>NOTE:</b>	<p>This call resets the respecting status bits in tsinfo. The storage section of TskStr must be sufficient in order to store the returned string. Max. 512 bytes will be returned.</p> <p>Task Message Strings can be generated in the programming environment of the Stand-Alone-Tasks by WRITE or WRITELN (chapter 6.6.78 and 6.6.79).</p>

#### 4.4.15 InitMcuErrorReport, initialisation error report

<b>DESCRIPTION:</b>	This functions explains in plaintext the error return values of the functions InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3() described below. A message box displays it on the screen. The user has then to close it.
<b>BORLAND DELPHI:</b>	procedure InitMcuErrorReport(error:integer);
<b>C:</b>	void InitMcuErrorReport (int error);
<b>VISUAL BASIC:</b>	Sub InitMcuErrorReport (ByVal error As Long)
<b>NOTE:</b>	PCAP command InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3()
<b>EXAMPLE:</b>	<i>initerror = InitMcuSystem3( ... );    // Start initialisation</i> <i>InitMcuErrorReport(initerror);        // In case of error, display error return value</i>

#### 4.4.16 InitMcuSystem, initialise mcu system

<b>DESCRIPTION:</b>	This function performs the complete software initialization for the drive system. The function call should be executed at the beginning of every PCAP application program at any case before any other PCAP calls. Inside this function, various PCAP basic functions are called. This includes initialization of the axis numbers {an} in the <i>tsrp</i> structure. If the <i>system.dat</i> system file has not yet been transferred onto the xPCI-800x board, this will be done here. At the end of the function, the axis parameters of all axes are read into the <i>tsrp</i> structure.																								
<b>BORLAND DELPHI:</b>	function InitMcuSystem(var tsrp:TSRP):integer;																								
<b>C:</b>	int InitMcuSystem(var TSRP far *tsrp);																								
<b>VISUAL BASIC:</b>	Function InitMcuSystem(DTSRP As TSRP) As Long																								
<b>NOTE:</b>	PCAP commands <i>txbf2()</i> , <i>mcuinit()</i> , structure/record type ROSI <b>Important:</b> This function has been written to be compliant with the PA 8000. You should use instead the functions InitMcuSystem2() or rather InitMcuSystem3().																								
<b>RETURN VALUE:</b>	The function can return the following values: <table border="1"> <thead> <tr> <th>Return value</th><th>Error description</th></tr> </thead> <tbody> <tr> <td><b>0</b></td><td>No error</td></tr> <tr> <td><b>31</b></td><td>No xPCI-800x controller found</td></tr> <tr> <td><b>32</b></td><td>The rw_MOS operating software has not been loaded or has been stopped. See PCAP command <i>BootFile()</i> or service program <i>mcfg.exe</i></td></tr> <tr> <td><b>33</b></td><td>Wrong operating system software. The file versions of the <i>mcug3.dll</i> and <i>rwmos.elf</i> files have incompliant revision states and do not match.</td></tr> <tr> <td><b>34</b></td><td>The driver <i>rnwmc.sys</i> (Windows NT 4.0, 2000) or <i>rnwmc.vxd</i> (Windows 95/98/Me) cannot be opened.</td></tr> <tr> <td><b>35</b></td><td>Error by mapping the physical xPCI-800x board memory.</td></tr> <tr> <td><b>36</b></td><td>Error by mapping in the physical xPCI-800x board memory.</td></tr> <tr> <td><b>37</b></td><td>Error by mapping out the physical xPCI-800x board memory.</td></tr> <tr> <td><b>38</b></td><td>xPCI-800x board cannot be accessed</td></tr> <tr> <td><b>39</b></td><td>xPCI-800x board mail-box-interface cannot be accessed</td></tr> <tr> <td><b>Iderr</b></td><td>Error return value from PCAP command <i>txbf()</i></td></tr> </tbody> </table>	Return value	Error description	<b>0</b>	No error	<b>31</b>	No xPCI-800x controller found	<b>32</b>	The rw_MOS operating software has not been loaded or has been stopped. See PCAP command <i>BootFile()</i> or service program <i>mcfg.exe</i>	<b>33</b>	Wrong operating system software. The file versions of the <i>mcug3.dll</i> and <i>rwmos.elf</i> files have incompliant revision states and do not match.	<b>34</b>	The driver <i>rnwmc.sys</i> (Windows NT 4.0, 2000) or <i>rnwmc.vxd</i> (Windows 95/98/Me) cannot be opened.	<b>35</b>	Error by mapping the physical xPCI-800x board memory.	<b>36</b>	Error by mapping in the physical xPCI-800x board memory.	<b>37</b>	Error by mapping out the physical xPCI-800x board memory.	<b>38</b>	xPCI-800x board cannot be accessed	<b>39</b>	xPCI-800x board mail-box-interface cannot be accessed	<b>Iderr</b>	Error return value from PCAP command <i>txbf()</i>
Return value	Error description																								
<b>0</b>	No error																								
<b>31</b>	No xPCI-800x controller found																								
<b>32</b>	The rw_MOS operating software has not been loaded or has been stopped. See PCAP command <i>BootFile()</i> or service program <i>mcfg.exe</i>																								
<b>33</b>	Wrong operating system software. The file versions of the <i>mcug3.dll</i> and <i>rwmos.elf</i> files have incompliant revision states and do not match.																								
<b>34</b>	The driver <i>rnwmc.sys</i> (Windows NT 4.0, 2000) or <i>rnwmc.vxd</i> (Windows 95/98/Me) cannot be opened.																								
<b>35</b>	Error by mapping the physical xPCI-800x board memory.																								
<b>36</b>	Error by mapping in the physical xPCI-800x board memory.																								
<b>37</b>	Error by mapping out the physical xPCI-800x board memory.																								
<b>38</b>	xPCI-800x board cannot be accessed																								
<b>39</b>	xPCI-800x board mail-box-interface cannot be accessed																								
<b>Iderr</b>	Error return value from PCAP command <i>txbf()</i>																								

#### 4.4.17 InitMcuSystem2, initialise mcu system (2<sup>nd</sup> method)

<b>DESCRIPTION:</b>	This function is identical to the <i>InitMcuSystem()</i> , except that the parameters <i>SystemFileName</i> and <i>TpuBaseAddress</i> are specified. <i>SystemFileName</i> contains the file name of the system file (usually system.dat) as well as path and drive information.
<b>BORLAND DELPHI:</b>	function InitMcuSystem2(var tsrp:TSRP; TpuBaseAddress: integer, var SystemFileName: string):integer;
<b>C:</b>	int InitMcuSystem2(struct TSRP *tsrp, int TpuBaseAddress, char *SystemFileName)
<b>VISUAL BASIC:</b>	Function InitMcuSystem2(DTSRP As TSRP, ByVal TpuBaseAddress As Long, ByVal filename As String) As Long
<b>RETURN VALUE:</b>	The function has the same return values as the function <i>InitMcuSystem()</i> <i>txbf2</i> implicitly called up.
<b>NOTE:</b>	See <i>InitMcuSystem()</i> , <i>TpuBaseAddress</i> has no meaning and is to be transferred with the value 0.

#### 4.4.18 InitMcuSystem3, initialise mcu system (3<sup>rd</sup> method)

<b>DESCRIPTION:</b>	This function is identical to <i>InitMcuSystem()</i> , except that the parameters <i>SystemFileName</i> , <i>rosi</i> , <i>TpuBaseAddress</i> and <i>BoardType</i> are specified. <i>SystemFileName</i> contains the file name of the system file (usually system.dat) as well as path and drive information.
<b>BORLAND DELPHI:</b>	function InitMcuSystem3(var tsrp:TSRP; var rosi:ROSI, TpuBaseAddress: integer, var SystemFileName: string; var BoardType: integer):integer;
<b>C:</b>	int InitMcuSystem3(struct TSRP *tsrp, struct ROSI *rosi, int TpuBaseAddress, char *SystemFileName, int *BoardType)
<b>VISUAL BASIC:</b>	Function InitMcuSystem3(DTSRP As TSRP, DROSI As ROSI, ByVal TpuBaseAddress As Long, ByVal filename As String, BoardType As Long) As Long
<b>RETURN VALUE:</b>	The function has the same return values as the function <i>InitMcuSystem()</i> . Further return values can returned by the function <i>txbf2</i> implicitly called up. The structure <i>rosi</i> is updated according to the system information returned by the control. The value <i>BoardType</i> informs about the control type. <i>BoardType</i> can contain the following values: 1 = PA 8000 (ISA board) 2 = PS 840 (ISA board) 4 = APCI-8001 16 (10 hex) = CPCI-8004 32 (20 hex) = APCI-8008 0 = unknown board or old RWMOS other values = more recent products
<b>NOTE:</b>	See <i>InitMcuSystem()</i> <i>TpuBaseAddress</i> has no meaning and is to be transferred with the value 0. As the initialisation function has currently the highest priority, the use of this function is recommended.

#### 4.4.19 ja, jog absolute

<b>DESCRIPTION:</b>	The axis channels selected in AS are moved absolutely to the target positions specified in <i>TSRP[n].tp</i> using a trapezoidal speed profile. The profile is generated using the axis-specific system parameters <i>jac</i> (jog acceleration), <i>jvl</i> (jog-velocity) and <i>jtl</i> (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 <i>mcfg.exe</i> utility program. The trajectory parameters are stated in the axis-specific units (distance, time) specified in <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	procedure ja(var as:AS; var tsrp:TSRP);
<b>C:</b>	void ja(struct AS far *as, struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub ja(DASEL As ASEL, DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].tp n = 0 .. Number of axes present-1
<b>NOTE:</b>	<p>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 (see chapter 2.2.7)</p> <p>You can set and interrogate the axis-specific parameters like accelerations and velocities at any time using write and read commands. They are not transferred automatically with ja.</p> <p><b>Important:</b> By calling out the function ja the element 0 of the global data structure TSRP must be entered, as ja() takes the index of the used TSRP structure elements from the AS structure entnimmt.</p>

#### 4.4.20 jhi, jog home index

<b>DESCRIPTION:</b>	<p>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 <i>tp</i> has been exceeded. The search run is carried out using a trapezoidal speed profile. The parameters for the profile generator are the system data <i>hac</i> and <i>hvl</i>, which can be set using <i>mcfg.exe</i> or the appropriate write commands. When the index signal (zero track) is detected, the motor is decelerated with the deceleration <i>hac</i> to velocity 0. The <i>tp</i> parameter is stated as a relative traverse distance in the axis-specific position unit. The search direction is determined by the sign of <i>tp</i>. 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.</p> <p>The command can be executed with the aid of the profile end flag (PE) in the <i>axst</i> register and the state of the index signal interrogated with the <i>digi</i> register (Chapter 4.4.51.1). The profile end flag remains set to 0 until the end of the search run.</p>
<b>BORLAND DELPHI:</b>	procedure jhi(var as:AS; var tsrp:TSRP);
<b>C:</b>	void jhi(struct AS far *as, struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub jhi(DASEL As ASEL, TSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].tp n = 0 .. Number of axes present -1
<b>NOTE:</b>	<p>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 <i>rdhvl()</i> and <i>wrhvl()</i>.</p> <p><b>Important:</b> By calling out the function jhi() the element 0 of TSRP must be entered, as jhi() takes the index of the used TSRP structure elements from the AS structure entnimmt.</p>

#### 4.4.21 jhl, jog home left

<b>DESCRIPTION:</b>	This command starts the reference search run for all axis channels specified in <i>AS</i> , in a negative traversing direction. The search run is executed with the aid of an endless trapezoidal speed profile. The axis-specific system data <i>hac</i> and <i>hvl</i> here serve as parameters for profile generation. If a digital input of the xPCI-800x board planned with REF function is activated at the axis channel selected, the search run will be terminated by decelerating (with <i>hac</i> ) the axis to a velocity of 0. This state can be interrogated in the <i>axst</i> register with the aid of the <i>pe</i> profile flag. The profile flag remains set to 0 until the end of the search run.
<b>BORLAND DELPHI:</b>	procedure jhl(var as:AS);
<b>C:</b>	void jhl(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub jhl(DASEL As ASEL)

#### 4.4.22 jhr, jog home right

<b>DESCRIPTION:</b>	This command functions in an identical way to the PCAP command <i>jhl()</i> , except that the search run is started in the positive traversing direction.
<b>BORLAND DELPHI:</b>	procedure jhr(var as:AS);
<b>C:</b>	void jhr(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub jhr(DASEL As ASEL)

#### 4.4.23 jr, jog relative

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>ja()</i> , except that the distance particular <i>tp</i> 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).
<b>BORLAND DELPHI:</b>	procedure jr(var as: AS; var tsrp:TSRP);
<b>C:</b>	void jr(struct AS far *as, struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub jr(DASEL As ASEL, DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].tp n = 0 .. number of existing axes-1
<b>NOTE:</b>	By calling out the function jr the element 0 of TSRP must be entered, as jr() takes the index of the used TSRP structure elements from the AS structure entnimmt.

#### 4.4.24 js, jog stop

<b>DESCRIPTION:</b>	The axis channels - selected in <i>AS</i> - are decelerated with the axis-specific time-delay <i>sdec</i> to velocity 0 and hold in position control. Until the end of deceleration the <i>pe</i> flag is reset in the <i>axst</i> register. You can set and interrogate the time-delay <i>sdec</i> at any time using write and read commands. The default value is specified in the <i>mcfg.exe</i> utility program.
<b>BORLAND DELPHI:</b>	procedure js(var as: AS);
<b>C:</b>	void js(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub js(DASEL As ASEL)
<b>NOTE:</b>	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 2.2.7]. The value <i>sdec</i> = 0 forces an immediate axis stop without braking ramp.

#### 4.4.25 lpr – Latch Position Registers

<b>DESCRIPTION:</b>	This command can start the recording of the graphical system analysis for one axis.
<b>BORLAND DELPHI:</b>	procedure lpr (var latch_infos: LATCH_INFOS);
<b>C:</b>	void lpr (struct LATCH_INFOS *latch_infos);
<b>VISUAL BASIC:</b>	Sub lpr (DLATCH_INFOS As LATCH_INFOS)
<b>RETURN VALUE:</b>	None
<b>EFFECT:</b>	After the command lprs has been executed the recording of the graphical system analysis is started. The recording parameters are given in latch_infos.
<b>NOTE:</b>	See also command lprs and graphical system analysis in mcfg <b>Important:</b> The data structure latch_infos must be aligned in 4 bytes.

#### 4.4.26 lprs – Latch Position Registers Synchronous

<b>DESCRIPTION:</b>	This command can start the recording of the graphical system analysis synchronously for one or several axes.
<b>BORLAND DELPHI:</b>	procedure lprs (var as: AS; var latch_infos: LATCH_INFOS);
<b>C:</b>	void lprs (struct AS *as, struct LATCH_INFOS *latch_infos);
<b>VISUAL BASIC:</b>	Sub lprs (DASEL As ASEL, DLATCH_INFOS As LATCH_INFOS)
<b>RETURN VALUE:</b>	None
<b>EFFECT:</b>	After the command lprs has been executed the recording of the graphical system analysis is started. The recording parameters are given in latch_infos. The element <i>san</i> of the data structure <i>latch_infos</i> has no significance with this command as the axes are specified in <i>as</i> .
<b>NOTE:</b>	See also command lpr and graphical system analysis in mcfg <b>Important:</b> The data structure latch_infos must be aligned in 4 bytes.

#### 4.4.27 lps, latch position synchronous

<b>DESCRIPTION:</b>	This command can be used to initiate a latch routine synchronized with the scan cycle of the axis channel selected in <i>an</i> . After call-up, the actual position <i>{rp}</i> is put into intermediate storage after every <i>mst</i> scan intervals. If a latch procedure has taken place, this will be displayed in the <i>axst</i> register in the <i>lpsf</i> flag (Bit No. 16). The PCAP read command <i>rdlp()</i> or the <i>lp</i> SAP axis qualifier can be used to read out the position from intermediate storage. Readout will also erase the <i>lpsf</i> flag in the <i>axst</i> register.
<b>BORLAND DELPHI:</b>	procedure lps(an: integer; mst: integer);
<b>C:</b>	void lps(int an, int mst);
<b>VISUAL BASIC:</b>	Sub lps(ByVal an As Long, ByVal mst As Long)
<b>NOTE:</b>	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 <i>mst</i> 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.

#### 4.4.28 mca, move circular absolute - smca, spool motion circular absolute

<b>DESCRIPTION:</b>	<p>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 <i>ac</i>, the trajectory velocity <i>vl</i> and the trajectory target velocity <i>tv</i>. The coordinates entered in <i>dtca1</i> and <i>dtca2</i> specify the circle's centre in an absolute system of units. Note that <i>dtca1</i> is assigned to the first axis programmed in AS and <i>dtca2</i> to the second axis specified in AS. The units for the trajectory parameters are selected with the PCAP command <i>ctru()</i>.</p> <p>The angle <i>phi</i> specifies the traverse angle to be run with the unit <u>degrees</u>. The sense of rotation is specified by the sign of the angle variable. Positive values signify anti-clockwise rotation and negative values signify clockwise rotation. The traverse angle range is not fixed to defined limits, i.e. part or multiple circles can be run as well.</p>
<b>BORLAND DELPHI:</b>	<pre>procedure mca(var as: AS; var cmp: CMP); procedure smca(var as: AS; var cmp: CMP);</pre>
<b>C:</b>	<pre>void mca(struct AS far *as, struct CMP far *cmp); void smca(struct AS far *as, struct CMP far *cmp);</pre>
<b>VISUAL BASIC:</b>	<pre>Sub mca(DASEL As ASEL, CMP As CMP) Sub smca(DASEL As ASEL, CMP As CMP)</pre>
<b>NOTE:</b>	Chapter 2.3 Interpolation with the xPCI-800x.

#### 4.4.29 mcr, move circular relative - smcr, spool motion circular relative

<b>DESCRIPTION:</b>	<p>This command is identical to the PCAP command <i>mca()</i>, except that the coordinates specified in <i>dtca1</i> and <i>dtca2</i> are incrementally (or relatively) referenced to the current motor position.</p>
<b>BORLAND DELPHI:</b>	<pre>procedure mcr(var as: AS; var cmp: CMP); procedure smcr(var as: AS; var cmp: CMP);</pre>
<b>C:</b>	<pre>void mcr(struct AS far *as, struct CMP far *cmp); void smcr(struct AS far *as, struct CMP far *cmp);</pre>
<b>VISUAL BASIC:</b>	<pre>Sub mcr(DASEL As ASEL, CMP As CMP) Sub smcr(DASEL As ASEL, CMP As CMP)</pre>
<b>NOTE:</b>	Chapter 2.3 Interpolation with the xPCI-800x.

#### 4.4.30 mca3d, move circular absolute three dimensional - smca3d, spool motion circular absolute three dimensional

<b>DESCRIPTION:</b>	<p>This function is used to carry out the circular interpolation of the 3 specified axis channels. There are not restrictions regarding axis selection. Circular interpolation is carried out on the basis of a trapezoidal speed profile, i.e. considering the maximum acceleration and maximum velocity. The trajectory acceleration <i>ac</i>, the trajectory velocity <i>vl</i> and the trajectory target velocity <i>tv</i> are used as interpolation parameters in <i>hmp3d</i>. The coordinates entered in <i>dtca1</i>, <i>dtca2</i> and <i>dtca3</i> specify the circle's center in absolute measurement system. Note that <i>dtca1</i> is assigned to the first axis programmed in AS, <i>dtca2</i> to the second axis and <i>dtca3</i> to the third axis. The units of the trajectory parameters are selected with PCAP command <i>ctru()</i>.</p> <p>The circle can be traversed in any wished level, which is specified in the surface normal in PN1, PN2 and PN3. The current start coordinates always remain in the given level.</p>
---------------------	---



	<p>The angle <i>phi</i> specifies the traverse angle to be run with the unit <u>Degree</u>. The sense of rotation is determined by the sign of the angle variable. Positive values signify anti-clockwise rotation and negative values clockwise rotation. The traverse angle range is not fixed to defined values, i.e. part or multiple circle can be runs as well.</p> <p>The data field <i>dtm[]</i> is not used here.</p>
<b>BORLAND DELPHI:</b>	<pre>procedure mca3d(var as: AS; var hmp3d: HMP3D); procedure smca3d(var as: AS; var hmp3d: HMP3D);</pre>
<b>C:</b>	<pre>void mca3d(struct AS far *as, struct HMP3D far *hmp3d); void smca3d(struct AS far *as, struct HMP3D far *hmp3d);</pre>
<b>VISUAL BASIC:</b>	<pre>Sub mca3d(DASEL As ASEL, HMP3D As HMP3D) Sub smca3d(DASEL As ASEL, HMP3D As HMP3D)</pre>
<b>NOTE:</b>	Chapter 2.3 Interpolation with the xPCI-800x.

#### 4.4.31 mcr3d, move circular relative three dimensional - smcr3d, spool motion circular relative three dimensional

<b>DESCRIPTION:</b>	This function is identical to the PCAP command <i>mca3d()</i> except that the coordinates specified in <i>dtca1</i> , <i>dtca2</i> and <i>dtca3</i> are incrementally or relatively referenced to the instantaneous motor positions.
<b>BORLAND DELPHI:</b>	<pre>procedure mcr3d(var as: AS; var hmp3d: HMP3D); procedure smcr3d(var as: AS; var hmp3d: HMP3D);</pre>
<b>C:</b>	<pre>void mcr3d(struct AS far *as, struct HMP3D far *hmp3d); void smcr3d(struct AS far *as, struct HMP3D far *hmp3d);</pre>
<b>VISUAL BASIC:</b>	<pre>Sub mcr3d(DASEL As ASEL, HMP3D As HMP3D) Sub smcr3d(DASEL As ASEL, HMP3D As HMP3D)</pre>
<b>NOTE:</b>	Chapter 2.3 Interpolation with the xPCI-800x.

#### 4.4.32 mcuinit, motion control unit initialisation

<b>DESCRIPTION:</b>	<p>This function is used to carry out various initialization routines inside the system driver <i>mcug3.dll</i>. It checks whether communication is possible between PC and xPCI-800x board. If this is the case, the <i>rw_MOS</i>-specific system data returned by the xPCI-800x board are entered in the structure/record <i>ROSI</i>, which can then be used to check the <i>rw_MOS</i>-specific system information for validity. If it has not proved possible to establish communication to the xPCI-800x, the entire <i>TOSI</i> structure will have the value 0</p>
<b>BORLAND DELPHI:</b>	<pre>procedure mcuinit(var rosi:ROSI);</pre>
<b>C:</b>	<pre>void mcuinit(struct ROSI far *rosi);</pre>
<b>VISUAL BASIC:</b>	<pre>Sub mcuinit(DROSI As ROSI)</pre>
<b>NOTE:</b>	<p>This command does not trigger a reset on the xPCI-800x board. This must be carried out with the PCAP commands <i>ra()</i> or <i>rs()</i>.</p> <p>You can use the <i>ROSI.sysfile_loaded</i> return value to ascertain whether the <i>system.dat</i> system file has already been transferred to the xPCI-800x board with the aid of the PCAP load command <i>txbf2()</i>. If this value is 0, then after a successful <i>mcuinit()</i> PCAP command the PCAP command <i>txbf2()</i> must be executed, so that you can work with the xPCI-800x board.</p> <p>The PCAP example programs provided include this command in the <i>InitMcuSystem()</i>, <i>InitMcuSystem2()</i> and <i>InitMcuSystem3()</i> functions, where the monitoring mechanism for system initialization is once more illustrated.</p> <p><b>Important:</b> <i>mcuinit()</i> is compliant to the PA 8000 and is to be entirely replaced by the <i>InitMcuSystem3()</i> command.</p>

#### 4.4.33 MCUG3\_SetBoardIntRoutine

<b>DESCRIPTION:</b>	With this function a user specific interrupt processing routine can be installed and activated.
<b>BORLAND DELPHI:</b>	function MCUG3_SetBoardIntRoutine (func : Pointer): integer;
<b>C:</b>	int MCUG3_SetBoardIntRoutine(InterruptRoutine func);
<b>VISUAL BASIC:</b>	Function MCUG3_SetBoardIntRoutine (ByVal func As Long) As Long
<b>PARAMETER:</b>	func is a function pointer onto the interrupt processing routine that was written by the user. It is declared (in C++) e.g. in the following manner: void CALLBACK EventHandler(int IRQLineBits) {}
<b>RETURN VALUE:</b>	No meaning
<b>NOTE:</b>	Within the interrupt processing routine the programming conventions of the Window operating system have to be observed. So, it is not allowed to generate window objects in a callback-handler. For Visual Basic 6.0 the additional module „MCUG3Interrupt.BAS“ is contained in the scope of delivery for the use of this function

#### 4.4.34 MCUG3\_ResetBoardIntRoutine

<b>DESCRIPTION:</b>	With this function a previously enabled user specific interrupt processing routine can be disabled.
<b>BORLAND DELPHI:</b>	function MCUG3_ResetBoardIntRoutine (): integer;
<b>C:</b>	int MCUG3_ResetBoardIntRoutine(void);
<b>VISUAL BASIC:</b>	Function MCUG3_ResetBoardIntRoutine () As Long
<b>NOTE:</b>	Before quitting the application the currently installed interrupt service routine must be desinstalled.

#### 4.4.35 mha, move helical absolute - smha, spool motion helical absolute

<b>DESCRIPTION:</b>	<p>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 <i>mca()</i> also apply to this command, except that the trajectory parameters are entered in the structure/record HMP. For additional axes specified in AS, the <i>dtm</i> parameter can be programmed as well. These are the absolute target positions for additional axes. While the first two axes perform a circular interpolation, the other ones execute a linear movement. All axes reach their target positions at the same moment.</p> <p>Unlike the circular interpolation the circle target point can be defined per target position instead through the circle angle. This case must be displayed by the user with a traverse angle value <math>\leq 1e-100</math>. The angle sign indicates the traverse direction.</p> <p>The required circle target point are defined in this case in <i>dtm</i> [0] and <i>dtm</i>[1] of HMP.</p> <p>In case the given target point is not located on the circle which results from the start point and the middle point, the target position is corrected.</p>
<b>BORLAND DELPHI:</b>	procedure mha(var as: AS; var hmp: HMP); procedure smca(var as: AS; var hmp: HMP);
<b>C:</b>	void mha(struct AS far *as, struct HMP far *hmp); void smha(struct AS far *as, struct HMP far *hmp);
<b>VISUAL BASIC:</b>	Sub mha(DASEL As ASEL, HMP As HMP) Sub smha(DASEL As ASEL, HMP As HMP)

#### 4.4.36 mhr, move helical relative - smhr, spool motion helical relative

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>mha()</i> , except that the distance particulars programmed in <i>dtca1</i> , <i>dtca2</i> and <i>dtma3</i> are referenced to the instantaneous motor position incrementally (or relatively).
<b>BORLAND DELPHI:</b>	procedure mhr(var as: AS; var hmp: HMP); procedure smhr(var as: AS; var hmp: HMP);
<b>C:</b>	void mhr(struct AS far *as, struct HMP far *hmp); void smhr(struct AS far *as, struct HMP far *hmp);
<b>VISUAL BASIC:</b>	Sub mhr(DASEL As ASEL, HMP As HMP) Sub smhr(DASEL As ASEL, HMP As HMP)

#### 4.4.37 mla, move linear absolute - smla, spool motion linear absolute

<b>DESCRIPTION:</b>	<p>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 <i>ac</i>, the trajectory velocity <i>vI</i> and the trajectory target velocity <i>tvI</i> for linear interpolation. The units for the trajectory parameters are selected with the <i>ctru()</i> command.</p> <p>Depending on the number of axes involved (<i>unoa</i>), you enter the axes you want in the <i>san</i> field and the corresponding traverse distances in the <i>dtm</i> field. Note that the traverse distance in the <i>dtm[n]</i> field is assigned to the axis number <i>n + 1</i>. 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.</p>
<b>BORLAND DELPHI:</b>	procedure mla(var as: AS; var lmp: LMP); procedure smla(var as: AS; var lmp: LMP);
<b>C:</b>	void mla(struct AS far *as, struct LMP far *lmp); void smla(struct AS far *as, struct LMP far *lmp);
<b>VISUAL BASIC:</b>	Sub mla(DASEL As ASEL, lmp As lmp) Sub smla(DASEL As ASEL, lmp As lmp)
<b>NOTE:</b>	Chapter 2.3 Interpolation with the xPCI-800x.

#### 4.4.38 mlr, move linear relative - smlr, spool motion linear relative

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>mla()</i> , except that the traverse distances specified in the <i>dtm</i> field are interpreted incrementally or relatively to the instantaneous motor position.
<b>BORLAND DELPHI:</b>	procedure mlr(var as: AS; var lmp: LMP); procedure smlr(var as: AS; var lmp: LMP);
<b>C:</b>	void mlr(struct AS far *as, struct LMP far *lmp); void smlr(struct AS far *as, struct LMP far *lmp);
<b>VISUAL BASIC:</b>	Sub mlr(DASEL As ASEL, lmp As lmp) Sub smlr(DASEL As ASEL, lmp As lmp)
<b>NOTE:</b>	Chapter 2.3 Interpolation with the xPCI-800x.

#### 4.4.39 ms, motion stop

<b>DESCRIPTION:</b>	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 <i>pe</i> flag in the <i>axst</i> 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 <i>sdec</i> .
<b>BORLAND DELPHI:</b>	procedure ms(var as: AS);
<b>C:</b>	void ms(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub ms(DASEL As ASEL)
<b>NOTE:</b>	Axes which are not interpolating jointly may reach the target point at different points in time.

#### 4.4.40 MsgToScreen, message to screen

<b>DESCRIPTION:</b>	This command disables or enables the screen messages of the DDL driver. If the parameter Enable = 0 the screen messages are disabled.
<b>BORLAND DELPHI:</b>	procedure MsgToScreen (Enable: integer);
<b>C:</b>	void MsgToScreen (long Enable);
<b>VISUAL BASIC:</b>	Sub MsgToScreen (ByVal Enable As Long)
<b>NOTE:</b>	This option is important for systems without user interface. If screen messages are enabled the system can otherwise wait for an entry which cannot be used. This command is available from the version 3.5.2.10.

#### 4.4.41 ol, open loop

<b>DESCRIPTION:</b>	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 xPCI-800x digital outputs planned with PAE function are de-activated for the axis channels programmed. Depending on the axis channels selected, the relays K2 (axis channel 1), K3 (axis channel 2) and K4 (axis channel 3) are switched off. [CM / Chapter 5.2.9]
<b>BORLAND DELPHI:</b>	Procedure ol(var as: AS);
<b>C:</b>	void ol(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub ol(DASEL As ASEL)
<b>NOTE:</b>	This command is used mainly in exceptional situations, like limit switch limitation, position error violation, etc.

#### 4.4.42 ra, reset axis

<b>DESCRIPTION:</b>	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 <i>wrjovr()</i> and <i>wrtrovr()</i> ) are set to the value 1.0. Any software limits planned will no longer be monitored for the axis channels selected in <i>ra()</i> .
<b>BORLAND DELPHI:</b>	Procedure ra(var as: AS);
<b>C:</b>	void ra(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub ra(DASEL As ASEL)
<b>NOTE:</b>	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. <b>Warning:</b> PAE outputs of other axes in the same output group which could have been set, are reset with this command.

#### 4.4.43 rdap, read axis parameters

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure rdap(var tsrp:TSRP);
<b>C:</b>	void rdap(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdap(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	all, i.e. TSRP[n].an .. TSRP[n].ifs
<b>RETURN VALUE:</b>	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.
<b>NOTE:</b>	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.

#### 4.4.44 rdaux, read auxiliary register

<b>DESCRIPTION:</b>	The function returns the axis-specific auxiliary register. [Chapter 6.3.3]
<b>BORLAND DELPHI:</b>	procedure rdaux (var tsrp:TSRP);
<b>C:</b>	void rdaux (struct TSRP *tsrp);
<b>VISUAL BASIC:</b>	Sub rdaux(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].aux
<b>NOTE:</b>	See also chapter 4.4.124

#### 4.4.45 rdaxst, read axis status

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure rdaxst(var tsrp:TSRP);
<b>C:</b>	void rdaxst(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdaxst(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].axst
<b>RETURN VALUE:</b>	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.

Table 12: Bit-decoded structure of the *axst* word

Bit No.	Name	Function
0 0000 0001	-	Not assigned, this flag has an undefined value.
1 0000 0002	<i>eo</i>	Emergency-Out Error-Flag: Has the value 1, when a digital input as EO planned is active.
2 0000 0004	<i>dnr</i>	1. Drive-Not-Ready error-flag: Has the value 1, when a digital input (as DR-planned) is inactive.
3 0000 0008	<i>lslh</i>	Limit-Switch Left Hardware error-flag: Has the value 1, when a digital input (as LSL_SMD, LSL_TOM or LSL_SMA planned) is active.
4 0000 0010	<i>lsrh</i>	Limit-Switch Right Hardware error-flag: Has the value 1, when a digital input (as LSL_SMD, LSR_TOM or LSR_SMA planned) is active.
5  0000 0020	<i>lsls</i>	2. 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 or SMA and the shp() command must have been executed beforehand.
6  0000 0040	<i>lsrs</i>	3. 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 or SMA and the shp() command must have been executed beforehand.
7  0000 0080	<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  0000 0100	<i>dhfe</i>	4. Data Handling error-flag: has the value 1, when a data error (e.g. inconsistent profile data) is detected by the <i>rw_MOS</i> operating system. 5. In certain cases, when this bit occurs, the control loops of the axis concerned in each case are opened. The resetting of this bit is only possible by a system restart ( <i>BootFile</i> ) or by the execution of the <i>ra()</i> [chapter 4.4.42] or <i>rs()</i> [chapter 4.4.107] commands. If necessary, also the system variable ErrorReg must be taken into consideration.

Bit No.	Name	Function
9	<i>cef</i>	Data Configuration error-flag. The <i>cef</i> flag is set when the information for operating modes, signal processing or CPU number on the xPCI-800x do not agree with the system data ( <i>system.dat</i> ). The configuration-check is carried out automatically after the following events: <ul style="list-style-type: none"> <li>• after every reset statement (e.g. PCAP command <i>rs()</i>)</li> <li>• after every transfer of the <i>system.dat</i> system file with the PCAP command <i>txbf2()</i>.</li> </ul> The cause of the error can be eliminated by saving the system data in the [Save Changes] menu.
0000 0200		
10..11		6. Not assigned, these flags always have a non-defined value .
12	<i>pe</i>	7. Profile-End status-flag: Has the value 1, when the end of the profile has been reached.
0000 1000		
13	<i>cl</i>	8. Closed-Loop status-flag: Has the value 1, when the axis channel is in position control.
0000 0200		
14	<i>ip</i>	9. 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 then the position differential contained in the axis-specific system parameter {ipw}.
0000 4000		
15	<i>ui</i>	10. User Input status-flag: Has the value 1, when a digital input (as UI-planned) is active.
0000 8000		
16	<i>lpsf</i>	The Latch Position Synchronous Flag indicates that latching has occurred synchronously to the sampling cycle [chapter 4.4.25], or that a digital input (planned with the LP function) has been activated (MCFG / Chapter 1.7.2.5). The flag is reset by reading the latched position LP, e.g. by the command <i>rdlp</i> .
0001 0000		
17	<i>reference d</i>	This flag indicated that the respecting axis is reduced with the command <i>shp</i> . The flag is reset at booting with the commands <i>rs()</i> , <i>ra()</i> or by writing on <i>rp</i> . At stepper motor axes the flag is also reset at opening and closing the control loop. This flag is only available from RWMOS version V2.5.3.16.
0002 0000		
18	<i>refh</i>	Ref-hardware input flag: Has value 1 if a digital input projected as REF is enabled. This flag is only available from RWMOS version V2.5.3.47.
0004 0000		
19	<i>saf</i>	Spooler-Asynchronous-Flag – indicates that the spooler of this axis in the interpolation compound is asynchronous. The flag is reset by <i>ResetAxis (ra)</i> or when the control loop is closed ( <i>cl</i> ). This flag is only available from RWMOS version V2.5.3.88.
0008 0000		
18..31		11. Not assigned, these flags always have a non-defined value and are reserved for future use.

#### 4.4.46 rdaxstb, read axis status bit

DESCRIPTION:	This function can be used to interrogate <u>one</u> piece of the axis status-information of the xPCI-800x board. The axis number must be specified in the <i>an</i> parameter (0, 1, ... MAXAXIS).
BORLAND DELPHI:	function rdaxstb(an:integer; bitnr:integer):integer;
C:	int rdaxstb(int an, int bitnr);
VISUAL BASIC:	Function rdaxstb(ByVal an As Long, ByVal bitnr As Long) As Long
RETURN VALUE:	The function returns the value 1 or if the corresponding input of <i>bitnr</i> is active. Assignment of <i>bitnr</i> to the axis status information involved is described in Table 12, 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:	See also PCAP command <i>rdaxst()</i>

#### 4.4.47 rdcbcnct, read common buffer CNC-Task

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.															
BORLAND DELPHI:	function rdcbcnct(var cbcnct:CBNCT):integer;															
C:	int rdcbcnct(struct CBNCT far *cbcnct);															
VISUAL BASIC:	Sub rdcbcnct(DCBNCT As CBNCT)															
RETURN VALUE:	The function rdcbcnct() has the following bit-coded return value: <table><tr><th>Bit number</th><th>Return value</th><th>Error description</th></tr><tr><td>0</td><td>0</td><td>No error</td></tr><tr><td>0</td><td>1</td><td>when invalid Task Number</td></tr><tr><td>1</td><td>0</td><td>No error</td></tr><tr><td>1</td><td>1</td><td>when maximum permitted buffer size exceeded. This means that the function normally returns the value 0.</td></tr></table>	Bit number	Return value	Error description	0	0	No error	0	1	when invalid Task Number	1	0	No error	1	1	when maximum permitted buffer size exceeded. This means that the function normally returns the value 0.
Bit number	Return value	Error description														
0	0	No error														
0	1	when invalid Task Number														
1	0	No error														
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>1,000</u> bytes. The record structure of CBNCT is described in chapter 4.3.2.9. PCAP command <i>wrcbcnct()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>															

#### 4.4.48 rdcd, read common double

<b>DESCRIPTION:</b>	This function can be used to read in predefined variables of the CNC task. The variables concerned are the <i>rw_SymPas</i> variables CD0 .. CD99. The first parameter here specifies the number <i>-index-</i> of the variable you want to have read in. The value range of <i>index</i> here is 0 to 999. The second parameter is a pointer to a field with 1,000 double variables.
<b>BORLAND DELPHI:</b>	procedure rdcd(ndx: integer; var cdbuf:CDBUF);
<b>C:</b>	void rdcd(int ndx, struct CDBUF far *cdbuf);
<b>VISUAL BASIC:</b>	Sub rdcd(ByVal ndx As Long, CDBUF As CDBUF)
<b>RETURN VALUE:</b>	The <i>rdcd()</i> command enters the current value of the relevant <i>CD</i> variable in the field specified with <i>index</i> .
<b>NOTE:</b>	The content of all common variables remains stored in memory even after a system reset operation, executed by the <i>rs()</i> command, for example. If you do not want this, you should set the variables concerned to the value you want when starting the program. <b>Special note:</b> With Index 100, variables 0 to 99 are read together. The variable with Index 100 cannot be read with rdcd. With Index 1000, variables 0 to 999 are read together.



#### 4.4.49 rdci, read common integer

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>rdcd()</i> , 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 <i>rw_SymPas</i> variables CI0 .. CI999.
<b>BORLAND DELPHI:</b>	procedure rdci(ndx: integer; var cibuf:CIBUF);
<b>C:</b>	void rdci(int ndx, struct CIBUF far *cibuf);
<b>VISUAL BASIC:</b>	Sub rdci(ByVal ndx As Long, CIBUF As CIBUF)
<b>NOTE:</b>	<b>Special note:</b> With Index 100, variables 0 to 99 are read together. The variable with Index 100 cannot be read with rdci. With Index 1000, variables 0 to 999 are read together.

#### 4.4.50 rdcncts, read computerized numeric controller task status

<b>DESCRIPTION:</b>	This command can be used to interrogate the current status of the CNC task selected in <i>TaskNr</i> (values 0..3). After this command has been executed, the results can be found in the structure/record <i>CNCTS</i> .
<b>BORLAND DELPHI:</b>	procedure rdcncts(TaskNr:integer; var cncts:CNCTS):integer;
<b>C:</b>	void rdcncts(int TaskNr, struct CNCTS far *cncts);
<b>VISUAL BASIC:</b>	Sub rdcncts(ByVal TaskNr As Long, CNCTS As CNCTS)
<b>RETURN VALUE:</b>	The return values obtained in CNCTS after <i>rdcncts()</i> has been executed are described in chapter 4.3.2.10.

#### 4.4.51 rddigi, read digital inputs

<b>DESCRIPTION:</b>	This function you can be used to interrogate the following signal states: The current status of the 16 xPCI-800x digital inputs 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 <ul style="list-style-type: none"> <li>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 <i>mcfg.exe</i> 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.</li> </ul>
<b>BORLAND DELPHI:</b>	procedure rddigi(var tsrp:TSRP);
<b>C:</b>	void rddigi(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rddigi(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].digi n = 0 .. Number of axes -1
<b>RETURN VALUE:</b>	The bit-encoded return value is located in the <i>digi</i> structure or record component and is structured as described in the table printed below.
<b>NOTE:</b>	There is no specified axis assignment for the digital inputs. Bits 16 ... 19 can be reset by means of the <i>rdigi()</i> command [chapter 4.4.71]. (MCFG / Chapters 1.7.2.5 and 1.7.2.5.1).

4.4.51.1 Axis-qualifier digi

The register *digi* can be used to check the state of the xPCI-800x digital inputs. Active inputs have the value 1 at the concerned bit position.

**Table 13: Bit-coded structure of the digi word**

Bit No.	Function	X1/Pin APCI-8001 APCI-8008
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 <b>APCI-8001/APCI-8008:</b> hardware strobe signal for latching the actual position (axis channel 1)	47
14	Input 15 <b>APCI-8001/APCI-8008:</b> hardware strobe signal for latching the actual position (axis channel 2)	48
15	Input 16 <b>APCI-8001/APCI-8008:</b> 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	<b>APCI-8008/CPCI-8004:</b> AEA alarm error encoder channel A	--
21	<b>APCI-8008/CPCI-8004:</b> AEB alarm error encoder channel B	--
22	<b>APCI-8008/CPCI-8004:</b> AEN alarm error encoder channel Index	--
23	<b>APCI-8008/CPCI-8004:</b> AES alarm error encoder group error	--
24	<b>CPCI-8004:</b> Input 17	--
25	<b>CPCI-8004:</b> Input 18	--
26	<b>CPCI-8004:</b> Input 19	--
27	<b>CPCI-8004:</b> Input 20	--
28	<b>CPCI-8004:</b> Input 21 and hardware strobe signal for latching axis channel 1	--
29	<b>CPCI-8004:</b> Input 22 and hardware strobe signal for latching axis channel 2	--
30	<b>CPCI-8004:</b> Input 23 and hardware strobe signal for latching axis channel 3	--
31	<b>CPCI-8004:</b> Input 24 and hardware strobe signal for latching axis channel 4	--

#### 4.4.52 rddigib, read digital input bit

<b>DESCRIPTION:</b>	This function can be used to interrogate the current state of <u>one</u> xPCI-800x digital input and other logic signals. The axis number must be specified in the <i>an</i> parameter (0, 1, ... <i>MAXAXIS</i> ).
<b>BORLAND DELPHI:</b>	function rddigib(an:integer; bitnr:integer):integer;
<b>C:</b>	int rddigib(int an, int bitnr);
<b>VISUAL BASIC:</b>	Function rddigib(ByVal an As Long, ByVal bitnr As Long) As Long
<b>RETURN VALUE:</b>	The function returns the value 1 or TRUE, if the corresponding input of <i>bitnr</i> is active.
<b>NOTE:</b>	Bit numbers 17..20 can be reset via the <i>rdigi()</i> command [Chapter 4.4.71], (MCFG / Chapters 1.7.2.5 and 1.7.2.5.1) and PCAP command <i>rddigi()</i> <b>Caution:</b> The bit number counting begins at 1.

Table 14: Assignment of *bitnr* to the various xPCI-800x digital inputs

'bitnr'	Function	X1/Pin APCI-8001 APCI-8008
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), axis-specific	--
21	<b>APCI-8008/CPCI-8004:</b> AEA alarm error encoder channel A	--
22	<b>APCI-8008/CPCI-8004:</b> AEB alarm error encoder channel B	--
23	<b>APCI-8008/CPCI-8004:</b> AEN alarm error encoder channel Index	--
24	<b>APCI-8008/CPCI-8004:</b> AES alarm error encoder group error	--
25	<b>CPCI-8004:</b> Input 17	--
26	<b>CPCI-8004:</b> Input 18	--
27	<b>CPCI-8004:</b> Input 19	--
28	<b>CPCI-8004:</b> Input 20	--
29	<b>CPCI-8004:</b> Input 21	--
30	<b>CPCI-8004:</b> Input 22	--
31	<b>CPCI-8004:</b> Input 23	--
32	<b>CPCI-8004:</b> Input 24	--
21..32	The flags that are not assigned depending on the control type have an undefined value and are reserved for future use.	--

#### 4.4.53 rddigo, read digital outputs

<b>DESCRIPTION:</b>	This command is used to read the current output status of the xPCI-800x digital outputs into the axis-specific structure/record component <i>digo</i> . The bits set there represent outputs set.
<b>BORLAND DELPHI:</b>	procedure rddigo(var tsrp:TSRP);
<b>C:</b>	void rddigo(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	TSRP[n].digo
<b>TSRP COMPONENTS:</b>	Sub rddigo(DTSRP As TSRP)
<b>RETURN VALUE:</b>	After this command has been executed the bit-coded return values are located in the structure/record component <i>digo</i> . This component has the structure/record defined in the PCAP-command wrdigo().

#### 4.4.54 rddigob, read digital output bit

<b>DESCRIPTION:</b>	This function can be used to interrogate the current state of <u>one</u> xPCI-800x digital output. The axis number must be specified in parameter <i>an</i> (0, 1, ... MAXAXIS-1).
<b>BORLAND DELPHI:</b>	function rddigob(an:integer; bitnr:integer):integer;
<b>C:</b>	int rddigob(int an, int bitnr);
<b>VISUAL BASIC:</b>	Function rddigob(ByVal an As Long, ByVal bitnr As Long) As Long
<b>RETURN VALUE:</b>	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 wrdigob().

#### 4.4.55 rddp, read desired position

<b>DESCRIPTION:</b>	The xPCI-800x 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 4.4.93 - <i>rdp()</i> ]] and this setpoint position must be identical, apart from tolerable deviations.
<b>BORLAND DELPHI:</b>	procedure rddp(var tsrp:TSRP);
<b>C:</b>	void rddp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rddp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].dp
<b>RETURN VALUE:</b>	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.
<b>NOTE:</b>	This setpoint position is also utilized for setpoint/actual-differential formation, for the automatic position error monitoring function.

#### 4.4.56 rddpoffset, read desired position offset

<b>DESCRIPTION:</b>	With this function the currently programmed value of the axis qualifier dpoffset can be read.
<b>BORLAND DELPHI:</b>	function rddpoffset (an: integer; var value: double): integer;
<b>C:</b>	int rddpoffset(int an, double *value);
<b>VISUAL BASIC:</b>	Function rddpoffset (ByVal an As Long, value As Double) As Long
<b>PARAMETER:</b>	With <i>an</i> , the axis channel which has to be read out is indicated (0, 1, ...). In <i>value</i> , the position offset which has to be written is returned in the axis-specific position unit.
<b>RETURN VALUE:</b>	0 at success, unequal 0 at failure, if e.g. RWMOS.ELF does not support this function yet.
<b>NOTE</b>	See also chapter 4.4.129

#### 4.4.57 rddpd – read desired position in display unit

<b>DESCRIPTION:</b>	The xPCI-800x profile generator computes an internal reference variable, referred to as the "setpoint position" (= desired position). This can be read with this command in the axis-specific display unit.
<b>BORLAND DELPHI:</b>	procedure rddpd(var tsrp:TSRP);
<b>C:</b>	void rddpd(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rddpd(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].dp
<b>RETURN VALUE:</b>	None
<b>EFFECT:</b>	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.
<b>NOTE:</b>	See also commands rddp, rdrp, rdrpd

#### 4.4.58 rddv, read desired velocity

<b>DESCRIPTION:</b>	This function returns the axis-specific setpoint velocity of the xPCI-800x profile generator. In best case the value read in corresponds to the real axis velocity (actual velocity).
<b>BORLAND DELPHI:</b>	procedure rddv(var tsrp:TSRP);
<b>C:</b>	void rddv(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rddv(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].dv
<b>RETURN VALUE:</b>	After the command has been executed, the setpoint velocity is available in the <i>dv</i> register with the axis-specific velocity unit.
<b>NOTE:</b>	The setpoint velocity can only be influenced by corresponding traversing commands.

#### 4.4.59 rddvoffset, read desired velocity offset

<b>DESCRIPTION:</b>	With this function, the currently programmed value of the axis qualifier dvoffset can be read.
<b>BORLAND DELPHI:</b>	function rddvoffset (an: integer; var value: double): integer;
<b>C:</b>	int rddvoffset(int an, double *value);
<b>VISUAL BASIC:</b>	Function rddvoffset (ByVal an As Long, value As Double) As Long
<b>PARAMETER:</b>	With <i>an</i> , the axis channel which has to be read out is indicated (0, 1, ...). In <i>value</i> , the currently set velocity value is returned in the axis-specific position unit.
<b>RETURN VALUE:</b>	0 at success, unequal 0 at failure, if e.g. RWMOS.ELF does not support this function yet.

#### 4.4.60 rdEffRadius – Read Effective Radius

<b>DESCRIPTION:</b>	The effective radius can be read with this command for a rotatory axis (see chapter 0).
<b>BORLAND DELPHI:</b>	rdEffRadius (an: integer; var value: double);
<b>C:</b>	void rdEffRadius (long an, double *value);
<b>VISUAL BASIC:</b>	Sub rdEffRadius (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The axis number is given in <i>an</i> . <i>The effective radius is returned in value</i> in the unit defined through PU.
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	see chapter 6.3.3

#### 4.4.61 rdepc, read EEPROM programming cycle

<b>DESCRIPTION:</b>	This function can be used to read the instantaneous number of xPCI-800x EEPROM programming cycles. The cycle number is increased by one in the EEPROM for every save operation in the TOOLSET program <i>mcfg.exe</i> . The EEPROM can be written at least 10,000 times.
<b>BORLAND DELPHI:</b>	Procedure rdepc(var tsrp:TSRP);
<b>C:</b>	void rdepc(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdepc(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].epc
<b>RETURN VALUE:</b>	After this command has been executed, the current programming cycle number is in the structure/record component <i>epc</i> .

#### 4.4.62 rdErrorReg, read Error Register

<b>DESCRIPTION:</b>	This function can be used to read the Error Register for the RWMOS operating system software.
<b>BORLAND DELPHI:</b>	procedure rdErrorReg(var ErrorReg: integer);
<b>C:</b>	void rdErrorReg (long *ErrorReg);
<b>VISUAL BASIC:</b>	Sub rdErrorReg (ErrorReg As Integer)
<b>RETURN VALUE:</b>	The bit-coded value of the Error Register is returned in ErrorReg. The function has no return value.
<b>NOTE:</b>	For the layout of the Error Register, see next chapter.

4.4.62.1 Register ErrorReg

The *ErrorReg* register displays various error states of the RWMOS operating system software. The register is bit-coded.

Table 15: Bit-coded construction of the ErrorReg word

Bit No.	Name	Function	Hex
0	errAxDef	Axis in AS was selected more than once in a positioning command	1
1	errTargetVel	Target velocity $\neq 0$ at spooler end, although ForbidTargetVel set: System has been reset	2
2	errUnit	An invalid unit was used	4
3	errCenterPoint	Invalid center point programmed for circle or a circle with a radius = 0 has been programmed	8
4	errSpooler Overrun	Spooler overrun detected for an axis	10
5	ProfileTooSmall	In spooler operation, at least two traverse profiles whose execution time is shorter than the scan time are executed consecutively. This may cause errors in the program flow and is not allowed.	20
6	SplineSizeErr	Too many spline sets loaded	40
7	RotationFail	Error in axis rotation	80
8	PciBusError	Error detected in Interrupt Cause Register of PCI bridge	100
9	CheckMonitor Screen	Incorrect output to Monitor Screen generated	200
10	SsfWait Refused	At least one SSF wait command was ignored, because the target velocity of the previous traversing command did not equal 0. This suggests a programming error in the user software!	400
11	SpoolerLoad Error	Error while writing on the spooler, as at the same time, a positioning profile was generated by the system. This may happen if, for example, a limit switch switches during the call of an interpolation command.	800
12	VelocityZero	This bit indicates that an interpolation command with a traversing velocity = 0 was detected. Depending on the bit InhibitProfileRefuse (register MODEREG Chapter 6.3.1.4), the profile is automatically rejected. This suggests a programming error in the user software or a configuration problem of the user!	1000
13	AccelZero	This bit indicates that an interpolation command with an acceleration = 0 was detected. Depending on the bit InhibitProfileRefuse (register MODEREG Chapter 6.3.1.4), the profile is automatically rejected. This suggests a programming error in the user software or a configuration problem of the user!	2000
14	LimitDefError	An incorrect limit value has been detected in mcpmax, mcpmin, mcpcp or mcpcn (incorrect numerical value).	4000
15	ZeroProfile	An interpolation command has been rejected because the indicated traverse distance is almost or equal 0.	8000
16	RadiusError	A circle or helix command has been rejected because the circle radius to be implemented is almost or equal 0.	10000
14..31		Reserved for future use; these flags have an undefined value	

#### 4.4.63 rdf, read filter

<b>DESCRIPTION:</b>	This command can be used to read in the current axis-specific PIDF filter coefficients of the xPCI-800x board. The default values of these coefficients are specified using the TOOLSET program <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	Procedure <code>rdf(var tsrp:TSRP);</code>
<b>C:</b>	<code>void rdf(struct TSRP far *tsrp);</code>
<b>VISUAL BASIC:</b>	<code>Sub rdf(DTSRP As TSRP)</code>
<b>TSRP COMPONENTS:</b>	TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv n = 0 .. Number of axis present -1
<b>RETURN VALUE:</b>	After the command has been executed, the return values are in the TSRP structure/record components listed above.
<b>NOTE:</b>	You will find further details on the PIDF filter in chapter 2.1.1.1, OM / Chapter 4.1.1, CM / Chapter 6.2 and PCAP command <i>uf()</i>

#### 4.4.64 rdGCR, read gear configuration register

<b>DESCRIPTION:</b>	With this function, the axis-specific Gear Configuration Register can be read. [Chapter 6.3.3]
<b>BORLAND DELPHI:</b>	procedure <code>rdGCR (an: integer; var value: integer);</code>
<b>C:</b>	<code>void rdGCR (long an, long *value);</code>
<b>VISUAL BASIC:</b>	<code>Sub rdGCR (ByVal an As Long, value As Long)</code>
<b>PARAMETER:</b>	With <i>an</i> , the axis channel which has to be read out is indicated (0, 1, ...). In <i>value</i> , the contents of the GCR register is returned.
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	See also document on the resource interface - GEAR

#### 4.4.65 rdgfh, read gear factor

<b>DESCRIPTION:</b>	This function returns the axis-specific gear factor {gf}. The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	Procedure <code>rdgfh(var tsrp:TSRP);</code>
<b>C:</b>	<code>void rdgfh(struct TSRP far *tsrp);</code>
<b>VISUAL BASIC:</b>	<code>Sub rdgfh(DTSRP As TSRP)</code>
<b>TSRP COMPONENTS:</b>	TSRP[n].gf
<b>RETURN VALUE:</b>	After the command has been executed, the factor is available in the <i>gf</i> field with the axis-specific unit.
<b>NOTE:</b>	The gear factor can be set at any time with the PCAP command <i>wrgfh()</i> .

#### 4.4.66 rdgfaux, read gear factor auxiliary channel

<b>DESCRIPTION:</b>	This function returns the axis-specific ratio of stepper motor resolution to encoder channel in stepper systems with encoder verification. The default value is 1.0; the value can only be changed at runtime.
<b>BORLAND DELPHI:</b>	function <code>rdgfaux (an: integer; var value: double) : integer;</code>
<b>C:</b>	<code>int rdgfaux(int an, double *value)</code>
<b>VISUAL BASIC:</b>	Function <code>rdgfaux (ByVal an As Long, value As Double) As Long</code>
<b>RETURN VALUE:</b>	After successful execution, the function returns 0. In this case, the axis-specific value of <i>gfaux</i> is available in <i>value</i> . With a return value $\neq 0$ , the value could not be read, because e.g. RWMOS.ELF does not support the command.
<b>NOTE:</b>	The factor can be set at any time with the PCAP command <i>wrgfaux()</i> . See also Chapter 6.3.3



#### 4.4.67 rdhac, read home acceleration

<b>DESCRIPTION:</b>	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> .
<b>BORLAND DELPHI:</b>	Procedure rdhac(var tsrp:TSRP);
<b>C:</b>	void rdhac(struct Tsrp far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdhac(DTSRP As Tsrp)
<b>TSRP COMPONENTS:</b>	TSRP[n].hac
<b>RETURN VALUE:</b>	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.
<b>NOTE:</b>	The reference travel acceleration can be set at any time with the PCAP command <i>wrhac()</i> .

#### 4.4.68 rdhvl, read home velocity

<b>DESCRIPTION:</b>	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> .
<b>BORLAND DELPHI:</b>	procedure rdhvl(var tsrp:TSRP);
<b>C:</b>	void rdhvl(struct Tsrp far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdhvl(DTSRP As Tsrp)
<b>TSRP COMPONENTS:</b>	TSRP[n].hvl
<b>RETURN VALUE:</b>	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.
<b>NOTE:</b>	The reference travel velocity can be set at any time with the PCAP command <i>wrhvl()</i> .

#### 4.4.69 rdifs, read interface status

<b>DESCRIPTION:</b>	This command can be used to read in status information of the xPCI-800x.
<b>BORLAND DELPHI:</b>	procedure rdifs(var tsrp:TSRP);
<b>C:</b>	void rdifs(struct Tsrp far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdifs(DTSRP As Tsrp)
<b>TSRP COMPONENTS:</b>	TSRP[n].ifs
<b>RETURN VALUE:</b>	The bit-coded return value is located in the structure/record component <i>ifs</i> and has the structure described in the table below.

##### 4.4.69.1 Axis qualifier ifs

This register can be used to interrogate various pieces of status information for the xPCI-800x. If the status or error information concerned is valid, this is indicated by the value 1 at the bit position involved. The bits represent important internal status information for the APCI-8001. Possible causes of errors can be problems at the voltage supply, EMC or hardware problems and should not actually occur. In case such an error occurs the controlling internal I/O interface is reset. A normal working process is then ensured once the controller is booted anew.

The status information must be controlled cyclically by an application program.

**Table 16: 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.
1	<i>cncrdy</i> : The CNC ready to operate relay is active (closed).
16	<i>pfe</i> : The Power Fail Error flag is set to "1" whenever the operating voltage at the xPCI-800x falls below a threshold voltage of 2.85V. 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 xPCI-800x 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_MOS</i> operating system software.
19	<i>scwdog</i> : The watchdog flag is set to „1“, if the watchdog safety logic (Secondary circle) has tripped the xPCI-800x.
20	<i>scpfe</i> : The Power Fail Error flag is always set to „1“, when the operating voltage at the xPCI-800x falls below a threshold of 4.75V. After the module is switched on, the flag is likewise set to „1“.
21..31	Not assigned, these flags always have the value 0

**Note:** In an initialization routine of the *rw\_MOS* firmware, error flags 16 ... 20 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 (*BootFile*). The flags can also be reset by the *rifs()* command [chapter 4.4.69].

#### 4.4.70 rdifsb, read interface status bit

<b>DESCRIPTION:</b>	This function can be used to interrogate <u>one</u> piece of APCI-8001 interface status information. The axis number must be specified in the <i>an</i> parameter (0, 1, ... <i>MAXAXIS-1</i> )
<b>BORLAND DELPHI:</b>	function rdifsb(an:integer; bitnr:integer):integer;
<b>C:</b>	int rdifsb(int an, int bitnr);
<b>VISUAL BASIC:</b>	Function rdifsb(ByVal an As Long, ByVal bitnr As Long) As Long
<b>RETURN VALUE:</b>	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 16, 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!
<b>NOTE:</b>	See also PCAP command <i>rdifs()</i>

#### 4.4.71 rdigi, reset digital inputs

<b>DESCRIPTION:</b>	This function can be used to clear axis-specific status information(s) filed in <i>digi</i> .
<b>BORLAND DELPHI:</b>	procedure rdigi(var tsrp:TSRP);
<b>C:</b>	void rdigi(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdigi(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].digi n = 0 .. number of axes -1
<b>NOTE:</b>	<i>rddigi()</i> [Chapter 4.4.51]

#### 4.4.72 rdipw, read in position window

<b>DESCRIPTION:</b>	This function returns the axis-specific In-Position Window.
<b>BORLAND DELPHI:</b>	procedure rdipw(var tsrp:TSRP);
<b>C:</b>	void rdipw(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdipw(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].ipw
<b>NOTE:</b>	After the command has been executed, the In-Position Window is available in the <i>ipw</i> register in the axis-specific position unit. PCAP command <i>wripw()</i>

#### 4.4.73 rdirqpc, read interrupt request PC

<b>DESCRIPTION:</b>	This command can be used to interrogate the instantaneous status of the interrupt source generated on the xPCI-800x board. If the interrupt is active, the function returns the value 1, otherwise the value 0.
<b>BORLAND DELPHI:</b>	function rdirqpc: integer;
<b>C:</b>	int rdirqpc(void);
<b>VISUAL BASIC:</b>	Function rdirqpc() As Long
<b>NOTE:</b>	The interrupt can be set or reset by the system variable <i>IRQPC</i> using an SAP program [chapter 6.3.1.1 - PC interrupt generation].

#### 4.4.74 rdjac, read jog acceleration

<b>DESCRIPTION:</b>	This command can be used to read in the axis-specific <i>jog</i> acceleration <i>jac</i> . The default value is specified using the TOOLSET program <i>mcfq.exe</i> .
<b>BORLAND DELPHI:</b>	procedure rdjac(var tsrp:TSRP);
<b>C:</b>	void rdjac(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdjac(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].jac
<b>RETURN VALUE:</b>	After the command has been executed, the <i>jog</i> acceleration is available in the <i>jac</i> field. The value is returned in the axis-specific acceleration unit.
<b>NOTE:</b>	The <i>jog</i> acceleration can be set at any time with the PCAP command <i>wrjac()</i> .

#### 4.4.75 rdJerkRel, read jerkrel

<b>DESCRIPTION:</b>	This command can be used to read in the axis-specific paramter <i>jerkrel</i> in <i>value</i> .
<b>BORLAND DELPHI:</b>	procedure rdJerkRel (an: integer; var value: double);
<b>C:</b>	void rdJerkRel (long an, double *value);
<b>VISUAL BASIC:</b>	Sub rdJerkRel (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	an = axis number (0..n) Double = free variable for function value
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	<i>jerkrel</i> has always a value from 0..1. See also chapter 6.3.3.

##### 4.4.75.1 Axis qualifier *jerkrel*

This variable can parameterise the acceleration characteristics for S-form speed profiles (jerk limitation). This factor is only effective when an S profile is selected (see register MODEREG chapter 6.3.1.4) and has the following meaning:

The acceleration defined for S profiles is constantly the medium acceleration above the whole acceleration/deceleration process. The maximum acceleration in the acceleration/braking ramp is calculated as follows:

$$a_{\max} = a * (1 + \text{jerkrel})$$

The value of *jerkrel* has the following consequence on the acceleration course.

0 = rectangular acceleration course

1 = triangular acceleration course

inbetween = trapezoidal acceleration course

**Example:** The value 0.2 is allocated to *jerkrel*.

- The acceleration has now a trapezoidal course for all profiles.
- The maximum acceleration in the middle of the trapez is 1.2 times faster as the set acceleration.

The medium acceleration above the whole acceleration/deceleration process is the programmed acceleration (jac at JOG commands or trac at MOVE commands).

Values between 0 and 1 are possible for *jerkrel*. The default value is 1. Values out of the range 0..1 are limited either to 0 or 1.

#### 4.4.76 rdjtvI, read jog target velocity

<b>DESCRIPTION:</b>	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> .
<b>BORLAND DELPHI:</b>	procedure rdjtvI(var tsrp:TSRP);
<b>C:</b>	void rdjtvI(struct TSPR far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdjtvI(DTSRP As TSPR)
<b>TSPR COMPONENTS:</b>	TSPR[n].jtvI
<b>RETURN VALUE:</b>	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.
<b>NOTE:</b>	The jog target velocity can be set at any time using the PCAP command <i>wrjtvI()</i> .

#### 4.4.77 rdjvI, read jog velocity

<b>DESCRIPTION:</b>	This command can be used to read in the axis-specific <i>jog</i> velocity <i>jvI</i> . The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	Procedure rdjvI(var tsrp:TSRP);
<b>C:</b>	void rdjvI(struct TSPR far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdjvI(DTSRP As TSPR)
<b>TSPR COMPONENTS:</b>	TSPR[n].jvI
<b>RETURN VALUE:</b>	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.
<b>NOTE:</b>	The jog velocity can also be set at any time using the PCAP command <i>wrjvI()</i> .

#### 4.4.78 rdledgn, read led green

<b>DESCRIPTION:</b>	
<b>APCI-8001:</b>	This function can be used to read in the current state of LED D29 (green).
<b>APCI-8008:</b>	This function can be used to read in the current state of LED D53 (green).
<b>CPCI-8004:</b>	This function can be used to read in the current state of LED D36 (green).
<b>BORLAND DELPHI:</b>	function rdledgn: integer;
<b>C:</b>	int rdledgn(void);
<b>VISUAL BASIC:</b>	Function rdledgn() As Long
<b>RETURN VALUE:</b>	The function's return value is 1, provided the LED is switched on, otherwise it is 0.
<b>NOTE:</b>	PCAP command <i>wrledgn()</i> , system variable <i>LEDGN</i>

#### 4.4.79 rdledrd, read led red

<b>DESCRIPTION:</b>	
<b>APCI-8001:</b>	This function can be used to read in the current state of LED D31 (red).
<b>APCI-8008:</b>	This function can be used to read in the current state of LED D56 (red).
<b>CPCI-8004:</b>	This function can be used to read in the current state of LED D38 (red).
<b>BORLAND DELPHI:</b>	function rdledrd: integer;
<b>C:</b>	int rdledrd(void);
<b>VISUAL BASIC:</b>	Function rdledrd() As Long
<b>NOTE:</b>	PCAP command <i>wrledrd()</i> , system variable <i>LEDRD</i>

#### 4.4.80 rdledyl, read led yellow

<b>DESCRIPTION:</b>	
<b>APCI-8001:</b>	This function can be used to read in the current state of LED D30 (yellow).
<b>APCI-8008:</b>	This function can be used to read in the current state of LED D55 (yellow).
<b>CPCI-8004:</b>	This function can be used to read in the current state of LED D37 (yellow).
<b>BORLAND DELPHI:</b>	function rdledyl: integer;
<b>C:</b>	int rdledyl(void);
<b>VISUAL BASIC:</b>	Function rdledyl() As Long
<b>NOTE:</b>	PCAP command <i>wrledyl()</i> , system variable <i>LEDYL</i>

#### 4.4.81 rdlp, read latched position

<b>DESCRIPTION:</b>	<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"> <li>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.</li> <li>2. If an <i>lps()</i> PCAP command [chapter 4.4.25] has previously been executed and the time delay specified there in the <i>mst</i> parameter has elapsed.</li> <li>3. In real time (max. 1 <math>\mu</math>s time delay) by means of default-setting xPCI-800x digital inputs.</li> </ol> <p>A new latching procedure will only be enabled after the latching input has been de-activated.</p> <p>In all these methods, the actual position {<i>rp</i>} of the motor axis is put into intermediate storage.</p> <p>In stepper motor systems or analog feedback with encoder verification, also the auxiliary channel AUX can be latched if the option "Use Encoder for position feedback" is activated (mcfg system data).</p>
<b>BORLAND DELPHI:</b>	procedure rdlp(var tsrp:TSRP);
<b>C:</b>	void rdlp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdlp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].lp
<b>RETURN VALUE:</b>	<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. real-time latching has top priority</p>
<b>NOTE:</b>	PCAP command <i>wrlp()</i>

#### 4.4.82 rdlpndx, read latched position index

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure rdlpndx(var tsrp:TSRP);
<b>C:</b>	void rdlpndx(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdlpndx(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].lp
<b>RETURN VALUE:</b>	After the function has been executed, the latch position is available in the <i>lp</i> register in the axis-specific position unit.
<b>NOTE:</b>	Latching of the incremental coder's zero track is helpful in the coder verification routine and for reference travel programming. PCAP command <i>wrlpndx()</i>

#### 4.4.83 rdlsn, read left spool memory

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure rdlsn(var tsrp:TSRP);
<b>C:</b>	void rdlsn(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdlsn(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].lsn
<b>NOTE:</b>	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. The required spool memory per command can be modified by the future operating system versions. It should not be used to determine the traverse profiles present in the spooler.

#### 4.4.84 rdMaxAcc – Read Maximum Acceleration Check

<b>DESCRIPTION:</b>	With this command the maximum axis-specific acceleration value ( <i>MaxAcc</i> ) can be read. This value can be used by RWMOS operating system software in order to limit the trajectory acceleration so that no axis involved in a linear interpolation exceeds the maximum acceleration accepted. If required, the trajectory acceleration can be reduced.
<b>BORLAND DELPHI:</b>	rdMaxAcc (an: integer; var value: double);
<b>C:</b>	void rdMaxAcc (long an, double *value);
<b>VISUAL BASIC:</b>	Sub rdMaxAcc (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axes is entered in <i>an</i> the maximum acceleration accepted is returned in <i>value</i> .
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	See chapter 4.4.150 and 6.3.3

#### 4.4.85 rdMaxVel – Read Maximum Velocity Check

<b>DESCRIPTION:</b>	With this command the maximum axis-specific velocity value ( <i>MaxVel</i> ) can be read for linear interpolation commands. The value can be used by RWMOS operating system software in order to limit the trajectory velocity so that no axis involved in a linear interpolation exceeds the maximum velocity accepted. If required the trajectory velocity can be reduced.
<b>BORLAND DELPHI:</b>	rdMaxVel (an: integer; var value: double);
<b>C:</b>	void rdMaxVel (long an, double *value);
<b>VISUAL BASIC:</b>	Sub rdMaxVel (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axes is entered in <i>an</i> the maximum velocity accepted is returned in <i>value</i> .
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	See chapter 4.4.151 and 6.3.3

#### 4.4.86 rdMCiS – Read Move Commands in Spooler

<b>DESCRIPTION:</b>	With this function the number of motion commands in the spooler of an axis.
<b>BORLAND DELPHI:</b>	procedure rdMCiS (an: integer; var value: integer);
<b>C:</b>	void rdMCiS (long an, long *value);
<b>VISUAL BASIC:</b>	Sub rdMCiS (an As Long, ByVal value As Long)
<b>PARAMETER:</b>	an = axis number
<b>RETURN VALUE:</b>	The number of motion commands in spooler of the corresponding axis is returned in value.
<b>COMMENT:</b>	This functionality is only available in versions from May 2002 and later.
<b>NOTE:</b>	This commands gives the current process state in Spooler.

#### 4.4.87 rdmcp, read motor command port

<b>DESCRIPTION:</b>	This command can be used to read in the current command values of the Motor-Command-Ports.
<b>BORLAND DELPHI:</b>	procedure rdmcp(var tsrp:TSRP);
<b>C:</b>	void rdmcp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdmcp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].mcp
<b>RETURN VALUE:</b>	<p>The return value is available in the <i>mcp</i> field after the command has been executed.</p> <p>In the case of servo axes, a value in the range -32,767 .. 32,767 is returned. This corresponds to a setpoint output voltage of approx. -10V .. +10V.</p> <p>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:</p> $tver = (mcp+1) * 2 / CLOCK;$ <p><i>Example: with mcp = 12,499 and CLOCK = 70MHz</i></p> $tver = 0.333ms \text{ and } f = 3kHz$ <p>Each time the time-delay <i>tver</i> elapses, the pulse signal is switched over, i.e. after <math>2*tver</math> a stepping signal with <math>f = 1 / (2*tver)</math> [Hz] is outputted. The value returned in <i>mcp</i> lies within the range of -1,048,575 .. +1,048,575.</p>

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 output, i.e. the motor is at a standstill.

#### 4.4.88 rdMDVel – Read Maximum Velocity Skip

<b>DESCRIPTION:</b>	The maximum axis-specific velocity jump ( <i>MDVEL</i> ) can be read with this command. The value is used by the Look-Ahead functionality of the operating system software RWMOS to reduce the trajectory velocity so that no axis involved in an interpolation exceeds the maximum velocity jump accepted.
<b>BORLAND DELPHI:</b>	rdMDVel (an: integer; var value: double);
<b>C:</b>	void rdMDVel (long an, double *value);
<b>VISUAL BASIC:</b>	Sub rdMDVel (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axis is given in <i>an</i> , the maximum velocity jump accepted of the axis-specific velocity unit is returned in <i>value</i> .
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	See Chapter 4.4.153 and 6.3.3

#### 4.4.89 rdModeReg – Read MODEREG

<b>DESCRIPTION:</b>	With this command the register MODEREG of the operating system software RWMOS can be read.
<b>BORLAND DELPHI:</b>	rdModeReg (var value: integer);
<b>C:</b>	void rdModeReg(long *value);
<b>VISUAL BASIC:</b>	rdModeReg (ByVal value As Long)
<b>PARAMETER:</b>	The ModeReg is returned in value.
<b>NOTE:</b>	See also chapter 6.3.1.4 and function wrModeReg chapter 4.4.154.

#### 4.4.90 rdmpe, read maximum position error

<b>DESCRIPTION:</b>	This function returns the axis-specific position error limit value.
<b>BORLAND DELPHI:</b>	procedure rdmpe(var tsrp:TSRP);
<b>C:</b>	void rdmpe(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdmpe(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].mpe
<b>NOTE:</b>	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>

#### 4.4.91 rdnfrax – read No-Feed-Rate-Axis

<b>DESCRIPTION:</b>	With this command the register NFRAX of the RWMOS operating system software is read.
<b>BORLAND DELPHI:</b>	Rdnfrax (var value: integer);
<b>C:</b>	void rdnfrax (long *value);
<b>VISUAL BASIC:</b>	Sub rdnfrax (ByVal value As Long)
<b>PARAMETER:</b>	In value NFRAX is returned
<b>NOTE:</b>	See also chapter 6.3.1 and function wrnfrax chapter 4.4.156



#### 4.4.92 rdPosErr, read Position Error

<b>DESCRIPTION:</b>	This function returns the axis-specific position error of the xPCI-800x controller actual value channel.
<b>BORLAND DELPHI:</b>	procedure rdPosErr (var an: integer; var value: double);
<b>C:</b>	void rdPosErr (long an, double *value)
<b>VISUAL BASIC:</b>	Sub rdPosErr (an As Long, value As Double)
<b>PARAMETER:</b>	<i>an</i> = Number of axes (0..n) <i>value</i> = read position error
<b>RETURN VALUE:</b>	After this function has been executed the position error of the axis <i>an</i> is available in the variables <i>value</i> in the axis-specific position unit.
<b>NOTE:</b>	The position error cannot be written in [Chapter 6.3.3]

#### 4.4.93 rdrp, read real position

<b>DESCRIPTION:</b>	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).
<b>BORLAND DELPHI:</b>	procedure rdrp (var tsrp:TSRP);
<b>C:</b>	void rdrp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdrp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].rp
<b>NOTE:</b>	After the function has been executed, the current position is available in the <i>rp</i> register in the axis-specific position unit

#### 4.4.94 rdrpd – read real position in display unit

<b>DESCRIPTION:</b>	The function returns the current axis-specific position (= real position) in axis-specific display unit. The position can be read at any time also during the running of an axis. Per scan cycle (1.28ms) a new actual value is available.
<b>BORLAND DELPHI:</b>	procedure rdrpd(var tsrp:TSRP);
<b>C:</b>	void rdrpd(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdrpd(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].rp
<b>RETURN VALUE:</b>	None
<b>EFFECT:</b>	After the execution of the function the real position is available in the field <i>rp</i> in the axis-specific display unit.
<b>NOTE:</b>	See also the commands rddp, rdrp, rddpd

#### 4.4.95 rdrv, read real velocity

<b>DESCRIPTION:</b>	The function returns the axis-specific actual velocity of the xPCI-800x controller actual value channel.
<b>BORLAND DELPHI:</b>	Procedure rdrv (var an: integer; var value: double);
<b>C:</b>	void rddv (int *an, double *value);
<b>VISUAL BASIC:</b>	Sub rddv (an As Long, value As Double)
<b>PARAMETER:</b>	<i>an</i> = Number of axes (0..n) <i>value</i> = Read velocity value
<b>RETURN VALUE:</b>	After the execution of the function the actual velocity is available in the variable <i>value</i> in the axis-specific velocity unit.
<b>NOTE:</b>	The actual velocity cannot be written in, yet can be read even if the control loop is open.

#### 4.4.96 rdSampleTime – Read Sample Time

<b>DESCRIPTION:</b>	This command can be used to determine the sample time of the control.
<b>BORLAND DELPHI:</b>	function rdSampleTime (var value: integer) as integer;
<b>C:</b>	void rdSampleTime(long *value);
<b>VISUAL BASIC:</b>	Function rdSampleTime (ByVal value As Long) As Long
<b>RETURN VALUE:</b>	1 for success, 0 for failure, where e.g. RWMOS.ELF does not yet support this function
<b>PARAMETERS:</b>	Sample time returned in $\mu$ s
<b>NOTE:</b>	The sample time is displayed as a whole number of $\mu$ s. The default value is 1280.

#### 4.4.97 rdsdec, read stop deceleration

<b>DESCRIPTION:</b>	This command returns the axis-specific stop deceleration <i>sdec</i> . The default value is specified using the TOOLSET program <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	procedure rdsdec(var tsrp:TSRP);
<b>C:</b>	void rdsdec(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdsdec(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].sdec
<b>RETURN VALUE:</b>	After the command has been executed the stop deceleration is available in the <i>sdec</i> field. The value is returned in the axis-specific acceleration unit.
<b>NOTE:</b>	The stop deceleration can be set at any time using the PCAP-command <i>wrsdec()</i> .

#### 4.4.98 rdsll, read software limit left

<b>DESCRIPTION:</b>	This function returns the axis-specific left software limit position.
<b>BORLAND DELPHI:</b>	procedure rdsll(var tsrp:TSRP);
<b>C:</b>	void rdsll(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	TSRP[n].sll
<b>TSRP COMPONENTS:</b>	Sub rdsll(DTSRP As TSRP)
<b>NOTE:</b>	After the function has been executed, the left software limit position is available in the <i>sll</i> register in the axis-specific position unit. PCAP command <i>wrsll()</i>

#### 4.4.99 rdslr, read software limit right

<b>DESCRIPTION:</b>	This function returns the axis-specific right software limit position.
<b>BORLAND DELPHI:</b>	procedure rdslr(var tsrp:TSRP);
<b>C:</b>	void rdslr(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdslr(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].slr
<b>NOTE:</b>	After the function has been executed, the right software limit position is available in the <i>slr</i> register in the axis-specific position unit. PCAP command <i>wrslr()</i>

#### 4.4.100 rdsfsp, read Slits / Stepperpulses

<b>DESCRIPTION:</b>	This function determines the axis-specific resolution per motor turn {slsp} at encoder or stepper motor system. The default value is determined with the TOOLSET program <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	function rdsfsp (an: integer; var value: double): integer;
<b>C:</b>	int rdsfsp (long an, double *value);
<b>VISUAL BASIC:</b>	Function rdsfsp (ByVal an As Long, value As Double) As Long

<b>TSRP-COMPONENTS:</b>	None
<b>RETRUN VALUE:</b>	1 when succesful, 0 when not successful, e.g.if the function RWMOS.ELF is not yet supported. After successful execution of the function, the factor slsp in value is available in units, which were selected in mcfg.
<b>NOTE:</b>	slsp can be set with the PCAP command <i>wrs/sp()</i> . See also axis qualifier slsp.

#### 4.4.101 rdtb, read target position

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	Procedure rdtb(var tsrp:TSRP);
<b>C:</b>	void rdtb(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdtb(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].tp
<b>NOTE:</b>	After the function has been executed, the target position of the last traversing command is available in the <i>tp</i> register in the axis-specific position unit. This command is used for monitoring purposes only.

#### 4.4.102 rdtbd – read target position in display unit

<b>DESCRIPTION:</b>	This function can be used to interrogate the target position) in the axis-specific display unit. The target position is always returned as an absolute position value.
<b>BORLAND DELPHI:</b>	Procedure rdtbd(var tsrp:TSRP);
<b>C:</b>	void rdtbd(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rdtbd(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].tp
<b>RETURN VALUE:</b>	none
<b>EFFECT:</b>	After the function has been executed, the target position of the last traversing command is available in the <i>tp</i> register in the axis-specific position unit. This command is used for monitoring purposes only.
<b>NOTE:</b>	See also commands rdtb, rddp, rdrp, rdrpd, rddpd

#### 4.4.103 rdtrovr, read trajectory override

<b>DESCRIPTION:</b>	This command reads a state variable of the currently set trajectory velocity correction value, which is taken into account for all interpolation commands ( <i>move</i> commands) and the correspondingly selected axes (PCAP command <i>utrovr()</i> ).
<b>BORLAND DELPHI:</b>	procedure rdtrovr(var value:double);
<b>C:</b>	void rdtrovr(double *value);
<b>VISUAL BASIC:</b>	Sub rdtrovr(value As Double)
<b>RETURN VALUE:</b>	After the command has been executed, the trajectory velocity correction value will be in the <i>value</i> variable.
<b>NOTE:</b>	PCAP commands <i>utrovr()</i> , <i>wrtrovr()</i> , <i>wrjovr()</i> , <i>rdtrovr()</i> and <i>rdjovr()</i>

#### 4.4.104 rdtrovrst, read trajectory override settling time

<b>DESCRIPTION:</b>	With this command the programmed override-settling-time (see wrtrovrst chapter 4.4.162) can be read out.
<b>BORLAND DELPHI:</b>	function rdtrovr(var value:double) : integer;
<b>C:</b>	int rdtrovr(double *value);
<b>VISUAL BASIC:</b>	Function rdtrovr(value As Double) as long

<b>RETURN VALUE:</b>	0 for success -1: command is not available in RWMOS version -4: time-out, reason unknown The set override settling time is returned in <i>value</i> .
<b>NOTE:</b>	PCAP command wrtrovrst

#### 4.4.105 rdzeroOffset, read zero offset

<b>DESCRIPTION:</b>	With this command the currently set axis specific zero offset can be read. The absolute value of the currently set axis specific zero offset is returned in <i>value</i> in the axis specific position unit. With the parameter <i>an</i> the axis index of the axis channel to be read (0..n) is indicated.
<b>BORLAND DELPHI:</b>	Function rdZeroOffset (an: integer; var value: double) : integer;
<b>C:</b>	Int rdZeroOffset (integer an, double *value);
<b>VISUAL BASIC:</b>	Function rdZeroOffset (ByVal an As Long, value As Double) As Long
<b>RETURN VALUE:</b>	<0: command is not supported by RWMOS.ELF =0: command executed successfully >0: time exceeded, command not executed
<b>NOTE:</b>	The zero offset can be set for example with the PCAP commands szpa (see chapter 4.4.118) or szpr (see chapter 4.4.119).

#### 4.4.106 rifs, reset interface status register

<b>DESCRIPTION:</b>	This command causes various error flags in the xPCI-800x interface status register <i>ifs</i> (error bits 16, 17, 18 - <i>pfe</i> , <i>wdog</i> and <i>iae</i> ) to be reset. Resetting should be performed only in exceptional situations, e.g. in an error monitoring routine.
<b>BORLAND DELPHI:</b>	procedure rifs(var tsrp:TSRP);
<b>C:</b>	void rifs(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub rifs(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].ifs
<b>NOTE:</b>	[chapter 4.4.69.1 - <i>rdifs()</i> ]

#### 4.4.107 RPtoDP, Real-Position to Desired-Position

<b>DESCRIPTION:</b>	This command enables the setpoint position {dp} of an axis to adopt the current position {rp}. The command is executed without delay. It only takes effect though if the relevant axes are not in a positioning profile, as otherwise the setpoint position will be immediately replaced by the computed value of the profile generation. However, it is possible to correct an axis traversing with a target velocity $\neq 0$ . The relevant axes serve as parameters.
<b>BORLAND DELPHI:</b>	procedure RPtoDP(var as: AS);
<b>C:</b>	void RPtoDP (struct AS far *as);
<b>VISUAL BASIC:</b>	Sub RPtoDP (DASEL As ASEL)
<b>RETURN VALUE:</b>	none
<b>NOTE:</b>	This command can be used when one or more axes are no longer regulated due to a position error caused, for example, by axis blocking. Once the error has been cleared, regulation can be continued from the previous position, even with traversing axes. This command is available from RWMOS.ELF V2.5.3.100 and mcug3.dll V2.5.3.80.

#### 4.4.108 rs, reset system

<b>DESCRIPTION:</b>	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 <i>mcfg.exe</i> . 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 <i>wrjovr()</i> and <i>wrtrov()</i> ) are set to the value 1.0.
<b>BORLAND DELPHI:</b>	procedure rs;
<b>C:</b>	void rs(void);
<b>VISUAL BASIC:</b>	Sub rs()
<b>NOTE:</b>	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 <i>ifs</i> are not influenced by this command. The contents of all common integers and double variables are retained

#### 4.4.109 scp – set controller params

<b>DESCRIPTION:</b>	This function is used for customised extensions and serves for transferring a parameter field of 15 x 15 floating points (predefined data structure CTRLPARAMS) axis-specifically to the control process.
<b>BORLAND DELPHI:</b>	procedure scp (an: integer; var ctrlparams: CTRLPARAMS);
<b>C:</b>	void scp (long an, struct CTRLPARAMS *ctrlparams);
<b>VISUAL BASIC:</b>	Sub scp (ByVal an As Long, DCTRLPARAMS As CTRLPARAMS)
<b>RETURN VALUE:</b>	None
<b>EFFECT:</b>	The values in CTRLPARAMS are transferred to the control system.
<b>NOTE:</b>	Use and significance of the data to be transferred are described according to the application.

#### 4.4.110 sdels, spooler delete synchronous

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure sdels(var as:AS);
<b>C:</b>	Void sdels(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub sdels(DASEL As ASEL)
<b>NOTE:</b>	The ongoing operation, like a traversing command, will be concluded.

#### 4.4.111 shp, set home position

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure shp(var tsrp:TSRP);
<b>C:</b>	Void shp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub shp(DTSRP As TSRP)

<b>TSRP COMPONENTS:</b>	TSRP[n].tp n = 0 .. Number of axes present-1
<b>NOTE:</b>	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.

#### 4.4.112 ssms, start spooled motions synchronous

<b>DESCRIPTION:</b>	<i>Spool</i> commands can be used to transfer commands to the individual axis channels of the xPCI-800x: 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.
<b>BORLAND DELPHI:</b>	procedure ssms(var as:AS);
<b>C:</b>	void ssms(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub ssms(DASEL As ASEL)
<b>NOTE:</b>	chapter 2.2.8.2 - Spool-Mode

#### 4.4.113 sstps, spooler stop synchronous

<b>DESCRIPTION:</b>	This command is used to interrupt command processing from the spooler for all axis channels selected in AS.
<b>BORLAND DELPHI:</b>	procedure sstps(var as:AS);
<b>C:</b>	void sstps(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub sstps(DASEL As ASEL)
<b>NOTE:</b>	The current command is completely processed. The commands that are in the spooler are preserved and can be continued with SSMS. But please pay attention if the spooler contains traverse commands with target velocities $\neq 0$ . In error situations, when the spooler contents are to be rejected, it is better to use direct commands to stop the axes concerned (ms or js), as these commands discontinue the current contour and at the same time reject the spooler contents.

#### 4.4.114 ssf, Spool-Special-Function

<b>DESCRIPTION:</b>	This commands allows to enter other commands as traverse commands in the spooler. The command you want to execute is entered with the parameter <i>command</i> .										
<b>BORLAND DELPHI:</b>	procedure ssf(an: integer; command: integer; value:double); far; stdcall;										
<b>C:</b>	void ssf(int axis, int command, double value);										
<b>VISUAL BASIC:</b>	Sub ssf(ByVal an As Long, ByVal command As Long, ByVal value As Double)										
<b>CALLING PARAMETER:</b>	The value <i>value</i> is entered to the axis defined in <i>axis</i> . The following commands are available at the moment: <table border="1" data-bbox="507 1765 1385 2020"> <thead> <tr> <th>Command</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0 .. 999</td><td>Describe CI-Variable with <i>Value</i>.</td></tr> <tr> <td>1000</td><td>Stop the spooler processing, this command is only carried out when the target velocity of the profile last entered is 0.</td></tr> <tr> <td>1001</td><td>Set digital outputs, the outputs to be set are specified bitwise in <i>Value</i>.</td></tr> <tr> <td>1002</td><td>Reset digital outputs, the outputs to be reset are specified bitwise in <i>Value</i>.</td></tr> </tbody> </table>	Command	Description	0 .. 999	Describe CI-Variable with <i>Value</i> .	1000	Stop the spooler processing, this command is only carried out when the target velocity of the profile last entered is 0.	1001	Set digital outputs, the outputs to be set are specified bitwise in <i>Value</i> .	1002	Reset digital outputs, the outputs to be reset are specified bitwise in <i>Value</i> .
Command	Description										
0 .. 999	Describe CI-Variable with <i>Value</i> .										
1000	Stop the spooler processing, this command is only carried out when the target velocity of the profile last entered is 0.										
1001	Set digital outputs, the outputs to be set are specified bitwise in <i>Value</i> .										
1002	Reset digital outputs, the outputs to be reset are specified bitwise in <i>Value</i> .										

1003	Stop the spooler processing during the time defined in <i>Value</i> . The time unit is 64 $\mu$ s. The real waiting time is multiplied several times by the scan time. This command is only carried out if the target velocity of the profile last entered is 0. The waiting process can be prematurely ended e.g. with the command SSMS.
1004	Stop the spooler processing until the inputs specified in <i>Value</i> are active. The inputs are specified in bit-coded form. This command is only carried out if the target velocity of the profile last entered is 0. The waiting process can be prematurely ended e.g. with the command SSMS. Only inputs of the respective axis group can be specified at a time.
1005	Stop the spooler processing until the value 0 is entered in the common variable CI99. The value entered in <i>Value</i> is first entered in CI99. When the command is to be executed the CI99 is set by default and must not be used for any other purpose (see also Chapter 4.4.114.1). <b>Note:</b> The PCAP command ClearCI99 <b>must</b> be used to delete CI99, because otherwise, the axes may be started asynchronously.
1006	Stop the spooler processing until the command was activated or executed with the same parameter for all bit-coded axes entered in <i>Value</i> . With this command the spooler processing can be synchronised by different axes (see also Chapter 4.4.114.1).
1015	The parameter value is added to CI99. Then the spooler processing is stopped until CI99 contains the value 0. The wait command is only executed if the target velocity of the previous profile is 0 (see also Chapter 4.4.114.1). <b>Note:</b> The PCAP command ClearCI99 <b>must</b> be used to delete CI99, because otherwise, the axes may be started asynchronously.
1025	In the variable CI99, the bit assigned to the axis is set. Then the spooler processing is stopped until this bit is reset in CI99. The wait command is only executed if the target velocity of the previous profile is 0 (see also Chapter 4.4.114.1). <b>Note:</b> The PCAP command ClearCI99 <b>must</b> be used to delete CI99, because otherwise, the axes may be started asynchronously.
1101	Activate PC interrupt request
1200	Write the Motor-Command-Port mcp of an axis in the system with the index 0..7
...	
1207	1200 = 1. axis, 1201 = 2. axis, etc.
2001	Reset the target velocity in the last spooled traverse profile.
10000	Set bits in CI-variable. The bits to be set are indicated in <i>Value</i> .
...	
10999	
11000	Reset bits in CI-Variable. The bits to be reset are indicated in <i>Value</i> .
...	
11999	
20000	Write on CD-variable with <i>value</i> .
...	
20999	

#### 4.4.114.1 Notes on SSF wait commands

Some of the above described wait commands use the common integer variable CI99.

Here it should be noted that these variables from the PCAP programming are written on via direct PCI memory access and thus asynchronously to the RWMOS operating system software. To achieve error-free synchronicity of axes after a profile continuation via an SSF wait command, the DLL command ClearCI99 must therefore be used.

In some of the commands specified above, bit-coded data, e.g. of inputs, outputs or axes, is expected. Here, the corresponding bit numbers are assigned to the relevant number of the value to be programmed.

Inputs 1 and 3, for example, are to be specified in bit-coded form. In this case, the hexadecimal value 5 must be programmed. This means it is possible to specify multiple axes, inputs or outputs in one data word.

<b>EXAMPLES:</b>	<code>ssf(A1, 125, 999);</code>	<code>// Write CI125 with the value 999</code>
	<code>ssf(A1, 1001, 1);</code>	<code>// Set output O1 at axis 1</code>
	<code>ssf(A1, 1002, 4);</code>	<code>// Reset output O3 at axis 1</code>

#### 4.4.115 startcnct, start numeric controller task

<b>DESCRIPTION:</b>	This command can be used to start a previously loaded SAP program. The CNC task selected in <i>TaskNr</i> (values 0..3) processes the SAP program right from its beginning. The PCAP command <i>txbf2()</i> can be used for loading.
<b>BORLAND DELPHI:</b>	procedure startcnct(TaskNr:integer);
<b>C:</b>	void startcnct(int TaskNr);
<b>VISUAL BASIC:</b>	Sub startcnct(ByVal TaskNr As Long)
<b>NOTE:</b>	A currently running SAP program will be stopped automatically before this command is executed. PCAP command <i>txbf2()</i>

#### 4.4.116 stepcnct, step numeric controller task

<b>DESCRIPTION:</b>	This command is used for executing an SAP program line by line.
<b>BORLAND DELPHI:</b>	procedure stepcnct (TaskNr:integer);
<b>C:</b>	void stepcnct(int TaskNr);
<b>VISUAL BASIC:</b>	Sub stepcnct(ByVal TaskNr As Long)
<b>NOTE:</b>	The PCAP command <i>stepcnct()</i> has not yet been implemented at present!

#### 4.4.117 stopcnct, stop numeric controller task

<b>DESCRIPTION:</b>	This command causes the SAP program currently being run to stop in the CNC task selected with <i>TaskNr</i> (values 0..3) and de-activates this CNC task. The SAP program can be continued with the SAP command <i>CONTCNCT()</i> or the PCAP command <i>contcnct()</i> .
<b>BORLAND DELPHI:</b>	procedure stopcnct(TaskNr:integer);
<b>C:</b>	void stopcnct(int TaskNr);
<b>VISUAL BASIC:</b>	Sub stopcnct(ByVal TaskNr As Long)
<b>NOTE:</b>	Any EVENT handlers enabled in the SAP program will no longer be processed after the <i>stopcnct()</i> command has been executed. Before this command is executed, the drive should be put into a safe operating state.



#### 4.4.118 szpa, set zero position absolute

<b>DESCRIPTION:</b>	This command sets an axis-specific virtual zero position. The parameter <i>Position</i> is specified in the axis-specific position unit. The parameter <i>an</i> specifies the axis number. 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.
<b>BORLAND DELPHI:</b>	procedure szpa(an: integer; Position: double);
<b>C:</b>	void szpa(int an, double Position);
<b>VISUAL BASIC:</b>	Sub szpa(ByVal an As Long, ByVal Position As Double)
<b>NOTE:</b>	By calling up szpa with the position value 0, a zero offset which has been possibly set can be deleted. The currently set position value of the zero offset can be read with the command rdZeroOffset Chapter 4.4.105). See also Bit ClearZeroPosition in the register ModeReg.

#### 4.4.119 szpr, set zero position relative

<b>DESCRIPTION:</b>	This commands sets an axis-specific virtual zero position to a relative position. The parameter <i>Position</i> is specified in the axis-specific position unit. The parameter <i>an</i> specifies the axis number. 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.
<b>BORLAND DELPHI:</b>	procedure szpr(an: integer; Position: double);
<b>C:</b>	void szpr(int an, double Position);
<b>VISUAL BASIC:</b>	Sub szpr(ByVal an As Long, ByVal Position As Double)
<b>NOTE:</b>	By calling up szpa with the position value 0, a zero offset which has been possibly set can be deleted. The currently set position value of the zero offset can be read with the command rdZeroOffset (Chapter 4.4.105). See also Bit ClearZeroPosition in the register ModeReg.

#### 4.4.120 txbf, transmit binary file

<b>DESCRIPTION:</b>	<p>This function is used to transfer the file specified in the string or character parameter to the xPCI-800x board. The specified file is first searched in the current working directory. Then the directories which are specified in the environment variable PATH are searched. There is an additional function (txbf()) in the function library for compatibility reasons: yet this function txbf() does not support any file name including drive or path information. When the function txbf2 is called up two special file types are essentially permitted.</p> <p>Firstly, the <i>system.dat</i> 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 <i>ncc.exe</i> command line compiler.</p> <p>Transferring the <i>system.dat</i> system file has the following results:</p> <p>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 <i>uf()</i>. These system data can also be edited in the TOOLSET program <i>mcf.exe</i>. Any system variables previously altered (e.g. axis-specific velocities, accelerations, etc.) are overwritten again by this command.</p> <p><b>Important!</b> 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 <i>startcnct()</i> or the PCAP command <i>STARTCNCT()</i>.</p>
---------------------	--

<b>BORLAND DELPHI:</b>	function txbf2(var filename:string):integer;																
<b>C:</b>	int txbf2(char far *filename);																
<b>VISUAL BASIC:</b>	Function txbf2(ByVal filename As String) As Long																
<b>RETURN VALUE:</b>	The function can return the following values: <table border="1"> <thead> <tr> <th>Return value</th><th>Error description</th></tr> </thead> <tbody> <tr> <td><b>0</b></td><td>No error</td></tr> <tr> <td><b>20</b></td><td>File cannot be opened. Possible causes are as follows: <ul style="list-style-type: none"> <li>- Invalid file name</li> <li>- File does not exist</li> <li>- Path and search drive is invalid</li> </ul> </td></tr> <tr> <td><b>21</b></td><td>The file too large for the CNC task main memory.</td></tr> <tr> <td><b>22</b></td><td>Invalid file type (Not a SAP file nor a system file)</td></tr> <tr> <td><b>23</b></td><td>Internal error when reserving memory.</td></tr> <tr> <td><b>24</b></td><td>Invalid task number is indicated. At a system file this error shows that an invalid or damages system file was used.</td></tr> <tr> <td><b>25</b></td><td>Data transfer error with remote systems (WebServices)</td></tr> </tbody> </table>	Return value	Error description	<b>0</b>	No error	<b>20</b>	File cannot be opened. Possible causes are as follows: <ul style="list-style-type: none"> <li>- Invalid file name</li> <li>- File does not exist</li> <li>- Path and search drive is invalid</li> </ul>	<b>21</b>	The file too large for the CNC task main memory.	<b>22</b>	Invalid file type (Not a SAP file nor a system file)	<b>23</b>	Internal error when reserving memory.	<b>24</b>	Invalid task number is indicated. At a system file this error shows that an invalid or damages system file was used.	<b>25</b>	Data transfer error with remote systems (WebServices)
Return value	Error description																
<b>0</b>	No error																
<b>20</b>	File cannot be opened. Possible causes are as follows: <ul style="list-style-type: none"> <li>- Invalid file name</li> <li>- File does not exist</li> <li>- Path and search drive is invalid</li> </ul>																
<b>21</b>	The file too large for the CNC task main memory.																
<b>22</b>	Invalid file type (Not a SAP file nor a system file)																
<b>23</b>	Internal error when reserving memory.																
<b>24</b>	Invalid task number is indicated. At a system file this error shows that an invalid or damages system file was used.																
<b>25</b>	Data transfer error with remote systems (WebServices)																
<b>NOTE:</b>	Normally, the <i>system.dat</i> system file need be loaded only once per system start. Please see the particulars given for the PCAP command <i>mcuinit()</i> in this context. If you want, you can specify drive and path names in the <i>filename</i> parameter.																

#### 4.4.121 txbfErrorReport, initialisation error report

<b>DESCRIPTION:</b>	This function gives in plaintext the error return value of the function <i>txbf2()</i> described above. A message box is displayed on screen and is to be closed after reading.
<b>BORLAND DELPHI:</b>	procedure txbfErrorReport(filename:PChar; error:integer);
<b>C:</b>	void txbfErrorReport (char *filename, int error);
<b>VISUAL BASIC:</b>	Sub txbfErrorReport (ByVal filename As String, ByVal error As Long)
<b>NOTE:</b>	PCAP commands InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3()
<b>EXAMPLE:</b>	<pre>txbferror = InitMcuSystem3( ... );           // Execute file transfer txbfErrorReport(..., initerror);           // In case of error, display error return  // value</pre>

#### 4.4.122 uf, update filter

<b>DESCRIPTION:</b>	You can use this command to set the xPCI-800x 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.
<b>BORLAND DELPHI:</b>	procedure uf(var tsrp:TSRP);
<b>C:</b>	void uf(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub uf(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv n = 0 .. Number of axis present -1
<b>NOTE:</b>	You will find more details on the PIDF filter in chapter 2.1.1.12, OM / Chapter 4.1.1, CM / Chapter 6.2 and PCAP command <i>rdf()</i> .

#### 4.4.123 utrovr, update trajectory override

<b>DESCRIPTION:</b>	The velocity override currently set is taken into account for all axis channels selected in AS.
<b>BORLAND DELPHI:</b>	procedure utrovr(var as:AS);
<b>C:</b>	void utrovr(struct AS far *as);
<b>VISUAL BASIC:</b>	Sub utrovr(DASEL As ASEL)
<b>NOTE:</b>	With this command, the last written trajectory override value is adopted for the selected axes. If the bit OvrMode is not set in the Modereg register, this value is adopted for the axis-specific variable jovr. You will find further information under the PCAP command <i>wrtrovr()</i> . Depending on the value set through the function <i>wrtrovrst()</i> , the override value is not adopted at once, but is adapted to the given ramp time.

#### 4.4.124 wraux, write auxiliary register

<b>DESCRIPTION:</b>	This function sets the axis-specific auxiliary register to the value set in <i>aux</i> .
<b>BORLAND DELPHI:</b>	procedure wraux (var tsrp:TSRP);
<b>C:</b>	void wraux (struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wraux (DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].aux
<b>NOTE:</b>	See also chapter 4.4.44 and 6.3.3

#### 4.4.125 wrbcnct, write common buffer CNC-Task

<b>DESCRIPTION:</b>	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 xPCI-800x board.												
<b>BORLAND DELPHI:</b>	function wrbcnct(var cbcnct:CBCNCT):integer;												
<b>C:</b>	int wrbcnct(struct CBCNCT far *cbcnct);												
<b>VISUAL BASIC:</b>	Sub wrbcnct(DCBCNCT As CBCNCT)												
<b>RETURN VALUE:</b>	The wrbcnct() function has the following bit-coded return value: <table border="1" data-bbox="507 1541 1385 1765"> <thead> <tr> <th>Bit-Nr</th><th></th></tr> </thead> <tbody> <tr> <td>0</td><td>0 No error</td></tr> <tr> <td>0</td><td>1 if task number invalid.</td></tr> <tr> <td>1</td><td>0 no error</td></tr> <tr> <td>1</td><td>if maximum permitted buffer size exceeded</td></tr> <tr> <td></td><td>This means that the function in normal circumstances returns the value 0.</td></tr> </tbody> </table>	Bit-Nr		0	0 No error	0	1 if task number invalid.	1	0 no error	1	if maximum permitted buffer size exceeded		This means that the function in normal circumstances returns the value 0.
Bit-Nr													
0	0 No error												
0	1 if task number invalid.												
1	0 no error												
1	if maximum permitted buffer size exceeded												
	This means that the function in normal circumstances returns the value 0.												
<b>NOTE:</b>	The CNC-task-specific buffer size is <u>1,000</u> bytes. The record structure for CBCNCT is shown in Chapter 4.3.2.9. PCAP command <i>rdcbcnct()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>												

#### 4.4.126 wrcd, write common double

<b>DESCRIPTION:</b>	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 <i>rw_SymPas</i> system variables CD0 .. CD99. The first parameter here specifies the number <i>ndx</i> of the double variable to be written. The value range of <i>ndx</i> 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.
<b>BORLAND DELPHI:</b>	procedure wrcd(ndx: integer; var cdbuf:CDBUF);
<b>C:</b>	void wrcd(int ndx, struct CDBUF far *cdbuf);
<b>VISUAL BASIC:</b>	Sub wrcd(ByVal ndx As Long, CDBUF As CDBUF)
<b>NOTE:</b>	The content of all common variables remains stored in memory even after a system reset operation, which is executed by the <i>rs()</i> 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.

#### 4.4.127 wrci, write common integer

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>wrcd()</i> , except that here the variables concerned are the <i>rw_SymPas</i> system variables CI0 .. CI999 of the LONGINT type.
<b>BORLAND DELPHI:</b>	procedure wrci(ndx: integer; var cibuf:CIBUF);
<b>C:</b>	void wrci(int ndx, struct CIBUF far *cibuf);
<b>VISUAL BASIC:</b>	Sub wrci(ByVal ndx As Long, CIBUF As CIBUF)
<b>NOTE:</b>	PCAP command <i>wrcd()</i>

#### 4.4.128 wrdigo, write digital outputs

##### Function description is only relevant for the APCI-8001 / APCI-8008

DESCRIPTION:	<p>This register can be used to set the digital outputs of the APCI-8001. It has to be considered that the digital outputs of the xPCI-800x are not grouped in an axis-specific way. If an output is to be set, this can be done by setting the respective bit. The bit-coded structure of the <i>digo</i> status word can be found in the table below:</p> <p>Table 17: Bit-coded structure of the status word <i>digo</i></p> <table><tr><th>Bit number</th><th>Function</th><th>Connector X1 / PIN</th></tr><tr><td>0</td><td>Output 1</td><td>26</td></tr><tr><td>1</td><td>Output 2</td><td>27</td></tr><tr><td>2</td><td>Output 3</td><td>28</td></tr><tr><td>3</td><td>Output 4</td><td>29</td></tr><tr><td>4</td><td>Output 5</td><td>30</td></tr><tr><td>5</td><td>Output 6</td><td>31</td></tr><tr><td>6</td><td>Output 7</td><td>32</td></tr><tr><td>7</td><td>Output 8</td><td>33</td></tr><tr><td>8..31</td><td>Not assigned</td><td>--</td></tr></table>	Bit number	Function	Connector X1 / 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	--
Bit number	Function	Connector X1 / 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	--																													
BORLAND DELPHI:	procedure wrdigo(var tsrp:TSRP);																														
C:	void wrdigo(struct TSRP far *tsrp);																														
VISUAL BASIC:	Sub wrdigo(DTSRP As TSRP)																														
TSRP COMPONENTS:	TSRP[n].digo																														

**Function description is only relevant for the CPCI-8004**

**DESCRIPTION:** This register can be used to set the digital outputs of the CPCI-8004. It has to be considered that the digital outputs of the xPCI-800x are not grouped in an axis-specific way. If an output is to be set, this can be done by setting the respective bit. The bit-coded structure of the *digo* status word can be found in the following table:

Table 18: Bit-coded structure of the status word *digo*

Bit number	Function	Connector X1 / PIN
0	Output 1	17
1	Output 2	18
2	Output 3	19
3	Output 4	37
4	Output 5	38
5	Output 6	39
6..31	Not assigned	--

**BORLAND DELPHI:** procedure wrdigo(var tsrp:TSRP);

**C:** void wrdigo(struct TSRP far \*tsrp);

**VISUAL BASIC:** Sub wrdigo(DTSRP As TSRP)

**TSRP COMPONENTS:** TSRP[n].digo

**4.4.129 wrdigob, write digital output bit****Function description is only relevant for the APCI-8001 / APCI-8008**

**DESCRIPTION:** This function can be used to set or reset one APCI-8001 digital output. The axis number must be specified in the *an* parameter (0, 1, ... MAXAXIS-1). The output is reset with the value 0 or FALSE.

Table 19: Assignment of *bitnr* to the respective xPCI-800x digital outputs

'bitnr'	Function	Connector X1 / 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	--

**BORLAND DELPHI:** procedure wrdigob(an:integer; bitnr:integer; value: integer);

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

**VISUAL BASIC:** Sub wrdigob(ByVal an As Long, ByVal bitnr As Long, ByVal value As Long)

**NOTE:** PCAP command *wrdigo()*

**Funktionsbeschreibung is only relevant for the CPCI-8004**

DESCRIPTION:

This function can be used to set or reset one CPCI-8004 digital output. The axis number must be specified in the *an* parameter (0, 1, ... MAXAXIS-1). The output is reset with the value 0 or FALSE.

Table 20: Assignment of *bitnr* to the respective xPCI-800x digital outputs

'bitnr'	Function	Connector X1 / PIN
1	Output 1	17
2	Output 2	18
3	Output 3	19
4	Output 4	37
5	Output 5	38
6	Output 6	39
7..32	Not assigned	--

BORLAND DELPHI:

procedure wrdigob(an:integer; bitnr:integer; value: integer);

C:

wrdigob(int an, int bitnr, int value);

VISUAL BASIC:

Sub wrdigob(ByVal an As Long, ByVal bitnr As Long, ByVal value As Long)

NOTE:

PCAP command *wrdigo()*

**4.4.130 wrdp, write desired position**

<b>DESCRIPTION:</b>	<p>You can use this command to write the axis-specific setpoint position (<i>dp</i>). This command is normally never 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 (<i>dp</i>) and the current position (<i>rp</i>), you must anticipate that the motor will be corrected to this position at maximum system acceleration.</p>
<b>BORLAND DELPHI:</b>	procedure wrdp(var tsrp:TSRP);
<b>C:</b>	void wrdp(struct TSRP far *tsrp) ;
<b>VISUAL BASIC:</b>	Sub wrdp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].dp
<b>NOTE:</b>	<p>Writing the setpoint position (<i>dp</i>) during execution of motion commands may lead to uncontrolled process behaviour and should therefore be avoided.</p> <p>PCAP command <i>rddp()</i></p>

**4.4.131 wrdp offset, write desired position offset**

<b>DESCRIPTION:</b>	With this command the axis specific set position offset ( <i>dpoffset</i> ) can be described.
<b>BORLAND DELPHI:</b>	function wrdpoffset (an: integer; var value: double): integer;
<b>C:</b>	int wrdpoffset(int an, double *value);
<b>VISUAL BASIC:</b>	Function wrdpoffset (ByVal an As Long, value As Double) As Long
<b>PARAMETER:</b>	<p>With <i>an</i> the axis channel, which has to be called, is indicated (0, 1, ...).</p> <p>In <i>value</i> the position offset, which has to be written, is transferred in the axis specific position unit.</p>
<b>RETURN VALUE:</b>	<p>0 at access,</p> <p>unequal 0 at failure, if e.g. RWMOS.ELF not yet supports this function.</p>
<b>NOTE:</b>	<p>In general here only small changes may be programmed, because the set point changes cause a jump of the axis. See also axis-qualifier <i>dpoffset</i> in Table 37. With a <i>dpoffset</i> value (see Chapter 4.4.132), the velocity offset of <i>dpoffset</i> can be parameterised, though.</p> <p>This register can be used e.g. for a regulation, overlaying the position controller, or for a spindle linearisation / spindle correction.</p>

#### 4.4.132 wrdvoffset, write desired velocity offset

<b>DESCRIPTION:</b>	With this command, the velocity offset ( <i>dvoffset</i> ) of the axis-specific setpoint position offset ( <i>dpoffset</i> ) can be written on.
<b>BORLAND DELPHI:</b>	function wrdvoffset (an: integer; var value: double): integer;
<b>C:</b>	int wrdvoffset(int an, double *value);
<b>VISUAL BASIC:</b>	Function wrdvoffset (ByVal an As Long, value As Double) As Long
<b>PARAMETER:</b>	With <i>an</i> , the axis channel which has to be accessed is indicated (0, 1, ...). In <i>value</i> , the velocity offset which has to be written is returned in the axis-specific position unit.
<b>RETURN VALUE:</b>	0 at success, unequal 0 at failure, if e.g. RWMOS.ELF does not support this function yet.
<b>NOTE:</b>	With the value 0, changes of <i>dpoffset</i> are immediately adopted. The default value is 0.

#### 4.4.133 wrEffRadius – Write Effective Radius

<b>DESCRIPTION:</b>	With the command the effective radius can be written for a rotatory axis.
<b>BORLAND DELPHI:</b>	wrEffRadius (an: integer; var value: double);
<b>C:</b>	void wrEffRadius (long an, double *value);
<b>VISUAL BASIC:</b>	Sub wrEffRadius (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axes is entered in and the effective radius is transmitted in value in the unit which is used per PU.
<b>NOTE:</b>	See chapter 6.3.3

#### 4.4.134 wrGCR, write gear configuration register

<b>DESCRIPTION:</b>	With this function, the axis-specific Gear Configuration Register can be written on. [Chapter 6.3.3]
<b>BORLAND DELPHI:</b>	procedure wrGCR (an: integer; var value: integer);
<b>C:</b>	void wrGCR (long an, long *value);
<b>VISUAL BASIC:</b>	Sub wrGCR (ByVal an As Long, value As Long)
<b>PARAMETER:</b>	With <i>an</i> , the axis channel which has to be read out is indicated (0, 1, ...). In <i>value</i> , the contents of the GCR register is returned.
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	See also document on the resource interface - GEAR

#### 4.4.135 wrgf, write gear factor

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure wrgf(var tsrp:TSRP);
<b>C:</b>	void wrgf(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrgf(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].gf
<b>NOTE:</b>	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 <i>gf</i> can be read with the PCAP command <i>rdgf()</i> .

#### 4.4.136 wrgfaux, write gear factor auxiliary channel

<b>DESCRIPTION:</b>	With this function, the axis-specific ratio of stepper motor resolution to encoder channel in stepper systems with encoder verification can be written. The default value is 1.0; the value can only be changed at runtime.
<b>BORLAND DELPHI:</b>	function wrgfaux (an: integer; var value: double) : integer;
<b>C:</b>	int wrgfaux(int an, double *value)
<b>VISUAL BASIC:</b>	Function wrgfaux (ByVal an As Long, value As Double) As Long
<b>RETURN VALUE:</b>	After successful execution, the function returns 0. In this case, the value in <i>value</i> could be successfully written to the axis <i>an</i> . With a return value $\neq 0$ , the value could not be written, because e.g. RWMOS.ELF does not support the command.
<b>NOTE:</b>	The factor can be read at any time with the PCAP command <i>rdgfaux()</i> . See also Chapter 6.3.3.

#### 4.4.137 wrhac, write home acceleration

<b>DESCRIPTION:</b>	You use this command to set the axis-specific maximum acceleration <i>hac</i> for all reference travel commands ( <i>home</i> commands). 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.
<b>BORLAND DELPHI:</b>	procedure wrhac(var tsrp:TSRP);
<b>C:</b>	void wrhac(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrhac(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].hac
<b>NOTE:</b>	The value currently set for <i>hac</i> can be read with the PCAP command <i>rdhac()</i> .

#### 4.4.138 wrhvl, write home velocity

<b>DESCRIPTION:</b>	You use this command to set the axis-specific maximum velocity with the aid of the <i>hvl</i> variable for all reference travel commands ( <i>home</i> commands). 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.
<b>BORLAND DELPHI:</b>	procedure wrhvl(var tsrp:TSRP);
<b>C:</b>	void wrhvl(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrhvl(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].hvl
<b>NOTE:</b>	The value currently set for <i>hac</i> can be read with the PCAP command <i>rdhvl()</i> .

#### 4.4.139 wripw, write in position window

<b>DESCRIPTION:</b>	This command can be used to alter (during the run time) the In-Position Window {ipw} specified using the TSW program <i>mcfg.exe</i> . The window is re-specified to the value set in <i>ipw</i> . The value is stated in the axis-specific position unit.
<b>BORLAND DELPHI:</b>	procedure wripw(var tsrp:TSRP);
<b>C:</b>	void wripw(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wripw(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].ipw
<b>NOTE:</b>	The "In-Position-Window" is monitored only when a value greater than 0.0 has been specified. (MCFG / Chapter 1.7.2.1.11) PCAP command <i>rdipw()</i>



#### 4.4.140 wrjac, write jog acceleration

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure wrjac(var tsrp:TSRP);
<b>C:</b>	void wrjac(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrjac(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].jac
<b>NOTE:</b>	The value currently set for <i>jac</i> can be read with the PCAP command <i>rdjac()</i> .

#### 4.4.141 wrJerkRel, write jerkrel

<b>DESCRIPTION:</b>	With this command the axis-specific parameter <i>jerkrel</i> can be written in.
<b>BORLAND DELPHI:</b>	procedure wrJerkRel (an: integer; var value: double);
<b>C:</b>	void wrJerkRel (long an, double *value);
<b>VISUAL BASIC:</b>	Sub wrJerkRel (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	an = Number of axes (0..n) value = value to be zu written
<b>RETURN VALUE:</b>	None
<b>NOTE:</b>	Only one value between 0 and 1 can be allocated to <i>jerkrel</i> . Lower or higher values are limited. See also chapter 4.4.75 and 6.3.3

#### 4.4.142 wrjovr, write jog override

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure wrjovr(var trsp:TSRP);
<b>C:</b>	void wrjovr(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrjovr(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].jovr
<b>NOTE:</b>	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. Whenn calling <i>utovr</i> and selected axis, the value of <i>jovr</i> is set also, if this was not switched off explicitly in the register variable <i>ModeReg</i> . PCAP command <i>rdjovr()</i>

#### 4.4.143 wrjtv, write jog target velocity

<b>DESCRIPTION:</b>	This command is used to set the axis-specific target velocity (jog) with the aid of the <i>jtv</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.
<b>BORLAND DELPHI:</b>	procedure wrjtv(var tsrp:TSRP);
<b>C:</b>	void wrjtv(struct TSPR far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrjtv(DTSRP As TSPR)
<b>TSPR COMPONENTS:</b>	TSPR[n].jtv
<b>NOTE:</b>	The value currently set for <i>jtv</i> can be read with the PCAP command <i>rdjtv()</i> .

#### 4.4.144 wrjvl, write jog velocity

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>wrhvl()</i> , except that here the maximum traversing velocity is specified using the <i>jvl</i> variable for all <i>jog</i> commands.
<b>BORLAND DELPHI:</b>	procedure wrjvl(var tsrp:TSRP);
<b>C:</b>	void wrjvl(struct TSPR far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrjvl(DTSRP As TSPR)
<b>TSPR COMPONENTS:</b>	TSPR[n].jvl
<b>NOTE:</b>	The value currently set for <i>jvl</i> can be read with the PCAP command <i>rdjvl()</i> .

#### 4.4.145 wrledgn, write led green

<b>DESCRIPTION:</b>	
<b>APCI-8001:</b>	This command can be used to switch the green SMD LED D29 on and off. It is switched on with the value 1 and switched off with the value 0.
<b>APCI-8008:</b>	This command can be used to switch the green SMD LED D53 on and off. It is switched on with the value 1 and switched off with the value 0.
<b>CPCI-8004:</b>	This command can be used to switch the green SMD LED D36 on and off. It is switched on with the value 1 and switched off with the value 0.
<b>BORLAND DELPHI:</b>	procedure wrledgn(value:integer);
<b>C:</b>	void wrledgn(int value);
<b>VISUAL BASIC:</b>	Sub wrledgn(ByVal value As Long)
<b>NOTE:</b>	This command is primarily used as a testing and diagnostic tool. The SMD-LED is located on the high back end of the solder side of the board.

#### 4.4.146 wrledrd, write led red

<b>DESCRIPTION:</b>	
<b>APCI-8001:</b>	as for PCAP command <i>wrledgn()</i> , but for the red LED D31
<b>APCI-8008:</b>	as for PCAP command <i>wrledgn()</i> , but for the red LED D56
<b>CPCI-8004:</b>	as for PCAP command <i>wrledgn()</i> , but for the red LED D38
<b>BORLAND DELPHI:</b>	procedure wrledrd(value:integer);
<b>C:</b>	void wrledrd(int value);
<b>VISUAL BASIC:</b>	Sub wrledrd(ByVal value As Long)

#### 4.4.147 wrledyl, write led yellow

<b>DESCRIPTION:</b>	
<b>APCI-8001:</b>	as for PCAP command <i>wrledgn()</i> , but for the yellow LED D30
<b>APCI-8008:</b>	as for PCAP command <i>wrledgn()</i> , but for the yellow LED D55
<b>CPCI-8004:</b>	as for PCAP command <i>wrledgn()</i> , but for the yellow LED D37
<b>BORLAND DELPHI:</b>	procedure wrledyl(value:integer);
<b>C:</b>	void wrledyl(int value);
<b>VISUAL BASIC:</b>	Sub wrledyl(ByVal value As Long)

#### 4.4.148 wrlp, write latched position

<b>DESCRIPTION:</b>	This command is used to set the axis-specific latch position to the value set in <i>lp</i> . The value is specified in the axis-specific position unit.
<b>BORLAND DELPHI:</b>	procedure wrlp(var tsrp:TSRP);
<b>C:</b>	void wrlp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrlp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].lp
<b>NOTE:</b>	PCAP command <i>rdlp()</i>

#### 4.4.149 wrlpndx, write latched position index

<b>DESCRIPTION:</b>	This command is used to set the axis-specific latch position of the zero track (index) to the value set in <i>lp</i> . The value is specified in the axis-specific position unit.
<b>BORLAND DELPHI:</b>	procedure wrlpndx (var tsrp:TSRP);
<b>C:</b>	void wrlpndx (struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrlpndx (DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].lp
<b>NOTE:</b>	PCAP command <i>rdlpndx()</i>

#### 4.4.150 wrMaxAcc – Write Maximum Acceleration Check

<b>DESCRIPTION:</b>	With this command you can write the maximum axis-specific acceleration ( <i>MAXACC</i> ). This value is used by the RWMOS operating system software to limit the trajectory acceleration so that no axis involved in a linear interpolation exceed the maximum acceleration accepted.
<b>BORLAND DELPHI:</b>	wrMaxAcc (an: integer; var value: double);
<b>C:</b>	void wrMaxAcc (long an, double *value);
<b>VISUAL BASIC:</b>	Sub wrMaxAcc (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axes is entered in <i>an</i> , the maximum acceleration accepted is transmitted in <i>value</i> in the axis-specific acceleration unit.
<b>NOTE:</b>	To check this function the bit 7 in MODEREG register must be set (see chapter 6.3.1.4). When set to 0 the check function is disabled for the axis involved. The function is only available with spooled commands.

#### 4.4.151 wrMaxVel – Write Maximum Velocity Check

<b>DESCRIPTION:</b>	With this command you can write the maximum axis-specific velocity ( <i>MAXVEL</i> ). This value is used by the RWMOS operating system software to limit the trajectory velocity so that no axes involved in a linear interpolation exceeds the maximum velocity accepted.
<b>BORLAND DELPHI:</b>	wrMaxVel (an: integer; var value: double);
<b>C:</b>	void wrMaxVel (long an, double *value);
<b>VISUAL BASIC:</b>	Sub wrMaxVel (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axes is entered in an, the maximum velocity accepted is transmitted in <i>value</i> in the axis-specific velocity unit.
<b>NOTE:</b>	To check this function the bit 7 in MODEREG register must be set (see chapter 6.3.1.4). When set to 0 the check function is disabled for the axis involved. The function is only available with spooled commands.

#### 4.4.152 wrmcp, write motor command port

<b>DESCRIPTION:</b>	<p>This command is used to write the Motor-Command-Port to the value set in the <i>mcp</i> 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.</p>
<b>APCI-8001:</b> <b>APCI-8008:</b> <b>CPCI-8004:</b>	<p>In the case of servo axes, <i>mcp</i> can be set to a value between -32,767 and +32,767. This value range corresponds to the analogue output voltage range of -10 V to +10 V. It may be necessary to allow for a planned inversion of the analogue output signal.</p> <p>In the case of stepping motor axes, <i>mcp</i> 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:</p> $f_{\text{Pulse}} = \text{CLOCK} / 2 / (\text{mcp} + 1)$ <p><i>Example: with mcp = 999 and CLOCK = 70MHz</i>  <math>f_{\text{Pulse}} = 35,000 [\text{Hz}]</math></p> <p>The CLOCK value is 70 MHz for the APCI-8001 and 66.66666 MHz for the APCI-8008.</p> <p>The value range of <i>mcp</i> lies between -1,048,574 and +1,048,574. The sign selects the desired traversing direction and influences the axis-specific directional signal. For the stepping signal <math>f_{\text{Pulse}}</math>, only the absolute value of <i>mcp</i> is determinant. Remember that the value 0 in <i>mcp</i> causes a stepping signal of 0 Hz, i.e. the motor halts.</p>
<b>BORLAND DELPHI:</b>	procedure wrmcp(var tsrp:TSRP);
<b>C:</b>	void wrmcp(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrmcp(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].mcp
<b>NOTE:</b>	<p>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.</p> <p>PCAP command <i>rdmcp()</i></p>

#### 4.4.153 wrMDVel – Write Maximum Velocity Skip

<b>DESCRIPTION:</b>	With this command you can write the maximum axis-specific velocity jump ( <i>MDVEL</i> ). This value is used by the look-ahead functionality RWMOS operating system software verwendet to limit the trajectory velocity so that no axis involved in an interpolation exceeds the maximum velocity accepted.
<b>BORLAND DELPHI:</b>	wrMDVel (an: integer; var value: double);
<b>C:</b>	void wrMDVel (long an, double *value);
<b>VISUAL BASIC:</b>	Sub wrMDVel (an As Long, ByVal value As Double)
<b>PARAMETER:</b>	The number of axes is entered in an, the maximum velocity accepted is transmitted in <i>value</i> in the activated position and time units (PU and TU) übergeben.
<b>NOTE:</b>	The look-ahead mode is activated by setting the bit 0 in MODEREG register (see chapter 6.3.1.4). The check function of the axis is disabled with the value 0. In this case please consider also the bit 6 of MODEREG. <b>Important:</b> In look-ahead mode the different profiles must be programmed with a target velocity > 0 (generally = maximum velocity) so that the look ahead is really effective.

#### 4.4.154 wrModeReg – Write MODEREG

<b>DESCRIPTION:</b>	With this command the register MODEREG of the RWMOS operating system software can be described.
<b>BORLAND DELPHI:</b>	wrModeReg (var value: integer);
<b>C:</b>	void wrModeReg(long *value);
<b>VISUAL BASIC:</b>	wrModeReg (ByVal value As Long)
<b>PARAMETER:</b>	bitcodierter value für ModeReg
<b>NOTE:</b>	With flags (bits) in the ModeReg register different options can be activated and monitored in RWMOS.ELF such as e.g. look-ahead, S profile etc. (see chapter 6.3.1.4).

#### 4.4.155 wrmpe, write maximum position error

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure wrmpe(var tsrp:TSRP);
<b>C:</b>	void wrmpe(struct TSRP far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrmpe(DTSRP As TSRP)
<b>TSRP COMPONENTS:</b>	TSRP[n].mpe
<b>NOTE:</b>	Position error monitoring is performed only if a value greater than 0.0 has been specified and the control loop is closed. (MCFG / Chapter 1.7.2.1.9) PCAP command <i>rdmpe()</i>

#### 4.4.156 wrnfx, write No-Feed-Rate-Axis

<b>DESCRIPTION:</b>	With this command on the NFRAX register of the RWMOS operating system software is written.
<b>BORLAND DELPHI:</b>	wrnfx (var value: integer);
<b>C:</b>	void wrnfx (long *value);
<b>VISUAL BASIC:</b>	Sub wrnfx (VYVal value As Long)
<b>PARAMETER:</b>	Bit coded value for NFRAX
<b>NOTE:</b>	In the register NFRAX so-called No-Feed-Rate axes can be defined bit coded. These axes are not used for the calculation of the velocity at interpolation commands; in spite of taking part in the interpolation. In this way the influence of other axes for the velocity in interpolation profiles can be prevented. See also rdnfrax function in chapter 4.4.91.

#### 4.4.157 wrp, write real position

<b>DESCRIPTION:</b>	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.
<b>BORLAND DELPHI:</b>	procedure wrp(var tsrp:TSRP);
<b>C:</b>	void wrp(struct Tsrp far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrp(DTSRP As Tsrp)
<b>TSRP COMPONENTS:</b>	TSRP[n].mpe
<b>NOTE:</b>	This command will cause the machine zero to be shifted automatically!

#### 4.4.158 wrsdec, write stop deceleration

<b>DESCRIPTION:</b>	This command is used to set the axis-specific stop deceleration <i>sdec</i> for the following: the PCAP command <i>js()</i> [chapter 4.4.24], SAP command <i>JS()</i> [chapter 6.6.26], the software end positions (MCFG / Chapters 1.7.2.1.10 and 1.7.2.2.3) planned with SMD and the digital inputs planned with LSL_SMD or LSR_SMD projected digital inputs (MCFG / Chapter 1.7.2.5). If <i>wrsdec()</i> 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.
<b>BORLAND DELPHI:</b>	procedure wrsdec(var tsrp:TSRP);
<b>C:</b>	void wrsdec(struct Tsrp far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrsdec(DTSRP As Tsrp)
<b>TSRP COMPONENTS:</b>	TSRP[n].sdec
<b>NOTE:</b>	The value currently set for <i>sdec</i> can be read with the PCAP command <i>rdsdec()</i> (see chapter 4.4.96).

#### 4.4.159 wrsll, write software limit left

<b>DESCRIPTION:</b>	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 <i>mcfg.exe</i> . The left software limit is reset to the value set in <i>sll</i> . The value is specified in the axis-specific position unit.
<b>BORLAND DELPHI:</b>	procedure wrsll(var tsrp:TSRP);
<b>C:</b>	void wrsll(struct Tsrp far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrsll(DTSRP As Tsrp)
<b>TSRP COMPONENTS:</b>	TSRP[n].sll

<b>NOTE:</b>	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. (MCFG / Chapter 1.7.2.1.10) PCAP commands <i>rdsl()</i> , <i>shp()</i> , SAP command <i>SHP()</i>
--------------	---

#### 4.4.160 wrslr, write software limit right

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>wrsll()</i> , but the right software limit is redefined with the value set in the <i>slr</i> parameter.
<b>BORLAND DELPHI:</b>	procedure wrslr(var tsrp:TSRP);
<b>C:</b>	void wrslr(struct TSPR far *tsrp);
<b>VISUAL BASIC:</b>	Sub wrslr(DTSRP As TSPR)
<b>TSPR COMPONENTS:</b>	TSPR[n].slr

#### 4.4.161 wrslsp, write Slits / Stepperpulses

<b>DESCRIPTION:</b>	This command sets the axis-specific resolution per motor turn {slsp}. The default value is determined with the TOOLSET program <i>mcfg.exe</i> .
<b>BORLAND DELPHI:</b>	procedure wrslsp (an: integer; var value: double);
<b>C:</b>	void wrslsp (long an, double *value);
<b>VISUAL BASIC:</b>	Sub wrslsp (ByVal an As Long, value As Double)
<b>TSPR COMPONENTS:</b>	None
<b>NOTE:</b>	slsp can be read with the PCAP command <i>rdslsp()</i> . See also axis qualifier slsp. For slsp only numeric values > 0.0 are allowed.

#### 4.4.162 wrtp – write target position

<b>DESCRIPTION:</b>	With this command you can write the axis-specific target position ( <i>tp</i> ). This command is usually only necessary and used in special cases.
<b>BORLAND DELPHI:</b>	procedure wrtp(var tsrp:TSRP);
<b>C:</b>	void wrtp(struct TSPR far *tsrp) ;
<b>VISUAL BASIC:</b>	Sub wrtp(DTSRP As TSPR)
<b>TSPR COMPONENTS:</b>	TSPR[n].tp
<b>NOTE:</b>	By writing the target position ( <i>tp</i> ) when motion commands are executed an uncontrolled process course can occur in certain cases. This is why it must be avoided. See also PCAP command <i>rdtp()</i> .

#### 4.4.163 wrtrovr, write trajectory override

<b>DESCRIPTION:</b>	This command sets the trajectory velocity correction value for all interpolation commands ( <i>move</i> commands). The <i>value</i> parameter must have a value greater than 0.0. All values smaller than 1.0 result in a reduction in the trajectory velocity. If <i>value</i> has a value greater than 1.0, this will be manifested in an increase in trajectory velocity. The correction value specified in <i>value</i> is placed in intermediate storage on the xPCI-800x board in a system variable and does not become operative until after execution of the PCAP command <i>utrovr()</i> , or the SAP command <i>UTROVR()</i> . The axis channels selected there will be decelerated or accelerated even during trajectory travel, depending on the <i>value</i> correction factor.
<b>BORLAND DELPHI:</b>	procedure wrtrovr(var value:double);
<b>C:</b>	void wrtrovr(double *value);

<b>VISUAL BASIC:</b>	Sub wrtrovr(value As Double)
<b>NOTE:</b>	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 <i>rs()</i> or the SAP command <i>RS</i> is executed, the override factor is initialized to the default value of 1.0. PCAP commands <i>wrtrovr()</i> , <i>wrjovr()</i> , <i>rdtrovr()</i> and <i>rdjovr()</i>

#### 4.4.164 wrtrovrst, write trajectory override settling time

<b>DESCRIPTION:</b>	With this command a „soft“ adaptation of the override value TROVR can be done after the calling of <i>utrovr()</i> realisieren. In the parameter „value“ before the calling of <i>utrovr()</i> a time in seconds is indicated that is the adaptation time between the values 0 and 1. In this way velocity jumps, which are caused by programming, can be avoided. Indicated times that are smaller than the sampling interval are not taken into consideration. The value 0 disables the function.
<b>BORLAND DELPHI:</b>	function wrtrovr(var value:double) : integer;
<b>C:</b>	int wrtrovr(double *value);
<b>VISUAL BASIC:</b>	Function wrtrovr(value As Double) as long
<b>RETURN VALUE:</b>	0 for success -1: command is not available in RWMOS version -4: time-out, reason unknown
<b>NOTE:</b>	Please also see the commands <i>rdtrovrst</i> , <i>wrtrovr</i> , <i>rdtrovr</i> , <i>utrovr</i> and <i>rw_SymPas</i> system parameter TROVRST Setting the Jog-Override using <i>wrjovr</i> is not affected by this function. See also Bit 25 in the MODEREG register (Section 6.3.1.4). Reading the TROVR parameter always returns the programmed target value, even during the adaptation phase. If the current effective override value is to be read during the adaptation phase, the current Jog-Override of one of the axes involved can be used.



## 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 xPCI-800x 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 White space

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 Symbols

*rw\_SymPas* recognizes the following kinds of symbol

*Symbol:*

*Keyword*

*Designator*

*Qualified*

*Labels*

*Constant*

*Operator*

*Punctuation character (including separators)*

*designator*

#### 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 21: 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

### a) 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.

### b) Designator upper and lower case

*rw\_SymPas* distinguishes between upper and lower-case letters, so that *Position*, *position* and *positioN* are different designators.

### c) 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 22: 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 xPCI-800x
A2.digo := $FFFFFFF;    // Set all outputs of xPCI-800x
```

**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 23 shows the formal definition of a constant

**Table 23: 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

a) 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.

b) 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.

c) Hexadecimal constants

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

d) 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).

e) 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  $1.7 \times 10^{-308}$  to  $1.7 \times 10^{308}$ .

f) 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.

g) Punctuation characters

Punctuation characters (also referred to as separators) are defined in *rw\_SymPas* as follows:

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

#### h) 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
```

#### i) Comma

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

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

#### j) 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).

#### k) 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 Types

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 = Type;

Type:

Boolean type

Integer type

Floating-point type

#### a) 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

#### b) Integer type

*rw\_SymPas* provides the integer types *Integer* and *Timer*.

**Table 24: The integer type and its value range**

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

#### c) 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 25: The floating-point types and their accuracy**

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

#### d) 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

##### a) 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. For 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



### a) Declaration section

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

#### ◆ 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

#### ◆ 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

#### ◆ 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

### b) 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.

##### a) 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«.

##### b) 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.

##### c) 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.

##### d) 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*.

#### a) Axis-type declaration

You can define variable axes with the AXIS type declaration. It is particularly useful for the design of sub-programmes (procedures) used several times and in which recurrent actions are to be executed for several axes.

Example:

```
var
    VA : AXIS;                // variable axis with name VA
    VA.an := 0;               // Assign axis number 0 to VA (important!)
    ol(VA);                   // open loop of axis 0
    for VA.an := 0 to 5 do cl(VA); // close loop axes 0 .. 5
```

**Note:** The axes numbers of the predefined axes specifiers (A1 .. An) that are defined by the system parameters can also be assigned anew in this way. This can yet cause a confused and incorrect programming. Should you operate with variable axes, the use of the corresponding variables with corresponding symbol character is recommended.

#### b) 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 control'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 µs.

*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 internal error flag to be set and in certain situations will trigger an abort of the *rw\_MOS* operating system.

A practical timer application might look like this:

```

Const
    s := 15625;           // 15,625 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 26 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 26: 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 27: 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(b)

**Table 28: 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:

$$i \bmod j = i - (i/j)*j;$$

#### 5.3.4.4 Logic operators

Table 29 shows the types of operand involved and the results of logic operations.

**Table 29: 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 `i shl j` and `i 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 30 shows the types of operand involved and the results of Boolean operations.

**Table 30: 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 31 shows the operand types involved and the results of relational operations.

**Table 31: 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(b).

### 5.3.5 Statements

This term “statement” stands for all constructs which agree an action which can be executed by the xPCI-800x board. 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 interested 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.



### a) 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

### a) 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*.

### b) 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*.

### c) 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 xPCI-800x 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 xPCI-800x 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 32: *rw\_SymPas* predefined system parameters

Name	Type	Abbr. meaning	Function
<b>BOARD TYPE</b>	integer	Board-Typ	Hardware version of the control type (see chapter 4.4.18)
<b>CI0..CI99</b>	integer	Common Integer 0..999	100 predefined integer variables for data exchange or for synchronization with a PC application program running in parallel. Further information at the PCAP commands <code>rdci()</code> and <code>wrci()</code> .
<b>CD0..CD99</b>	double	Common Double 0..999	100 predefined double variables. Otherwise as for CI0..CI99. Further information at the PCAP commands <code>rdcd()</code> and <code>wrcd()</code> .
<b>DTCA1</b>	double	Distance-to-Center A1	Indication of medium point for helical profiles and 3 D circles for the X circle axis
<b>DTCA2</b>	double	Distance-to-Center A2	Indication of medium point for helical profile and 3 D circles for the Y circle axis
<b>DTCA3</b>	double	Distance-to-Center A3	Indication of medium point for 3 D circles for the Z circle axis
<b>ERROR REG</b>	integer	error register	Bit coded error register in which internal error states of RWMOS.ELF are indicated.
<b>IRQPC</b>	boolean	Interrupt Request PC	PC interrupt request, active when TRUE
<b>LEDGN</b>	boolean	Led green	Green LED on xPCI-800x, switched on when TRUE
<b>LEDRD</b>	boolean	Led red	Red LED, otherwise as for LEDGN
<b>LEDYL</b>	boolean	Led yellow	Yellow LED, otherwise as for LEDGN
<b>LET</b>	double	Latch End Time	Time for the recording of the graphical system analysis in seconds from the moment LST. See the commands LPR and LPRS. Basically 1000 values are always recorded. The value entered in LET is always rounded up in integral multiples of $1000 \cdot T_A$ . $T_A$ is set to 1.28ms as a standard.
<b>LST</b>	double	Latch Start Time	Moment for the beginning of the recording the graphical system analysis in seconds from the moment of the calling up. See the commands LPR and LPRS.
<b>MODEREG</b>	integer	Mode Register	Bit coded register to control the operating system functionality (see chapter 6.3.1.4)
<b>NFRAX</b>	integer	No-Feed-Rate-Axis	In this variable the axes can be defined as bit coded. They are not utilized for the calculation of the trajectory velocity by interpolation movements.
<b>NOA</b>	integer	Number of Axis	This system variable includes the number of the axes actually available in the system and cannot be written.
<b>OSVERSION</b>	integer	Operating system – version information	The predefined system parameter OSVERSION (Type) returns the current operating system version number of the <code>rwmo.elf</code> file while the SAP program is running. The version number contains a primary number and secondary number. Yet the primary number is incremented in 1000 steps and the secondary in 1 steps. The version number 253042 e.g. means that the primary number is 2.5.3 and the secondary is 042.
<b>PHI</b>	double		Traverse angle for circular and helical profiles
<b>DTCA1</b>	double	Distance-to-Center A1	Center point for helical profile with target point for the X axis
<b>DTCA2</b>	double	Distance-to-Center A2	Center point for helical profile with target point for the Y axis
<b>NFRAX</b>	integer	No-Feed-Rate-Axis	Axes can be coded in bits, which are not utilized by interpolation for the calculation of the trajectory velocity.
<b>PN1</b>	double	Plane-Normal	Surface normals for MCA3D command. Additional Information at the command MCA3D.

Name	Type	Abbr. meaning	Function
PN2	double	Plane-Normal	Surface normals for MCA3D command. Additional Information at the command MCA3D.
PN3	double	Plane-Normal	Surface normals for MCA3D command. Additional Information at the command MCA3D.
PU	integer	Position Unit	Index for position unit
SSFP	integer	Spool-Special-Function-Parameter	Function parameter for specific functions in the spool operating mode. Additional information at the SAP command SSF
TRAC	double	Trajectory Acceleration	Trajectory acceleration for linear, circular and helical profiles
TROVR	double	Trajectory Override	Trajectory velocity correction value
TROVRST	double	Trajectory Override Settling Time	Time for the settling of the 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 (PCI-Interrupt) is automatically given with the aid of the Plug & Play properties integrated on the xPCI-800x board and manage through the system driver mcug3.dll. The user has only to define a PCAP user routine with predefined structure and to inform the driver. Once the xPCI-800x board has generated a hardware interrupt, the corresponding user routine is called up automatically. The mcug3.dll driver is structured in such a way that other PCI interrupts, which use the same interrupt sources can be called up as well.

The driver software that is contained in the scope of delivery, supplies functions in order to easily install an interrupt service routine and, if required, to uninstall it (see chapters 0 and 4.4.34).

#### 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 33: System parameter PU**

Value	Unit	Abbr. meaning
0	Mm	Millimeter
1	Inch	Inch
2	M	Meter
3	Rev	Revolution
4	Deg	Degree
5	Rad	Radiant
6	Counts	Counts
7	Steps	Steps

**Table 34: 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 used 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.

### 6.3.1.3 ERRORREG

In this bit-coded register, runtime errors of the RWMOS operating system software are specified. The bit assignment can be found in Table 15 in Chapter 4.4.62.1.

The ERRORREG register is only reset if it is written on with 0 or by a system boot.

### 6.3.1.4 MODEREG

With the bit coded register different options of the operating system software RWMOS.ELF can be set. As a standard all bits are set to 0.

**Note:** When a bit is to be set or reset in this register, you must pay attention that the other bits are not modified. For this it is necessary to read MODEREG before writing, to process the content with boolean operations and then to write again. Bits are set with boolean Or connection, and reset with And connection. Bits which are currently not assigned must not be used as they are reserved for future extensions.

**Table 36: Bit coding MODEREG**

Bit # / Hex	Name	Description
0 / 0001H	<b>LookAhead</b>	With this bit the look-ahead functionality of the RWMOS operating system software is activated. The given target velocity of interpolation profiles is limited so that the maximum axis-specific velocity jump MDVEL is exceeded by no axis and that all axes are at rest by the end of the interpolation travel. This mode is only valid for the commands SMLA, SMLR, SMCA, SMCR, SMHA, SMHR. In addition, the mode limits the trajectory velocity in circle commands so that the axis-specific Jog acceleration (jac) is not exceeded for no axis involved. The maximum velocity is calculated as follows: $\sqrt{(\text{Kreisradius} * \text{jac})}$ .
1 / 0002H	<b>S-Profil</b>	By setting this bit the acceleration and braking ramp are run with S-form velocity rise/drop. This option can be parameterised with the axis qualifier JERKREL.
2 / 0004H		Not assigned
3 / 0008H	<b>WkzRadKorr</b>	Tool-Radius-correction (only by optionTC) The Tool-Radius-correction is described in a separate manual. Ask for it if you need it.
4 / 0010H		Not assigned
5 / 0020H	<b>AutoSpool</b>	When travel profiles are spooled in SAP programs, the spooler is checked by activated option. Once the spooler is full, the spooler processing is automatically started by the axes selected in the current axes. Further profiles are only entered when memory is available. This option can be used for the following travel commands:

Bit # / Hex	Name	Description
		SMLA, SMLR, SMCR, SMCA, SMHA, SMHR and G01 by DIN66025
6 / 0040H	<b>NoTriangle</b>	Triangular profiles in look-ahead mode are disabled. In case a sub-profile cannot reach the programmed trajectory velocity, the current start velocity remains. This enable a correcter running for short travel profile parts without continuous acceleration and braking phases.
7 / 0080H	<b>ChkMaxVel</b>	In this mode, the trajectory velocity and the trajectory acceleration of all axes are limited by spooled linear interpolation commands so that no axis exceeds the maximum values set in MAXVEL and MAXACC.
8 / 0100H	<b>ExactTargetPos</b>	Usually at the end of an travel profile which has a target velocity of 0, the setpoint position is rounded up in integral value of the system resolution. It is for example by stepper motor systems a step or by encoder systems an encoder counting pulse. It can cause an error by connecting relative profiles with each other. This rounding up can be switched off by setting this bit.
9 / 0200H	<b>ShortestRotatoric Distance</b>	When this bit is set by rotary axes, Jog absolute commands (JA) are run in the direction in which the shortest traverse distance is required.
10 / 0400H	<b>RotatoricUnit</b>	When this bit is set, the target position / the traverse distance are given in the axis-specific rotatory unit for rotatory axes which must be travelled with translatory axes per interpolation command.
11 / 0800H	<b>ForbidTargetVel</b>	If this bit is set, a system reset (rs) is executed, if the traverse profiles are terminated with a target speed $\neq 0$ . In this case in the system variable ErrorReg Bit 1 is set.
12 / 1000H	<b>CenterAlwaysRel</b>	Circle centers at G-Codes G02 and G03 are to be interpreted always a relative coordinates.
13 / 2000H	<b>NoLsmCheck</b>	By setting this flag, the automatic spooler monitoring at G01 of the G-Code interpreter (McuWIN) can be switched off.
14 / 4000H to 23 / 80000H		Currently not assigned.
24 / 0100 0000H	<b>SimulationMode</b>	With this bit, the control can be set into the simulation mode. In this mode no position sizes are given to the drive systems, the profile of the actual position is simulated. <b>Caution:</b> A drift of the axes must be avoided from the user, as in this mode, the bearing controller is disabled.
25 / 0200 0000H	<b>OverMode</b>	When this bit is set, the Jog-Override of the selected axes will not be influenced when calling the commando ctru.
26 / 0400 0000H	<b>StopAtWriteln</b>	When this bit is set, the SAP command writeln causes to stop the corresponding task. In this case in the register "running" of the data structure CNCTS (Section 4.3.2.10) additionally Bit 2 is set. This mode can be used for the complete processing of output strings in an overlapping program.
27 / 0800 0000H	<b>ClearZeroPosition</b>	When this bit is set, the zero offset set with szpa / szpr is deleted. However, the current position values remain the same. When the bit is set, the reference position can thus be moved at any time, for example.
28 / 1000 0000H	<b>JSatSAF</b>	JOG Stop at Spooler-Asynchronous Flag: When this bit is set in the event that an SAP flag appears in the AXST register, all axes programmed with the stop deceleration will be stopped.
29 / 2000 0000H	<b>InhibitProfile Refuse</b>	Usually, interpolation positioning profiles without traverse distance or with velocity/acceleration = 0 are automatically rejected and an error message in the fwsetup monitor screen is generated. Using this bit, the output of an error message during the rejection of positioning profiles can be disabled.
Following bits		Currently not assigned, reserved for future use



### 6.3.2 Axis specifiers

The various axis channels are referenced with a symbolic name. You can choose these names quite freely in the *mcf*.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 37: Axis qualifiers

Name	Type	Abbr. meaning	Function
<b>an</b>	integer	axis number	The axis qualifier an contains the axis number of the axis designator involved. The qualifier can be used in relation with „variable“ axis name. [Chapter 5.3.3.1]
<b>aux</b>	double	Auxiliary Register	The content of this register is dependent from the option. If the system includes the optionEV (Encoder Verification), the encoder count by stepper motor systems can be accessed through this variable. In this case the unit of the register is Counts.
<b>axst</b>	integer	axis status	Error, state and profile flags (wordwise)
<b>digi</b>	integer	digital inputs	Digital inputs of the xPCI-800x (wordwise) Various flags of this register can be erased by assigning any desired value to this register [chapter 4.4.51.1].
<b>digo</b>	integer	digital outputs	Digital outputs of the xPCI-800x (wordwise)
<b>dp</b>	double	desired position	Setpoint position of the axis channel
<b>dpoffset</b>	double	desired position offset	In this register, a position offset for the position controller can be entered in the axis-specific user unit. This register can be used for a cascade control e.g. by steppers with encoder verification. This register is available for stepper systems from the version 2.5.2.23 of RWMOS, for servo systems from the version 2.5.2.29 of RWMOS.
<b>dv</b>	double	desired velocity	Setpoint velocity of the axis channel
<b>dvoffset</b>	double	desired velocity offset	In this register, a velocity offset of the position offset (dpoffset) for the position controller can be entered in the axis-specific user unit.
<b>effradius</b>	double	Effektiv Radius	When rotary axes are involved in translatory interpolation travels: axis-specific parameter for conversion of rotatory values in translatory ones, (Surface area processing) [Chapter 2.3.4]
<b>epc</b>	integer	EEPROM programming cycles	Number of programming cycles
<b>gcr</b>	integer	gear configuration register	With this register, the Gear functionality of the APCI-8001 can be controlled. This register is also described in the manual “Resource Interface”.
<b>gf</b>	double	gear factor	The axis-specific gear factor can be accessed using this variable. An assignment to this value may only be made in special cases.
<b>ifs</b>	integer	interface status	Interface status flags of the xPCI-800x (wordwise) Various flags of this register can be erased by assigning any desired value to this register [see chapter 4.4.69.1].
<b>hac</b>	double	home acceleration	Acceleration for <i>home</i> commands
<b>hvl</b>	double	home velocity	Velocity for <i>home</i> commands
<b>ipw</b>	double	In position window	Position-dependent target window
<b>jac</b>	double	jog acceleration	Acceleration for <i>jog</i> commands
<b>jerkrel</b>	double	Jerk Relativ	Parameter for S velocity profile
<b>jovr</b>	double	jog override	Velocity factor
<b>jttl</b>	double	jog target velocity	Target velocity for <i>jog</i> commands
<b>jvl</b>	double	jog velocity	Velocity for <i>jog</i> commands
<b>kd</b>	double		PIDF filter coefficient for differentiation
<b>kfca</b>	double		PIDF filter coefficient for forward compensation for acceleration
<b>kfcv</b>	double		PIDF filter coefficient for forward compensation for velocity
<b>ki</b>	double		PIDF filter coefficient for integration

Name	Type	Abbr. meaning	Function
<b>kp</b>	double		PIDF filter coefficient for amplification
<b>kpl</b>	double		PIDF filter coefficient for add. phase lead
<b>lp</b>	double	latched position	latched position value
<b>lpndx</b>	double	latched position index	latched position value with index signal (zero track)
<b>lsm</b>	integer	left spool memory	free spool area [Bytes]
<b>maxacc</b>	double	maximum acceleration	Axis-specific maximum acceleration in ChkMaxVel mode
<b>maxvel</b>	double	maximum velocity	Axis-specific maximum velocity in ChkMaxVel mode
<b>mcis</b>	integer	Move Commands in Spooler	This register shows how many traverse commands are currently included in the spooler. Thus the processing state of the spooler is checked. This information can be used, when the current process must be continued after interruption (see also PCAP command rdMCiS).
<b>mcp</b>	integer	Motor Command Port	Servomotors: Setpoint value for analogue port stepper motors: Stepper signal for stepper motor performance end levels. Additional description of the commands: wrmcp (chapter 4.4.152) and rdmcp (chapter 4.4.87).
<b>mdvel</b>	double	maximum velocity skip	Axis-specific maximum velocity jump in Look-ahead mode
<b>mpe</b>	double	maximum position error	Maximum permitted position error
<b>poserr</b>	double	position error	The current axis-specific position error in the user unit is shown in this register. This is the value $dp - rp$ calculated in real-time.
<b>pprev</b>	double	Pulses Per Revolution	The number of encoder pulses per revolution (drive side) can be read in this register. On linear axes or stepper motor axes, 0 is returned here.
<b>rp</b>	double	real position	Actual position of the axis channel
<b>rv</b>	double	real velocity	Actual velocity of the axis channel [Chapter 4.4.95], can only be read not assigned
<b>sdec</b>	double	stop deceleration	Stop deceleration of the axis channel
<b>sf</b>	integer	special function	Application-specific register.
<b>sll</b>	double	Software limit left	Left software limit
<b>slr</b>	double	Software limit right	Right software limit
<b>slsp</b>	double	Slits or stepper Pulses	In this register the number of encoder slits per turn (drive side) or the number of steps per turn at stepper motors can be read or set. Quadruplication and units correspond with the values set in mcfg.
<b>tp</b>	double	target position	Target position of axis channel
<b>zerooffset</b>	double	Zero-Offset	Lately set zero point switch

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 38: Structured axis qualifiers**

Name	Type	Abbr. meaning	Function
<b>digib</b>	boolean	digital-input-bit	Digital inputs of the xPCI-800x (bitwise)
<b>digob</b>	boolean	digital-output-bit	Digital outputs of the xPCI-800x (bitwise)
<b>ifsb</b>	boolean	interface-status-bit	Status flags of the xPCI-800x (bitwise)
<b>axstb</b>	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 *rddigib()* 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 39: Abbreviations**

Name	Description
<b>A1</b>	Symbolic name for the first axis channel. This name can be freely selected in mcfg.exe. Is mainly used for examples
<b>A2</b>	Symbolic name for the second axis channel. Otherwise as for A1.
<b>Spec</b>	Axis specifier, such as A1 or A2
<b>Qual</b>	Axis qualifier, such as digi, digib, digo, digob, axst etc.
<b>Pos</b>	Position setpoint value (data type: double)
<b>Event</b>	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. Here it must be observed that the respective events are not checked anymore, if a task is terminated. If a continuous event monitoring is necessary, the respective task must stay in an endless loop.

### 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 (see MCFG / Chapter 1.7.2.5). 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
    CIO := 999;           // Common Variable
                           // signals program abort
    abort;                // Abort application program
end;

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

```

### 6.4.2 Event procedure 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 (MCFG / Chapter 1.7.2.5).

### 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\_SMD, LSL\_TOM, LSL\_SMA, LSL\_SMD, LSR\_TOM, LSR\_SMA or LSR\_SMD function is activated (MCFG / Chapter 1.7.2.5).

#### 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 (MCFG / Chapter 1.7.2.5).

#### 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* (MCFG / Chapter 1.7.2.1.9)

#### 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 (MCFG / Chapter 1.7.2.5). 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:

Procedure name	Priority
EVEO	highest priority
EVDNR	↓
EVLSH	
EVLSS	
EVMPE	
EVUI	lowest priority

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:

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

### 6.6.2 ABORT, abort

<b>DESCRIPTION:</b>	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> .
<b>NOTE:</b>	After the command has been executed, the enabled EVENT handler procedures will no longer be processed.
<b>EXAMPLE:</b>	<i>ABORT;</i>

### 6.6.3 ABS, absolute function

<b>DESCRIPTION:</b>	The function returns the absolute value of <i>value</i> .
<b>DECLARATION:</b>	<code>abs(value:double)</code>
<b>RESULT TYPE:</b>	double
<b>EXAMPLE:</b>	<pre>... var     d1, d2: double; ... d1 := -5.0; d2 := ABS(d1);           // d2 := 5.0</pre>

### 6.6.4 ACOS, arc cosine function

<b>DESCRIPTION:</b>	The function returns the arc cosine of <i>value</i> . The argument <i>Value</i> must lie within the range [-1..+1]. The return value has the unit rad and lies within the limits [0..pi].
<b>DECLARATION:</b>	<code>acos(value:double)</code>
<b>RESULT TYPE:</b>	double

### 6.6.5 ASIN, arc sine function

<b>DESCRIPTION:</b>	The function returns the arc sine of <i>value</i> . The argument <i>Value</i> must lie within the range [-1..+1]. The return value has the unit rad and lies within the limits [-pi/2..+pi/2].
<b>DECLARATION:</b>	<code>asin(value:double)</code>
<b>RESULT TYPE:</b>	Double

### 6.6.6 ATAN, arc tangent function

<b>DESCRIPTION:</b>	The function returns the arc tangent of <i>value</i> . The return value has the unit rad and lies within the limits [-pi/2..+pi/2].
<b>DECLARATION:</b>	<code>atan(value:double)</code>
<b>RESULT TYPE:</b>	Double

### 6.6.7 AZO, activate zero offsets

<b>DESCRIPTION:</b>	PCAP command <code>azo()</code>
<b>FUNCTION PARAMETERS:</b>	Integer constant in the value range of 0..4
<b>EXAMPLE:</b>	<pre>const Offsets1 = 1;  azo(Offsets1); // Activate zero offsets Set 1</pre>



### 6.6.8 CL, close loop

<b>DESCRIPTION:</b>	PCAP command <i>cl()</i> [chapter 4.4.6]
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>CL(A1, A2);    // Bring Axis Channels 1 and 2 into position control</i>

### 6.6.9 CLV

<b>DESCRIPTION:</b>	PCAP command <i>clv()</i> [Kapitel 4.4.9]
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SIMULTANEOUS FUNCTION:</b>	Ja
<b>EXAMPLE:</b>	<i>clv (A1, A2);    // Bring axis channels 1 and 2 into position contro</i> <i>js (A1, A2);    // then stop the axes immediately</i>

### 6.6.10 CONTCNCT, continue CNC-Task

<b>DESCRIPTION:</b>	This command continues the CNC task transferred in the parameter.
<b>FUNCTION PARAMETERS:</b>	Integer constant in the range of 0..3
<b>NOTE:</b>	The command can be used to continue a stopped SAP program. An SAP program which has been stopped with the SAP command <i>ABORT</i> can only be <u>restarted</u> (i.e. not continued) with the SAP command <i>STARTCNCT()</i> or the PCAP command <i>startcnct()</i> . Automatic continuation of stopped tasks is not possible either.
<b>EXAMPLE:</b>	<i>...</i> <i>const</i> <i>        TASK0 = 0;</i> <i>...</i> <i>CONTCNCT(TASK0);            // continue Task 0</i> <i>CONTCNCT(1);                // continue Task 1</i>

### 6.6.11 COS, cosine function

<b>DESCRIPTION:</b>	The function returns the cosine of <i>value</i> . The argument <i>Value</i> is interpreted as an angle in the unit rad ( $0..2\pi = 0..360$ ) degrees.
<b>DECLARATION:</b>	<i>cos(value:double)</i>
<b>RESULT TYPE:</b>	Double
<b>NOTE:</b>	<i>Sin()</i> , <i>Tan()</i> -function
<b>EXAMPLE:</b>	<i>...</i> <i>var</i> <i>        d1, d2: double;</i> <i>...</i> <i>d1 := 3.1415;</i> <i>d2 := COS(d1); // d2 := -1.0 (rounded)</i>

### 6.6.12 COSH, hyperbolic cosine function

<b>DESCRIPTION:</b>	The function returns the hyperbolic cosine of <i>value</i> .
<b>DECLARATION:</b>	<code>cos(value:double)</code>
<b>RESULT TYPE:</b>	Double

### 6.6.13 DISEV, disable event

<b>DESCRIPTION:</b>	disables the event handler specified
<b>FUNCTION PARAMETERS:</b>	<i>Event</i>
<b>REFERENCES:</b>	Chapter 6.4 and SAP command <i>ENEV()</i>
<b>EXAMPLE:</b>	<code>DISEV(EVEO);                   // ignore emergency out handler</code>

### 6.6.14 ENEV, enable event

<b>DESCRIPTION:</b>	enables the event handler specified.
<b>FUNCTION PARAMETERS:</b>	<i>Event</i>
<b>REFERENCES:</b>	Chapter 6.4 and SAP command <i>DISEV()</i>
<b>EXAMPLE:</b>	<code>ENEV(EVEO);                   // enable emergency out handler</code>
<b>NOTE:</b>	The released event-handler is not active anymore if the task is terminated or stopped.

### 6.6.15 EXP, exponential function

<b>DESCRIPTION:</b>	The function returns the value $e^{value}$ , where <i>e</i> is the base of the natural logarithm (2.718281...).
<b>DECLARATION:</b>	<code>exp(value:double)</code>
<b>RESULT TYPE:</b>	Double
<b>NOTE:</b>	Function <i>Ln()</i>

### 6.6.16 JA, jog absolute

<b>DESCRIPTION:</b>	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 <i>jac</i> to the velocity <i>jvl</i> and moved to the specified target position <i>Pos</i> . In addition, you can use the <i>jtl</i> parameter to specify a target velocity. The trajectory parameters are specified in the axis-specific units.
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i> and <i>Pos</i>
<b>SYSTEM PARAMETERS:</b>	Qualifier: <i>jac</i> , <i>jvl</i> and <i>jtl</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>REFERENCES:</b>	PCAP command <i>ja()</i> , SAP command <i>JAW()</i>
<b>NOTE:</b>	PCAP command <i>ja()</i>
<b>EXAMPLE:</b>	<code>JA(A1:=100.0);                   // Move Axis 1 absolutely to position 100</code> <code>JA(A1:=100.0, A2:=100.0);</code>

### 6.6.17 JAW, jog absolute waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>JA()</i> and PCAP command <i>ja()</i> , 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.	
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>	
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>jac, jvl</i> and <i>jtv</i>	
<b>SIMULTANEOUS FUNCTION:</b>	Yes	
<b>REFERENCES:</b>	<i>JA</i>	
<b>NOTE:</b>	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.	
<b>EXAMPLE:</b>	<pre>JAW(A2:=-1000.0);           // Move Axis 1 absolutely to Position -1000.0 and                              // wait until the profile end is reached JAW(A1:=1e3, A2 := 1.3e4);</pre>	

### 6.6.18 JHI, jog home index

<b>DESCRIPTION:</b>	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 <i>Pos</i> is exceeded.	
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>	
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>hac</i> and <i>hvl</i>	
<b>SIMULTANEOUS FUNCTION:</b>	Yes	
<b>REFERENCES:</b>	PCAP command <i>jhi()</i> , SAP command <i>JHIW()</i>	
<b>EXAMPLE:</b>	<i>JHI(A1 := 1.0, A2 := 1.5);</i> // Start reference search run for axes 1 and 2.	

### 6.6.19 JHIW, jog home index waiting

<b>DESCRIPTION:</b>	This command is identical to PCAP command <i>jhi()</i> and SAP command <i>JHI()</i> . In addition, the system waits for the profile end of the axes involved.	
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>	
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>hac</i> and <i>hvl</i>	
<b>SIMULTANEOUS FUNCTION:</b>	Yes	
<b>NOTE:</b>	SAP command <i>JA()</i>	
<b>EXAMPLE:</b>	<i>JHIW(A1 := 5.0);</i>	

### 6.6.20 JHL, jog home left

<b>DESCRIPTION:</b>	The reference search run on a digital input planned with REF for all selected axis channels is started towards the left traversing direction.	
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>	
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>hac</i> and <i>hvl</i>	
<b>SIMULTANEOUS FUNCTION:</b>	Yes	
<b>REFERENCES:</b>	PCAP command <i>jhl()</i> , SAP command <i>JHLW()</i>	
<b>EXAMPLE:</b>	<i>JHL(A1);</i>	

### 6.6.21 JHLW, jog home left waiting

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>jhl()</i> and SAP command <i>JHL()</i> . In addition, the system waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>hac</i> and <i>hvl</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>REFERENCES:</b>	PCAP command <i>jhl()</i> , SAP command <i>JHL()</i>
<b>NOTE:</b>	SAP command <i>JA()</i>
<b>EXAMPLE:</b>	<i>JHLW(A2);</i>

### 6.6.22 JHR, jog home right

<b>DESCRIPTION:</b>	The reference search run on a digital input planned with REF for all selected axis channels is started towards the right traversing direction.
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>hac</i> and <i>hvl</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>REFERENCES:</b>	PCAP command <i>jhr()</i> , SAP command <i>JHRW()</i>
<b>EXAMPLE:</b>	<i>JHR(A2);</i>

### 6.6.23 JHRW, jog home right waiting

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>jhr()</i> and SAP command <i>JHR()</i> . In addition, the system waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>hac</i> and <i>hvl</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>NOTE:</b>	SAP command <i>JA()</i>
<b>EXAMPLE:</b>	<i>JHRW(A1);</i>

### 6.6.24 JR, jog relative

<b>DESCRIPTION:</b>	For description, please consult PCAP command <i>jr()</i> .
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>jac</i> , <i>jvl</i> and <i>jtv</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>JR(A1 := 100);</i>

### 6.6.25 JRW, jog relative waiting

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>jr()</i> and the SAP command <i>JR()</i> . In addition, the system waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>jac</i> , <i>jvl</i> and <i>jtv</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>REFERENCES:</b>	JR

### 6.6.26 JS, jog stop

<b>DESCRIPTION:</b>	For description, please consult PCAP command <i>js()</i> .
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>sdec</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>JS(A1);</i>

### 6.6.27 JSW, jog stop waiting

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>js()</i> and the SAP command <i>JS()</i> . In addition, the system waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>sdec</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>JSW(A1);</i>

### 6.6.28 LN, natural logarithm function

<b>DESCRIPTION:</b>	The function returns the natural logarithm of <i>value</i> , i.e. the power by which the constant 2.71828... must be raised to obtain <i>value</i> .
<b>DECLARATION:</b>	<i>ln(value:double)</i>
<b>RESULT TYPE:</b>	Double
<b>NOTE:</b>	Values smaller than/equal to 0.0 for <i>value</i> are not defined mathematically. In this case the function has no valid return value. Function <i>Exp()</i>

### 6.6.29 LPR, latch position registers

<b>DESCRIPTION:</b>	Start the data recording of an motion process for one axis (see graphical system analysis in mcfg).
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEMPARAMETER:</b>	PU, TU, LST, LET
<b>SIMULTANEOUS FUNCTION:</b>	No
<b>EXAMPLE:</b>	<i>LPR (A1);</i>

### 6.6.30 LPRS, latch position registers synchronous

<b>DESCRIPTION:</b>	Start the synchronous data recording of an motion process for several axes (see graphical system analysis in mcfg).
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEMPARAMETER:</b>	PU, TU, LST, LET
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>LPRS (A1, A2, A3);</i>

### 6.6.31 MCA, move circular absolute - SMCA, spool motion circular absolute

<b>DESCRIPTION:</b>	PCAP command <i>mca()</i> , <i>smca()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI</i>
<b>EXAMPLE:</b>	<i>MCA(A1 := 50.0, A2 := 0.0, PHI := 720.0);</i> <i>SMCA(A1 := 0.0, A2 := 10.0, PHI := 0.1);</i>

### 6.6.32 MCAW, move circular absolute waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MCA()</i> , except that here the system also waits for the profile end of the two axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI</i>
<b>REFERENCES:</b>	PCAP command <i>mca()</i>

### 6.6.33 MCA3D, move circular absolute three-dimensional SMCA3D, spool move circular absolute three-dimensional

<b>DESCRIPTION:</b>	PCAP command <i>mca3d()</i> , <i>smca3d()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3</i>
<b>EXAMPLE:</b>	<i>MCA3D(A1 := 50.0, A2 := 0.0, A3 := 0.0, PN1 = 1.0, PN2 = 0.0, PN3 = 1.0, PHI := 720.0);</i> <i>// Circle rotated by 45 degrees around A2</i>

### 6.6.34 MCA3DW, move circular absolute three-dimensional waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MCA3D()</i> , except that here the system also waits for the profile end of the two axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3</i>
<b>REFERENCES:</b>	SAP command <i>mca3d()</i>

### 6.6.35 MCR3D, move circular relative three-dimensional SMCR3D, spool move circular relative three-dimensional

<b>DESCRIPTION:</b>	PCAP command <i>mcr3d()</i> , <i>smcr3d()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3</i>
<b>EXAMPLE:</b>	<i>MCR3D(A1 := 50.0, A2 := 0.0, A3 := 0.0, PN1 = 1.0, PN2 = 0.0, PN3 = 1.0, PHI := 720.0);</i> <i>// Circle rotated by 45 degrees around A2</i>

### 6.6.36 MCR3DW, move circular relative three-dimensional waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MCR3D()</i> , except that here the system also waits for the profile end of the two axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3</i>
<b>REFERENCES:</b>	SAP command <i>mcr3d()</i>

### 6.6.37 MCR, move circular relative - SMCR, spool motion circular relative

<b>DESCRIPTION:</b>	PCAP command <i>mcr()</i> , <i>smcr()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI</i>
<b>EXAMPLE:</b>	<i>MCR(A1 := 50.0, A2 := 0.0, PHI := 360.0);</i> <i>SMCR(A1 := 0.0, A2 := 10.0, PHI := 45.0);</i>

### 6.6.38 MCRW, move circular relative waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MCR()</i> , except that here the system also waits for the profile end of the two axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI</i>

### 6.6.39 MHA, move helical absolute - SMHA, spool motion helical absolute

<b>DESCRIPTION:</b>	PCAP command <i>mha()</i> , <i>smha()</i> <i>If the circle is to be defined by the target point, the target coordinates are to be allocated to the axis specifiers and the center point coordinates to the system parameters DTCA1 and DTCA2. If the circle is specified by the traverse angle, the center point coordinates are allocated to the axis specifiers of the circle axes.</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)</i>
<b>EXAMPLE:</b>	<i>MHA(A1 := 50.0, A2 := 0.0, PHI := 720.0, A3 := 10);</i> <i>SMHA(A1 := 0.0, A2 := 10.0, PHI := 0.1, A3 := 10);</i> <i>// Circle in anticlockwise direction with Radius 10</i> <i>MHR(A1 := 0.0, A2 := 0.0, PHI := 0.0, A3 := 10, DTCA1 := -10, DTCA2 := 0);</i> <i>// Semi-circle in clockwise direction with Radius 10</i> <i>SMHR(A1 := 20.0, A2 := 0.0, PHI := -1e-100, A3 := 10, DTCA1 := 10, DTCA2 := 0);</i>

### 6.6.40 MHAW, move helical absolute waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MHA()</i> , except that here the system also waits for the profile end of all axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)</i>

### 6.6.41 MHR, move helical relative - SMHR, spool motion helical relative

<b>DESCRIPTION:</b>	PCAP command <i>mhr()</i> , <i>smhr()</i> Here the target point cannot be specified to run the circle.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)</i>

### 6.6.42 MHRW, move helical relative waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>SAP</i> command <i>MHR()</i> , except that here the system also waits for the profile end of all axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL, PHI, (if necessary DTCA1, DTCA2)</i>

### 6.6.43 MLA, move linear absolute - SMLA, spool motion linear absolute

<b>DESCRIPTION:</b>	description is provided at the PCAP commands <i>mlla()</i> or <i>smla()</i> .
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>MLA(A1:=1000.0, A2:=3.2e2);</i> <i>SMLA(A1:=100.0, A2:=-335.0);</i>

### 6.6.44 MLAW, move linear absolute waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MLA()</i> , except that here the system also waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>MLAW(A1:=-0.3e3, A2:=100.4);</i>

### 6.6.45 MLR, move linear relative - SMLR, spool motion linear relative

<b>DESCRIPTION:</b>	The description is provided at the PCAP commands <i>mlr()</i> or <i>smlr()</i> .
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>MLR(A1:=2000.0, A2:=3.2e2);</i> <i>SMLR(A1:=300.0, A2:=-35.3);</i>



### 6.6.46 MLRW, move linear relative waiting

<b>DESCRIPTION:</b>	This command is identical to the SAP command <i>MLRW()</i> , except that here the system also waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SYSTEM PARAMETERS:</b>	<i>TRAC, TRVL, TRTVL</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>MLRW(A1:=-3.45e3, A2:=100.4e-1);</i>

### 6.6.47 MS, motion stop

<b>DESCRIPTION:</b>	The description is provided at the PCAP command <i>ms()</i> [Chapter 4.4.39].
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	None
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>MS(A1, A2);</i>

### 6.6.48 MSW, motion stop waiting

<b>DESCRIPTION:</b>	This command is identical to the PCAP command <i>ms()</i> and SAP command <i>MS()</i> , except that here the system also waits for the profile end of the axes involved.
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	None
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>MSW(A1, A2);</i>

### 6.6.49 OL, open loop

<b>DESCRIPTION:</b>	PCAP command <i>ol()</i> [Chapter 4.4.41]
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>OL(A1, A2);     // Open position control loop of A1 and A2</i>

### 6.6.50 POWER

<b>DESCRIPTION:</b>	The function returns the value of <i>base</i> to the <i>exponent</i> .
<b>DECLARATION:</b>	<i>sqrt(base, exponent : double)</i>
<b>RESULT TYPE:</b>	double
<b>NOTE:</b>	Function is available from RWMOS.ELF V2.5.3.93
<b>EXAMPLE:</b>	<pre>... var     d1, d2: double;  ... d1 := 2.0; d2 := 3.0; d2 := POWER(d1, d2); // d2 := 8.0</pre>

### 6.6.51 RA, reset axis

<b>DESCRIPTION:</b>	PCAP command <i>ra()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>RA(A1, A2);     // Reset axes A1 and A2</i>

### 6.6.52 RDCBD, read COMMON BUFFER double function

<b>DESCRIPTION:</b>	The function returns a floating-point value with double accuracy from the CNC-task-specific COMMON BUFFER. The <i>offset</i> 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 xPCI-800x board CPU system to access this correctly, <i>offset</i> must always be word-oriented, i.e. have a value which is divisible by 8.
<b>DECLARATION:</b>	<i>RDCBD(offset:integer)</i>
<b>RESULT TYPE:</b>	double
<b>NOTE:</b>	The CNC-task-specific buffer size is <u>1,000 bytes</u> . PCAP commands <i>rdcbcnct()</i> and <i>wrcbcnct()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>
<b>EXAMPLE:</b>	... var <i>cbd: double;</i> ... <i>cbd := RDCBD(500);             // Read in double variable from offset 500</i>

### 6.6.53 RDCBI, read COMMON BUFFER integer function

<b>DESCRIPTION:</b>	The function returns an integer value from the CNC-task-specific COMMON BUFFER. The <i>offset</i> 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 xPCI-800x board CPU system to access this correctly, <i>offset</i> must always be word-oriented, i.e. have a value which is divisible by 4.
<b>DECLARATION:</b>	<i>RDCBI(offset:integer)</i>
<b>RESULT TYPE:</b>	integer
<b>NOTE:</b>	The CNC-task-specific buffer size is <u>1,000 bytes</u> . PCAP commands <i>rdcbcnct()</i> and <i>wrcbcnct()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i> .
<b>EXAMPLE:</b>	... var <i>cbi: integer;</i> ... <i>cbi := RDCBI(500);             // Read in integer variable from offset 500</i>

### 6.6.54 RDCBS, read COMMON BUFFER single function

<b>DESCRIPTION:</b>	The function returns a floating-point value with single accuracy from the CNC-task-specific COMMON BUFFER. The <i>offset</i> 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 xPCI-800x board CPU system to access this correctly, <i>offset</i> must always be word-oriented, i.e. have a value which is divisible by 4.
<b>DECLARATION:</b>	RDCBS( <i>offset</i> :integer)
<b>RESULT TYPE:</b>	single
<b>NOTE:</b>	The CNC-task-specific buffer size is <u>1,000 bytes</u> . PCAP commands <i>rdcbcnc()</i> and <i>wrcbcnc()</i> , SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>
<b>EXAMPLE:</b>	... var <i>cbs</i> : single; ... <i>cbs</i> := RDCBS(500);                   // Read in single variable from offset 500

### 6.6.55 RPTODP, Real-Position to Desired-Position

<b>DESCRIPTION:</b>	PCAP command <i>RPToDP()</i>
<b>NOTE:</b>	The relevant axes must not be in a positioning profile, i.e. the profile end flag in the axis status register must be set.
<b>EXAMPLE:</b>	<i>RPTODP</i> (X, Z);

### 6.6.56 RS, reset system

<b>DESCRIPTION:</b>	PCAP command <i>rs()</i>
<b>NOTE:</b>	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.
<b>EXAMPLE:</b>	<i>RS</i> ;     // reset complete axis system

### 6.6.57 SHP, set home position

<b>DESCRIPTION:</b>	PCAP command <i>shp()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i> , <i>Pos</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<i>SHP</i> (A2:=1000.0);

### 6.6.58 SIN, sine function

<b>DESCRIPTION:</b>	The function returns the sine of <i>value</i> . The argument <i>Value</i> is interpreted as an angle in the unit rad ( $0..2\text{Pi} = 0..360$ ) degrees.
<b>DECLARATION:</b>	<code>sin(value:double)</code>
<b>RESULT TYPE:</b>	double
<b>NOTE:</b>	<i>Cos()</i> , <i>Tan()</i> function
<b>EXAMPLE:</b>	<pre>... var     d1, d2: double;  ... d1 := 3.1415; d2 := SIN(d1);           // d2 := 0.0 (rounded)</pre>

### 6.6.59 SINH, hyperbolic sine function

<b>DESCRIPTION:</b>	The function returns the hyperbolic sine of <i>value</i> .
<b>DECLARATION:</b>	<code>Cos(value:double)</code>
<b>RESULT TYPE:</b>	double

### 6.6.60 SQR, square function

<b>DESCRIPTION:</b>	This function returns the square of <i>value</i> .
<b>DECLARATION:</b>	<code>sqr(value:double)</code>
<b>RESULT TYPE:</b>	double
<b>NOTE:</b>	Available only in RWMOS and compiler versions from 05.10.2007
<b>EXAMPLE:</b>	<pre>... var     d1, d2: double;  ... d1 := 9.0; d2 := SQR(d1); // d2 := 81.0</pre>

### 6.6.61 SQRT, square root function

<b>DESCRIPTION:</b>	The function returns the square root of <i>value</i> .
<b>DECLARATION:</b>	<code>sqrt(value:double)</code>
<b>RESULT TYPE:</b>	double
<b>NOTE:</b>	Negative values of <i>value</i> are not defined mathematically. In this case, the function does not have a valid return value.
<b>EXAMPLE:</b>	<pre>... var     d1, d2: double;  ... d1 := 9.0; d2 := SQRT(d1); // d2 := 3.0</pre>



### 6.6.65 STARTCNCT, start CNC-Task

<b>DESCRIPTION:</b>	This command starts the CNC task transferred in the parameter and executes the SAP program stored there from its beginning.
<b>FUNCTION PARAMETERS:</b>	Integer-constant in range 0..3
<b>NOTE:</b>	An SAP program can also start itself automatically from the beginning with this command.
<b>EXAMPLE:</b>	<pre>... const     Task1 = 1; ... STARTCNCT(Task1);</pre>

### 6.6.66 STOP, stop

<b>DESCRIPTION:</b>	<p>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.</p> <p>The application program can be resumed by means of the <i>contcnct()</i>-PCAP command, the <i>CONTCNCT()</i>-SAP command or in the TOOLSET program <i>mcfg.exe</i>.</p>
<b>NOTE:</b>	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.
<b>EXAMPLE:</b>	<i>STOP; // Stops the SAP program</i>

### 6.6.67 STEPCNCT, stop CNC-Task

<b>DESCRIPTION:</b>	This command executes a program line in the indicated CNC task.
<b>FUNCTION PARAMETER:</b>	Integer constant in the range 0..3
<b>NOTE:</b>	EVENT handling procedures that were possibly released, will not be processed anymore after executing the program line. Before the execution of the command, a valid program must be loaded. See also PCAP command <i>stepcnct</i> (chapter 4.4.116).
<b>EXAMPLE:</b>	<pre>... const     Task3 = 3; ...  STEPCNCT(Task3);</pre>

### 6.6.68 STOPCNCT, stop CNC-Task

<b>DESCRIPTION:</b>	This command halts the CNC task transferred in the parameter and thus halts the SAP program stored in it as well.
<b>FUNCTION PARAMETERS:</b>	Integer constant in the range 0..3
<b>NOTE:</b>	Any EVENT handling procedures enabled will no longer be processed by the correspondingly selected task after executing <i>STOPCNCT()</i> . See also chapter 6.6.66.
<b>EXAMPLE:</b>	<pre>... const     Task3 = 3; ...  STOPCNCT(Task3);</pre>

### 6.6.69 STOPTOSS,

<b>DESCRIPTION:</b>	This command transfers the CNC-task, which has been transferred in the parameter, from the stop-state to the step-state, however without executing a program line in the indicated task.
<b>FUNCTION PARAMETERS:</b>	Integer constant in the range 0..3
<b>NOTE:</b>	If the indicated task is not in the stop-mode, the command has no influence. This command is required especially for the single-step processing by using several SAP programming tasks.
<b>EXAMPLE:</b>	<pre>... const     Task3 = 3; ...  STOPTOSS(Task3);</pre>

### 6.6.70 SZPA – Set Zero Position Absolut

<b>DESCRIPTION:</b>	Set a virtual zero position. The command is described at the PCAP command <i>szpa</i> (Chapter 4.4.118). You can use SZPA for stepper motor systems only from the version 2.5.2.32 of RWMOS.ELF.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>REFERENCES:</b>	PCAP command <i>szpa()</i> , <i>szpr()</i> , SAP command <i>SZPR</i>
<b>EXAMPLE:</b>	SZPA (X := 100, Y := -20);

### 6.6.71 SZPR – Set Zero Position Relativ

<b>DESCRIPTION:</b>	Set the virtual zero position in a relative position. This command described at the PCAP command <i>szpr</i> (Chapter 4.4.119). You can use SZPR bei Schrittmotorsystemen for stepper motor systems only from the 2.5.2.32 of RWMOS.ELF.
<b>FUNCTION PARAMETERS:</b>	<i>Spec, Pos</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>REFERENCES:</b>	PCAP command <i>szpa()</i> , <i>szpr()</i> , SAP command <i>SZPA</i>
<b>EXAMPLE:</b>	SZPR (X := 100, Y := -20);

### 6.6.72 TAN, tangent function

<b>DESCRIPTION:</b>	The function returns the tangent of <i>value</i> . The argument <i>Value</i> is interpreted as an angle in the unit rad ( $0..2\pi = 0..360$ ) degrees.
<b>DECLARATION:</b>	<code>tan(value:double)</code>
<b>RESULT TYPE:</b>	Double
<b>NOTE:</b>	<i>Sin()</i> , <i>Cos()</i> function
<b>EXAMPLE:</b>	<pre>... var     d1, d2: double; ... d1 := 0.5; d2 := TAN(d1); // d2 := 0.5463 (rounded)</pre>

### 6.6.73 TANH, hyperbolic tangent function

<b>DESCRIPTION:</b>	The function returns the hyperbolic tangent of <i>value</i> .
<b>DECLARATION:</b>	<code>tan(value:double)</code>
<b>RESULT TYPE:</b>	Double

### 6.6.74 UF, update filter

<b>DESCRIPTION:</b>	PCAP command <i>uf()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	Qualifiers: <i>kp, ki, kd, kpl, kfca, kfcv</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>NOTE:</b>	For updating the PIDF filter coefficients, all the qualifiers listed above must be initialized before executing the command.
<b>EXAMPLE:</b>	<pre>... 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); ...</pre>



### 6.6.75 UTROVR, update trajectory override

<b>DESCRIPTION:</b>	PCAP command <i>utrovr()</i>
<b>FUNCTION PARAMETERS:</b>	<i>Spec</i>
<b>SYSTEM PARAMETERS:</b>	<i>TROVR</i>
<b>SIMULTANEOUS FUNCTION:</b>	Yes
<b>EXAMPLE:</b>	<pre>... TROVR := 0.9;           // Trajectory velocity override = -10% UTROVR(A1, A2);         // Reduced trajectory velocity for axes A1 and A2 ...</pre>

### 6.6.76 WRCBI, write COMMON BUFFER integer procedure

<b>DESCRIPTION:</b>	<p>The procedure describes a memory location of the integer type with the value of <i>value</i> in the CNC-task-specific COMMON BUFFER. The <i>offset</i> parameter is a byte offset referenced to the first element (Element 0) of the COMMON BUFFER.</p> <p>The integer data type occupies 4 bytes in the COMMON BUFFER. To enable the xPCI-800x board CPU system to access this correctly, <i>offset</i> must always be word-oriented, i.e. have a value which is divisible by 4.</p>
<b>DECLARATION:</b>	<code>WRCBI(offset:integer; value:integer)</code>
<b>NOTE:</b>	<p>The CNC-task-specific buffer size is <u>1,000 bytes</u>.</p> <p>PCAP commands <i>rdcbcnct()</i> and <i>wrcbcnct()</i>, SAP commands <i>RDCBx()</i> and <i>WRCBx()</i></p>
<b>EXAMPLE:</b>	<pre>WRCBI(500, -1000);      // Write integer variable from offset 500                         // with value -1000</pre>

### 6.6.77 WRCBS, write COMMON BUFFER single procedure

<b>DESCRIPTION:</b>	<p>The procedure describes a memory location of the single type (floating-point number with single accuracy) with the value of <i>value</i> in the CNC-task-specific COMMON BUFFER. The <i>offset</i> parameter is a byte offset referenced to the first element (Element 0) of the COMMON BUFFER.</p> <p>The single data type occupies 4 bytes in the COMMON BUFFER. To enable the xPCI-800x board CPU system to access this correctly, <i>offset</i> must always be word-oriented, i.e. have a value which is divisible by 4.</p>
<b>DECLARATION:</b>	<code>WRCBS(offset:integer; value:single)</code>
<b>NOTE:</b>	<p>The CNC-task-specific buffer size is <u>1,000 bytes</u>.</p> <p>PCAP commands <i>rdcbcnct()</i> and <i>wrcbcnct()</i>, SAP commands <i>RDCBx()</i> and <i>WRCBx()</i>.</p>
<b>EXAMPLE:</b>	<pre>WRCBS(500, 3.99);       // Write single variable from offset 500                         // with value 3.99</pre>

### 6.6.78 WRCBD, write COMMON BUFFER double procedure

<b>DESCRIPTION:</b>	<p>The procedure describes a memory location of the "double" type (floating-point number with double accuracy) with the value of <i>value</i> in the CNC-task-specific COMMON BUFFER. The <i>offset</i> parameter is a byte offset referenced to the first element (Element 0) of the COMMON BUFFER.</p> <p>The double data type occupies 8 bytes in the COMMON BUFFER. To enable the xPCI-800x board CPU system to access this correctly, <i>offset</i> must always be word-oriented, i.e. have a value which is divisible by 8.</p>
<b>DECLARATION:</b>	WRCBD( <i>offset</i> :integer; <i>value</i> :double)
<b>NOTE:</b>	<p>The CNC-task-specific buffer size is <u>1,000 bytes</u>.</p> <p>PCAP commands <i>rdcbcnct()</i> and <i>wrcbcnct()</i>, SAP commands <i>RDCBx()</i> and <i>WRCBx()</i></p>
<b>EXAMPLE:</b>	<pre>WRCBD(500, 100.2e-128);    // Write double variable from offset 500                            // with value 100.2e-128</pre>

### 6.6.79 WRITE

<b>DESCRIPTION:</b>	Adding a partial string to the current task specific string output.
<b>FUNCTION PARAMETER:</b>	<i>Diverse</i>
<b>NOTES:</b>	<p>The function can be called with an undefined number of parameters, which can be of the type string constant, integer, double or boolean. String constants are strings that are limited in <i>rw_SymPas</i> by superior commas and in the G-Code programming by inverted commas. The single parameters are separated by commas. Numeric or boolean parameters can be also expressions. The call of this function sets Bit 0 in the system variable <i>tskinfo</i>. Information about the state of the string output see Chapter 4.4.13. The reading of the task specific output string is done with the PCAP function <i>gettskstr()</i>, see chapter 4.4.14.</p>
<b>EXAMPLE RW_SYMPAS:</b>	<pre>write ('This is a string: ', C10); write ('Istposition: ', A1.rp);</pre>
<b>EXAMPLE G-CODES:</b>	N0100 write "This a string", C10

### 6.6.80 WRITELN

<b>DESCRIPTION:</b>	Adding a partial string to the current task specific string output and concluding the output string.
<b>FUNCTION PARAMETER:</b>	<i>Diverse</i>
<b>NOTE:</b>	<p>The function can be called with an undefined number of parameters, which can be of the type string constant, integer, double or boolean. String constants are strings that are limited in <i>rw_SymPas</i> by superior commas and in the G-Code programming by inverted commas. The single parameters are separated by commas. Numeric or boolean parameters can be also expressions. If after this command <i>write</i> or <i>writeln</i> is called again, the previous output string will be overwritten. The call of this function sets Bit 1 in the system variable <i>tskinfo</i>. For information about the state of the string output, see chapter 4.4.13. The reading of the task specific output string is done with the PCAP function <i>gettskstr()</i>, see chapter 4.4.14. If Bit 26 is set in the MODEREG register (chapter 6.3.1.4), the respective CNC task will be stopped by this command.</p>
<b>EXAMPLE RW_SYMPAS:</b>	<pre>writeln ('This is a string: ', C10); writeln ('Istposition: ', A1.rp);</pre>
<b>EXAMPLE G-CODES:</b>	N0100 writeln "This is a string", C10

### 6.6.81 WT, wait timer

<b>DESCRIPTION:</b>	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.
<b>FUNCTION PARAMETERS:</b>	Integer values with a unit of 64 $\mu$ s
<b>NOTE:</b>	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 <i>WT()</i> calls with shorter wait times (perhaps in a loop).
<b>EXAMPLE:</b>	<pre>... CONST sec = 15625; ... WT(5*sec);    // Wait 5s ...</pre>

## 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

<b>DESCRIPTION:</b>	<p>This compiler command instructs the compiler to read in the file designated by <i>filename</i>. Basically, the compiler behaves as if the text read is in place of the {\$I} command. <i>rw_SymPas</i> 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.</p> <p><b>Note:</b> If in the <i>mcfg.exe</i> 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.</p>
<b>SYNTAX:</b>	{ \$I Filename }

### 6.7.2 Task selection

<b>DESCRIPTION:</b>	You can use this compiler command to specify the task ( <i>TaskNr</i> , values 0..3) in which the SAP program involved is to be run. The information is stored in the autocode file " <i>filename.cnc</i> ". The PCAP command <i>txbf()</i> is used to transfer this file automatically into the right task.
<b>SYNTAX:</b>	<code>{TASK <i>TaskNr</i>}</code>
<b>NOTE:</b>	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
<b>EXAMPLE:</b>	<pre>... const     Task1 = 1; ...  {\$TASK Task1};      // or {\$TASK 1}</pre>

### 6.7.3 Full system compiling

<b>DESCRIPTION:</b>	The command selects the compiler option FULLSYSTEM.
<b>SYNTAX:</b>	<code>{FULLSYSTEM}</code>
<b>NOTE:</b>	If the SAP program concerned does not contain this statement, the option currently selected will be utilized for compiling. This statement is to be entered at the beginning of the source text file. This command is available for mcfg from the version V2.5.2.13 and for ncc from the version V2.5.2.9.

## 6.8 SAP run time errors

When operating Stand Alone programs, different errors may occur. In this case the running task is stopped. The error number and the number of the line where the error occurred are then entered in the data structure CNCTS (See chapter 4.3.2.10). The error numbers and possible causes of error are listed below.

**Table 40: SAP run time errors**

<b>Error #</b>	<b>Description</b>
<b>1 / 0001</b>	The arithmetic operation is not correct for the used data type
<b>2 / 0002</b>	Uncorrect data type.
<b>4 / 0004</b>	Uncorrect internal operation code. This can be caused by a compatibility problem between mcfg/ncc and RWMOS.ELF.
<b>8 / 0008</b>	Stack overflow. Program too big or internal problem for the RWMOS operating system software.
<b>16 / 0010</b>	Stack underflow. This error can be an internal problem for the RWMOS operating system software.
<b>32 / 0020</b>	Unknown event handler. It can be caused by a compatibility problem between mcfg/ncc and RWMOS.ELF.
<b>64 / 0040</b>	Uncorrect NC command. This can be caused by a compatibility problem between mcfg/ncc and RWMOS.ELF
<b>128 / 0080</b>	Address injury in NC program. This error can be caused by an internal problem by the RWMOS operating system software.
<b>256 / 0100</b>	Address injury in NC program through wrong parameter settings
<b>512 / 0200</b>	Error by using the AT interface.
<b>1024 / 0400</b>	A common variable has been addressed outside the correct range.
<b>2048 / 0800</b>	Uncorrect index by double access to the common buffer (cannot be divided by 8).
<b>4096 / 1000</b>	Incorrect SAP command. This error can be caused by a compatibility problem with controllers of another generation.
<b>8192 / 2000</b>	Error in cutting speed interpolation
<b>16384 / 4000</b>	Use of non-permitted axes in interpolation with G-codes
<b>32768 / 8000</b>	Invalid parameter in arithmetic operation, e.g. mod 0