

MTX 14VRS

Programming Manual

Reference Book
R911342634

Edition 08



Title	MTX 14VRS Programming Manual
Type of Documentation	Reference Book
Document Typecode	DOK-MTX***-NC**PRO*V14-RE08-EN-P
Internal File Reference	RS-32b2b2ff1c3483ba0a6846a50090ad76-8-en-US-3
Change record	Edition 08, 2020-10 Refer to tab. 1-1 "Change record" on page 1
Copyright	© Bosch Rexroth AG 2020 All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Liability	The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.
Editorial Department	Development Software and Control Firmware - Control Functions CNC, ReLa (MiSc/MePe)

Table of Contents

	Page
1 About this documentation.....	1
1.1 Validity of the documentation.....	1
1.2 General information.....	2
1.3 Required and supplementing documentation MTX.....	3
1.3.1 Selection/compilation.....	3
1.3.2 Configuration.....	4
1.3.3 Commissioning.....	4
1.3.4 Operation.....	5
1.3.5 OEM Engineering.....	6
1.3.6 AddOns.....	6
1.4 Information representation.....	7
1.4.1 Safety instructions.....	7
1.4.2 Symbols used.....	7
1.4.3 Names and abbreviations.....	7
1.5 Customer feedback.....	8
2 Important instructions on use.....	9
2.1 Intended use.....	9
2.1.1 Introduction.....	9
2.1.2 Areas of use and application.....	9
2.2 Unintended use.....	10
3 Basics of NC programming.....	11
3.1 Introduction.....	11
3.2 Basic information on standard and CPL programming.....	12
3.2.1 Programming options.....	12
3.2.2 Highlighting CPL elements within a part program.....	13
3.2.3 CPL elements during manual data input, buffered NC block specification and non-linked sub-routines	13
3.2.4 Complying with times of interpretation between CPL and standard language elements.....	14
3.3 Linking of NC programs.....	14
3.3.1 Link table.....	14
3.3.2 Linking at program selection or at program runtime.....	15
3.3.3 Linking subroutines - Postlinking.....	16
3.3.4 Influencing the linking process using DIN/CPL IDs.....	16
3.4 Basic NC program components.....	16
3.4.1 Program Block.....	16
3.4.2 Commands.....	17
3.4.3 Additional conditions.....	17
3.5 Program words.....	18
3.5.1 Overview.....	18
3.5.2 Program words of NC functions.....	19
3.5.3 Program words as parameters.....	22
3.5.4 Programming attributes.....	22

Table of Contents

	Page
3.5.5 Using separators between two partial words	24
3.6 Program end.....	25
3.7 Effect of program words.....	25
3.7.1 Modal.....	25
3.7.2 Non-modal	26
3.8 Special program design elements for program design	27
3.8.1 Channel ID.....	27
3.8.2 Block numbers.....	27
3.8.3 Blank lines in the program code.....	28
3.8.4 Comments in a part program.....	28
3.8.5 Notes in the user interface.....	29
3.8.6 Jumps in the program run	30
3.9 Subroutines.....	31
3.9.1 Overview.....	31
3.9.2 Subroutine call with P-address.....	32
3.9.3 Subroutine Call Without P Address.....	33
3.9.4 Local subroutines.....	34
3.9.5 Self-defined subroutine calls with G and M codes	36
3.9.6 Self-defined modal subroutine calls.....	37
3.9.7 Subroutine call in CPL via the CALL command.....	38
3.9.8 Transferring parameters to subroutines.....	39
3.9.9 Configuration and error recovery.....	45
3.10 Macro programming.....	47
3.10.1 Overview.....	47
3.10.2 Macros with parameters.....	47
3.11 Label programming and jump instructions.....	48
3.11.1 Overview.....	48
3.11.2 Labels in standard NC blocks and CPL blocks.....	49
3.11.3 GoAhead (GOA) Forward jump to a standard NC block.....	50
3.11.4 GoBack (GOB) Backward jump to a standard NC block	51
3.11.5 GoCond (GOC) Conditional jump to a standard NC block.....	51
3.11.6 GoTo Unconditional jump to a standard NC block	52
3.11.7 CPL jump (GOTO) Jump to any program block.....	53
3.11.8 RetTo subroutine return to any program block.....	54
3.12 Decision and branch instructions.....	55
3.12.1 Overview.....	55
3.12.2 "Skip block" function	55
3.12.3 CPL instruction: CASE-LABEL...LABEL-OTHERWISE-ENDCASE.....	56
3.12.4 CPL instruction: IF-THEN-ELSE-ENDIF.....	57
3.13 Repeat statements.....	58
3.13.1 Overview.....	58
3.13.2 CPL instruction: FOR-STEP-TO-NEXT.....	58
3.13.3 CPL instruction: REPEAT-UNTIL.....	59
3.13.4 CPL instruction: WHILE-DO-END.....	59
3.14 Variable programming.....	60
3.14.1 Variable names.....	60

	Page
3.14.2	Variable groups..... 61
3.14.3	Variable types..... 66
3.14.4	Variables within standard NC programming..... 68
3.14.5	Variable ERRNO for evaluating errors in CPL functions "ERRNO"..... 69
3.15	General system information..... 70
3.16	Reserved command words..... 70
3.17	CPL commands..... 72
3.17.1	Value assignment..... 72
3.17.2	Mathematical operations..... 73
3.17.3	Logic operations..... 76
3.17.4	Conversion between numerical systems..... 77
3.17.5	Relational operations " =, >=, >, <>, <=, <"..... 77
3.18	More CPL basic elements..... 78
3.18.1	Constants..... 78
3.18.2	Code characters..... 78
3.19	Commands for NC block synchronization 78
3.19.1	Overview..... 78
3.19.2	Synchronization functions of block preparation..... 79
3.19.3	Synchronization functions at the block execution time..... 82
3.19.4	Writing system date at execution time..... 94
4	Overview on modal function groups..... 97
5	NC functions with syntax according to DIN 66025 (incl. extensions)..... 99
5.1	Overview..... 99
5.2	G-codes..... 101
5.2.1	Straight line interpolation in rapid traverse "G0"..... 101
5.2.2	Linear interpolation in the feed "G1" 102
5.2.3	Circular/helical/helical-N interpolation "G2, G3"..... 104
5.2.4	Dwell time "G4"..... 109
5.2.5	Tangential circle exit "G12"..... 110
5.2.6	Tangential circle entry "G5", "G13"..... 110
5.2.7	Spline programming "G6"..... 111
5.2.8	Path slope ON "G8", Path slope OFF "G9"..... 120
5.2.9	Jerk-limited velocity control "G8(SHAPE...)", "G9(SHAPE...)", "G9(ASHAPE...)", "G9(X... , Y... , ...)", "AsynchrShapeOrder, ASO"..... 121
5.2.10	No plane "G16"..... 125
5.2.11	Plane switching "G17, G18, G19"..... 126
5.2.12	Advanced plane switching "G17(...), G18(...), G19(...)"..... 127
5.2.13	Free selection of planes (independent of the WCS) "G20"..... 128
5.2.14	Contour move programming "G31" 129
5.2.15	Thread cutting "G33"..... 131
5.2.16	Milling cutter path correction "G40, G41, G42"..... 135
5.2.17	Contour transitions for milling cutter path correction: Arc "G43", Intersection "G44" Arc..... 137
5.2.18	Feed correction: Cutter contact point "G45", cutter center point "G46"..... 139

Table of Contents

	Page
5.2.19	Tool length correction "G47, G48"..... 139
5.2.20	Programmable zero offset "G52"..... 141
5.2.21	Zero offsets (ZO) "G53", "G53.1-G59.1" to "G53.5-G59.5"..... 142
5.2.22	Exact stop ON/OFF "G61, G62"..... 145
5.2.23	Tapping without compensation chuck "G63"..... 147
5.2.24	Inch programming "G70", Local inch programming "INCH(...)"..... 148
5.2.25	Metric programming "G71"..... 149
5.2.26	Approaching the reference point coordinates "G74"..... 149
5.2.27	Approaching the reference point "G74(HOME)"..... 150
5.2.28	Traveling to probe "G75"..... 150
5.2.29	Approaching the fixed machine axis position "G76"..... 152
5.2.30	Asynchronous subroutines: Repositioning individual coordinates "G77"..... 152
5.2.31	Compensation switching ON "G78", Compensation switching OFF "G79"..... 153
5.2.32	Absolute dimension programming "G90", Relative dimension programming "G91", Local absolute dimension programming "AC(...)", Local relative dimension programming "IC(...)"..... 155
5.2.33	Time programming "G93"..... 156
5.2.34	Feed programming (per minute) "G94"..... 157
5.2.35	Absolute velocity programming "G94(LINS)" with path-proportional velocity adaptation..... 158
5.2.36	Absolute velocity programming "G94(LINT)" with time-proportional velocity adaptation..... 159
5.2.37	Incremental velocity programming "G94(...)" with acceleration adaptation..... 160
5.2.38	Absolute velocity programming "G94(...)" with acceleration adaptation..... 162
5.2.39	Feed programming (per revolution) "G95"..... 163
5.2.40	Incremental velocity programming "G95(...)" with acceleration adjustment..... 164
5.2.41	Feed programming per rotary axis revolution "G95(FRA=<Axis>)"..... 166
5.2.42	Constant cutting velocity "G96", Direct speed programming "G97"..... 166
5.2.43	3D Tool radius correction "G140, G141, G142"..... 170
5.2.44	Placement: Inclined plane "G151 - G159.5"..... 173
5.3	M-codes..... 180
5.3.1	Interrupt program (program stop) "M0, M00"..... 180
5.3.2	Conditional program abort (cond. program halt) "M1, M01"..... 181
5.3.3	Exiting the program (program end) "M2, M02, M30"..... 181
5.3.4	Spindle clockwise rotation "M3, M103, M203", Spindle clockwise rotation and coolant ON "M13, M113, M213"..... 182
5.3.5	Spindle counterclockwise rotation "M4, M104, M204", Spindle counterclockwise rotation and coolant ON "M14, M114, M214"..... 183
5.3.6	Spindle stop "M5, M105, M205"..... 184
5.3.7	Orienting the spindle/positioning the spindle "M19, M119, M219"..... 184
5.3.8	Automatic gear selection "M40, M140, M240"..... 186
5.3.9	Manual gear selection "M41...44, M141...144, M241...244"..... 186
5.3.10	Gear idle selection "M48, M148, M248"..... 187
5.4	Feed and speed programming..... 187
5.4.1	Speed setting or modulo speed setting "<Axis>=S<Value>"..... 187
5.4.2	F-address "F"..... 189
5.4.3	F2-address "F2"..... 189
5.4.4	Velocity of asynchronous axes "FA"..... 190
5.4.5	Omega address (feed) "Omega"..... 191
5.4.6	Programming spindle speed "S, SSPG"..... 191

	Page
5.5 Tool correction.....	192
5.5.1 D-correction "D".....	192
5.5.2 ED compensation "ED".....	193
6 NC functions with high-level language syntax.....	195
6.1 Overview.....	195
6.2 Programming effective radius factors "ActRadFact, ARF".....	196
6.2.1 Effect.....	196
6.2.2 Programming.....	196
6.3 Adjusting modal NC functions "Adjust, ADJ".....	197
6.4 Programmable distance-dependent signal.....	200
6.4.1 General information.....	200
6.4.2 Activate search for a mark "ADSENA".....	200
6.4.3 Deactivate the search for a mark "ADSENA".....	201
6.4.4 Set mark "ADSLAB".....	201
6.4.5 Reset switching signal "ADSCLR".....	201
6.5 Extending the retrace forward program block "AdvanceAppend, ADVA".....	202
6.6 Replacing the retrace forward program block "AdvanceSubstitute, ADVS".....	203
6.7 Area monitoring "area, ARA".....	203
6.8 Asynchronous subroutines.....	209
6.8.1 Asynchronous subroutines: log off "ASPCLR".....	209
6.8.2 Asynchronous subroutines: switch off "ASPDIS".....	209
6.8.3 Asynchronous subroutines: switch on "ASPENA".....	210
6.8.4 Asynchronous subroutines: Define point of return to contour "ASPRTP".....	210
6.8.5 Asynchronous subroutines: register "ASPSET".....	211
6.8.6 Asynchronous subroutines: triggering via the program "ASPSTA".....	212
6.9 Assigning logical axis name "AssLogName, ALN".....	212
6.10 Calibrating axis kinematics.....	213
6.10.1 Backward transformation "ATBWD".....	213
6.10.2 Optimizing parameters "ATCAL".....	214
6.10.3 Forward transformation "ATFWD".....	216
6.10.4 Reading parameter from NC "ATGET".....	217
6.10.5 Writing parameters to the NC "ATPUT".....	218
6.11 Executing active auxiliary functions of all groups "AUXFUNC".....	220
6.12 Changing maximum axis acceleration "AxAcc, AAC", save maximum axis acceleration "AxAcc-Save, AAS".....	221
6.13 Loading axis error correction table "AxCompTab".....	222
6.14 Axis coupling "AxCouple, AXC".....	224
6.15 Axis feed coupling "AxisFeedCouple, AFC".....	228
6.16 Changing the maximum axis jerk "perform, AJK", save maximum axis jerk "AxJerkSave, AVS".....	231
6.17 Switching off the C-axis mode for spindles "AxisToSpindle, ATS".....	233
6.18 Switching off the C-axis mode for spindles and wait "AxisToSpindleWait, ATSW".....	233
6.19 Replacing axes "AxisReplace, AXR".....	234
6.20 Changing maximum axis velocity "AxVel, AVE", Save maximum axis velocity "AxVelSave, AVS".....	235
6.21 Parameters of the B-spline approximation "BsaPar, BAP".....	236
6.22 Placement: Workpiece position correction "BcsCorr, BCR".....	236

Table of Contents

	Page
6.23	Next block monitoring with buffered NC block input "ChkNxtBlk, CNB"..... 238
6.24	Chamfer programming "ChLength", "CHL", "CHLL", "ChSection", "CHS", "CHSL"..... 239
6.25	Automatic detection of corners and lines "CLD"..... 241
6.26	Collision monitoring "Collision, CLN"..... 242
6.27	G0 interpolation configuration "ConfG0, CG0"..... 244
6.28	Selecting the axis transformation "Coord, CRD"..... 245
6.29	Sharpening internal corners "Cornering, COR"..... 246
6.30	Spline coupling table "CoupleSplineTab, CST"..... 247
6.31	Curve parameter interpolation "CurveParamInterpolation, CPI"..... 249
6.32	Customer-specific velocity profile "CVP"..... 250
6.33	Activating D-correction tables "DcTSel, DCS"..... 250
6.34	Applying axis settings from MP "DefAxis, DAX"..... 251
6.35	Resetting channel spindles to MP "DefSpindle, DSP"..... 251
6.36	Block transition without deceleration "DefTangTrans, DTT, DTT"..... 252
6.37	Diameter programming "DiaProg, DIA", Radius programming "RadProg, RAD"..... 252
6.38	Delayed response to drive errors "DERDelay"..... 254
6.39	Dimension of the working plane "DimPlane, DimP"..... 254
6.40	Distance control for digitalization "DistCtrl, DCR"..... 256
6.41	Triggering and exiting diagnostic recording "DREC_START", "DREC_STOP"..... 257
6.42	End position coupling "EndPosCouple, EPC"..... 258
6.43	Feed computation: Hiding of axes "FeedAd, FAD"..... 259
6.44	Feed forward "FeedForward, FFW"..... 260
6.45	Feed forward for asynchronous axes "FFWAxis, FFWA"..... 262
6.46	Feed computation: Feed group "FeedGrp, FDG"..... 263
6.47	Feed computation: Feed group type "FeedGrp, FDGT"..... 264
6.48	Flying measurement "FlyMeas, FME"..... 265
6.49	Measurement at fixed stop "FsProbe, FSP"..... 266
6.50	Moving to fixed stop "FsMove, FSM", "FsTorque, FST", "FsReset, FSR"..... 268
6.51	Straightness and angle error compensation "GeoComp, GCT"..... 271
6.52	Applying the axis "GetAxis, GAX"..... 271
6.53	Creating channel spindles "GetSpindle, GSP"..... 272
6.54	Flying block transition via high-speed signal "HsBlkSwitch, HSB"..... 273
6.55	Block switching with stop via high-speed signal "HsBlkSwitch(.., HSSTOP=..), HSB(.., HSSTOP=..)"..... 276
6.56	Online correction in workpiece coordinates "HWOC", "HWOCDIS"..... 277
6.57	Initialization "Flying measurement" "InitMeas, IME"..... 278
6.58	Inverse time programming "InvTime, ITIM"..... 279
6.59	Selecting the exact stop window "IPS1, IPS2, IPS3"..... 279
6.60	Determining system axes to specify the optimum SHAPE Filter in Case of Enabled Jerk Limita- tion "JerkAxList, JAL"..... 280
6.61	Activating the jerk limitation "JerkControl, JKC"..... 281
6.62	Jog mode in workpiece coordinates "JogWCSSelect"..... 282
6.63	Disabling the velocity limit due to the sum of radial and tangential acceleration "KinLimAcc, KLA"..... 283
6.64	Disabling kinematic NC block splitting "KinSplitDisable, KSD"..... 283
6.65	CO programming "CvProg, CVP"..... 284
6.66	Lag compensation "LCP"..... 285

	Page
6.67	Dividing the traversing block: Length of path segments "LEN"..... 286
6.68	Path velocity-dependent laser power control "LFP", "LFConf, LFC"..... 287
6.69	Linear to spline switching "LinearToSpline, LTS"..... 289
6.70	Linear-circular-spline-conversion "LCTS"..... 289
6.71	Zeroing the modulo axis "LinModZp, LMZ"..... 290
6.72	Velocity profiles (ramp functions)..... 291
6.73	Main spindle switching "MainSp, MSP"..... 294
6.74	Input tool: Mirroring "Mirror(...), MIR(...)"..... 294
6.75	Measuring on a block "MOB"..... 297
6.76	Measuring on a contour (MOC)..... 298
6.77	Modulo calculation for incrementally traveling endless axes "Modulo, MOD"..... 301
6.78	Modulo offset for endless axes "ModuloShift, MODS"..... 302
6.79	Changing the touch probe edge configuration (MPEDGE)..... 302
6.80	Nibbling "Nibble, NIB"..... 303
6.81	Dividing the traversing block: Number of path segments "NUM"..... 305
6.82	Orientation programming "O()", ROTAX()", "Phi, Theta, Psi"..... 305
6.82.1	Overview of the orientation programming using O(...)..... 305
6.82.2	General description..... 306
6.82.3	Linear orientation motion with axis programming..... 306
6.82.4	Linear orientation motion with coordinate programming..... 308
6.82.5	Vector orientation..... 309
6.82.6	Tensor orientation..... 312
6.83	Feed 100% "OvrDis, OVD", "OvrEna, OVE"..... 315
6.84	Changing the maximum path acceleration "PathAcc, PAC"..... 316
6.85	Activating placement tables "PmTSel, PMS"..... 317
6.86	Programming polar coordinates..... 318
6.86.1	Defining the pole "PolarPol, POP"..... 318
6.86.2	Programming polar coordinates..... 318
6.87	Defining the reflection/rotation point "PoleSet, PLS"..... 322
6.88	Position-dependent high-speed output "PosDepHSOut, PHS"..... 322
6.89	Positioning type for endless axes "PosMode, PMD", Local positioning type for endless axes "DC", "ACP", "ACN"..... 324
6.90	Precision programming "PrecProg, PRP"..... 326
6.91	Program processing depending on the block preparation "PREP"..... 327
6.92	Stroke release time..... 329
6.92.1	Stroke release time (interpolation end point) "PtBlkEnd, PTE"..... 329
6.92.2	Setting the stroke release time to default "PtDefault, PTD"..... 331
6.92.3	Stroke release time (Inpos window) "PtInpos, PTI"..... 332
6.93	PTP traversing motion "PtpMove, PTP"..... 333
6.94	Punching "Punch, PUN"..... 335
6.95	Maximum radial acceleration "RadialAcc, RAC"..... 337
6.96	Recording of the reverse vector as system data "RecordRevVec, RRV, RRV"..... 338
6.97	Torque reduction "RedTorque, RDT"..... 339
6.98	Program continuation with block pre-run after program abort "ReentContBlk, RCB"..... 340
6.99	Backing up the program starting point for re-entry after program cancelation "ReEntryRef, RERF"..... 341
6.100	Removing the axis from the axis group "RemAxis, RAX"..... 343

Table of Contents

	Page
6.101 Removing logical axis name "RemLogName, RLN".....	344
6.102 Removing channel spindles "RemSpindle, RSP".....	345
6.103 Block repetition "REPEAT, RPT".....	345
6.104 Asynchronous subroutines.....	346
6.104.1 Defining the repositioning behavior "REPOSDEF".....	346
6.104.2 Defining point of return to contour in asynchronous subroutine "REPOSTP".....	346
6.105 Retrace, RETRACE.....	347
6.106 Retrace switching lock LockRetrace, LOCKRETRACE.....	349
6.107 Extending the retrace backward program block"ReverseAppend, REVA".....	350
6.108 Replacing the retrace backward program block"ReverseSubstitute, REVS".....	351
6.109 Input tool: Rotating "Rotate(...), ROT(...)".....	351
6.110 Rounding corners.....	353
6.110.1 Rounding corners specifying the deviation "RoundEps, RNE".....	353
6.110.2 Rounding corners with radius specification "Rounding", "RND", "RNDL".....	354
6.111 Synchronizing system axis coupling "SACSYNC".....	355
6.112 Input tool: Scale"Scale(...), SCL(...)".....	357
6.113 Selected additional coordinate coupling.....	359
6.113.1 Selected additional coordinate coupling "SelCrdCouple, SCC".....	359
6.113.2 Selected additional coordinate coupling with table "SelCrdCoupleTab, SCCT".....	360
6.114 Suppressing the collision monitoring "SCLN".....	361
6.115 Setting program position "SetPos, SPS".....	362
6.116 Programmed contour shift "Shift, SHT".....	363
6.117 Speed limitation "SMin, SMN", "SMax, SMX".....	363
6.118 Adjusting spindles in block pre-run init adjustment program "SpAdjSp".....	364
6.119 Determining the spindle syntax for the block pre-run adjustment program "SpAdjStr".....	365
6.120 Enabling/transferring reserved spindle "SpAdmin, SPA".....	366
6.121 Defining (activating) the coupling group, removing (deactivating) the coupling group"SpCouple- Config, SPCC".....	366
6.122 Coupling distance of slave spindle "SpCoupleDist, SPCD".....	368
6.123 Synchronous mode error window "SpCoupleErrWin, SPCE".....	368
6.124 Angular offset with active coupling "SpCouplePosOffs, SPCP".....	369
6.125 Waiting for angular offset "SpCouplePosOffs_Wait, SPCP_WAIT".....	370
6.126 Synchronous window "SpCoupleSyncWin, SPCS".....	370
6.127 Waiting for synchronous mode "SpCouple_Wait, SPC_WAIT".....	371
6.128 Defining/removing spindle groups "SPG.., SPGALL".....	371
6.129 Writing into a system data queue "SSDQ".....	372
6.130 System spindle programming "SSp.....".....	375
6.130.1 General information.....	375
6.130.2 Enabling/applying the reserved spindle "SSpAdm".....	375
6.130.3 Disabling management of the spindle motion for specific channels "SSpAdmOff".....	375
6.130.4 Changing the gear range "SSpGear".....	376
6.130.5 Limiting the maximum speed "SSpMax".....	376
6.130.6 Limiting the minimum speed "SSpMin".....	377
6.130.7 Switching between position/speed interface "SSpMode".....	377
6.130.8 Motion programming "SSpMove (SSPM)".....	377
6.130.9 Spindle orientation "SSpOri (SSPO)".....	378

	Page
6.130.10 Speed programming "SSpSpeed (SSPS)"	378
6.130.11 Tool-specific spindle limitations "SSpToolLim"	379
6.131 Additional function to orient/position spindles "SpindleHome, SPHOME"	380
6.132 Enabling C-axis mode for spindles "SpindleToAxis, STA"	380
6.133 Enabling C-axis mode for spindles and wait "SpindleToAxisWait, STAW"	381
6.134 Waiting for spindle programming at end of block "SpindleWait, SPWAIT"	382
6.135 Selecting a waiting strategy "SpindleWaitMode, SWM"	382
6.136 Rounding corners with splines	383
6.136.1 Rounding corners with splines (modal) "SplineCornering, SCO"	383
6.136.2 Rounding corners with splines (local) "SplineCorneringLocal, SCOL"	384
6.137 Defining the spline type "SplineDef, SPD"	385
6.138 Programmable path split "Split, SPT"	386
6.139 Spindle: Switching between position and speed interface "SpMode, SPM"	387
6.140 Parameterizing the static tool orientation "StatToolOri, STO"	388
6.141 Suppressing single step modes (on/off) "StepModeDisable, SMD"/"StepModeEnable, SME"	390
6.142 Tool-specific spindle limits "SToolLim"	392
6.143 Traveling in TCS "TCM(, ,)"	393
6.144 Tangential tool control "TangTool, TTL"	393
6.145 Tangential tool orientation "TangToolOri, TTO"	396
6.146 Retracting from tapping "TappRet1, TappRet2"	399
6.147 Spindle selection for tapping without compensation chuck "TappSp, TSP"	399
6.148 Adjusting the axis configuration "TargetAxisConf, TAX"	400
6.149 TCS definition in program coordinates "TcsDef, TCS"	401
6.150 Special functions for thread cutting "ThreadSet, TST"	404
6.150.1 General information	404
6.150.2 Configuring the retract data ThreadSet(...)	404
6.150.3 Configuring the dynamics ThreadSet(DYN)	404
6.150.4 Enabling the retraction ThreadSet(RON)	405
6.150.5 Switching the spindle mode ThreadSet(SPC)	405
6.150.6 Influencing the channel interface signal ThreadSet(TCI)	405
6.151 Program coordinate offset "Trans, TRS", Additive Program Coordinate Offset "ATrans, ATR"	405
6.152 Volumetric compensation	407
6.152.1 Enabling and disabling "VCPEna"	407
6.152.2 Loading the table "VCPLoad"	408
6.153 Virtual drives "VirtAxisPos, VAP, VAPR"	409
6.154 Diagnostics of velocity control "VREC_START", "VREC_STOP"	409
6.155 Applying the axis, wait, if required "WaitAxis, WAX"	412
6.156 Writing Sercos parameters with extended Sercos ID number "Writeld3, WID3"	413
6.156.1 General information	413
6.156.2 Writing parameters to one or several drives (for spindles and asynchronous axes)	413
6.156.3 Writing parameters to one or several drives (for synchronous axes)	414
6.156.4 Writing extended ident numbers of an ID list to a drive	414
6.156.5 Writing values of a parameter list to a drive	414
6.157 Activate zero offset tables "ZoTSel, ZOS"	415

Table of Contents

	Page
7 CPL commands	417
7.1 Axes and coordinates	417
7.1.1 Overview	417
7.1.2 Commands to read names and properties	419
7.1.3 Commands to read coordinate and axis positions	424
7.1.4 Commands to read measured values	431
7.2 Spindles	434
7.2.1 Overview	434
7.2.2 Commands to read names	434
7.3 Corrections and input helps	436
7.3.1 Zero point offsets	436
7.3.2 Position and contour offset	441
7.3.3 Workpiece position correction and placements (inclined plane)	442
7.3.4 Input helps	448
7.3.5 Tool corrections	449
7.4 Control data and processes	453
7.4.1 Control version	453
7.4.2 Time recording	453
7.4.3 Machine parameters	456
7.4.4 Simple system data	456
7.4.5 System data of structured types	475
7.4.6 Sercos Interface	479
7.4.7 PLC interface	484
7.4.8 Active functions	487
7.4.9 NC interface (NCS)	488
7.4.10 General XML tables	545
7.4.11 Tool database	546
7.4.12 Errors and warnings	555
7.5 Files and directories	560
7.5.1 General information	560
7.5.2 Directories	562
7.5.3 File operations	564
7.5.4 Editing the file	569
7.5.5 File properties	578
7.6 Character string processing	581
7.6.1 Basics	581
7.6.2 Type conversion	583
7.6.3 Changing the character string	586
7.6.4 Programming examples	593
7.7 Communicating with a client	595
7.7.1 MMC	595
8 Appendix	597
8.1 Tabular overview on NC functions	597
8.1.1 Alphanumerically arranged according to long form	597
8.2 ASCII character set	623

	Page
8.3 Additional key codes.....	626
8.4 Optimizing the machining time.....	627
8.4.1 Measures directly influencing processing velocity.....	627
8.4.2 Measures indirectly influencing the processing velocity.....	629
8.4.3 Processing delay due to faulty programming.....	631
 9 Service and support.....	 633
 Index.....	 635

1 About this documentation

Editions of this documentation

Edition	Release date	Note
01	2014-07	First edition
02	2015-04	Modifications implemented
03	2016-05	Fast positioning for laser cutting - Leap Frog
04	2017-01	Extension of the general language syntax for interruptible ASUPs: ASPENA, ASPDIS Extension of the basics (jump commands): RetTo return from the subroutine to a label in the main program or subroutine
05	2018-03	CVP: Switching off the internal velocity control KLA: Switching off the simultaneous incorporation of radial and tangential acceleration MOB: Measuring a workpiece position Hermite splines: Use a hermite spline to specify an externally generated velocity profile
06	2018-09	Compensation in chapter "Annex" Description of the new NC function "Automatic linear circular spline conversion" (WP-00070601: Smoothing of helix contours using the B-spline approximation) in chapter "Overview modal NC function groups", "NC functions with high-level language syntax", "CPL commands" and "Annex"
07	2020-02	Chapter "NC functions with high-level language syntax" corrected, Defdb00204822 resolved Chapter "NC functions with high-level language syntax" corrected, Defdb00173892 resolved Chapter "Basic principles of NC programming", "Variable programming" corrected, Defdb00202878 resolved Additional function SPHOME supplemented in chapter "NC functions with high-level language syntax", "Additional function to orient/position spindles "SpindleHome, SPHOME""
08	2020-10	Chapter "NC functions with syntax according to DIN 66025" corrected, Defdb00203516 resolved Chapter "NC functions with high-level language syntax" corrected, Defdb00203516 resolved

Tab. 1-1: Change record

1.1 Validity of the documentation

Overview on target groups and product phases

The following graphic refers to the bordered activities, product phases and target groups of the present documentation.

The target group "Programmer" can "optimize and test" in the product phase "Commissioning" using this documentation.

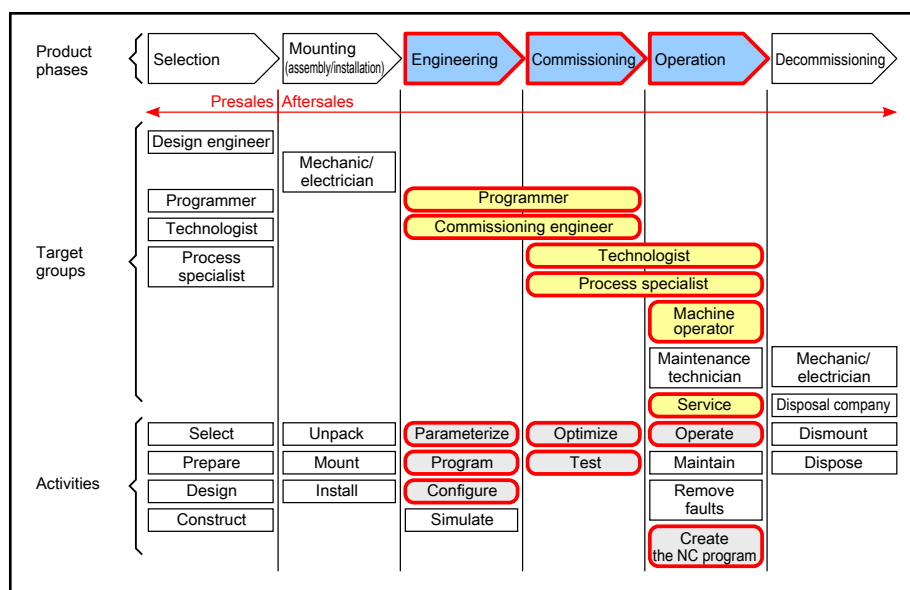


Fig. 1-1: Assigning this documentation to the target groups, product phases and target group activities

Target group This documentation is intended for users commissioning an MTX control.

This documentation supports the user during the following phases:

- Commissioning
- Operation

1.2 General information

This documentation describes the standard programming of the MTX control. Apart from the basics of NC programming, the use of NC functions according to DIN 66025 as well as the NC functions with high-level language syntax and CPL functions are described.

The manual provides support during the application/activation of various control functions.

It focuses on the procedures required for the parameterization.

This requires that the hardware of CNC, drives, PLC and voltage supply units are already integrated and checked according to your application documentation including wiring and accuracy. This refers especially to all safety-relevant features such as E-STOP circuits, release contacts, limit switches.

Ideally, the software of all Sercos drives is already commissioned. However, this is not imperative, since the control is able to configure any drives connected at startup via a parameter download. Thus, the fitting initialization files have to be available on the control.

To adjust the initialization files, have the corresponding documents available (such as lists with the necessary Sercos parameter settings for all drives and drive documentation).

For a better overview, all functions are assigned to different superordinate topics (see "Table of Content").

For each function, the following information is provided:

- Purpose or area of application
- Prerequisites/restrictions on use
- Interference options.

Numerous functions affect several domains at the same time (machine parameters, NC functions for programming, interface signals, etc.). Thus, many functions can also be described completely or partially in other manuals of this product.



Refer to the respective manual.
The following manuals supplement this manual:

- Machine parameters: "MTX Machine Parameters"
- NC and CPL functions: "MTX Programming Manual"
- Interface signals: "MTX PLC Interface"

Furthermore, have the drive documentation at hand.

You require good knowledge of the

- standard user interface of the control
- Windows user interface of the PC control panel
- tools for the machine parameter configuration.

These tools can modify the control configuration. Therefore, you have to be familiar with all required tools!

⚠ CAUTION

Due to an incorrect or unintended use caused by insufficiently trained or untrained staff, serious damage to the machine, loss of data and even personal injury can result!

Thus, only qualified staff that is sufficiently trained may start and operate the system and set or modify configuration parameters!

The personnel also has to be able to detect the risks resulting from changing parameters and the inherent general risks of mechanical, electrical or electronic equipment.

Incorrect or unintended use by insufficiently trained or untrained staff can cause serious damage to the equipment! Bosch Rexroth is not liable for any of those damages and consequential damages caused.

1.3 Required and supplementing documentation MTX

1.3.1 Selection/compilation

Documentation titles with type codes and part numbers

MTX 14VRS System Description DOK-MTX***-SYS*DES*V14-PRRS-EN-P, R911342640 This documentation describes the MTX control. It includes the designs, technical data, interfaces as well as the configuration of the control components.
MTX 14VRS SafeLogic System Overview DOK-MTX***-SL**SYS*V14-PRRS-EN-P, R911341698 This documentation describes the use of the safety control SafeLogic in the MTX.

Tab. 1-2: MTX documentation overview - Selecting/composing

1.3.2 Configuration

Documentation titles with type codes and part numbers

MTX 14VRS Machine Parameters DOK-MTX***-MA*PAR**V14-RERS-EN-P, R911342630 This documentation describes structure and the adjustment of available parameters of the MTX. It also includes the functions of the NC configurator and its operation.
MTX 14VRS PLC interface DOK-MTX***-PLC*INT*V14-PRRS-EN-P, R911342622 This documentation describes the interface signals and the program function blocks for the integrated PLC.
MTX 14VRS Functional Description - Basics DOK-MTX***-NC*F*BA*V14-RERS-EN-P, R911342626 This documentation describes the basic functions of the MTX. The basic commissioning steps and the control functions are provided as description and handling instruction.
MTX 14VRS Functional Description - Extension DOK-MTX***-NC*F*EX*V14-RERS-EN-P, R911342628 This documentation describes the extended functions of the MTX. The basic commissioning steps and the control functions are provided as description and handling instruction.
MTX 14VRS Functional Description - Special Functions DOK-MTX***-NC*F*SP*V14-RERS-EN-P, R911342624 This documentation describes the special functions of the MTX. The basic commissioning steps and the control functions are provided as description and handling instruction.
MTX Free Form Surface Milling DOK-MTX***-FREEFORM***-APRS-EN-P, R911341435 This documentation describes the free form surface milling process with the MTX control. CNC programs generated by a CAD/CAM system are used as basis for the entire process. Overview over the following topics: Description of the MTX or CNC function for freeform surface milling, strategy during the NC parameterization process, boundary conditions when generating CNC programs with CAM software.
MTX Conversion of MTX Projects DOK-MTX***-PROCONV****-PRRS-EN-P, R911342484 This documentation provides support during the conversion of MTX 1.x projects to MTX 2G. This project planning manual provides information on how to convert a project and identifies potential difficulties during the conversion.

Tab. 1-3: MTX documentation overview - Configuring

1.3.3 Commissioning

Documentation titles with type codes and part numbers

IndraWorks 13VRS Simulation DOK-IWORKS-SIMU****V13-APRS-EN-P, R911339256 This documentation describes the functions of the simulation components View3D, virtual control panel, virtual control and its operation in IndraWorks.
MTX 14VRS Commissioning DOK-MTX***-STARTUP*V14-CORS-EN-P, R911342620 This documentation describes the commissioning of the MTX control. Apart from a complete overview, commissioning and configuration of the axes and the user interface as well as the PLC data are described.

IndraWorks 14VRS Basis Libraries IndraLogic 2G DOK-IL*2G*-BASLIB**V14-LIRS-EN-P, R911343920 This documentation describes the system-comprehensive PLC libraries.
IndraWorks 14VRS Field Buses DOK-IWORKS-FB*****V14-APRS-EN-P, R911341485 This documentation describes the field bus and local periphery connections supported by the XLC, MLC and MTX systems. The focus of this documentation lies in the configuration, parameterization, commissioning and diagnostics of different periphery connections. It is the basis for the online help.
IndraWorks 14VRS WinStudio 7.4 DOK-IWORKS-WINSTUD*V14-APRS-EN-P, R911341585 This "User Manual and Technical Reference Book" supports the user in achieving the best results with the "Rexroth WinStudio"™ software. The document provides technical information and step-by-step instructions to create web-enabled HMI/SCADA programs.
IndraWorks 14VRS Software Installation DOK-IWORKS-SOFTINS*V14-CORS-EN-P, R911344286 This documentation describes the IndraWorks installation.
IndraWorks 14VRS Engineering DOK-IWORKS-ENGINEE*V14-APRS-EN-P, R911343566 This documentation describes the use of IndraWorks in which the engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.
IndraWorks 12VRS FDT Container DOK-IWORKS-FDT*CON*V12-APRS-EN-P, R911334398 This documentation describes the IndraWorks FDT Container functionality. It includes the activation of the functionality in the project and working with DTMs.
IndraWorks 14VRS PLC Programming System IndraLogic 2G DOK-IWORKS-IL2GPRO*V14-APRS-EN-P, R911343571 This documentation describes the PLC programming tool IndraLogic 2G and its use. The documentation includes the basic use, first steps, visualization, menu items and editors.
IndraWorks 14VRS HMI DOK-IWORKS-HMI*****V14-APRS-EN-P, R911343569 This documentation describes the functions, configuration and operation of the user interfaces IndraWorks HMI Engineering and IndraWorks HMI Operation.

Tab. 1-4: MTX documentation overview - Commissioning

1.3.4 Operation

Documentation titles with type codes and part numbers

MTX 09VRS Shop Floor Programming Turning and Milling DOK-MTX***-SF*PROG*V09-AWxx-EN-P, R911324377 This documentation describes the operation and programming of the graphic NC programming of turning and milling as well as the workpiece simulation.
MTX 12VRS Block Pre-Run DOK-MTX***-BLK*RUN*V12-APRS-EN-P, R911334379 This documentation explains to the machine manufacturer how to setup the "Block pre-run" function at the machine for the end user.

MTX 14VRS Programming Manual

DOK-MTX***-NC**PRO*V14-RERS-EN-P, R911342634

This documentation describes the standard programming of the MTX control. Apart from the basics of NC programming, the use of NC functions according to DIN 66025 as well as the NC functions with high-level language syntax and CPL functions are described.

MTX 14VRS Standard NC Operation

DOK-MTX***-NC*OP***V14-APRS-EN-P, R911342632

This documentation describes the operation of the standard user interface of the NC control of the MTX. It includes the operation of the interface, the NC program development as well as the tool management.

MTX 14VRS Multitouch

DOK-MTX***-MULTI***V14-APRS-EN-P, R911384735

The Multitouch user interface of the NC control MTX is described in this documentation.

MTX 14VRS Standard NC Cycles

DOK-MTX***-NC*CYC**V14-PRRS-EN-P, R911342638

This documentation describes the application of the standard cycles of the different technologies for the MTX control.

MTX 14VRS NC Simulation

DOK-MTX***-NC*SIM**V14-APRS-EN-P, R911379629

This documentation describes the NC simulation for the MTX control.

xx

Corresponding edition

*Tab. 1-5:**MTX documentation overview - Operating*

1.3.5 OEM Engineering

Documentation titles with type codes and part numbers

MTX 13VRS Automation Interface

DOK-MTX***-AUT*INT*V13-APRS-EN-P, R911337274

This documentation describes the script-based access to IndraWorks project data via the interface of the Automation Interface. Different objects including code examples are described. The Automation Builder is also described in this manual.

MTX 14VRS OPC Communication

DOK-MTX***-OPC*COM*V14-PRRS-EN-P, R911342636

This documentation describes the syntax and the structure of the items for the communication with Bosch Rexroth devices.

IndraWorks OPC UA Communication

DOK-IWORKS-OPC*UA*****-APRS-EN-P, R911379309

This documentation describes the OPC UA communication of the MLC and MTX control systems.

*Tab. 1-6:**MTX documentation overview - OEM engineering*

1.3.6 AddOns

Documentation titles with type codes and part numbers

MTX 11VRS Action Recorder

DOK-MTX***-ACR*****V11-APxx-EN-P, R911329943

This documentation describes the MTX action recorder. It includes the installation and commissioning as well as interface signals, application and operation.

MTX 14VRS Efficiency Workbench MTX cta, MTX ega

DOK-MTX***-EWB*****V14-APRS-EN-P, R911343916

This documentation describes the mode of operation and the area of application of the analysis tool MTX cta and MTX ega.

MTX Remote Condition Monitoring

DOK-MTX***-RCM*****V01-APRS-EN-P, R911334383

This documentation describes the operation of the Remote Condition Monitoring System.

MTX visIREC User Documentation

DOK-MTX***-VISIREC*V01-APRS-EN-P, R911344242

This documentation describes the analysis tool visIREC used to optimize the free form surface milling process. 2D or 4D display of path-related data. 2D or 4D display of coordinate-related data. Analysis of critical areas (path and orientation deviation). Comparing the programmed NC blocks to the interpolated NC blocks.

xx Corresponding edition
Tab. 1-7: MTX documentation overview - AddOns

1.4 Information representation

1.4.1 Safety instructions

If there are safety instructions in the documentation, they contain certain signal words (Danger, Warning, Caution, Notice) and sometimes a safety alert symbol (according to ANSI Z535.6-2006).

The signal word draws attention to the safety instruction and indicates the risk potential.

The signal graphics (warning triangle with exclamation mark), added in front of the signal words Danger, Warning and Caution refers to hazards to persons.

CAUTION

In case of non-compliance with this safety instruction, minor or moderate injury can occur.

NOTICE

In case of non-compliance with this safety instruction, material or property damage may occur.

1.4.2 Symbols used

Note Notes are represented as follows:



This is a note for the user.

Tip Tips are represented as follows:



This is a tip for the user.

1.4.3 Names and abbreviations

Term	Explanation
IWE	IndraWorks Engineering
IWO	IndraWorks Operation
OWG	Optical waveguide

NC	Numerical Control
OEM	Original Equipment Manufacturer
Profibus	Communication connection
Sercos	Communication connection

Tab. 1-8: Terms and abbreviations

1.5 Customer feedback

Customer requests, comments or suggestions for improvement are of great importance to us. Please email your feedback on the documentations to Feedback.Documentation@boschrexroth.de. Directly insert comments in the electronic PDF document and send the PDF file to Bosch Rexroth.

2 Important instructions on use

2.1 Intended use

2.1.1 Introduction

Bosch Rexroth products are developed and manufactured according to the state-of-the-art. The products are tested prior to delivery to ensure operating safety and reliability.

The products may only be used as intended. If they are not used as intended, situations occur that result in damage to property or injury to persons.



Bosch Rexroth shall not assume any warranty, liability or payment of damages in case of damage resulting from a non-intended use of the products; the use shall solely bear all risks from unintended use of the products.

Before using Bosch Rexroth products, the following requirements have to be met to guarantee the intended use of the products:

- Anybody handling Bosch Rexroth products in any way is obliged to read and consent to the relevant safety instructions and the intended use.
- Hardware products may not be altered and have to remain in their original state; i.e. no structural changes are permitted. It is not permitted to decompile software products or alter source codes.
- Do not install damaged or defective products or use them in operation.
- It has to be ensured that the products have been installed as described in the relevant documentation.



Ensure that the data present in the control or entered or read in by the user is correct before applying it to exclude unwanted axis motion. It can be the following invalid or old data:

- Part programs
- ZO tables
- Compensation tables
- Tool tables
- Permanent CPL variables
- Remanent PLC data
- Permanent system data

2.1.2 Areas of use and application

The MTX control is used to

- program contour and machining technology (path feed, spindle speed, tool change) of a workpiece.
- guide a machining tool along a programmed path.

Feed drives, spindles and auxiliary axes of a machine tool are activated via Sercos interface.



This additionally requires I/O components for the integrated PLC which - together with the actual CNC - control the machining process as a whole and also monitors this process with regard to technical safety.

It may only be operated with the explicitly specified hardware component configurations and combinations and only with the software and firmware specified in the appropriate documents and functional descriptions.

The CNC system MTX provides the perfect CNC system solution for cutting and forming for the following technologies:

- Rotate
- Milling
- Drilling
- Grinding
- Bending
- Nibbling
- Punching
- Contour cutting
- Handling

2.2 Unintended use

Using the CNC system MTX outside the above-referenced areas of application or under operating conditions other than the conditions described in the documentation and the specified technical data, is defined as "unintended use".

The CNC system MTX must not be used if ...

- it is subjected to operating conditions not corresponding to the specified ambient conditions. Operation under water, under extreme temperature fluctuations or under extreme maximum temperatures is prohibited.
- Furthermore, the CNC system MTX shall not be used for applications not expressly approved of by Bosch Rexroth. Therefore, please read the information given in the general safety instructions!
- The CNC system MTX may not be used in systems or machines connected to the internet via an unsecure network connection. Otherwise, malfunctions or a control failure can result due to unauthorized access.

3 Basics of NC programming

3.1 Introduction

An NC control receives all information required for the machining of a workpiece on a machine tool via an NC program (part program).

The structure of such an NC program is variable so that nearly any kind of workpiece can be machined with a wide range of technologies (milling, turning, grinding, etc.). The part program does not only contain information about movements describing the path of the tool relative to the workpiece but also information about technologies.

Motion information is subdivided into individual elementary contour elements (straight lines, circles, spirals, splines, nurbs, etc.).

The control can then carry out the movements for each of these geometrically simple contour elements in one machining step provided that all machining steps are specified in the NC program in the correct order and with all required boundary conditions. Among other things, the required boundary conditions consist of technology functions (velocities, speeds, etc.) and auxiliary machine functions (e.g. for coolants and axis clamping).



Basic guidelines regarding the structure of an NC program can be found in DIN 66025.

The content of DIN 66025 "Program Structure for Numerically Controlled Machine Tools" (part 1 and 2) corresponds to the international standard: ISO/DIS 6983 and ISO/DP 6983 "Numerical control of machines".

The CNC system MTX manages NC programs in the "file system" of the control. It is also possible to connect external drives and to run programs directly from there.

For more information about the **File system** and **Access rights** as well as information about creating and editing part programs, refer to the MTX Operating Instructions.

Each NC program has an NC program name which has to meet the general regulations for file names.

A maximum of 28 characters is admissible for names of NC programs, including file name extensions. All letters, numbers as well as the special characters "." and "_" are admissible.

- The standard file name extension of the MTX for the NC program is ".npg".
- Internally, there is no difference between the file name and the file name extension.
- A distinction is made between upper-case and lower-case letters.
- The identifiers "." and ".." are not permitted as file names.
- File names have to be unique within a directory. However, files with the same name can exist in different directories.
- In exceptional cases, file names with up to 30 characters are admissible (provided that the program concerned is never linked).

3.2 Basic information on standard and CPL programming

3.2.1 Programming options

The control provides two ways of programming:

- Standard or DIN programming
- CPL programming (CPL: Customer Programming Language).

Standard or DIN programming can be used to describe the motion sequences and their boundary conditions (geometry, kinematics, dynamics, corrections, etc.). Standard programming is a pure command language to control movements at the machine and to activate specific machine functions.

The syntax of the MTX CNC system consists of commands that are specified in DIN 66025 (G and M-codes) as well as of important extensions regarding G-codes and additional general language-like syntax elements.

The basic elements of standard programming are the so-called NC functions; a programming syntax is assigned to each NC function.

Additional parameters assigned to an NC function can be used to set the parameters of the function.

Example:

NC function: G2 Circular interpolation, clockwise
 Parameter: I, J, K, R Center coordinates, radius

CPL programming is similar to the standard of the BASIC general language but also contains Pascal-like structure elements. That is why it is easy to learn.

CPL is a real programming language, thus providing an alternative way of programming. It includes program elements that lie outside the focus of the machine control; special system functions permit access to the system data of the control.

CPL programming offers the following options:

- Symbolic programming with variables
- Character string processing
- File processing
- Mathematical operators:
 +, -, *, /,
 and trigonometric functions, ...
- Relational operators:
 =, <, >, ...
- Logic operations:
 NOT, AND, OR, ...
- Control structures to control the program run:
 REPEAT, WHILE, FOR, IF, CASE, GOTO, ...
- System functions to determine internal system states:
 positions, active functions, tool data, interface signals, ...
- Process services:
 program selection, control reset, program start, operation mode specification

These options allow the creation and storage of any machining procedures in variable notations.

In accordance with the existing formal input stipulations, CPL instructions are usually written in upper-case letters.

Using CPL results in:

- Shortening of repeat procedures in NC programs and similar program segments, and
- Status-dependent program variants as a result of access to control system data.

An important difference between standard and CPL programming is that all CPL portions are initiated as early as the time of block preparation directly when reading the corresponding program line. The CPL portions thus no longer exist for further block preparation and at the time of interpolation of the NC block.

3.2.2 Highlighting CPL elements within a part program

Both CPL and standard programming can be used within a part program. Due to reasons of uniqueness and, more importantly, in order to be able to provide efficient program interpretation, the CPL portions of the program must be highlighted:

- Program blocks that exclusively contain CPL elements are to be highlighted directly at the start of the block by a positive value without a positive/negative sign (line number) (see [chapter 3.8.2 "Block numbers" on page 27](#)).

Example:

20 AX1\$="U"	In CPL block 20, assign a value of "U" to the string variable "AX1\$".

- If CPL elements are programmed **within a standard NC block** (e.g. due to symbolic variable programming), the CPL portions are to be enclosed **by square brackets** "[" and "]"). This allows a parameterizable value to be assigned to NC functions and to parameters of NC functions. Therefore, only expressions that can be placed on the right side of an "=" sign within CPL (variables, mathematical expressions, system functions that supply a corresponding value) are permitted within the square brackets.

Example:

N10 G1 Y[PCS("X")]	In NC block N10, the y-axis is to be traveled to the workpiece position that was last programmed for the x-axis.
N20 ROT([360/A+SIN(B)])	In NC block N20, activate the Rotate input tool. The rotation angle is defined with the help of a CPL expression.
30 XPOS=100.5	In CPL block 30, a value of 100.5 is first assigned to the variable "XPOS".
N40 G1 X[XPOS] Y10	In NC block N40, the content of the variable "XPOS" is transferred as a coordinate value to the x-axis.

3.2.3 CPL elements during manual data input, buffered NC block specification and non-linked subroutines

All CPL commands can be used without global and local variables during manual data input, buffered NC block specification or non-linked subroutines.

Exceptions are:

- Label programming and jump instructions ([chapter 3.11 "Label programming and jump instructions" on page 48](#))

- Decision and branch instructions ([chapter 3.12 "Decision and branch instructions" on page 55](#))
- Repeat statements ([chapter 3.13 "Repeat statements" on page 58](#))

3.2.4 Complying with times of interpretation between CPL and standard language elements

Due to the special identification of CPL language elements, the programmer can immediately detect which program elements have already been initiated during the block preparation.

Standard programming elements do not have any external effect before the time of the block execution at the machine.

Example:

N523 KvProg (X[@K VX])	At the time when block N523 is executed, the CO value that is used for the x-axis is the one that was set for the permanent variable @K VX at the time of the block preparation.

Until the time when block N523 is executed, however, the variable @K VX might be set to a totally different value (for example, one that was written by another channel)!

The "WAIT" function should be used in this case to avoid uncertainties. The WAIT command blocks the block preparation until the NC block programmed immediately before has been completely processed. The block preparation is thus synchronized with the active status (interpolation time) of the control. Then, processing continues with the block preparation of the subsequently programmed block. At this time, no other prepared blocks exist, since these have all already been processed.

Example:

WAIT	Synchronize block preparation
N523 KvProg (X[@K VX])	At the time when block N523 is executed, the CO value that is used for the x-axis is the one that was set for the permanent variable @K VX at the time of the block preparation. However, since the block preparation has now been synchronized, it is now the active value of the variable.

3.3 Linking of NC programs

3.3.1 Link table

After the program has been selected, the program is checked for its syntax, possible jump targets and subroutines. Furthermore, corresponding management structures that the control accesses during runtime are created for CPL variables. This process is called **linking** (or "preparing").

The result of a successful linking procedure is the generation of a **link table** for the corresponding NC program. All link tables of the MTX CNC system are saved in a special directory that is specified in the machine parameter 3080 00004. The name of the link table of a program is determined by the name of the part program plus the extension ".l" (l: link).

During control startup, it searches the associated NC program for all existing link tables. It is searched with the search path set in the parameter NCO/ FileOrg/SrchPathSubProg (MP 3080 00001). Link tables for which no part program is found are deleted.



The search path sequence should be set in a way that the paths containing the most NC programs are as much as possible in the beginning of the sequence of the parameter NCO/FileOrg/SrchPathSubProg (MP 3080 00001).

If an NC program that is already linked is selected again, the MTX CNC system uses an existing link table provided that the part program has not been modified in the meantime. Linking is carried out again after a program has been changed.

3.3.2 Linking at program selection or at program runtime

There are two options to select and process a not yet linked program: The linking variant available for program selection up to the MTX 12VRS and the linking available at program runtime for the MTX 13VRS and higher.

Linking at program selection, executes a linking procedure. The program can be started after linking. Linking at program runtime starts a linking procedure at program selection, but does not wait until the linking procedure is completed (in contrast to linking at program selection). Thus, the program can be started immediately.

The parameter NCO/FileOrg/ChFileOrg/Ch/KindOfLinking (MP 7070 00030) defines whether linking can take place at program runtime.

If the selected program is linked at runtime, program parts that are not yet linked (processed by linking) are executed. This can cause short delays in the program execution.

If such delays cause trouble during the production sequence, there are two options to avoid linking at program runtime:

1. General linking deactivation at program runtime: Set the parameter NCO/FileOrg/ChFileOrg/Ch/KindOfLinking (MP 7070 00030) for the corresponding channel to 1. It is always linked at program selection.
2. Program-by-program (or subroutine-by-subroutine) deactivation of the linking at program runtime: Program "(CPL)" in the first line of the program/subroutine. The complete subroutine including the subroutines called in that subroutine are completely linked. This can also be performed at program execution time (see [chapter 3.3.3 "Linking subroutines - Postlinking" on page 16](#)).

If the program is linked at program runtime, either the ID used up to now "(DIN)" is ignored in the first line of the program/subroutine or the ID of the subroutine call "P <UP> DIN".

If the program is linked at selection or if a subroutine/program is completely linked due to the "(CPL)" ID, the "(DIN)" ID remains as usual. The corresponding subroutine is excluded from the linking and may thus not contain any CPL instructions requesting a linking procedure.



For big NC programs, the complete linking can cause noticeable delays of the program selection. This is thus to be avoided.



The link tables creation in the mode "Link at program runtime" has low priority. At a high control load, for example if multiple channels are running, the link table is created with some delay. If the program is aborted before completing the link table, the link table generation is aborted and thus not saved.



Linking at program selection causes a syntax check. If a program is selected in the "Link at program runtime" mode, the syntax is not checked at program selection. A possible syntax error is thus only known after some time. When changing a creating NC programs, it is recommended to link the program completely once by program selection (the "(CPL)" ID in the first line can be programmed).

3.3.3 Linking subroutines - Postlinking

If subroutines are called in the program to be linked, the MTX CNC system checks whether there are any existing link tables. If yes, such subroutines are not re-linked. Thus, the linking procedure can be significantly shortened.

If a part program does not contain CPL portions (jumps, CPL variables, CPL expressions, etc.) but only NC blocks (DIN) and subroutine calls, the program does not need to be explicitly linked before being executed.

In this case, the toggle softkey "CPL Prog / DIN Prog" can be set to "DIN Prog" when selecting a program. If necessary, subroutines are re-linked when the program is executed; in certain cases, this may lead to delays in the processing flow.

If subroutine calls are programmed using CPL variables (e.g. P[UP\$]), the linking procedure for this subroutine is not executed before the program is executed, since only at this time the variable name can be initiated. Therefore, a subroutine that is called in this manner is always re-linked if a link table does not already exist.

3.3.4 Influencing the linking process using DIN/CPL IDs

Under the prerequisites mentioned above, linking can also be influenced directly using instructions in the part program, regardless of the current position of the toggle softkey "CPL Prog / DIN Prog".

For doing so, the keyword "**(DIN)**" has to be programmed at the beginning of the first program line. Also in this case, subroutines are re-linked, if required, when the program is executed.

Each (DIN) command can also be programmed in the first line of a subroutine; as a result, a subroutine that is marked in this way will not be linked.

Alternatively, the keyword "**DIN**" (without parentheses) can also be written directly behind the subroutine call in the calling program (also see [chapter 3.9 "Subroutines" on page 31](#)).

Analogously to the (DIN) command, there is the keyword "**(CPL)**" which is also programmed at the beginning of the first program line. This forces the generation of a link table for the corresponding program, even if it has been selected using the "DIN Prog" setting or if it was called as a subroutine with the DIN ID.

3.4 Basic NC program components

3.4.1 Program Block

An NC program consists of at least one **program block**.

The following applies to program blocks:

- A maximum of 1 contour element (e.g. straight line, arc) may be programmed for each program block.
- Blank lines before or after a program block are permitted to improve the organization/readability of the program code.

- A program block must not contain more than 512 characters.
- A program block is terminated by the ASCII character <LINEFEED>.
- A program block consists of at least one **program word** (e.g. NC function, function parameter).

3.4.2 Commands

Commands are program words that directly or indirectly affect the tool path, the program run, the status or the mode of reaction of the control. All NC functions are typical statements.

Available NC functions, together with the corresponding required syntax rules, can be found in [chapter 6 "NC functions with high-level language syntax" on page 195](#).

Path functions play a special role among instructions:

Path functions describe in which way a position is to be approached (e.g. straight line, circle, with or without interpolation of the participating axes, approaching movement in feed or rapid traverse, etc.).

Examples: G0, G1, G2

Path functions are often programmed in the same block as position, path or radius specifications. In such cases, path functions always also initiate traversing motions.

Similarly, position or path specifications that are programmed in a program block without a path function always initiate traversing motions since there is always an active path function.

Example: Path function with coordinate specification

G1	X40	Y50	Traverse with feed to X40/Y50
			Coordinate specification
			Path function

3.4.3 Additional conditions

Additional conditions are program words used to set required technologies or processing boundary conditions for the machine.

Important program words that function as additional conditions include:

F<Number>	influences the feed of synchronous axes
FA<Number>	influences the feed of asynchronous axes
S<Number>	influences the spindle speed
M<Number>	activates M-functions (e.g. gear range selection, rotating direction of the spindle, subroutine calls).
	auxiliary functions are also often programmed as M-functions.
T<Number>	chooses tools

The program words are explained in detail in [chapter 6 "NC functions with high-level language syntax" on page 195](#).

Example: Path information with additional conditions

G01 G71 X40 Y50 F250 S500 T05 M03

Additional conditions

Traverse with feed to X40/Y50;
with programmed F value
(feed= 250 mm/min) and
S value (speed = 500 mm/min);
with clockwise spindle rotation
provide tool T05 in
tool magazine.



Although such additional conditions are not used directly to describe the contour or the path, they can nevertheless affect or initiate movements on the machine (e.g. tool magazine movements)!

3.5 Program words

3.5.1 Overview

A program word is:

1. An **NC function** or
2. a **parameter with a value or a parameter list**. A parameter is addressed in the part program using its syntax (address).



NC block synchronization commands and subroutine calls are special cases of NC functions since they can be programmed together with a CPL parameter list.

Each program word always consists of one or two **partial words** which can be combined as follows:

Program word	Substring 1	Partial word 2	Examples
NC function	Function syntax	-	AxAcc G0 G52.1
	Function syntax	Parameter list	AxAcc(...) G0(NIPS)
	Function syntax	Value	D5 M777
Synchroniza- tion command	Command syntax	-	OFFSTOPA
	Command syntax	CPL parameter list	WPV[@9=10]
Subroutine	UP syntax, SR name	-	P UP1 G4711
	UP syntax SR names	CPL parameter list	P UP2[7,@25]

Program word	Substring 1	Partial word 2	Examples
Parameter	Parameter syntax	Value	X-23.45 Y=AC(40) S250 F5000
	Parameter syntax	Parameter list	O(0,0,1) ROTAX(0,45)

Tab. 3-1: Combination of partial words



G- and M-codes can occur with or without the partial word 2:

- For G and M-codes with fixed internal functions, the numerical value is a component of the syntax, e.g. G0, G17, G54, M0, M3, M19, ...
- For user-defined functions (subroutine calls and auxiliary functions), the numerical value is not a component of the syntax but represents the value for the function. These functions are configured within the machine parameters.

3.5.2 Program words of NC functions

The following applies in general:

- The syntax of NC functions can consist of G and M-codes as well as of general language elements.

Examples:

"G0", "G41", "G141", "G52.0", "M30", "Mirror(...)"

All available NC functions, together with the corresponding syntax rules, can be found in [chapter 6 "NC functions with high-level language syntax" on page 195](#).

- NC functions can contain additional parameters with which the method of operation of the NC function can be controlled.

Three cases are distinguished:

- Parameters that are programmed as an independent program word in the NC block, e.g. "G2 X10 I1.3 J2.5 G94 F1000". These parameters are mainly parameters that are defined in DIN 66 025.
- Parameters that are programmed as a program word, but within a **parameter list with specific syntax elements** that is enclosed in parentheses, e.g. "KvProg(X1.2,Y1.2)".
- Parameters within a **parameter list without specific syntax elements** that is enclosed in parentheses, for which only the value at the defined position is programmed, e.g. "Coord(0,1)", "Rotate(45)", "GetAxis(Z1,Z,W,REV)".
- Parameter lists can be optional and contain at least one valid parameter. If several parameters exist for one parameter list, they are to be separated from one another using commas.



In this documentation, existing parameters that cannot be used simultaneously (alternative parameters) are marked with the "|" symbol between the two alternatives. The "|" character is not programmed.

- Parameters of NC functions can be optional. If such parameters are not programmed, default settings that are permanently coded or stored in the machine parameters are used in general.

Example:

Syntax rule: `TangToolOri({ SYM<s>} , { ANG<a>})`

Due to the syntax rule, it can be seen that the parameters "SYM" and "ANG" are optional parameters. When programming, replace the wildcard <s> by an appropriate value for "SYM" and the wildcard <a> by an appropriate value for "ANG". The syntax rule generally specifies which values are permitted for <s> and <a>.

For instance, one possible programming option for the syntax rule shown above would be `"TangToolOri(SYM4)"`.



In this documentation, the following is identified

- optional parameters are marked by curly brackets and
- Place holders for the values to be programmed are marked by angled brackets.

Curly or angled brackets are not programmed.

Parameter lists without specific syntax elements:

All the parameters in this list type are constants that are transferred directly as a numerical value or name. They must be programmed in a precisely defined order since the meaning of each parameter is specified solely by its position within the list.

Example: Changing the order of the parameters may result in a completely different effect:

<code>GetAxis (X1,X,Y3,Y)</code>	The axes with the system names X1 and Y3 are transferred to the channel where they are assigned the channel names "X" and "Y".
<code>GetAxis (X1,Y3,X,Y)</code>	The axes with the system names X1 and X are transferred to the channel where they are assigned the channel names "Y3" and "Y".

Parameters in a parameter list without specific syntax elements can be optional.

The following applies in such cases:

- Optional parameters at the beginning and in the middle of the list are not required; however, the corresponding comma must be programmed. Only by doing so, the position of the parameter in the list remains unique.
- Optional parameters at the end of the list are not required; the list is then terminated with ")."

Example:

`GetAxis(<Phy.Ax1> {,<Log.Ax1>} {,<Phy.Ax2>} {,<Log.Ax2>} {,...})`

Programmed without parameters in the middle: `GetAxis(X,,Z)`

Programmed without parameters at the end: `GetAxis(X)`

Using **CPL expressions in parameter lists** without specific syntax elements:

If individual elements are to be transferred as a CPL expression, each element is to be enclosed by square brackets.

Example: Variable names as transfer parameters

10 AX1\$="U"	String variable "AX1\$" is assigned the content "U".
20 AX2\$="V"	String variable "AX2\$" is assigned the content "V".
:	
N100 G17 ([AX1\$], [AX2\$])	G17 opens the working plane with the axes U and V.

Parameter lists with specific syntax elements:

The order of all parameters in this list type does not matter since the meaning of a parameter is determined by its programmed syntax.

Example:

"KvProg (X1.2, Y1.4, Z1.6) " and

"KvProg (Z1.6, Y1.4, X1.2) "

have the same effect since the individual CO values can be uniquely assigned using the axis addresses X, Y and Z.

CPL expressions can be used for both syntax elements and numerical values. These must be enclosed in square brackets: "[" and "]".

Example:

Example:

20 value =1.2	Assign value: 1.2 to the "Value" variable.
N30 KvProg (Y[VALUE], X1.2)	Program CO values of the axes Y and X. The y-axis receives the value contained in the variable "VALUE" while the x-axis receives a value of 1.2 (constant).

Example:

40 ACHSB\$="X":FACT%=2	Assign the value "X" to the "ACHSB\$" string variable and assign the value 2 to the "FACT %" integer variable.
N50 Scale ([AXISB\$] [FACT%])	Activate the scaling by a factor of 2 for the "X" axis. Corresponds to the programming: Scale(X2)

Example:

60 DIM PARAMETER\$(10)	Create a character array for a string with a maximum length of 10 characters.
70 PARAMETER\$="X2"	
N80 AxAcc ([Parameter\$])	Assign an axis acceleration of 2 m/s m/s ² to axis "X".

3.5.3 Program words as parameters

Parameters with following value: *In general, the following applies to such program words:*

- An address always starts with a letter and can consist of several characters.
- Program words with an address and numbers are used for programming, for example:
 - Axis and coordinate names (e.g. X..., Y..., Z..., B...)
 - Radii (R...) and interpolation parameters (I..., J..., K...)
 - Feed rate / time values (F...)
 - Spindle speeds and cutting velocities (S..., Si=...)
 - External tool corrections (ED...)
 - D-corrections (D...)
 - Auxiliary functions (M..., T...)
- Leading zeros do not have to be programmed.
- Non-integers are written with a decimal point; trailing zeros may be omitted (e.g. "X100.500" corresponds to "X100.5")
- If a plus sign (or none at all) is programmed, the following value is always interpreted as being positive. A negative sign declares a negative value.

Example: Program word consisting of address letters and a number
(here: coordinate value of the x-axis)

X-2407.0458

post-decimal dot value

pre-decimal dot value

sign

address letter

Parameter with following parameter list:

In general, the following applies to such program words:

- As a rule, the same statements as for program words consisting of an address and a number apply here.
- The parameter list is programmed behind the address and is enclosed by parentheses. The individual elements are to be separated by commas.
- The elements in the parameter list are constants that are transferred directly as a numerical value or a name. They must be programmed in a precisely defined order since the meaning of each parameter is specified solely by its position within the list.
- If individual elements are to be transferred as CPL expressions, each element is to be enclosed by square brackets.
- For example, program words with an address and a parameter list are used for programming
 - Spline coefficients (z. B. X(...)..., Y(...)..., Z(...)..., B(...)...)
 - Vector orientation (O(...), ROTAX(...))

3.5.4 Programming attributes

When programming axis and coordinate positions, it is also possible to provide a programming attribute for the programmed position value.

Without specifying a programming attribute, the position value is interpreted according to the current modal machine status.

With a programming attribute, the modal status can be overridden locally in accordance with the axes/coordinates.

The CNC system MTX knows the following attributes:

AC(...):	The programmed position specification is interpreted absolutely and independently from G90/G91.
IC(...):	The programmed position specification is interpreted incrementally and independently from G90/G91.
DC(...):	The position programmed for an endless axis is approached on the shortest path, regardless of the settings in the machine parameters and the programming of the "PosMode" function.
ACP(...):	The position programmed for an endless axis is approached with a positive rotation direction, regardless of the settings in the machine parameters and the programming of the "PosMode" function.
ACN(...):	The position programmed for an endless axis is approached with a negative rotation direction, regardless of the settings in the machine parameters and the programming of the "PosMode" function.
INCH(...):	The position specification programmed for an asynchronous axis is interpreted in inches, irrespective of the unit filed in the configuration data. The attribute can also be applied to the asynchronous feed (FA address), the programmed value of which is then interpreted in inches/min, irrespective of the linear axis unit configured for asynchronous axes.
MM(...):	The position specification programmed for an asynchronous axis is interpreted in millimeters, irrespective of the unit filed in the configuration data. The attribute can also be applied to the asynchronous feed (FA address), the programmed value of which is then interpreted in mm/min irrespective of the linear axis unit configured for asynchronous axes.
S(...):	Velocity programming for asynchronous endless axes in rpm or MOD/min

Examples:

```
N10 G90 G1 F1000 X10 Y=IC(15)
```

The x-axis travels absolutely to position 10, the y-axis incrementally travels 15 mm further.

```
N20 G91 X=AC(15) Y5
```

Despite G91, the x-axis travels to the absolute position 15, the y-axis travels incrementally 5mm further.

```
N30 B=DC(90)
```

The B-axis (endless axis) is to travel to the 90-degree position following the shortest path.

```
N40 B=ACP(350)
```

The B-axis is to travel to the 350-degree position with a positive direction of rotation (traverse path: 260 degrees).

N50 B=ACN(0)

The B-axis is to be positioned at 0 degrees with a negative direction of rotation (traverse path: 350 degrees).

N100 B=S(1000)

Rotate asynchronous B-rotary axis with 1000 rpm

3.5.5 Using separators between two partial words

Description: *Each partial word can consist of one or more characters (character string). Each programmable character can be classified according to the following groups:*

- Letters:
"A" - "Z", "a" - "z"
- Numbers (including decimal point!):
"0" - "9", "."
- Characters that act as separators:
" ", "=", "+", "-", "(", ")"
- Other special characters (is not discussed further)

A separator has to be programmed between 2 adjacent partial words if:

- The partial word in front ends with a letter **or** with a number and
- if the following partial word starts with a letter **or** a number.

For example, this is the case

- if an additional syntax follows an NC function without a value and a parameter list
- or -
- if a parameter syntax ends with a number and a numerical value is to be assigned to the parameter.

Examples:

-
1. N10 OVE FeedForward(...)	suitable separator: " "
2. N20 X2=2	suitable separator: "="

In the second case, "" or "+" can also be used as a separator; "-" can also be used in case of a negative value assignment.

Using separators can also be relevant for parameter lists with syntax-specific elements!

Special features of axis and coordinate designations:

The desired axis and coordinate identifiers for the control can be specified in the machine parameters:

- Parameter MAIN/Dr/SysDrName (MP 1003 00001) system axis name
- Parameter CHAN/Ch/Coord/Acs/ChAx/ChAxName (MP 7010 00010) channel axis name
- Parameter CHAN/Ch/Coord/Acs/OptAxName/OptChAxName (MP 7010 00020) optional axis names of the channel
- Parameter CHAN/Ch/Coord/Wcs/LinCoord und CHAN/Ch/Coord/Wcs/OriCoord (MP 7080 00010) coordinate names of the channel
- Parameter CHAN/Ch/Coord/Mcs/CartCoord (MP 7080 00020) Cartesian coordinate names of the channel

Axis and coordinate designations always start with a letter and can

- consist of one or more letters where the resulting character string must not have the same appearance as an NC function.

Examples: "X", "PALLET"

- consist of one or more letters and end with a number.
Examples: "X1", "PALLET1"
- In this case, a suitable separator or a blank space must be programmed between the designation and the following value: "=", "+", "-".

Example: The axis identifiers "X" and "X2" are defined.

N40 G1 X10	Axis X travels to position 10
N50 G1 X20	Axis X travels to position 20
N60 G1 X2.5	Axis X travels to position 2.5
N70 G1 X=2.2	Axis X travels to position 2.2
N80 G1 X 2.8	Axis X travels to position 2.8
N90 G1 X2	Axis X travels to position 2
N100 G1 X2 1	Axis X2 travels to position 1
N110 G1 X2=2.8	Axis X2 travels to position 2.8
N120 G1 X2+3	Axis X2 travels to position 3
N130 G1 X2-2.4	Axis X2 travels to position -2.4

3.6 Program end

The program and/or subroutine end is reached in the following cases:

- at the end of the file
- or -
- In a program block containing "M2", "M02" or "M30".
For details concerning the so-called M-functions, refer to section [chapter 5 "NC functions with syntax according to DIN 66025 \(incl. extensions\)" on page 99](#).

Example:

```

:
N250 ...      N250 is the last program block
M30           End of program.
```

At the end of a subroutine, the system returns to the calling program. All modal states are retained ("modal": see page [chapter 3.7.1 "Modal" on page 25](#)).

At the end of a main program, the system returns to the beginning of this program and waits for the next "NC start". If "M2", "M02" or "M30" has been used as the end of a main program, the modal states are set to the M30 portion of the init string.

3.7 Effect of program words

3.7.1 Modal

Description: Program words can have a "modal" or "a non-modal" effect.

"modal" means that a program word remains effective in every subsequent program block until

- the same program word is programmed with a different value,

- another program word is programmed that cancels its effect
- or -
- the function of the program word is specifically switched off.



The term "self-sustaining" is sometimes used as synonym for "modal".

Example of an NC part program:

N10 F1000	Set the feed rate to 1000 mm/min. F1000 acts modally.
N20 G0 X0 Y0	Linear interpolation in rapid traverse to position X0/Y0. G0 acts modally.
N30 Z100	Linear interpolation in rapid traverse to position Z100.
N40 G1(IPS1) X10 Y10	Linear interpolation (in feed; with fine-scale positioning window) with 1000 mm/min to position X10/Y10. G1 cancels the effect of G0. G1(IPS1) acts modally.
N50 X20	Linear interpolation (in feed; with fine-scale positioning window) with 1000 mm/min to position X20/Y10.
N60 G1(IPS2) X30 Y30	Linear interpolation (in feed; with rough positioning window) with 1000 mm/min to position X30/Y30. G1 was programmed with a different value (IPS2). G1(IPS2) acts modally.
N70 X40 Y40 F500	Linear interpolation (in feed; with rough positioning window) with 500 mm/min to position X40/Y40. F was programmed with a different value (500). F500 acts modally.
N80 G0 X0 Y0	Linear interpolation in rapid traverse to position X0/Y0. G0 cancels the effect of G1(...). G0 acts modally.
N90 Scale(X2,Y2)	Activate the scaling function. Scaling acts modally.
:	Scaling remains active.
N200 Scale()	Deactivate the scaling function.

3.7.2 Non-modal

"Non-modal" means that a program word has an effect only in the program block in which it was programmed.



The terms "non self-sustaining" or "local" are used as synonym for "non modal".

Example of an NC part program:

N10 G1 F1000	Activate linear interpolation. G1 acts modally. Set the feed rate to 1000 mm/min. F1000 acts modally.
N20 G75.1 X100 Y100	Travel against first measuring probe. (G75.1) acts non-modally. G75.1 overrides the modal function G1 in this block.
N30 Z100	Linear interpolation at feed to Z100. G1 remains modally active.

3.8 Special program design elements for program design

3.8.1 Channel ID

Description: Use this option to specify the channel the corresponding program may be exclusively used in. Starting the program in another channel generates a run-time error.

The channel ID is written at the beginning of the program.

Syntax:

\$<ChannelNumber>

Tab. 3-2: Channel ID syntax

Example:

N10 \$2	This program can run only in channel 2.
:	Program instructions
M30	Program end

3.8.2 Block numbers

- Program blocks **can** be marked with a block number to improve the readability of the program code.
If unique block numbers are assigned, it is possible to program jumps to these block numbers.
- Program blocks which only contain CPL elements **must** be marked with a block number. Such blocks are also called **CPL blocks**.

The following applies:

- Block numbers must always be programmed as the first NC word in a program line.
- Block numbers of **standard NC blocks** consist of the address letter "N" and an immediately following positive number
(Example: N10, N10.2).
Note that CPL expressions within standard NC blocks must be enclosed by square brackets.
- Block numbers of **pure CPL blocks** only consist of a positive number without a positive/negative sign
(Example: 10, 11.9).
The CPL command or specification of the block is programmed behind the block number.
- If a CPL block concludes with ":" It must be followed by another CPL block that does not have a line number.

3.8.3 Blank lines in the program code

Blank lines can be used to structure the program; thus, they increase the readability of the program. Blank lines are skipped by the control.

3.8.4 Comments in a part program

Comments are skipped by the control when the program runs.

Use comments to

- document the program code or to insert explanations.
- provide comments for entire program lines or for individual parts.

Program comments facilitate and accelerate subsequent editing, e.g. if the program needs to be changed. However, each comment character increases the program file by 1 byte.

Providing comments for entire program lines:

It is possible to completely hide any program line, regardless of whether it is a standard NC block or a CPL block.

To provide comments for entire lines:

- program a semicolon ";" at the beginning of the block
- or -
- enclose the entire line in parentheses "(" and ")".

Remarks in a standard NC block:

It is possible to program a remark at any position within a standard NC block:

- Program a semicolon ";" where the comment starts. From the semicolon to the end of the line, the control interprets the program line as comment.
- Enclose completed remarks within a standard NC block with parentheses "(" and ")". Thus, it is also possible to program "nested comments", for example to hide a sequence within a program line that already contains a comment with parentheses.

Parentheses must not be used behind functions for which a parameter list with parentheses can be optionally programmed (e.g. G0, G1, G61) but for which this parameter list has not been programmed.

Example:

N10 G0 (<comment text>)	Invalid programming!
N20 G0 (NIPS) (<comment text>)	Valid programming
• Behind a function with a possible optional parameter list, it is possible to initiate a comment by "/" if the list is missing. Such a comment can be optionally terminated with "\\".	

Comments in a CPL block:

Within a CPL block, introduce a comment with "REM". Starting at the REM command and ending at the end of the line, the control interprets the program line as a remark.

Syntax:

REM <CommentText>

Tab. 3-3: REM syntax

Example:

```
...
10 REM ***UP to unmask the status word ***
...
```

Examples: Comments in the part program

:	Program code
; <Comment>	Comment line
;N10 <Standard NC block>	Hide standard NC block
;20 <CPL>	Hide CPL block
(<Comment>)	Comment line
(N30 <Standard NC block>)	Hide standard NC block
(40 <CPL>)	Hide CPL block
N50 G1 X0 Y0 ; <comment>	Comment in a standard NC block
N60 G1(IPS) (<comment>) F1000	Complete comment in a standard NC block.
N70 X10 (Y10 (<comment>))	"Nested comment" in a standard NC block.
N80 G0 //<comment>	Comment behind a function for which optional parameters could be programmed.
N90 G0 //<comment>\\ X0 Y0	Completed comment behind a function for which optional parameters could be programmed.
100 REM <comment>	Comment in a CPL block.

3.8.5 Notes in the user interface

Use note programming to display note texts on the NC user interface (max. 80 characters).

Thus, it is possible, for example, to provide the machine operator with the following while the program is running

- information about the current program status, or
- handling instructions.

There are two possible types of notes:

- **channel-specific notes:**
These are deleted if the program is deselected or if the channel is reset.
Syntax: MSG (<note text>)
- **Cross-channel notes:**
These can be deleted with an overall reset.
Syntax:
GMSG (<Note text>)

It is also possible to program a note to provide the machine operator with a handling instruction. To do this, program e.g. "M0" in the same line or in the following line. Thus, it is ensured that the program is interrupted immediately after the note text has been output. The rest of the program does not run until "NC start" is pressed.

Example:

:	Program code
N60 (MSG probe workpiece !)	Output channel-specific note.

N70 M0	Wait for NC start button.
:	Program code

For reasons of compatibility, some alternative syntax variants exist for note programming; in terms of functions, they are all equal:

- Syntax variants for channel-specific notes:
 (MSG<Information text>)
 (*MSG<Information text>)
 (MSG,<Information text>)
 (*MSG,<Information text>)
- Syntax variants for cross-channel notes:
 (GMSG<Information text>)
 (GMSG,<Information text>)

If no note text (or space) is programmed with MSG or GMSG, the pending notes are deleted. The MSG function deletes the channel-dependent notes created by MSG or PRN(0,...). GMSG can delete channel-independent notes.

Example:	:	Program code
	N60 MSG(Probe workpiece!)	Output channel-specific note.
	N70 M0	Wait for NC start button.
	N80 MSG()	Deleting note.
	:	Program code

3.8.6 Jumps in the program run

One consequence of an increasing program size is the increased importance of a "clean programming process".

This mainly includes the following:

- structured programming,
- fault tolerance, and
- software ergonomics.

Generally speaking, structured programs tend to be clearer in their overall architecture. Bundling program segments that serve related purposes or contain frequently used functions for (parameterized) subroutines or for a single jump destination which is provided with a comprehensible identifier (label) does not only result in improved readability but also in more efficient work since these programs can also be used in other programs.

The following options are provided:

- **Subroutine calls**
 (refer to [chapter 3.9 "Subroutines" on page 31](#))

Use subroutines if a certain processing section occurs several times in an identical or similar form within processing.

Program this section only once (if necessary, with the possibility of transferring parameters), save it as a program and simply call it when needed.

This saves programming code and memory space. In addition, the programs become clearer and easier to maintain.

- **Jump instructions**

(refer to [chapter 3.11 "Label programming and jump instructions" on page 48](#))

Use jump instructions to continue the program at other locations within the current program depending on defined events (e.g. calculation results).

- **Decision/branch instructions**

(refer to [chapter 3.12 "Decision and branch instructions" on page 55](#))

Use such commands if individual program blocks, different program sections or entire subroutines are to be executed depending on certain conditions.

- **Repeat instructions**

(refer to [chapter 3.13 "Repeat statements" on page 58](#))

Use repeat instructions if program sections or entire subroutines are to be run repeatedly. Usually, it is normally also possible to enter the repetition rate.

3.9 Subroutines

3.9.1 Overview

Subroutines (SR) are programs that are called with the help of a subroutine call. Once the execution of a subroutine has been concluded, the calling program will continue to run from the point where the SR was called. A program calling itself is called a "recursive" subroutine call.

The main program (MP) is the program from which execution jumps into the first subroutine level (SR level).

Formally, no differentiation is made between main programs and subroutines; however, only subroutines can have transfer parameters.

The following applies:

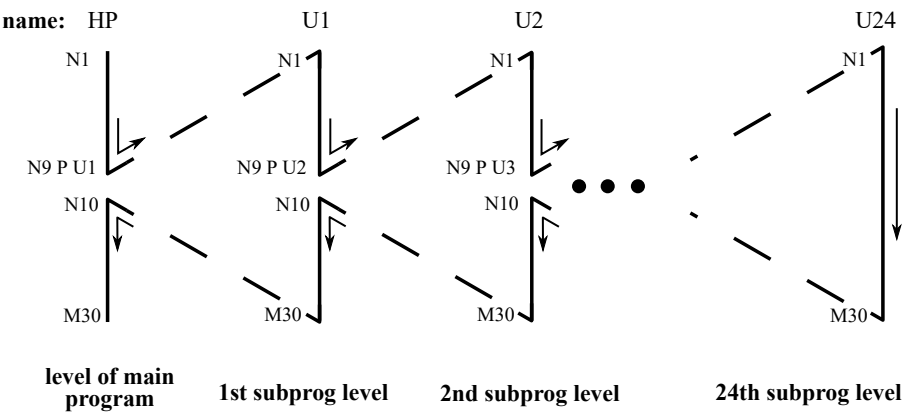
- Subroutines can contain standard NC blocks and CPL blocks.
- Each part program can be called as a subroutine by other programs. However, a program is incapable of calling itself as a subroutine (recursive calling is not possible).
- The calling program can transfer parameters to a subroutine.
- The maximum nesting depth is 24, i.e. the control can keep a maximum of 24 subroutine levels open at the same time.
- Subroutine names are also case sensitive.



For information about ending a subroutine, see [chapter 3.6 "Program end" on page 25](#).

The maximum nesting depth can neither be configured directly nor extended. But different machine parameters can impede the reaching of the maximum nesting depth. For more detailed information, refer to [chapter 3.9.9 "Configuration and error recovery" on page 45](#).

Example: The maximum subroutine nesting allowed is represented schematically. N1 is the first block of the program, the next subroutine is called in block N9. M30 completes the respective (sub)routine.



There are locally and modally effective subroutines:

- Generally, subroutines are **local** effective. In this case, the subroutine is called once at the point of calling.
- If, on the other hand, a **modally** effective subroutine is activated, then it is called again at every following programmed traversing motion until it is deselected again. This option is used, for example, in drilling cycles.

The CNC system MTX provides the following variants for subroutine calls:

- Call with P address and subroutine names from a standard NC block (optionally with path information).
- Call only with subroutine name without path information and without P address from a standard NC block.
- A subroutine as self-defined G- or M-function in a standard NC block.
- Activation of a modal subroutine with self-defined syntax from a standard NC block.
- Call using the CALL function from a CPL block.

3.9.2 Subroutine call with P-address

- Description:**
- The subroutine name is directly programmed behind the P-address. It is also possible to program the directory in which the subroutine is located.
 - The subroutine name with or without path can be programmed behind the P address with the help of the CPL variable. Therefore, the CPL variable has to be enclosed by square brackets.
 - To improve readability, it is possible to program a space character as a separator between the P address and the subroutine name.
 - The subroutine call must be programmed at the block end.
- Traversing motions programmed in the same block are executed prior to the subroutine call (see example).
- No more than one subroutine call must be programmed in one block.
 - The subroutine call is local (not modal).



Using subroutine calls via the CPL variables such as "P [UP\$]" cause an extended programming runtime.

Syntax:

P{<path>}<Name> {DIN} or P[<CPL variable>] {DIN}	
with	
<Path>	Directory in which the subroutine is located.

<Name>	Name of the program to be called.
<CPL var.>	CPL string variable which includes the name of the subroutine with or without path as a value.
DIN	Optional Prevents the subroutine from being linked. "Simple" CPL instructions (see chapter 3.2.3 "CPL elements during manual data input, buffered NC block specification and non-linked subroutines " on page 13) in the non-linked subroutine are permitted. Subroutine calls are allowed. These are also postlinked. For further information refer to the CALL command (chapter 3.9.7 "Subroutine call in CPL via the CALL command" on page 38).

Tab. 3-4: Syntax with P address

Example:

```

:
N40 P Hole pattern          Call the program "drill pattern".
N50 X100                    Block N50 follows after the end of the SR.
:
N140 G0 X10 Y0 PUP1        First, positioning to X10/Y0 in rapid traverse.
N150 Z0                      Subsequently, call program "UP1".
:                             Block N150 follows after the end of the SR.
200 DIM UP$ (20)
210 UP$="hole pattern"
N220 P[UP$]

```

3.9.3 Subroutine Call Without P Address

Description: *Subroutines can also be called **without** preceding P address:*

- Only the subroutine name is programmed directly.
- The subroutine name can also be programmed with the CPL variable. Therefore, the CPL variable has to be enclosed by square brackets.
- It is not possible to provide any path information.
- The subroutine call must be programmed at the block end.
Traversing motions programmed in the same block are executed prior to the subroutine call (see example).
- No more than one subroutine call must be programmed in one block.
- The subroutine call is local (not modal).

Syntax:

<Name> or [<CPL var.>]	
with	
<Name>	Name of the program to be called.
<CPL var.>	CPL string variable containing the name of the subroutine without path.

Tab. 3-5: Syntax without P address



Ensure that this programming option does not cause any confusions with the normal syntax!

If the subroutine to be executed starts with a valid syntax, it must be called with the P address.

Therefore, always assign unique names to subroutines to avoid misinterpretations by the control interpreter.

Example:

:	
N40 XUP	Call the program "XSR".
N50 X100	Block N50 follows after the end of the SR.
:	
100 DIM UP\$ (20)	
110 UP\$="XUP"	
N120 [UP\$]	
N100 X1UP	Attention!
:	The UP name "X1UP" programmed here leads to a syntax error since "X1" is interpreted as a coordinate of an axis named "X" and since a program named "UP" does not exist.
:	
:	
:	
:	
N140 G0 X10 Y0 XUP	First, positioning to X10/Y0 in rapid traverse.
N150 Z0	Then call the program "XSR".
:	Block N150 follows after the end of the SR.



In subroutines called with the CPL variable, it is **not** possible to use the MTX block pre-run dialog.

3.9.4 Local subroutines

Description: In the MTX 11V02 or higher, subroutines can also be defined within the files of the main program. A local subroutine starts with the command **LPS <SRName>** (LocalProgStart) and is completed with the command **PEND** (ProgramEnd). Within one program sequence, a local subroutine can be called using the command **LP <SRName>** (LocalProg).

- To call local subroutines, the name of the local subroutine is directly programmed behind the LP address.
- The subroutine name can also be programmed with the CPL variable. Therefore, the CPL variable has to be enclosed by square brackets.
- The subroutine call must be programmed at the block end.
Traversing motions programmed in the same block are executed prior to the subroutine call (see example).
- No more than one subroutine call must be programmed in one block.
- The local subroutine call is local (not modally) effective.
- Position or address parameter can be transmitted to local subroutines. The parameters are transferred as for "regular" subroutines.

- The local subroutine call LP may contain the parameter **DIN**. The parameter DIN indicates that the local subroutine exclusively contains DIN or "simple CPL instructions" and does thus not have to be linked (refer to [chapter 3.2.3 "CPL elements during manual data input, buffered NC block specification and non-linked subroutines "](#) on page 13).
- Instead of programming the DIN/CPL ID when calling the local subroutine, this ID can also be put in round brackets and programmed in the first program line of the local subroutine.
- Local subroutines cannot be called as asynchronous subroutines (AS-UP).
- Local subroutines can be nested and also contain ordinary subroutine calls.
- Local subroutines must be located at the end of file after the main program has been completed.
- The program end is identified by M30. If no M30 is programmed, the first LPS programming is implicitly interpreted as program end.
- Contour definition for machining cycles must be located in front of the first LPS command in the file.

Application: Using local subroutines means reducing the link times of big part programs with nearly exclusive DIN programming (e.g. free-form surfaces). The DIN parts are "sourced out" into a local subroutine that is then called using the **DIN** parameter.

Syntax:

<Name> or [<CPL var.>]{DIN}	
with	
<Name>	Name of the program to be called.
<CPL var.>	CPL string variable containing the name of the subroutine without path.
DIN	Optional Prevents the subroutine from being linked. Use this parameter only if the subroutine does not contain any CPL blocks. Subroutine calls are allowed. These are postlinked if required.

Tab. 3-6: Local subroutines

Example:

```

:
10 I% = 5
N10 G0 X0 F1000
20 IF (...)
:
LP Up1 (2,3.4)           Calling the local subroutine Up1 with position pa-
                           rameters (P1, P2)
:
X10 LP Up2               Traversing Motion on X10 in rapid traverse and
                           subsequent call of the local subroutine Up2
:

```

LP Up3 (MOD2, X3.4)	Calling the local subroutine Up3 with address parameters (MOD, X)
M30	Program end
LPS Up1	Start of the local subroutine Up1
:	
PEND	End of the local subroutine Up1
LPS Up2	Start of the local subroutine Up2
:	
PEND	End of the local subroutine Up2
LPS Up3	Start of the local subroutine Up3
(ADR_PARA)	Label for transmitting address parameters
:	
PEND	End of the local subroutine Up3



In subroutines called with the CPL variable, it is **not** possible to use the MTX block pre-run dialog.

3.9.5 Self-defined subroutine calls with G and M codes

In addition to the subroutine calls described above, the control allows the user to define:

- "non-modal" subroutine calls using the M address
(see parameter NCP/SubProg/NonModalMFunc/MFunc/MFuncCode (MP 3090 00003) and parameter NCP/SubProg/NonModalMFunc/MFunc/MFuncSubProg (MP 3090 00004)) and
- "non-modal" subroutine calls using the G address
(see parameter NCP/SubProg/NonModGFunc/GFunc/GFuncCode (MP 3090 00001) and parameter NCP/SubProg/NonModGFunc/GFunc/GFuncSubProg (MP 3090 00002))

Subroutines can be assigned for a maximum of 64 user-defined G-codes and a maximum of 8 user-defined M-codes; these subroutines are called as local (non-modal) subroutines when programming the corresponding G or M-code.

The G and M-codes configured as subroutine calls may not collide with permanently defined G and M-codes.



Contact the relevant system administrator for the subroutine calls that are configured on the specific machine.

*The following applies to the **programming** of self-defined SR calls:*

- Only the corresponding G or M-code is programmed in the part program. The subroutine that is called as a result is determined by the configuration in the machine parameters.
- The subroutine call must be programmed at the block end.
Traversing motions programmed in the same block are executed prior to the subroutine call.

- No more than one subroutine call must be programmed in one block.
- The subroutine call is local (not modal).

3.9.6 Self-defined modal subroutine calls

Subsequent to their initial call, modal subroutines will continue to be automatically executed after each traversing motion specified by a standard NC block. This will continue until they are deselected via a special NC function.

Modal subroutine calls, including the syntax of the switch-off function, are configured in the machine parameters (see parameter NCP/SubProg/ModaSubProg/SwiOn/SwiOnSynSubProg (MP 3090 00005) et seq.).

A maximum of 63 modal subroutines can be set. The names of the modal subroutines are assigned to the freely definable syntax. In addition, the maximum number of parameters that can be transferred to the corresponding modal subroutine must be specified.

The syntax of the modal subroutine calls must not collide with the permanently defined NC functions.



Contact the relevant system administrator for the subroutine calls that are configured on the specific machine.

*The following applies to the **programming** of modal SR calls:*

- Only the respectively configured syntax of the modal subroutine is programmed in the part program. The subroutine that is called as a result is determined by the configuration in the machine parameters.
- Only 1 subroutine call may be programmed in a block.
- The subroutine call is modal, i.e. the subroutine is called again after every programmed traversing motion until it is deselected.
- Modal subroutines are deactivated with the deselection syntax.
- If a modal subroutine is programmed while another modal subroutine is already active, it is executed once at a programmed location like a non-modal subroutine.
- If the same modal subroutine is reprogrammed without programming the deselection syntax before, only the new parameters are applied. The program is started with the next traversing motion, but with the new parameters. This corresponds to a behavior as if the deselection syntax was programmed before a new modal subroutine parameterization takes place.

*The following applies for the **parameters** of modal subroutines:*

- Programmed parameters are inherited from SR call to SR call (the parameters programmed once apply for each modal subroutine call until they are deselected).
- If a programmed parameter of such a modal subroutine is changed once, the **changed** value is inherited.
- Up to <n> programmed parameters are inherited. <n> is the number of transfer parameters for the respective modal subroutine defined in parameter NCP/SubProg/ModaSubProg/SwiOn/NofParSubProg (MP 3090 00007). If more than <n> parameters are programmed, they only apply to the first subroutine call. The value is not assigned anymore in the following modal calls.
- If a modal subroutine is active and programmed again (without previous deselection), the newly programmed parameters apply additionally to

the old, inherited parameters. This applies until the maximum number (see above) of parameters that can be inherited, is reached.

Note: Parameter values not programmed in the call are never inherited!

3.9.7 Subroutine call in CPL via the CALL command

Description: The CALL command provides the possibility to call subroutines from CPL programs.

The following applies to programming:

- The subroutine programmed after the CALL command is called directly.
- The CALL command must be programmed in an individual CPL block.
- The subroutine call is local.

The CALL keyword is followed by the program name. This, in turn, may be followed by transfer parameters enclosed in square brackets and, to conclude the instruction, the "DIN" identifier (to influence the link process).



In a CPL block that includes the CALL instruction, a ":" must not be used. Subsequent CPL commands must be programmed in a new CPL block.

Example:

```
50 IF A% = 1 THEN
51 CALL P999
52 ENDIF
:
```

**Influencing the LINK process
("preparing") with the help of "DIN"
identifiers:**

If calling a subroutine with CALL and programming the "DIN" identifier as end, the control excludes the subroutine thus called from the linking process. For example, the linking process of a main program that includes numerous SR calls can be significantly accelerated in this manner.

Example:

```
50 IF A% = 1 THEN
51 CALL P999 DIN          SR "P999" is not linked
52 ENDIF
M30
```

It is strongly recommended to include the "DIN" identifier in the program only if the called subroutine

- consists only of DIN blocks or simple CPL instructions (see [chapter 3.2.3 "CPL elements during manual data input, buffered NC block specification and non-linked subroutines "](#) on page 13) and
- does not call any further subroutines.

In the event that a subroutine containing CPL elements was excluded from the linking due to the "DIN" identifier and contains invalid CPL elements, the control displays a corresponding error message at program runtime.

As an alternative, it is also possible to insert the "DIN" identifier **as a remark** in the first line of the subroutine to be called. The control does not link the program.

Example: "DIN" identifier in the program to be called

Example: "DIN" identifier in the program to be called

```

N10 (DIN)                SR "P999" is not linked
N20 ...
:

```

3.9.8 Transferring parameters to subroutines

General information

There are two options to transfer parameters to a subroutine:

- via **CPL parameter list** or
- via **Address parameter list**.

The CPL parameter list is used to access the transferred values in the subroutine using the local variables P1 to P <n>. The assignment parameter value variable name is performed by the programming sequence in the calling program. In case of the address parameter list when calling the subroutine, the variable name in the subroutine is specified. Thus, the programming sequence in the call is not important and the "talking" variable name can be used in the call as well as in the subroutine. They are to be seen as equivalent with regard to their functions.



Do not use variable names beginning with a "P" and followed by a digit.

Address parameter list

With the help of an **address parameter list**, it is possible to transfer parameters to a subroutine. For this purpose, the parameters are programmed in a list enclosed by round brackets "(" and ")" immediately behind the subroutine call. The individual parameters are separated by commas.

The parameters to be transferred are programmed in the calling program using a combination of parameter address word and parameter value that might be separated with a space or a "=".

Examples: The following examples for transferring the Real value "3.4" to the address "A" of the subroutine are equivalent:

```

N10 P Subroutine(A 3.4)
N20 P Subroutine(A=3.4)
N30 P Subroutine(A3.4)

```

Unambiguity of address names

In the N30 example, the separation between the address parameter and value is ambiguous: The value ".4" could be assigned to the variable "A3" and the value "3.4" could be assigned to the address (CPL variable) "A". The control searches for the last alpha-numeric character to separate the address from the value and interprets the remainder as value.

If the value ".4" should be assigned to the variable "A3", a space or "=" is to be programmed between "A3" and ".4":

```
N40 P subroutine(A3 .4).
```

This especially applies to the transfer of strings. They have to be different from the address parameters.

Strings should always be programmed between inverted commas ("). If a string is written between inverted commas and an additional inverted comma

is required in this string, the additional inverted comma must be preceded by a backslash (\). Backslashes in turn are represented by two backslashes.

If strings are very simple (no leading or closing blanks, no comma, no "brackets close"), the inverted commas may be omitted. In this case, backslashes (\) are interpreted as normal backslashes.

Example:

String parameters

```
N10 P Up(ST="A string (with parentheses)")
N20 P Up(ST="A string with \"Text\".")
N30 P Up(ST="Pfad: \\mnt\\Test.npg")
N40 P Up(ST=SimpleString)
```

In these examples, the assignment symbol (=) can be replaced by a blank.



To use the address parameter list to transfer the parameter to a subroutine, the keyword "**(ADR_PARA)**" must be programmed in the first line. An additional ID "(DIN)" ensures that the subroutine is not linked and that no parameter values are transferred.

Permitted parameter addresses

- CPL variables of the subroutine

Permitted parameter values

- Numbers
- String constants
- Arithmetic CPL expressions (in the CPL brackets "[" and "]")

In order to transfer several parameters, the address word and the parameter value pairs are separated from each other using a comma:

```
N50 P subroutine(A3, B5.2).
```

Two or more consecutive commas are not allowed.

In contrast to the CPL parameter list, the parameter sequence is not important. Non-programmed addresses are not assigned in the subroutine.

A value can be transferred to every CPL variable in the subroutine. No special variable names such as P1, P2% etc. (as it is the case for the CPL parameter list) have to be used.

Example 1:

Using the same variable names

Main program:

```
10 A=5.5
N20 P Subroutine(A=3.4)
N30 P Subroutine(A=[A])
```

Subroutine:

```
(ADR_PARA)
N10 G0 X0
N20 G1 X[A]
```

Explanation:

The main program assigns a local variable "A" with a value of 5.5. In block N20, the subroutine is called and the value 3.4 is assigned to the local variable "A" **of the subroutine**. In the N20 block, the subroutine moves the axis X from the position 0 to 3.4 determined by the local variable "A" of the subroutine. When calling the subroutine a second time in the N30 block, the value of the local variable "A" of the **calling program** is assigned to the local variable "A" of the **subroutine**.

General rule:

On the left, in front of the delimiter, there is the local variable of the subroutine. On the right of the delimiter, in square brackets, there are the local variables of the calling program or other expressions allowed.

Example 2:**Using several parameters, string transfer****Main program:**

```
10 A=5.5
N20 P Subroutine(A=3.4, B[A*3], UP="ASubroutine")
...
```

Subroutine:

```
(ADR_PARA)

10 IF B=NUL THEN
20 B=0
30 ENDIF

N10 G1 X[A] Y[B]

40 DIM UP$(20)

N20 P [UP$]
```

Explanation:

The main program assigns a local variable "A" with a value of 5.5. In block N20, the subroutine is called and the value 3.4 is assigned to the local variable "A" **of the subroutine**. The subroutine assigns the value 0 to the variable B if it was not programmed. Subsequently, axis X is traveled to position 3.4 and axis Y to position $3 * 5.5 = 16.5$ in the N10 block, since 3.4 was assigned to A and 16.5 to B when calling a subroutine. The third parameter SR specifies that another subroutine called "OneSubroutine" is called in the subroutine.



If the calling program is selected without linking and if a string is transferred to a subroutine, the PDIM command (instead of the DIM command) has to be used (also refer to the CPL parameter list).

CPL parameter list

Description: With the help of a **CPL parameter list**, it is possible to transfer parameters to a subroutine. For this purpose, the parameters are programmed in a list enclosed by square CPL brackets "[" and "]" immediately behind the subroutine call. The individual parameter values are separated by commas.

The parameters transferred during the subroutine call are always addressed via the variables P1, P2, P3, etc. in accordance with the order of the parameter transfer.

The parameters may also be addressed by means of, for example, P1TEST, P2XYZ, etc. However, the upper-case letters following P1, P2, etc., are ignored (P1 = P1TEST = P1XYZ).

The following parameters are permitted:

- Numbers (for floating point numbers smaller than 1, zero has to be programmed in front of the point, e.g. 0,134.)
- CPL string constants (in quote marks: "<CPL string constant>")
- CPL variables
- Arithmetic CPL expressions

Example:

- P1 has a value of 2.75 in SR P999,
- P2 has the value of variable X% at the time of the parameter transfer
- P3 has a value of 0.

If P2 is to represent an integer in the subroutine as well, simply attach a % character to P2. This mode of identifying the type of variable can also be used with the other types of variables.

In the subroutine, the value of the individual parameters can be assigned to additional variables.

Main program:

```
50 IF A% = 1 THEN
51 Call P999 [2.75, X%, 0]   Subroutine call with parameter transfer
52 ENDIF
M30
```

Subroutine P999:

```
1 FACTOR=P1 : XVALUE%=P2% : KORRTAB%=P3%
N1 G1 X[XWERT%*FAKTOR]
N2 G22 K[KORRTAB%]
```

The PDIM command has to be used if:

- a subroutine with a string constant as transfer parameter **and**
- the calling program is selected without linking.

Syntax:

PDIM <ParameterName>(<ArraySize>)	
	If the array size programmed is too small or missing entirely, the control displays the part program error "INVALID VARIABLE".

Tab. 3-7: PDIM syntax

Example: Main program:

```

N10 (DIN)
:
N50 P UP["Test"]
M30
Subroutine:
10 PDIM P1$(4)
M30

```

The string variable P1\$ has the value "TEST".

Transferring parameters to mModal subroutines

Modal subroutines cannot be parameterized via transfer parameters like non-modal subroutines. The following is to be taken into consideration:

The number of parameters to be transferred to a modal subroutine is specified for every modal subroutine in the parameter NCP/SubProg/ModaSubProg/SwiOn/NofParSubProg (MP 3090 00007). The value set there also specifies the number of parameter values that should be passed on to the next modal subroutine call.

When using the CPL parameter list, the parameter values of the variables P1 .. Pn (n is specified in the parameter NCP/SubProg/ModaSubProg/SwiOn/NofParSubProg (MP 3090 00007)) is transferred and passed on to the next subroutine call after the completion of the modal subroutine.

If the address parameter list is used, the first n programmed addresses are passed on to the next call. Addresses that are not programmed are not passed on.

NC block conversion in a parameter list

In case of some machining processes, it is required that a fast traversing motion, e.g. a G0 infeed motion is not executed directly but e.g. is replaced by an alternative infeed motion. For example, this is the case for height-controlled laser cutting machines if the height control is not dynamic enough to prevent a collision of the tool head with the workpiece in case of a high infeed velocity. Instead of a linear motion, the height-controlled axis/coordinate has to be "raised". This example is known as "Leap Frog". As the NC programs often cannot be rewritten, CNC support is recommended to cover this use case. Of course, this solution can also be applied to other use cases.

The NC function "SubPConv" converts the syntax of the subsequently programmed coordinates and the values to an address subroutine parameter list. For the use case described, a subroutine (e.g. "LeapFrog") has to be developed which expects the time of G0 infeed motion as parameters and that generates other motions. To fulfill this requirement and to avoid that the NC program has to be changed, an NC macro has to be configured additionally which replaces the NC function "G0" by "SubPConv LeapFrog". The NC function "SubPConv" acts like a regular P-subroutine call - the subroutine name can be found behind the NC function and should contain an absolute path specification (to avoid ambiguity due to the search order).

To convert a typical G0 infeed motion of type "G0 X100 Y220", the NC string "SubPConv LeapFrog X100 Y220" is generated first in this example by replacing the macro. In the second step, the conversion function converts the string behind the subroutine program name to a valid subroutine program syntax list: "SubPConv LeapFrog (X=100, Y=220)".

Thus, the subroutine program "LeapFrog" can access the transferred CPL variables "X" and "Y" and to generate a new traversing motion.

Restrictions: Axis values or coordinate values programmed before the NC function "SubPConv" are machined first in the same NC block. For example, "X100 Y200 SubPConv LeapFrog" traverses the axes on (100,200), subsequently, the "LeapFrog" subroutine is called to which no parameters are transferred.

If an axis or coordinate list is interrupted by a syntax element that is not axis or coordinate, the parameter list conversion is completed before this syntax element. The axis or coordinate command programmed behind this syntax element is then executed before the subroutine call. Example: "SubPConv LeapFrog X100 G91 Y200" results in relative (**G91**) positioning in Y around 200 and in a subroutine call with the parameter list "LeapFrog (X=100)".

If the syntax replacement of (e.g.) "G0" is applied by "SubPConv LeapFrog" using the macro functionality, this applies across systems. However, it is not always desired or possible to call a subroutine (e.g. in InitString) so that unintentional error messages can sporadically be reported. Make sure that the macro search text cannot occur in critical or sensitive situations!

Use case LeapFrog

Assumption 1: Each infeed motion in rapid traverse programmed with G0 should execute a "LeapFrog". G00 and G000 is not influenced to guarantee a normal, linear infeed motions.

Assumption 2: The process and machine-specified subroutine "LeapFrog" accepts 2 parameters: X and Y, as the (Cartesian) infeed motions take place in these coordinates.

Macro definition (e.g. /usrfep/macro/MacroDef_G0.xml)

Program:

```
<MacroDef>
  <Macro Type="DIN" Overlay="Yes">
    <Name>G0</Name>
    <Replace>SubPConv LeapFrog</Replace>
    <Comment>Replace every occurrence of G0 with "SubPConv LeapFrog"</Comment>
  </Macro>
</MacroDef>
```

Subprogram/usrfep/LeapFrog

Program:

```
(ADR_PARA)

; total retract height
10 RETRACTHEIGHT = 10
; height which has to be moved vertically
20 VLIFT = 2

; read current position (start point)
30 CUR_X = PCS("X")
40 CUR_Y = PCS("Y")
50 CUR_Z = PCS("Z")

; if no coordinate is given, the endpoint equals the start point
60 IF (X = NUL) THEN
70   X = CUR_X
80 ENDIF

90 IF (Y = NUL) THEN
100  Y = CUR_Y
110 ENDIF

; calculate the length of the desired movement in X/Y plane
130 DX = X - CUR_X
140 DY = Y - CUR_Y
150 LENG = SQRT(DX * DX + DY * DY)

; do some spline calculation (application dependent)
120 L_PARAM=2*(RETRACTHEIGHT - VLIFT)
```

```

; external influence: If @1 == 0, then enable Leap Frog,
; else do a linear G0 movement without Leap Frog

160 IF (@1 = 1) THEN
N170 G000 X=[X] Y=[Y]
180 ELSE
    ; Leap-Frog-Logik

190 IF (LENG < 2*RETRACTHEIGHT) THEN
    ; too short for leap frog. Do NOT use rapid mode (as the
    ; height control may not be dynamic enough even for short
    ; movements
N200 G1 X=AC([X]) Y=AC([Y])
210 ELSE
    ; generate 3 movements:
    ; vertikal upwards
    ; horizontal
    ; vertikal downwards
    ; use spline cornering (SCOL) to increase corner velocity
N220 G000 Z=AC([RETRACTHEIGHT + CUR_Z]) SCOL(L1=[L_PARAM], L2=[L_PARAM])
N230 G000 X=AC([X]) Y=AC([Y]) SCOL(L1=[L_PARAM], L2=[L_PARAM])
N240 G000 Z=AC([CUR_Z])
250 ENDIF

260 ENDIF

```

*Example:***Programming examples**

N10 G0 X100 Y200 **generated (equivalent)** N10 P LeapFrog(X=100, Y=200)

N20 G0 X100 **generated (equivalent)** N20 P LeapFrog(X=100). The Y-coordinate retains its position.

N30 G00 X100 is not changes (G00 is not replaced by the macro definition file), i.e. no LeapFrog subroutine is executed.

N40 G0 X100 M992 Y100 **generates** N40 P LeapFrog (X=100) M992 Y100, i.e. axis Y is first traversed with the original G-code (e.g. G1). Subsequently, the LeapFrog subroutine is called.

3.9.9 Configuration and error recovery

The default setting of the NC control cannot allow more than 8 subroutine levels. If more than eight subroutine levels are used, the following error messages can occur. The steps required to recover the error are listed.

Channel Error 2085**Error text:**

Too many open files

Cause:

Every subroutine is opened while linking or editing. The number of open files per channel is restricted by a machine parameter.

Recovery:

Increase machine parameter 7060 00210.



The machine parameter should be set to the maximum nesting depth reached by the programs plus 3, plus the number of the files opened in the CPL.

Channel error 1927**Error text:**

The block preparation is limited to NNN blocks.

Cause:

Every subroutine jump used an NC block. If too many subroutine jumps are programmed without any traversing motion, the NC blocks used are not enabled soon enough. That causes a so-called resource deficit.

Recovery:

- Increasing the parameter /NCO/LookAh/Ch/NofBlkPrep (MP 7060 00110)
- or -
- disable the block look-ahead for the respective area with G9. Then, enable the block look-ahead with G8 if required
- or -
- program the traversing motions between the subroutine jumps if possible.
- Parameter <AdditionalNCBlockMem> in the startup.xml file

The CNC control requires memory especially for the NC block data. The memory requirement is estimated using the configuration and is reserved during control startup for this purpose. In some use cases, this evaluation cannot be sufficient so that the waiting state of the channel 20 (waiting for resources) or the channel error 1927 is generated in case of lack of memory.

If these effects occurs, the memory area for the NC block data has to be increased. This should be realized via parameter AdditionalNCBlockMem in the "startup.xml" file. The parameter is specified in bytes and added to the internally determined memory size.

Due to the variety of applications and configurations of the CNC control, it is not possible to specify a value valid for all applications. As a rule of thumb, 10000 bytes per configured NC block in the channel (Macoda parameter 7060 00110) can be assumed. For example, if 300 blocks have been configured for channel 1 and 500 have been configured for channel 2, a good value would be $(300+500)*10000 = 8000000$ - provided that NC programs are processed simultaneously in both channels and a very high number of blocks is actually required.

Example of the entry for the "startup.xml" file: <AdditionalNCBlockMem>8000000</AdditionalNCBlockMem>

The template of the "startup.xml" file is contained in the "/feprom" directory. The changed file has to be stored in the directories "/usrfep" or "/" (root directory) of the control.

The option to increase the memory size by creating another unused channel with a high number of blocks is not recommended. The memory area is increased by the additional channel but the memory requirement and the administrative expense is also increased by the additional channel. The parameter contained in "startup.xml" is much more memory-efficient.

Channel error 2716 Error text:

Not sufficient runtime memory available (Stack).

Cause:

The local CPL variables of the called program are saved on the CPL stack when the subroutine is called. If this error occurs, the CPL stack memory required is higher than the value set.

Recovery:

- Parameter NCO/Mem/ChMemCh/CplStack (MP 7070 00010), increase index 1
- or -
- if possible, use less or smaller CPL variables

Channel error 825 or 465

Error text:

Insufficient memory for program XXXX

Cause:

The link table memory is insufficient for the program specified.

Recovery:

Parameter NCO/Mem/ChMem/Ch/CplLinkBuff (MP 7070 00010), increase index 2

3.10 Macro programming

3.10.1 Overview

Macros are input tools replacing parts of the NC block.

Macros consist of:

- A name,
- a replacement text, and
- Attributes (DIN or CPL).

Macros are configured in definition files. The exact function and the procedure is explained in "MTX Functional Description", chapter "Macro programming".

Macros are generally only active in the context (DIN or CPL) for which they have been configured. Apart from that, macros are used like CPL or NC words.

Example:

Using macros

Macro name:	G0
Replacement text:	ACC(X3.0, Y3.0, Z3.0) G0
Programmed block:	N10 G0(NIPS) M5
Acts like:	N10 ACC(X3.0, Y3.0, Z3.0) G0(NIPS) M5

More complex examples are contained in the "MTX Functional Description".



Macros can override NC functions. Thus, existing part programs may react differently than expected.

Macros overriding NC functions should be avoided.

It is recommended to give macros unique and recognizable names.

3.10.2 Macros with parameters

In order to design macros in a more flexible form, the user is able to define parameterizable macros.. If a macro with parameters is to be called in the

part program, the parameters must follow the macro name in round brackets. The parameters are separated by commas. If a parameter is to contain commas or round brackets, it must be written between inverted commas. If inverted commas are to be transferred in the parameter text as such, they are to be provided with a preceding backslash ("").

Example:

Using macros with inverted commas

Macro name:	Note
Replacement text:	PRN#(0,{1})
Programmed block:	10 Note(" \"Value A\$=\" , A\$ ")
Acts like:	10 PRN#(0, "Value A\$=" , A\$)

Parameters are inserted into the replacement text as character string exactly in the same form as they have been programmed. There is no check for syntactic correctness.

Example:

Using macros with parameters

Macro name:	Travel_X
Replacement text:	G0 X[{1}]
Programmed block:	N10 Travel_X(@J) M5
Acts like:	N10 G0 X[@J] M5
Macro name:	Move
Replacement text:	G0(NIPS) ACC({2}) {1}
Programmed block:	N10 Move(X100 Y100, "X3.0, Y3.0, Z3.0") M5
Acts like:	N10 G0(NIPS) ACC(X3.0, Y3.0, Z3.0) X100 Y100 M5

More complex examples are contained in the "MTX Functional Description".

3.11 Label programming and jump instructions

3.11.1 Overview

Jump instructions can be used to continue the program execution at certain entry points.

The CNC system MTX provides the following functions for jump instructions :

-
- Label programming for standard NC blocks (DIN)
 - Label programming for CPL blocks
 - GoAhead (GOA): jump forward to a standard NC block
 - GoBack (GOB): jump backward to a standard NC block

- GoCond (GOC): conditional jump to a standard NC block
- GoTo: unconditional jump to a standard NC block
- CPL jump (GO-TO): jump to any program block
- RetTo: Subroutine return to any program block



The CPL jump GOTO is processed much more quickly as the address of the jump target is already calculated during the counter-clockwise rotation.

The jump target, however, is only determined during the program runtime for DIN commands. The label has to be searched in the program which takes longer the further the jump target is away from the GOTO command.

It is recommended to use the CPL command.

It may not be jumped to a

CASE-LABEL...LABEL-OTHERWISE-ENDCASE instruction using the CPL string GOTO.

The standard NC jump commands GoAhead, GoBack, GoCond and GoTo must not be used to jump from or into a CPL repetition command or branch command.

This affects CPL commands:

- REPEAT UNTIL
- WHILE – DO – END
- FOR STEP TO NEXT
- IF THEN ELSE ENDIF
- CASE-LABEL...LABEL-OTHERWISE-ENDCASE INSTRUCTION USING THE CPL STRING GOTO.

3.11.2 Labels in standard NC blocks and CPL blocks

A label is a jump marker for a jump instruction. The CNC system MTX differentiates between labels in a standard NC block (DIN) and a CPL block.

Label programming in a standard NC block (DIN):

- The target of the jump must always be programmed directly at the start of the block.
- In blocks with a block number, the target of the jump is located directly behind the block number and must be separated by a space.
- The label name can consist of 2 to 32 characters. Letters, underlines and numbers are allowed; however, the first two characters must not be numbers. A distinction is made between upper-case and lower-case letters.
- At the target of the jump, a colon must be programmed behind the label name.



Due to reasons of compatibility, DIN labels should exclusively be programmed **behind the block number**. If a label is directly programmed at the beginning of the block and it starts with the same character sequence as the NC function, a syntax error occurs.

Since there are always new NC functions in new versions, it might occur that existing program does not run anymore in newer versions due to syntax errors.

It is strongly recommended to program DIN labels behind the block to avoid this problem.

Label programming in a CPL block:

- The target of the jump is located directly behind the block number and must be separated by a space.
- The label name consists of a decimal point followed by ASCII characters, starting with a capital letter.
- A label must not be a variable.

3.11.3 GoAhead (GOA) Forward jump to a standard NC block

Description: Continues the program execution at a jump target (label) without any conditions.

The following applies:

- The target of the jump must be defined.
- As seen in relation to the current program block, the target of the jump must be located in the direction of the end of the file.

For the required label programming, see page [chapter 3.11 "Label programming and jump instructions"](#) on page 48.

Syntax:

GoAhead <Label>	
Short form: GOA	
with	
<Label>	Jump target name 2 to 32 characters Letters, underlines and numbers are allowed; however, the first two characters must not be numbers.

Tab. 3-8: Syntax of GoAhead

Example:

```

:
N40 GoAhead LABEL1      Jump forward to the target "LABEL1".
:
N80 LABEL1:             Label programming of the jump target "LABEL1".
:

```



The standard NC jump commands GoAhead must not be used to jump into a CPL repetition or branch instruction or out of a CPL repetition or branch instruction.

For the relevant CPL commands, see page [chapter 3.11.1 "Overview"](#) on page 48.

3.11.4 GoBack (GOB) Backward jump to a standard NC block

Description: Continues the program execution at a jump target (label) without any conditions.

The following applies:

- The target of the jump must be defined.
- As seen in relation to the current program block, the target of the jump must be located in the direction of the start of the file.

For the required label programming, see page [chapter 3.11 "Label programming and jump instructions" on page 48](#).

Syntax:

GoBack <Label>	
Short form: GOB	
with	
<Label>	Jump target name 2 to 32 characters Letters, underlines and numbers are allowed; however, the first two characters must not be numbers.

Tab. 3-9: Syntax of GoBack



Note that infinite loops may be accidentally generated when programming backward jumps!

Example:

```

:
N40 LABEL1:           Label programming of the jump target "LABEL1".
:
N80 GoBack LABEL1     Return to the target "LABEL1".
:                     If no additional jump command is programmed be-
                       tween N40 and N80, the program runs through an in-
                       finite loop between N40 and N80!
  
```



The standard NC jump command GoBack must not be used to jump to or from a CPL repeat or CPL branch instruction.

For the relevant CPL commands, see page [chapter 3.11 "Label programming and jump instructions" on page 48](#).

3.11.5 GoCond (GOC) Conditional jump to a standard NC block

Description: Continues the program execution at a specified block number if the "conditional jump" input signal was active at the channel interface at the time of the block preparation.

The following applies:

- The entered block number must exist.
- The entered block number can have any position within the program file in relation to the current program block.

Syntax:

GoCond N<Number>
Short form: GOC

with	
<Number>	Block number Number in integer or real format with max. 15 digits. If the target block number is programmed with preceding zeros (e.g. "N0020 ..."), preceding zeros must be programmed here as well. ("GoCond N0020").

Tab. 3-10: Syntax of GoCond

Example:

<pre> 10 WAIT N20 GoCond N090 : : : : N090 ... : </pre>	Stop block preparation until all blocks before N20 have been processed. Then jump to N090 if the interface signal "conditional jump" is active at the time of preparation of block N20.



The standard NC jump commands GoCond must not be used to jump into a CPL repetition or branch instruction or out of a CPL repetition or branch instruction.

For the relevant CPL commands, see [page chapter 3.11 "Label programming and jump instructions" on page 48](#).

Special features and restrictions:

- Except for a possibly programmed preceding block number, no additional program words are permitted in the same block.
 - Changes in the channel interface signal "conditional jump" within the time between the block preparation and the block execution are not taken into account.
- If this behavior is not permitted for the relevant application, the user has to program a "WAIT" function in the previous program line.



Note that infinite loops may be accidentally generated when jumping in the direction of the start of the file!

3.11.6 GoTo Unconditional jump to a standard NC block

Description: Continues the program execution at any block number without any conditions.

The following applies:

- The entered block number must exist.
- The entered block number can have any position within the program file in relation to the current program block.

Syntax:

GoTo N N<Number>	
with	
<Number>	Block number Number in integer or real format with max. 15 digits. If the target block number is programmed with preceding zeros (e.g. "N0020 . . ."), preceding zeros must be programmed here as well. ("GoTo N0020").

*Tab. 3-11: GoTo syntax***Example:**

```

:
N40 GoTo N080           Forward jump to block N080.
:
N080 GoTo N40           Backward jump to block N40.
:
                        If no further jump command is programmed between
                        N40 and N080, the program runs through an infinite
                        loop between N40 and N080!

```

Special features and restrictions:

Except for a possibly programmed preceding block number, no additional program words are permitted in the same block.



Note that infinite loops may be accidentally generated when jumping in the direction of the start of the file!



The standard NC jump commands GoTo must not be used to jump into a CPL repetition or branch instruction or out of a CPL repetition or branch instruction.

For the relevant CPL commands, see page [chapter 3.11.1 "Overview"](#) on page 48.

3.11.7 CPL jump (GOTO) Jump to any program block

Description: Continues the program execution at a jump target without any conditions.

The following applies:

- A CPL block number, a standard NC block number or a "label" (jump marker) can be specified as a jump target.
- The target of the jump can have any position within the program file in terms of the current program block.

Syntax:

GOTO <Target>

*Tab. 3-12: GOTO syntax***Example:**

```

10 GOTO N20           Jump to block N20
N20 X100
30 GOTO 120           Jump to CPL block 120
...
120 GoTo .ZIEL1       Jump to label .ZIEL1

```

...

150 .ZIEL1



It may not be jumped to a CASE-LABEL...LABEL-OTHERWISE-ENDCASE instruction using the CPL string GOTO

For label programming conditions, see page [chapter 3.11 "Label programming and jump instructions"](#) on page 48.

3.11.8 RetTo subroutine return to any program block

Description: Executes a subroutine return by one or multiple levels without any conditions and continues the program sequence at the requested target block.

The following applies:

- Either a block number or a jump label (for standard NC block or for CPL block individually) is specified as target block
- The target block can have any position within the target program file with regard to the position of the subroutine call
- If no other parameter than the target block is specified, there is a return by exactly one program level
- Optionally, returning to multiple program levels is possible. Therefore, use a level parameter to either specify the difference to the current program level (negative values) or to specify a program level (positive values) as target level

Syntax:

RetTo (<TargetBlock>{, <TargetLevel>})	
with	
<Target block>	Block number or jump label (for standard NC block or for CPL block individually)
<TargetLevel>	Absolute or incremental program level > 0: Return to the specified absolute program level = 0: Return to the main program < 0: Return to the specified number of program levels If no target level is specified, it is equivalent to -1, i.e. return by one program level

Tab. 3-13: Syntax RetTo

Special features and restrictions: The RetTo is not permitted

- in the channel initialization programs ("Init program") and their subroutines
- in the main program
- in the operation mode "Manual Data Input" as direct entry or in the first program level of one of the programs called there

Additionally, the RetTo causes an error if

- no target block is specified
- the target block was not found

- an absolute target level is not lower than the current program level
- the number of program levels to be returned is higher than the current program level for an incremental target level

Local subroutines are an individual program level and are thus handled like all other subroutines.

Examples:

N100 RetTo (N500)	Return by one level to the standard NC block N500
N100 RetTo (500)	Return by one level to the CPL block 500
N100 RetTo (STDLABEL)	Return by one level to the standard NC block with the jump label "STDLABEL"
N100 RetTo (.CPLLABEL)	Return by one level to the CPL block with the jump label ".CPLLABEL"
N100 RetTo (N500, -1)	Return by one level to the standard NC block N500 (identical to the first example)
N100 RetTo (500, -2)	Return by two levels to the CPL block 500
N100 RetTo (STDLABEL, 2)	Return to program level 2 to the standard NC block with the jump label "STDLABEL"
N100 RetTo (.CPLLABEL, 0)	Return to the main program to the CPL block with the jump label ".CPLLABEL"



With RetTo, it may not be jumped into a CASE-LABEL...LABEL-OTHERWISE-ENDCASE instruction.



With RetTo, it may not be jumped out of the CASE-LABEL...LABEL-OTHERWISE-ENDCASE instruction when returned to a standard NC block jump label.

For label programming conditions, see page [chapter 3.11 "Label programming and jump instructions" on page 48](#).

3.12 Decision and branch instructions

3.12.1 Overview

Decision and branch instructions are used to execute individual program blocks and sections or entire subroutines in relation to certain events.

The CNC system MTX provides the following possibilities:

- "Skip block" function for standard NC blocks (DIN)
- CPL instruction IF-THEN-ELSE-ENDIF
- CPL instruction CASE-LABEL...LABEL-OTHERWISE-ENDCASE

3.12.2 "Skip block" function

Description: With the help of this function, individual standard NC blocks (DIN) can be skipped by the control. To do so, program the "/" sign at the beginning of the corresponding program lines.

The marked program blocks are skipped only if the interface signal "qCh_BlockSlash" (skip block) is set for the bit interface of the corresponding channel.

Example:

```

:                               The interface signal "skip block" is activated.
:
/N100 ...                       Block N100 is ignored.
:                               The interface signal "skip block" is deactivated.
:
/N300 ...                       Block N300 is processed.
:

```



The "skip block" function can be used only for standard NC blocks (DIN)!

3.12.3 CPL instruction: CASE-LABEL...LABEL-OTHERWISE-ENDCASE

Description: Within a program, it is often necessary to request **more than two states** of an integer expression or an integer variable. In such cases, a query by means of an IF command is only possible with the use of several nested IF commands. This requires additional computing time and worsens the readability and maintainability of the program.

These drawbacks may be avoided by using the CASE structure:

```

CASE <integer expression> OF
LABEL <int. constant>[ , <int. constant>][ : <instruction>]
<Instruction>
:
LABEL ...
:
[OTHERWISE <command>
<Instruction>
:]
ENDCASE

```

Subsequent to the CASE command, the program run branches to the LABEL command in which one of the <integer constants> is equal to the value of the <integer expression>. Now, all commands up to the next occurrence of the LABEL or OTHERWISE commands are carried out. The program then branches directly to the ENDCASE instruction.

If there is no LABEL statement that fulfills this condition, the program branches to the OTHERWISE statement or (if OTHERWISE has not been programmed) directly to the ENDCASE statement.

The <Command> area of a CASE structure may include all CPL commands. A maximum of 10 CASE structures can be nested.

Examples:

```

10 CASE A% OF
20 LABEL 0 : Y=1
30 LABEL 2
40 Y=Y*Y
50 LABEL 4 : Z=Y*Y
60 Y=Z*Z
70 OTHERWISE Y=0

```

```
80 ENDCASE
```

```
10 CASE (INT(X/Y)+C%) OF
20 Label 1.2 : X=1 : Y=2
30 Label 4.8
40 X=2 : Y=4
50 LABEL 0
60 X=0 : Y=1
70 OTHERWISE X=0 : Y=0
80 ENDCASE
```

```
10 CASE INTFELD%(1,2) OF
20 LABEL 1,2,3 : GOTO .MARK1
30 LABEL 4,5,6 : GOTO .MARK2
40 OTHERWISE GoTo .ENDE
50 ENDCASE
```



The standard NC jump commands GoAhead, GoBack, GoCond, GoTo, must not be used to jump from or into a CASE-LABEL...LABEL-OTHERWISE-ENDCASE command.



It may not be jumped to a CASE-LABEL...LABEL-OTHERWISE-ENDCASE instruction using the CPL string GOTO

3.12.4 CPL instruction: IF-THEN-ELSE-ENDIF

Description: This function is a simple conditional branch instruction:

"IF a specific condition is fulfilled, THEN perform the routine or ELSE perform the other routine."

Syntax:

IF <condition> THEN <routine> [ELSE <alternative routine>]	
ENDIF	<i>The following applies:</i> • The condition is located in the same line as "IF" and is concluded by "THEN" in the same line. • Both the THEN and the ELSE routines are program branches that do not have to be processed in every case. • If the ELSE component is omitted, the program continues to run immediately after processing the ENDIF command provided that the condition is not fulfilled.

Tab. 3-14: IF-THEN-ELSE-ENDIF syntax

Similar to the abort conditions for loop commands, the condition for the IF command may contain arithmetic, trigonometric and logic operations. Here, nesting is possible, too.

The IF command must always be concluded with an ENDIF command since otherwise the end of the routine or the end of the alternative routine will not be detected. Since the placement of the ENDIF command depends on the

program processing logic, the control sometimes fails to reliably detect and interpret a missing ENDIF command. This results in ambiguous error messages. It is therefore good practice for the programmer to verify the completeness of the IF command.

Example:

```
...
10 X = 1
20 .START
30 IF X>=100 THEN
40 GOTO .ENDE
50 ELSE X=X+2.75
60 GOTO .START
70 ENDIF
...
90 .ENDE
...
```



The standard NC jump commands GoAhead, GoBack, GoCond, GoTo may not be used to jump into or out of an IF-THEN-ELSE-ENDIF instruction.

3.13 Repeat statements

3.13.1 Overview

If one or more program blocks are to be processed repeatedly in relation to certain conditions, this can be programmed with the help of **CPL repeat instructions**. The multiple repetition of the program is also known as a loop.

The CNC system MTX provides the following possibilities:

- CPL command FOR-STEP-TO_NEXT
- CPL command REPEAT-UNTIL
- CPL command WHILE_DO_END

3.13.2 CPL instruction: FOR-STEP-TO-NEXT

Description: If the abort condition for the repeat instruction is to be a direct consequence of the processing of the routine, a tracking counter would be required, for example.

This counter requires no specific programming for the FOR NEXT loop. A counting variable (integer) is determined whose starting and end counter has to be specified. If the counting step width deviates from 1, the step width (STEP) can be specified separately.

Syntax:

FOR <CountingVar.>=<InitialValue> [STEP <StepWidth>]
TO <EndValue><Routine>
NEXT [<Count.Variable>]

Tab. 3-15: FOR-STEP-TO-NEXT syntax

Example:

```
10 FOR I%=1 TO 18
20 XSINUS(I%)=SIN(I%*10)
30 NEXT I%
```

After the end of the loop, the counting variable is provided with a value greater than the end value (max. step width).

In this example, the sine values for 0 to 180 degrees are written to the XSINE field. The "I%" that was attached to "NEXT" in line 30 serves for clarification purposes only and may be omitted.

It is also possible to program FOR NEXT loops with a **variable step width**. In this case, the step width variable should be the same type of variable as the counting variable.

Example: *Program:*

```
10 OPENW(1,"P222",130)
20 STEP%=2 : START%=1 : END%=3500 : NJUST
30 FOR COUNTER%=START% STEP STEP% TO END%
40   STEP%=ROUND(STEP%*SQRT(STEP%))
50   PRN#(1,"COUNTER: ",COUNTER%,"step width: " ,STEP%)
60 NEXT
70 CLOSE(1)
```

Subsequent to this program execution, the following is displayed in the "P222" file:

Example: *Program:*

```
COUNTER:      1 STEP WIDTH:      3
COUNTER:      4 STEP WIDTH:      5
COUNTER:      9 STEP WIDTH:     11
COUNTER:     20 STEP WIDTH:     36
COUNTER:     56 STEP WIDTH:    216
COUNTER:    272 STEP WIDTH:   3175
COUNTER:   3447 STEP WIDTH:  178902
```



The standard NC jump commands GoAhead, GoBack, GoCond, GoTo may not be used to jump into or out of a FOR STEP TO NEXT instruction.

3.13.3 CPL instruction: REPEAT-UNTIL

Description: The REPEAT loop can be used if the abort condition for the repeat command is to be queried only after the first time that the routine has been processed.

Syntax:

```
REPEAT <routine> UNTIL <condition>
```

Tab. 3-16: REPEAT-UNTIL syntax

Example:

```
:
30 REPEAT
40 X=X+1                      Loop until X = 100
50 UNTIL X=100
:
```



The standard NC jump command GoAhead, GoBack, GoCond, GoTo, must not be used to jump into or out of a REPEAT-UNTIL command.

3.13.4 CPL instruction: WHILE-DO-END

Description: If the abort condition for the repeated instruction is to be queried before the initial loop iteration: "While the condition is fulfilled, execute the routine!".

The WHILE loop has the following structure:

Syntax:

WHILE <condition> DO <routine> END

*Tab. 3-17: WHILE-DO-END syntax***Example:**

```

:
30 WHILE SD(9)=0 DO    Wait loop until SD(9) assumes a value of 0.
40 I=I+1
50 END
:

```

Tab. 3-18:

The standard NC jump command GoAhead, GoBack, GoCond, GoTo, must not be used to jump into or out of a WHILE-DO-END command.

3.14 Variable programming

3.14.1 Variable names



Variable programming is included in the scope of CPL languages!

Variable programming in the CPL is used to design programs so that parameters can be set within them and thus adapting the program execution to the current conditions.

- A variable is any symbol name to which a few special boundary conditions apply:
- - Variable names must be unique.
 - Variable names may not be identical to reserved CPL command words.

Due to the formal separation between standard NC and CPL programming, however, variable names can theoretically be identical to the names of NC functions or NC function parameters; for example, a variable "X" might be defined although there is already an axis with the designation "X" in the system.

- The variable name consists of any order of upper-case letters and numbers; the first character must be an upper-case letter. A variable name which consists of the letter "N" and is only followed by numbers is not a valid variable name. This character string corresponds to the identification of an NC block.



Since only the first 8 characters of the name of the variable are used to distinguish the name, only these 8 characters are termed significant (except for: definable permanent variables).

Do not use variable names beginning with a "P" and followed by a digit.

- There is a total of 3 variable groups that specify the validity range of the variables. The variable group is specified by appending an identifier at the beginning of the variable name. This character is always counted among the significant characters of the name!

There are the following **variable groups** with the corresponding identifiers:

- Local variables: no special identifier
- Global variables: "#"
- Permanent variables: "@"
- The **type of variable** is always specified by appending an specific identifier to the end of the name of the variable. This also applies if the variable name exceeds the number of significant characters.

Win the declaration files for definable permanent variables ("wmhperm.dat" and "anwperm.dat"), this ID is omitted. The respective **CPL data type** is to be specified instead.

There are the following variable types with the corresponding identifiers:

Variable type	CPL data type	Specific variable identification
Integer	INT	"%"
Floating point number with simple accuracy	REAL	no special identification
Floating point number with double accuracy	DOUBLE	"!"
Logic variable	BOOL	"?"
Characters	CHAR	"\$"

Tab. 3-19: Variables

Examples:

Local, global and permanent variables:

10 NUMBER1% = 1	Local variable of CPL type INT
20 #NUMBER2% = 2	Global variable of CPL type INT
30 @36% = 3	Permanent variable of CPL type INT
40 @ABCD% = 4	Defined permanent variable of CPL type INT

3.14.2 Variable groups

Introduction

Declarations with regard to the effective range of variables are required due to the option of using subroutines as well as due to the possible requirement to commit the values of variables to intermediate storage irrespective of the program being executed. For this purpose, a distinction is made between the following groups of variables.

Local variables

Local variables take effect only within the program for which they have been declared. When this program reaches the end of program (EP), the variables are deleted, thus releasing the occupied memory. At a subroutine call, a variable name local to the calling program is not "visible" to the subroutine. For this reason, the same variable can also be declared as a local variable in the subroutine without consequential interference due to the similarity of their respective names. Upon return to the calling program, the original local variable is again available with the value from the moment the subroutine was called.

Global variables

Global variables are identified by a preceding # (number sign, gate or hash) character. Once a value has been assigned to a global variable, it can be accessed, read and/or modified from within all program parts for the remainder of the entire program. Global variables are deleted subsequent to end of program (EP).

Permanent variables

Permanent variables are identified by a leading @ character that is followed by the name of the variable. They can be addressed by any active program. Permanent variables are stored to a separate memory area and are always retained after the end of a program, after a control reset and after an activation or deactivation. Deletion is possible only through direct overwriting.

Permanent variables of the CPL type INT can be addressed under the designations of @1 to @100 (for detailed information on the CPL type INT, see [chapter 3.14.3 "Variable types" on page 66](#)). To improve program readability, the indication of such permanent variables can also be augmented by appending letters to the number.

In addition, the permanent one-dimensional array variable @_R can be used with 100 elements of the CPL type DOUBLE. The two permanent variables @_RES_DOUBLE and @_RES_DWORD are reserved for internal applications and should not be used.

Definable permanent variables

Definable permanent variables are also identified by a leading @ character that is followed by the name of the variable.

The differences to "permanent variables" are as follows:

- Definable permanent variables are not automatically declared a component of the system software but must be **manually declared** via user entry in the files named "wmhperm.dat" (for proprietary data supplied by the machine tool manufacturer) and "anwperm.dat" (for end user-specific data). The declaration syntax is discussed under the section of file structure of "wmhperm.dat" and "anwperm.dat".

During system startup, the control first searches for the files in the root directory, then in the user FEPRM, and finally in the FEPRM.

The control interprets the file identified by the first occurrence of the respective file name, using the entries found therein to create "definable permanent variables", provided that they do not already exist. Existing "definable permanent variables" that are not declared in one of the above named files are deleted.

The maximum number of possible definable permanent variables is limited by the available memory capacity. In the event that no more memory capacity is available for generating variables, the control displays a corresponding error message.

- The names of "definable permanent variables" always start with the **@ character and a character string**. This character string consists of one upper-case letter, followed by any combination of upper-case letters or numbers.

In the case of the "definable permanent variables", the first **16 characters** of the name of the variable are significant. If they differ only from the 17th character, the CPL interprets them as a single variable!

- Defined permanent variables may be of the CPL types **INT**, **REAL**, **DOUBLE**, **BOOL** or **CHAR**. The type of variable is specified by appending an identifier to the end of the name of the variable.

This specification must be entered in the part program:

- @ABCD% def. perm. Variable of CPL type INT
 - @EFGH def. perm. Variable of CPL type REAL type (without specific identification)
 - @IJKL! def. perm. Variable of CPL type DOUBLE
 - @MNOP? def. perm. Variable of CPL type BOOL
 - @QRST\$ def. perm. Variables of CPL type CHAR
- One-dimensional and two-dimensional **arrays** may be used.
The **maximum array index** for array variables of CPL types INT, REAL, DOUBLE or BOOL is **65535**. For array variables of CPL type CHAR, it is **1024**.

Examples:

@WZNR% (1) =4	The first variable (with index 1) of the one-dimensional array @WZNR of type INT is assigned a value of 4.
@WZKOR (2, 2) =0.2	The variable (with the indices 2,2) within the two-dimensional array @WZKOR (@TCor) of CPL type REAL is assigned a value of 0.2.

- Estimating the available **number** of newly definable permanent variables:

total memory space for permanent variables: 100 Kbytes (102400 bytes)

Pos.	Reserved for	Memory space in bytes	Note
1	All permanent variables	102400	Total memory
Of which the following are reserved for			
2	@1 - @100 (permanent variables)	800	Data type: INT To improve program readability, the indication of such permanent variables can also be augmented by appending letters to the number.
3	Administrative information	24	
4	All definable permanent variables	101576	(4) = (1) - (2) - (3)
Of which the following are reserved for			
5	@_R	823	Permanent array variable with 100 elements of CPL type DOUBLE
6	@_RES_DOUBLE	40	Permanent variable of CPL type DOUBLE, reserved for internal applications
7	@_RES_DWORD	35	Permanent variable of CPL type INT, reserved for internal applications
8	New definable permanent variables	100678	(8) =(4) - (5)- (6) - (7)

Tab. 3-20: Memory capacity for all permanent variables

Basics of NC programming

Pos.	Reserved for	Memory space in bytes	Note
9	The names of the definable permanent variables	max. 16	1 byte per character
10	The value of the definable permanent variables	1, 4 or 8	INT: 4 bytes DOUBLE: 8 byte REAL: 4 bytes BOOL: 1 byte
11	Administrative information	20	
12	A definable permanent variable of CPL type DOUBLE with a name length of 16 characters	44	e.g.: maximum memory allocation (9) + (10) + (11)

Tab. 3-21: Memory capacity for definable permanent variables

Number of "definable permanent variables" of the types DOUBLE and INT:

Variable type	Number of variables	Note
CPL type DOUBLE with a name length of max. 16 characters	2288	$100678/44 = 2288$
CPL type INT with a name length of max. 16 characters	2516	$100678/(16+4+20) = 2516$
CPL type INT with a name length of max. 8 characters	3146	$100678/(8+4+20) = 3146$
Array variables with name lengths of max.16 characters, CPL type INT	25160	$(100678-16-20)/4 = 25160$
Array variables with name lengths of max.16 characters, CPL type DOUBLE	12580	$(100678-16-20)/8 = 12580$

Tab. 3-22: Number of the "definable permanent variables"

File structure of the declaration files "wmhperm.dat" and "anwperm.dat":

The files may contain only declarations of "definable permanent variables". Each variable is declared in a separate line and has to be completed with a **semicolon**. There can be an optional comment at the end of the line.

A line of declaration always exhibits the following structure:

```
DEF <variable type> @ <variable name>; [<comment>]
```



The separator ";" has to be written even if there is no comment!

Examples of "wmhperm.dat" and "anwperm.dat"

```
DEF INT @CDEF;
DEF INT @ABCD           ; simple variable of CPL type INT
DEF REAL @EFGH          ; simple variable of CPL type REAL
DEF DOUBLE @IJKL        ; simple variable of CPL type DOUBLE
DEF BOOL @MNOP          ; simple variable of CPL type BOOL
DEF CHAR @PSTR1(3)      ; Variable of CPL type CHAR with the length 3
DEF INT @WZNR(9)        ; 1-dimensional array of CPL type INT with 9 variables
```

```

DEF REAL @WZKOR(9,2) ; 2-dimensional array of CPL type REAL with 18 variables
DEF CHAR @PSTR2(9,2) ; 2-dimensional array of CPL type CHAR with 9 partial strings of 2 characters each

```

Sample applications of permanent variables:

```

10 @1 = 1
15 @2_ZAEHLER = 2
20 @ABCD% = 3
25 @EFGH = 4.1
30 @IJKL! = 5.12345
35 @MNOP? = TRUE
40 @PSTR1$ = "ABC"
45 @WZNR%(2) = 6
50 @WZKOR(3,2) = 7.6
55 @PSTR2$(3) = "DE"

```

Structured variables

Structured variables can be replaced by system data (SD) but are retained for reasons of compatibility (see [chapter 7 "CPL commands" on page 417](#)).

Structured variables are characterized by various structure levels that are separated by a period ("."). Structured variables always start with "SV.".

All structured variables that are to be accessed in a CPL command must be "manually" **declared** via user entry in the file named **machdef.dat**. During system startup, the control first searches for the file "machdef.dat" in the root directory, then in the user FEPRM, and finally in the FEPRM. The file found first is evaluated when the NC is starting up; the management data and the memory for the structured variable user data are created at the same time. The memory is created again and all values are preset to 0 during every startup.

The file "machdef.dat" is only used for the definition of structured variables. Each definition occupies a separate line and concludes with a semicolon.

A definition line always exhibits the following structure:

```
DEF<Variable type> SV.<Variable name> ; [<Comment>]
```

<Variable type> and <variable name> are strings that can consist of lower-case or upper-case letters, numbers, a dash and an underline.



For all variable names, the first 14 characters are significant (without preceding "SV."). If they differ only beyond that point, CPL or PLC interpret them as a single variable!

Example:

```
DEF DBT1Rec_t SV.A;    Definition of a structured variable of type "DBT1Rec_t"
```

CPL commands can access both SV.A and all structure components of SV.A. The various structure components are separated from one another by ".".

```
10 SV.A.Hd=DBSEA("DBT1",-1,-1,"K1=1",FOUND%)
```

3.14.3 Variable types

Integer variable (INT)

An variable of CPL type INT requires a memory of 32 bits. It is identified by a "%" character appended to the name of the variable. The value range extends from -2.147.483.647 through +2.147.483.647.

```
10 NUMBER% = 4
```

INTEGER variable

Floating point variable (REAL)

If no special identification is appended to the name of the variable, the variable is interpreted as real variable of single precision.

In this case, the variable occupies 32 bits of memory space. The value range is $\pm 10^{38}$. This corresponds to 7 significant digits.

```
10 PI = 3.141593
```

simple precision REAL variable

Floating point variable (DOUBLE)

If an exclamation mark "!" is appended to the name of the variable, the variable is interpreted as real variable with **double precision**.

In this case, the variable occupies 64 bits of memory space. The value range is $\pm 10^{308}$. This corresponds to 15 significant positions.

```
10 PI! = 3.141592653589793
```

double-precision REAL variable

Logical variable (BOOL)

These variables are identified by a "?" question mark that is appended to the name of the variable. Logical variables (Boolean variables) can only assume the value **TRUE** or **FALSE**. They are used to store logical states or conditions required throughout the course of program execution.

```
10 START? = FALSE
```

BOOLEAN variable

Array variable (ARRAY)

ARRAY variables allow to reserve a one- or two-dimensional array under a single variable name in the memory, with the array consisting of one or more variables of the same type.

Array definitions are possible for variables of the CPL types INT, REAL, DOUBLE, BOOL and CHAR. To enable access to the individual array elements of an array, the array index and/or indices are specified in addition to the name of the array variable.

Example: Dimensioning an ARRAY variable

```
10 DIM FELDVAR (2,3)
```

INTEGER constants for field sizes (index)
variable name (REAL variable)
DIM command word

Access to an array variable

```

100 FELDVAR(1,1) = MCS(1)
110 FELDVAR(2,1) = PCS(1)
120 FELDVAR(1,2) = MCS(2)
130 FELDVAR(2,2) = PCS(2)
140 FELDVAR(1,3) = MCS(3)
150 FELDVAR(2,3) = PCS(3)

```

Prior to the initial access to the array variables, the index range or the array size must be dimensioned with integer constants:

- Array size of the array variable of CPL type INT and REAL:
max. 65536
- Array size of the array variables of CPL type CHAR:
max. 1024

```
DIM <VariableName> (<array size1>[,<array size2>])
```



Dimensioning with DIM must not be applied to "definable permanent variables". Instead, the dimensioning of these variables occurs in the file "wmhperm.dat" or "anwpwerm.dat".

Variables of CPL type CHAR and string variables

A CPL type CHAR variable is identified by a following "\$" sign. This type of variable can accommodate a single character as well as a complete character string.

However, character string commands (see section "Processing Character Strings") are possible only if a character string is stored to a one-dimensional or two-dimensional array of CPL type CHAR variables. This requires that the field be declared through a DIM command.

Each variable of CPL type CHAR in this array then contains only one character of the character string.

A one-dimensional array comprised of variables of CPL type CHAR is called a STRING variable. No index is entered when accessing a one-dimensional variable of CPL type CHAR. However, when accessing a two-dimensional variable, an index has to be entered.

Example:

```

1 REM string variable AB (length 10)
2 DIM AB$(10)
3 REM 3 string variables CD (length 5 each)
4 DIM CD$(3,5)
5 AB$ = "Z"
6 CD$(2) = "ABC"

```

Overview of variables

Variable group	Variable name	CPL type		Arrays possible (X=yes)
Local	Max. 8 significant characters	%	INT	X
			REAL	X
		!	DOUBLE	X
		?	BOOL	X
		\$	CHAR	X
Global#	incl. "#" character, max. 8 significant characters	%	INT	X
			REAL	X
		!	DOUBLE	X
		?	BOOL	X
		\$	CHAR	X
Permanent@	1 - 100			
Definable permanent@	max. 16 significant characters	%	INT	X
			REAL	X
		!	DOUBLE	X
		?	BOOL	X
		\$	CHAR	X

Tab. 3-23: Overview of variables

3.14.4 Variables within standard NC programming

Frequently, NC functions or parameters of NC functions with variables are to be parameterized in a part program within standard NC programming (DIN).

While assigning such values, note that CPL expressions within standard NC programming must be enclosed by square brackets "[" and "]".

Example: Parameterization of standard NC blocks:

```

10 Angle = 45
20 VAL1 = 1.5
30 VAL2 = 1.5
40 XPOS = 10.2
50 YPOS = 5.73
60 FEEDRATE = 1000
N70 Rotate([ANGLE])
N80 Scale (X[VAL1],Y[VAL2])
N90 G1 X[POS] Y[POS] F[FEEDRATE]
```



The block number cannot be parameterized using CPL variables!



None of the addresses calling a subroutine are intended for variable syntax!

3.14.5 Variable ERRNO for evaluating errors in CPL functions "ERRNO"

The CPL variable ERRNO, in which the called function returns errors, can be transferred as a transfer parameter to a CPL function generating a runtime error in the case of an error.

The CPL variable ERRNO is an optional parameter that can be programmed in those CPL functions in which it has been entered as a parameter. It can be programmed at any position there.

If ERRNO is not programmed, the CPL function generates a runtime error or a warning if an error occurs.

If ERRNO is programmed, the CPL function does **not** generate a runtime error or a warning. In this case, the CPL variable ERRNO must be evaluated in the CPL program and a runtime error (using SETERR) or a warning (using SETWARN) must be generated explicitly.

If the CPL function has been executed directly, the value of ERRNO is 0. Errors are acknowledged through negative values. The meaning of the negative values is generally valid; however, not all values are relevant for every CPL function. The error values possible for a CPL function are specified for the appropriate function.

List of generally valid error values:

0:	Access OK
-1:	Parameter error
-2:	Coordinate/axis/spindle does not exist
-3:	Invalid coordinate/axis in channel
-4:	Axis is not a pseudo-coordinate
-5:	Channel does not exist
-6:	Function can only be called in its own channel
-7:	Data could not be read
-8:	Name of source file, including path, is too long
-9:	Access to source file not possible
-10:	Name of target file, including path, is too long
-11:	File name (source or target) invalid
-12:	Copying not possible
-13:	Data could not be written
-14:	XML table exists
-15:	XML table not found
-16:	Invalid file extension
-17:	Invalid table type
-18:	Insert position faulty
-19:	Maximum number of axes exceeded
-20:	Incorrect root tag
-21:	File access not possible
-22:	Invalid key
-23:	Data block is locked
-24:	no read access for the file
-25:	no write access for the file
-26:	Command with invalid parameter value

-27:	Command with invalid parameter type
-28:	Error during access to XML table
-29:	Incorrect array index
-30:	Insufficient memory
-31:	Search condition incorrect
-32:	One data block of this database table is locked in the channel
-33:	Data block not locked in the channel
-34:	Variable on the right of the assignment not assigned
-35:	Internal error at system data access
-36:	Sercos service channel blocked
-37:	Unknown Sercos ident number
-38:	Invalid Sercos ident number
-39:	Invalid axis number
-40:	Freeze block does not exist
-41:	Command not permitted
-42:	Limit value violated
-43:	Invalid element number
-44:	No measured value available for the axis
-45:	The measured value of the coordinate cannot be calculated. There is no measured value for at least one axis that is part of the axis transformation or involved in a rotation

Example: Read the position of channel axis X in the first channel. One of the following four lines (all with the same effect) can be programmed:

```
10 POS = ACS ("X", 1, 1, ERRNO)
10 POS = ACS ("X", 1, ERRNO, 1)
10 POS = ACS ("X", ERRNO, 1, 1)
10 POS = ACS (ERRNO, "X", 1, 1)
```

3.15 General system information

3.16 Reserved command words

The keywords listed here must be standing alone or have to be separated by a special symbol. This identifies them as command words. The selection of names for variables must not encompass any reserved instruction words!

Example:	<code>GOTO 10</code>	Jump to line 10
	<code>GOTO10</code>	Symbol name (variable); if alone, it leads to the error message "Runtime error 2167 = missing" since a value assignment for the variable "GOTO10" is expected.

Keywords:

A:	ABS ACOS ACS AND APOS ASC ASIN ATAN ATAN2 AXADR\$ AXINF AXO AXP	B:	BCD BIN BITIF	C:	CALL CASE CHR\$ CLOCK CLOSE CLRWARN COF COS CPROBE	D:	DATE DBLOAD DBMOVE DBSAVE DBSEA DBSEAX DBTAB DBTABX DBTABXL DCT DIM DIRCR DIRDEL DIRINF DO DPC
E:	ELSE END ENDIF ENDCASE EOF ERASE ERRNO	F:	FALSE FILEACCESS FILECOPY FILEDATE FILENO FILEPOS FILESIZE FOR FXC FXCR FXDEL FXINS	G:	GETERR GOTO	I:	IF INP# INSTR INT
L:	LABEL LEN LJUST	M:	MCA MCASE\$ MCODS MCOPS MCS MID\$ MMC	N:	NCF NEXT NJUST NOT NUL	O:	OF OPENR OPENW OTHERWISE OR

P:	PCS PCSPROBE PDIM PLC PMT PMV PPOS PRN# PROBE	R:	REM REPEAT REWRITE ROUND	S:	SCL SCS SCSL SCS3L SD SDLOAD SDR SDSAVE SEEK SETERR SETWARN SIN SPOS SQRT STEP STR\$	T:	TAN TCV THEN TIME TO TRIM\$ TRUE
U:	UNTIL	V:	VAL VARINF VERSINF\$	W:	WAIT WCS WHILE	X:	XOR XTAB XTABCR
Z:	ZOCDEL ZOCINS ZOT ZOTCR ZOV						

Tab. 3-24: Keywords

3.17 CPL commands

3.17.1 Value assignment

Assignment

Local as well as global variables can be associated with values. This is accomplished with the use of the "=" (equals) sign.

Examples: Value assignment for a Boolean variable

```
10 START? = FALSE
```

value
assignment character
(logical) variable

Value assignment for a real variable

```

1 X1MIN! = 2097.876

```

value (max. 7 digits)

assignment character

double-precision REAL variable

Value assignment between variables

```

1 XSET = X1MIN!

```

value (double-precision REAL variable)

assignment character

simple-precision REAL variable

The variable to assign a value to has to be positioned on the left of the assignment symbol and the respective value on the right. This declaration must be used with caution especially in cases where the value of one variable is to be assigned to another variable.

NUL

If no value has been assigned to a variable, it has the value of NUL. As a consequence, the statement <variable>= NUL is true. This signifies that the equal sign can also be used in expressions representing comparisons or conditional operations.

To delete a specific local or global variable, assign the NUL value. A permanent variable cannot be deleted but requires overwriting.

Example: *Delete variable*

```

1 XSOLL = NUL
2 IF XSOLL = NUL THEN
3     PRN#(0,"Variable not assigned.")
4 ENDIF

```



For variables without an assigned value or with the value deleted by assigning NUL, no value can be assigned to other variables! However, querying the variables to NUL is always possible.

3.17.2 Mathematical operations

Simple functions

In addition to the assignment of a value in the form of a constant expression (numerical) or as a variable, it is also possible to assign the value of a CPL expression to a variable. A CPL expression may contain functions using both constants and variables.

The four basic arithmetical operations are the simplest functions:

- Addition "+"
- Subtraction "-"
- Multiplication "***"
- Division "/"

As a rule, "multiplication and division take priority over addition and subtraction", i.e. multiplication and division is carried out at first, followed by addition and subtraction. It is also possible to use parentheses, the nesting of which,

to a depth of seven, can be used with simple expressions (containing no function calls).

Example:

```
1 I% = 25: XACTUAL = 10
2 XCOMMAND = 150/(100-I%)+XACTU-  XCOMMAND has a value of 12
  AL
```

Arithmetical functions with an effect on variables, constants or CPL expressions can still be called. These have to be placed in parentheses immediately behind the respective instruction word. The function always refers to the internal numerical representation of the input value. This representation can be verified during program execution with the use of "program check". In the case of nested expressions, and particularly when these contain function calls, the maximum possible nesting depth must be considered. It is dependent upon the memory capacity required by the expressions within parentheses during their respective execution.

ABS

Specifies the absolute value of the input value, i. e. negative values become positive while positive values remain positive.

Example:

```
1 I% = -125
2 XVALUE = 2*SQRT(ABS(100+I%))  XWERT has value 10
```

EXP

Calculates the exponential function regarding an arbitrary programmed basis or a default basis e (e=2.718281828...) if no <Basis> (optional parameter) is specified.

Syntax:

EXP(<Exponent> [, <Basis>])

Tab. 3-25: EXP syntax

Example: *Program:*

```
1 A! = EXP(2)    : REM calculates e^2 and assigns the result to A!
2 B! = EXP(3.2) : REM calculates 2^3 and assigns the result to B!
```

INT

INT converts the input value (REAL) to an integer number (CPL type INT) by skipping the digits after the decimal point (rounding down). The input value may be a constant or a variable.

Example:

```
1 XVALUE% = INT(10.9)  XWERT has value 10
```

RANDOM

This function generates a random integer number. The value is between 0 and 32767 by default.

The value range of the random number can be specified by the optional parameters <MinValue> and <MaxValue>.

It has to apply:

<MaxValue> - <MinValue> <= 32767

Syntax:

RANDOM ([[<MinValue>] [, <MaxValue>]])	
<MinValue>	optional: Minimum value for the random number (positive number)
<MaxValue>	optional: Maximum value for the random number (positive number)
ERRNO	CPL variable If the CPL variable ERRNO is programmed at any position in the parameter list, the command does not generate any internal run-time error. The error is returned by a corresponding return value of the variable. The random number 0 is generated in case of error. The following return values are possible: 0: Access OK. -1: Parameter error If the CPL variable ERRNO is not entered, a runtime error is generated when an parameter error occurs.

Tab. 3-26: *RANDOM syntax*

Example:

```

1 REM random number 0 or 1
2 I%=RANDOM(0,1)
3 REM random number between 10 and 100
4 I%=RANDOM(10,100)

```

ROUND

ROUND converts the input value into an integral number (CPL type INT) by rounding it up or down. The input value can be a real expression.

Example:

```

1      XVALUE%      = XWERT has value 11
ROUND(10.9)

2 XWERT% = ROUND(5.5) XWERT has value 6

3      XVALUE%      = XVALUE has a value of 5
ROUND(5.49)

```

SQRT

Forms the square root of an input value. The input value must not be a negative value; this is not defined.

Example:

```

1 I% = 44

2 XSET = 4*SQRT(100+I%) XSOLL has value 48

```

SIN, COS, TAN, ASIN, ACOS, ATAN

If trigonometric functions process angles in terms of conventional degrees, it is useful to mark these angles as double-precision real variables.

The trigonometric functions SIN, COS, TAN have angles in degree as input variables and provide numerical values as results. Accordingly, the ASIN, ACOS, ATAN inverse functions have numerical values as input variables and angles (in degrees) as results.

It is useful to declare the angles as double-precision real variables.

The following trigonometric functions can be used:

-
•	SIN	Sine function
•	COS	Cosine function
•	TAN	Tangent function
•	ASIN	Arc sine function
•	ACOS	Arc cosine function
•	ATAN	Arc tangent function
-

Example:

1	Angle! = 30	
2	ERG! = SIN(ANGLE!)	ERG! has a value of 0.5
3	Angle! = ASIN(ERG!)	ANGLE! has a value of 30
4	Angle! = ATAN(1/1)	ANGLE has a value of 45 degrees
5	ANGLE! = ATAN(1/0)	Not defined due to division by zero
6	ANGLE! = ATAN(0/-1)	ANGLE! has a value of 0 degrees
7	Angle! = ATAN(-1/-1)	ANGLE has a value of 45 degrees

ATAN2

Forms the arc tangent in the correct quadrant, i.e., the result is within a range of ± 180 degrees. To achieve this, ATAN2 must be called with two arguments. The single arc tangent (ATAN) does not provide for this possibility. Its result is always within a range of ± 90 degrees.

There is the following relationship between ATAN and ATAN2:

$\alpha_1 = \text{atan}(\text{opposite leg}/\text{adjacent leg}) = \text{atan}(y/x)$

with $-90 < \alpha_1 < 90$

$\alpha_2 = \text{atan2}(\text{opposite leg}, \text{adjacent leg}) = \text{atan2}(y, x)$ with $-180 < \alpha_2 < 180$

Example:

-
ANGLE = ATAN2(1,1)	ANGLE has a value of 45 degrees
ANGLE = ATAN2(1, 0)	ANGLE has a value of 90 degrees
Angle = ATAN2(0, -1)	ANGLE has a value of 180 degrees
ANGLE = ATAN2(-1, -1)	ANGLE has a value of -135 degrees
-

3.17.3 Logic operations

General information

Logical operations are binary with logical variables and decimal with integer variables. As depicted in the diagram below, logic operations can be represented with the usual operating symbols, i.e. the "." and the "+" symbols (**not in CPL, however**). Here as well, "multiplication/division calculations take priority over addition/subtraction calculations", i.e. AND operations take priority over OR operations. Parentheses may be nesting to a depth of seven.

NOT, AND, OR, XOR

CPL provides four types of operation functions:

- NOT function **NOT**
- AND function **AND**
- OR function **OR**

- EXCLUSIVE OR function **XOR**

E1 1 O-A NOT link $E1 = \overline{A}$	E1 E2 & A AND link $E1 \cdot E2 = A$	E1 E2 >1 A OR link $E1 + E2 = A$	E1 E2 =1 A XOR link $E1 \cdot \overline{E2} + \overline{E1} \cdot E2 = A$
NOT	AND	OR	XOR
E1 E2 A 0 - L L - 0	0 0 L 0 L 0 0 0 L L 0 0	0 0 L L L 0 L L L L L L	0 0 L L L 0 L L L L L 0

Fig. 3-1: Logic operations

Logic operations can be used for bit masking.

Example: Is bit 0 set in @20?

```

...
20 IF (@20 AND 1) <> 0 THEN GOTO .SET
30 ELSE GOTO . UNSET ENDIF
...

```

3.17.4 Conversion between numerical systems

BCD

Converting the binary value in BCD format.

Syntax: `<BCD value>=BCD (<binary value>)`

Tab. 3-27: BCD syntax

Example: 1 BCD_VALUE = BCD(49) BCD_VALUE has a value of 73

BIN

Converting BCD-coded numbers in binary value.

Syntax: `<binary value>=BIN (<BCD value>)`

Tab. 3-28: BIN syntax

Example: 1 BIN_VALUE = BCD(49) BIN_VALUE has a value of 31

3.17.5 Relational operations" =, >=, >, <>, <=, <"

The following relational operators are permitted:

=	Equals
>=	Greater than or equal to
>	Greater than
<>	unequal to
<=	Less than or equal to
<	Less than

Relational operations are used to describe the **relation** ("fulfilled" or "not fulfilled") of a **condition** (e.g. for the commands REPEAT UNTIL, WHILE-DO-END, IF-THEN-ELSE-ENDIF).

3.18 More CPL basic elements

3.18.1 Constants

General information

If numerical values are declared for program execution and are to remain unchanged (constant), such values may be entered in the commands as a numerical expression.

Integer constant (INT)

Integers are written without decimal points.

Example: `NUMBER% = 4` Integer constant (CPL type INT)

Floating-point constant (DOUBLE)

Real numbers (decimal numbers or fractions) are identified by a decimal point (floating point). For floating point numbers smaller than 1, zero has to be programmed in front of the point, e.g. 0,134.

Example: `PI = 3.141593` Floating point constant (CPL type REAL)

Floating point constants are always double the precise, i.e. shown with 15 digits.

Character string constant

A character string constant is limited by quotation marks (").

Example: `EXAMPLE$ = "This is a character string"` STRING constant

3.18.2 Code characters

CPL uses the following code characters:

! ? , " [] < - / &
 @ % \$: () = > + *

The comma is normally used as a separator. It is used as a grammatical punctuation mark only within character strings. In decimal numbers, the period is used as a decimal point and, for jump targets, as a label. Within character strings, the period is interpreted as a grammatical punctuation mark.

3.19 Commands for NC block synchronization

3.19.1 Overview

NC block synchronization commands can be used to:

- synchronize NC programs with specific events.
- synchronize block preparation with program execution.
- limit Look Ahead to a certain number of blocks.
- Synchronize NC programs with each other in different channels.
- Synchronization functions are applied either at the time of block preparation or at the active point in time of block execution.

Synchronization functions of the block preparation:

-
• WAIT (without parameters):	It pauses the block preparation until all previous blocks have been processed.
• CPL function WAIT (,<idle time>):	The block preparation is paused for the time specified.
• CPL function WAIT(BITIF(...)):	The block preparation waits for a certain signal within the PLC NC bit interface.
• BlkNmb:	It limits the Look Ahead to the programmed number of blocks.

Synchronization functions at the active point in time:

-
• WAITA / WAITO:	It waits for a certain state on the PLC-NC bit interface.
• WPV / WPVE:	It waits for the value of a permanent CPL variable.
• SPV / SPVE:	It writes permanent CPL variables.
• ASTOPA / ASTOPO:	It stops movement until certain axis positions are reached.
• BSTOPA / BSTOPO:	It stops movement until certain axis positions in the basic workpiece coordinate system (BCS) are reached.
• WSTOPA / WSTOPO:	It stops movement until certain axis positions in the current workpiece coordinate system (WCS) are reached.
• OFFSTOPA / OFFSTOPO:	It cancels stop conditions in the control channel.

3.19.2 Synchronization functions of block preparation

WAIT (without parameters)

Description: The "WAIT" function **stops the block preparation** until all the blocks programmed before the WAIT block have been completely processed. This is absolutely required if current machine data or process-based data is to be subsequently accessed within the program.

Chronologically, block preparation – a phase in which the individual program lines are analyzed and interpreted – is always carried out prior to execution on the machine. The time period elapsing between preparation and execution is not constant, but depends on many parameters (feed, distance traveled, Look Ahead, etc.).

Therefore, if the program is to respond to a machine- or process-related actual status (e.g. current actual position, signal at the bit interface, etc.), the WAIT instruction must be used to ensure that the previously mentioned time period is equal to "0" exactly at the moment of evaluation.

Example: **WAIT (without parameters)**

```

N10 X0
N100 (MSG, still running)
N20 X150

```

```

30 WAIT                               Block processing paused
40 XPOS = MCS(1)-150
50 IF XPOS < 0.0001 THEN               "Position reached" is output at X=150
    (MSG, position reached)
70 ENDIF

```



The "WAIT" function (without) parameter can be programmed in standard NC blocks (DIN) and in CPL blocks. In a CPL block that includes a WAIT instruction, a ":" must not be programmed. Subsequent CPL commands have to be written to a new CPL block.

CPL function: WAIT(<idle time>)

Description: It stops the block preparation until the programmed waiting time has elapsed.

Syntax:

WAIT(,<idle time>[,<ResVar>])	
with	
<Idle time>	Waiting period in ms without decimal places. The waiting period may also be programmed as an integer arithmetic expression.
<Event var>	optional integer variable After the waiting period has elapsed, 1 is assigned to <ResVar>.

Tab. 3-29: WAIT syntax

Example: WAIT with waiting period

```

10      WAIT(, Block processing is paused for 1000 ms. Subsequently, the integer
1000,E%) value of "1" is assigned to the E% variable.

```

```

10 WAIT(TIME How long block processing is stopped depends on the content of
%)           the integer variable TIME%. No value is returned.

```

Special features and restrictions: The function is skipped during pre-run.

CPL function: WAIT(BITIF(...))

Description: Stops block preparation until a certain status occurs on the PLC NC bit interface.

A time span can optionally be programmed. In this case, the block preparation is paused until the state occurred at the bit interface or until the time span elapsed.

Syntax:

WAIT(<BITIF condition>[, [<timeout>] [, <ResVar>]]) with	
<BITIF condition>	Specifies the condition that is to be checked on the PLC NC bit interface. This requires the following syntax: [NOT () BITIF (<Parameter>)] [= <State>]
	<Parameter> Transfer parameter of the BITIF function (for a description see BITIF function, chapter 7 "CPL commands" on page 417).
	<State> Boolean expression that is compared to the result of the BITIF function. If <State> is not programmed, the comparison is made with TRUE.
	If the condition is fulfilled, the block preparation continues.
<Timeout>	Waiting period in ms without decimal places. The block preparation waits until the BITIF condition has been fulfilled or the Timeout time has elapsed.
<Event var>	Optional integer variable: Returns the time when the wait event occurred: 0: Waiting condition was already met upon the call 1: Waiting time elapsed but the waiting condition was not met 2: Waiting condition was met within the waiting time

Tab. 3-30: WAIT syntax

Example: WAIT(BITIF(...))

10 WAIT (BITIF (1, 1, 1) = TRUE)	Wait until the second axis-related input signal of the first axis is set..
11 WAIT (BITIF (2, 0, 2) = (E1? OR E2?))	Wait until the 3rd channel-based input signal of the 2nd channel has the value of the logical expression (E1? OR E2?).
12 WAIT (NOT BITIF (3, 2, 1) , , C%)	Wait until the fourth spindle-based input signal of the first spindle is FALSE.
20 IF C%=0 THEN 21 PRN# (0, "BED. OK") 22 ENDIF	The variable C% provides a value of either "0" if the condition had already been fulfilled when WAIT was called.
30 WAIT (BITIF (4, 4, 1) = E7? , 250, ERG%)	Wait until the fifth axis-based input signal of the first axis has been set. Wait until the axis assumes the value of the variable E7? or until 250 ms have elapsed.

```

40 IF ERG%=0 THEN
41 PRN#(0,"Not waited")
42 ENDIF

50 IF ERG%=1 THEN
51 PRN#(0,">250ms waited")
52 ENDIF

```

The variable ERG% returns the value "0" if the condition had already been fulfilled when WAIT was called.

The variable ERG% returns the value "1" if the condition had not been fulfilled during waiting.

Special features and restrictions: The function is skipped during pre-run.

BlkNmb (BNB)

Description: Using the standard NC function "BlkNmb", the maximum number of blocks in the block preparation function can be limited.

For example, the "BlkNmb" function can be used to control further processing of measuring results in the part program determined at runtime.

Syntax:

BlkNmb<Num>	Restriction ON
BlkNmb(0)	The max. permitted number of blocks depends on the parameter /NCO/LookAh/Ch/NofBlkPrep (MP 7060 00110).
Short form: BNB . . . with	
<Disp>	Max. number of desired blocks Integer Input range: Greater than or equal to 0. 0: The number of blocks depends on MP 7060 00110.

Tab. 3-31: BlkNmb (BNB) syntax

- Special features and restrictions:**
- If more than <No> blocks have been prepared when the function is called, the block preparation pauses until the number of prepared blocks <No> drops below .
 - If more blocks are entered in <No> than the number of blocks planned in the control configuration, the effective number of blocks is automatically reduced to the setting in MP 7060 00110.

3.19.3 Synchronization functions at the block execution time

General information

One program per channel can run in the CNC system MTX. If the individual processing segments are divided into different individual programs and if these programs run on different channels, the processing sequence of all individual programs can be controlled by the synchronization functions at the block execution time.

The synchronization function at the block execution time with bit events (REV, SEV, WEV and WREV) are programmed with **DIN parameter lists**.



Bit events have system-wide valid levels (0 or 1) which can be modified or used with the NC functions REV (ResetEvent), SEV (SetEvent), WEV (WaitForEvent) and WREV (WaitForAndResetEvent).

All other synchronization functions at the block execution time have **CPL parameter lists** (similar to subroutines). In these, lists of CPL expressions are programmed that must be enclosed in square brackets: "[" and "]".

In general, the following applies:

- System-wide valid bit events are used in REV, SEV, WEV and WREV. The engineer has to ensure that there are no interactions.
- The permanent CPL variables used in the functions "WPV", "WPVE", "SPV", "SPVE" are valid throughout the system. Therefore, the engineer has to ensure that incorrect usage does not cause unintended interactions.
- In the framework of the NC functions offered, only the following simple types of permanent CPL variables are permitted:
 - INT
 - BOOL
 - REAL
 - DOUBLE

For arrays, only individual elements may be addressed!



Functions WEV, WREV, WAITA, WAITO, WPV, WPVE implicitly cause a falling edge at the end of the block. Incorrectly set synchronization points can cause damages at the machine.

Check the program sequence for possible synchronization problems prior to the actual processing.

Waiting for a bit event: WEV

Description: With the "WEV" function, it is waited at runtime (or the continuous block switching impedes) until a certain bit event occurs.

Syntax:

WEV (<EventNo> {, EventNo ...})	
With:	
<EventNo>	Number of an event bit. Valid 1 - 96. Any number of <EventNr> can be programmed in one WEV.

Tab. 3-32: WEV syntax

The events within the WEV are disjunctive (or) linked. Several WEVs can be programmed in one block. All WEVs programmed separately are conjunctive (and) linked. Several programming of one event does not generate an error.

If an event is already pending, the WEV is simply skipped. If the user waits (reacting on edge), the event has to be deleted before with REV. That can be carried out in the same block.

Example:

N10 X10 WEV(10,12)	It is first traveled to X10 and then, the program waits till event 10 or event 12 occurs. If either event 10 or event 12 already occur while traveling to X10 or before, it is not waited at the end of the block. (A down ramp is still generated).
N10 WEV(10,12)	The block is completed when the events (10 or 12)
WEV(16, 18) WEV(20)	and (16 or 18) and 20 are present.

WEV/WREV/SEV/REV

With the "WEV", "SEV" and "REV" functions, for example, two channels can be synchronized at runtime with the help of bit events

If several functions are programmed in one block, they are applied in the following sequence:

- 1. REV
- 2. SEV
- 3. WEV/WREV

With the block N10 REV(10) WEV(10) can thus be waited for the edge of event 10.

Channel 1 runs the following sequence:

Channel 1 Program:

```
;Channel 1
N10 REV(20, 21)      Deletes events 20 and 21
N20 G1 F1000 X100 Y100 Approaches position 1
N30 WEV(20)
;Waits till event 20 is present. It is synchronized at runtime with channel 2 N30..
N30 X200 Y200        Approaches position 2
N40 SEV(21)          Triggers event 21, channel 2 may continue
```

Channel 2 runs the following sequence:

Channel 2 Program:

```
;Channel 2
N20 G1 F1000 Z100 W100 Approaches position 1
N30 SEV(20)            Triggers event 21, channel 1 may continue
N30 Z200 W200          Approaches position 2
N40 WEV(21)
;Waits till channel 1 triggers event 21. It is synchronized at runtime with channel 1 N40..
```

Waiting for bit event deletion: WREV

Description: With the "WREV" function, it is waited at runtime (or the continuous block switching impedes) until a certain bit event occurs. If this condition is met, all bit events that have been waited for, are deleted.

Syntax:

WREV (<EventNo> {, EventNo ...})	
With:	
<EventNo>	Number of an event bit. Valid 1 - 96. Any number of <EventNr> can be programmed in one WREV.

Tab. 3-33: WREV syntax

In relation to the waiting condition, the WREV acts analogously to WEV (see [chapter "Waiting for a bit event: WEV" on page 83](#)). All bit events programmed in the waiting condition are deleted when the condition is met. The waiting condition can be combined from WEV and WREV.



While waiting for the condition to be fulfilled and the deletion of the bit event, some time passes during which some channel might execute blocks.

Example: A N10 WREV(11) is programmed in channel 1. The blocks N20 SEV(11) and then N20 WEV(11) come in channel to. If SEV was executed, the WREV waiting condition is fulfilled and continued in channel 1. Some time passes before bit event 11 is deleted. If WEV was executed before in channel 2, channel 2 does not wait (since bit event 11 is still active). If deletion is executed first, channel 2 waits.

In those cases, it is recommended that a synchronization is programmed over two different bit events.

Example:

N10 X10 WREV(10,12)	It is first traveled to X10 and then, the program waits till event 10 or event 12 occurs. If either event 10 or event 12 already occur while traveling to X10 or before, it is not waited at the end of the block. (A down ramp is still generated). Then, the bit events 10 and 12 are deleted.
N10 WREV(10,12) WEV(16, 18) WREV(20)	The block is completed when the events (10 or 12) and (16 or 18) and 20 are present. Then, the bit events 10, 12 and 20 are deleted.

Triggering a bit event: SEV

Description: The "SEV" function triggers (sets) a bit event at runtime. Thus, the level of the programmed event is set to 1 (irrespective whether the value was 0 or 1). Several SEVs can be programmed in one block.

Syntax:

SEV (<EventNo> {, <EventNo> ...})	
With:	
<EventNo>	Number of an event bit Valid 1 - 96 Any number of <EventNr> can be programmed in one SEV.

Tab. 3-34: SEV syntax

Example:

N10 SEV(13)	Event 13 is set.
N10 SEV(15, 16)	The events 15 and 16 are set.
N10 SEV(15) SEV(16)	The events 15 and 16 are set.

Deleting a bit event: REV

Description: The "REV" function deletes a bit event at runtime. Several REVs can be programmed in one block.

Syntax:

REV (<EventNr> {, <EventNr> ...})	
With:	
<EventNo>	Number of an event bit Valid 1 - 96 Any number of <EventNr> can be programmed in one REV. Special cases: REV(ALL) or REV() deletes all event bits.

Tab. 3-35: REV syntax

Example:	N10 REV(13)	Event 13 is deleted.
	N10 REV(15, 16)	The events 15 and 16 are deleted.
	N10 REV(15) REV(16)	The events 15 and 16 are deleted.
	N10 REV(ALL)	All events are deleted.

Waiting for states on the PLC NC bit interface: WAITA / WAITO

Description: The function "WAITA / WAITO" starts the waiting process at runtime until one or more of a maximum of 16 interface signals have reached a specified value.

Depending on the operation, the following can be programmed for several interface signals:

- **WAITA:**
"AND operation" of the individual signals
Wait until **all** interface signals assumed the specified value.
- **WAITO:**
"OR operation" of the individual signals
Wait until **at least one** interface signal assumed the specified value.

Syntax:

Wait for each of the specified signals: WAITA[BITIF(<Parameter>){=<State>}, BITIF(<Parameter>){=<State>},..., {<Timeout>}]	
Wait for one of the specified signals: WAITO[BITIF(<Parameter>){=<State>}, BITIF(<Parameter>){=<State>},..., {,<Timeout>}]	
With:	
<BITIF>	The BITIF function requests the PLC-NC Interface. Optionally, 2 to 16 interface signals can be queried simultaneously.
<Parameter>	Transfer parameter of the BITIF function (for a description, refer to BITIF function, chapter 7 "CPL commands" on page 417)
<State>	Boolean expression that is compared to the result of the BITIF function. If <State> is not programmed, the comparison is made with TRUE. If the condition is fulfilled, the block preparation continues.
<Timeout>	Optional time in ms; default = 0. If <Timeout> elapses before the respective condition has been fulfilled, a warning is generated and the system continues waiting. If timeout is not programmed or is equal to 0, no warning is generated.

Tab. 3-36: *BlkNmb (BNB) syntax*

Example: Program 1 in channel 1 processes the end face of a turned part.
Program 2 in channel 2 is to cut a slot into this end face and has to **wait** for program 1 to release the turned part for program 2. Program 2 is enabled by **setting** specific interface signals. When the interface signals have reached

the state, channel 1 transmits the release to channel 2. While program 2 is processed, program 1 waits for program 2 to continue its machining task.

N10 WAITO[BITIF(10,1,1)=FALSE,
BITIF(11,1,2)]

It waits actively until BITIF(10,1,1) assumes a value of 0 or BITIF(11,1,2) assumes a value of 1.

N10 WAITA[BITIF(10,1,1)=FALSE,
BITIF(11,1,2)]

It waits actively until BITIF(10,1,1) assumes a value of 0 and BITIF(11,1,2) assumes a value of 1.

The following applies to **WAITA, WAITO**:

If WAITA and WAITO are programmed in one NC block, the block execution is paused until both conditions are fulfilled. The WAITO condition is evaluated first.



The functions "WAITA", "WAITO", "WPV", "WPVE" implicitly cause a falling edge at the end of the block.

Incorrectly set synchronization points can cause damages at the machine.

Check the program sequence for possible synchronization problems prior to the actual processing.

Special features and restrictions: The function is skipped during pre-run.

Waiting for a permanent CPL variable value: WPV / WPVE

Description: The function "WPV / WPVE" starts the waiting process at runtime until a permanent CPL variable has reached a specified comparison value.

The comparison value can be determined at different times:

- **WPV:**

The comparison value is a CPL expression which is calculated during runtime exactly at the point in time at which the NC block is activated. A change in the comparison value after the NC block activation has no effect. The comparison value is compared with the value of the permanent variables.

Due to the evaluation at runtime, only a simple CPL expression is permitted.

- **WPVE:**

The comparison value is a CPL expression which was determined at preparation time, but not compared with the value of the permanent variable until runtime.

Syntax:

The comparison value is determined at **runtime**:

WPV [<Perm. CPL variable><ComparisonOperator><Simple CPL Expression>{,<Timeout>}]

The comparison value is already determined during **preparation time**:

WPVE [<Perm. CPL Variable><ComparisonOperator><CPL expression>{,<Timeout>}]

With:

<Perm. CPL Variable>	Permanent variable, marked by "@" followed by a variable name. If these are the elements of an array, only a constant or a permanent integer variable (CPL type INT) is allowed as an index for WPV. With WPVE, the index can be a constant or any CPL integer variable.												
<Relational operator>	The following comparison operators are possible: <table> <tr> <td>=</td><td>The permanent CPL variable is equal to the value of the CPL expression. Recommendable only in case of integer or Boolean values.</td></tr> <tr> <td>< ></td><td>The permanent CPL variable is not equal to the value of the CPL expression. Recommendable only in case of integer or Boolean values.</td></tr> <tr> <td><</td><td>The permanent CPL variable value is less than the value of the CPL expression.</td></tr> <tr> <td>≤</td><td>The permanent CPL variable value is less than or equal to the value of the CPL expression.</td></tr> <tr> <td>></td><td>The permanent CPL variable value is greater than the value of the CPL expression.</td></tr> <tr> <td>≥</td><td>The permanent CPL variable value is greater than or equal to the value of the CPL expression.</td></tr> </table>	=	The permanent CPL variable is equal to the value of the CPL expression. Recommendable only in case of integer or Boolean values.	< >	The permanent CPL variable is not equal to the value of the CPL expression. Recommendable only in case of integer or Boolean values.	<	The permanent CPL variable value is less than the value of the CPL expression.	≤	The permanent CPL variable value is less than or equal to the value of the CPL expression.	>	The permanent CPL variable value is greater than the value of the CPL expression.	≥	The permanent CPL variable value is greater than or equal to the value of the CPL expression.
=	The permanent CPL variable is equal to the value of the CPL expression. Recommendable only in case of integer or Boolean values.												
< >	The permanent CPL variable is not equal to the value of the CPL expression. Recommendable only in case of integer or Boolean values.												
<	The permanent CPL variable value is less than the value of the CPL expression.												
≤	The permanent CPL variable value is less than or equal to the value of the CPL expression.												
>	The permanent CPL variable value is greater than the value of the CPL expression.												
≥	The permanent CPL variable value is greater than or equal to the value of the CPL expression.												
<simple CPL expression>	To avoid any influence of the movement generation, only simple CPL expressions may be evaluated at runtime. A simple CPL expression is a mathematical expression consisting of permanent CPL variables, constants and the mathematical operations allowed in CPL. <i>Permitted data types are:</i> • INT • BOOL • REAL • DOUBLE												
<CPL expression>	Any mathematical expression in the CPL programming language.												
<Timeout>	Optional time in ms; default = 0. If <Timeout> elapses before the respective condition has been fulfilled, a warning is generated and the system continues waiting. If timeout is not programmed or is equal to 0, no warning is generated.												

Tab. 3-37: WPV / WPVE syntax

Example:	N10 WPV[@9=10]	The program waits at the active point in time until the permanent variable @9 assumes a value of 10.
	N10 WPVE[@8=(5*#VAR2%)]	The expression "5 * #VAR2%" is evaluated at preparation time. The value thus determined is compared to the permanent variables @8 at run-time. As long as @8 does not correspond to the value determined, no new NC block is applied.

WPV/SPV

With the "WPV" and "SPV" functions, for example, two channels can be synchronized at runtime with the help of permanent CPL variables

Channel 1 runs the following sequence:

Program:

<pre>;Channel 1 N20 G1 F1000 X100 Y100 N30 WPV(@20=5) N30 X200 Y200 N40 SPV(@20=6)</pre>	<pre>travel to position 1 wait till @20=5, here synchronizing at runtime with channel 2 N30.. travel to position 2 set @20=6, now channel 2 can continue running</pre>

Channel 2 runs the following sequence:

Program:

<pre>;Channel 2 N20 G1 F1000 Z100 W100 N30 SPV(@20=5) N30 Z200 W200 N40 WPV(@20=6)</pre>	<pre>travel to position 1 set @20=5, now channel 1 can continue running travel to position 2 wait till @20=6, here synchronizing at runtime with channel 1 N40..</pre>

Special features and restrictions: The function is skipped during pre-run.

Writing a permanent CPL variable: SPV / SPVE

Description: Using the function "SPV / SPVE", a value is assigned to a permanent CPL variable by writing at runtime.

The value can be determined at different times:

- **SPV:**

The value to be assigned to the permanent variable is only defined at runtime.

Due to the evaluation at runtime, only a simple CPL expression is permitted.

- **SPVE:**

The value to be assigned of the permanent variables is determined at preparation time (CPL interpretation time) but not assigned to the permanent CPL variable before runtime.

Syntax:

The value to be assigned is determined at **runtime**:

SPV [<Perm. CPL Variable> =<Simple CPL Expression>]

The value to be assigned is determined at **preparation time**:

SPVE [<Perm. CPL Variable> =<CPL Expression>]

With:

<Perm. CPL Variable>	Permanent variable, marked by "@" followed by a variable name.
<simple CPL expression>	To avoid any influence of the movement generation, only simple CPL expressions may be evaluated at runtime. A simple CPL expression is a mathematical expression consisting of permanent CPL variables, constants and the mathematical operations allowed in CPL. <i>Permitted data types are:</i> • INT • BOOL • REAL • DOUBLE
<CPL expression>	Any mathematical expression in the CPL programming language.

Tab. 3-38: SPV / SPVE syntax

Example:	N10 SPV[@6=1]	A value of 1 is assigned to permanent variable "@6" at runtime.
	N10 SPV[@5=(7*(@PERMVAR1% + 5))]	The value of the expression (7*(@PERMVAR1% + 5)) is determined at runtime and then assigned to @5.
	N10 SPVE[@5=(7*#VAR1%)]	The value of the expression (7 *#VAR1%) is determined at preparation time and assigned to @5 at runtime.

Motion stop until axis position is reached: ASTOPA / ASTOPO

Description: Using the "ASTOPA / ASTOPO" function, it is possible to synchronize movements between channels. Depending on the **position** of one or more axes of the **axis coordinate system ACS** in a channel, the synchronous movement in another channel (for exceptions, special cases) is stopped and resumed.

Restrictions:

- The axes used for synchronization must belong to a channel other than the channel to be controlled. Otherwise a self-holding locking may occur.
- The channel to be controlled must be in "Automatic" or "MDI" mode.
- If AND and OR conditions have been specified for the channel at the same time, there is a channel stop if the respective condition is met for at least one of the two functions.

Special case: ASTOP stops the calling channel

- ASTOPA/ASTOPO can stop the channel in which it is programmed if the axis/axes to be monitored is/are not assigned to this channel. The axis/axes has/have to be programmed using the system axis name(s) or number(s)

One or several conditions to stop the channel can be specified for each channel to be controlled:

- **ASTOPA:**

As long as **all conditions** are fulfilled, the synchronous motion of the channel to be controlled is stopped (logical AND operation).

- **ASTOPO:**

As long as **at least one condition** is fulfilled, the synchronous motion of the channel to be controlled is stopped (logical OR operation).



If new AND and OR conditions are specified, all previous conditions in this channel become invalid.

Syntax:

AND conditions:	
ASTOPA [<ChannelNumber> , <Cond.1> { , <Cond.2> } { . . { , <Cond.8> } . }]	
OR conditions:	
ASTOPO [<ChannelNumber> , <Cond.1> { , <Cond.2> } { . . { , <Cond.8> } . }]	
With:	
<Channel number>	Number of the channel to be controlled (1..n). Integer value or integer variable.
<Bed.1, Bed.2...Bed.8>	Specification of 1 to 8 conditions as follows: <Axis><Comparison operator><Comparison value> With:
<Axis>	System or channel axis name programmed as CPL string constant or as CPL string variable.
<Relational operator>	Operators allowed: < , ≤ , > , .
<Comparison value>	Real value or real CPL expression. The value is determined at preparation time and remains modally effective.

Tab. 3-39: Syntax



A maximum of 4 other channels can be stopped from one channel by AND/OR conditions.

Example:

Using axis names and numbers:

10 AXISNO% = 2	<i>Definition:</i>
20 AXISNAME\$ = "X"	• Axis number
30 STOPCHAN% = 2	• Axis name
:	• Channel number
N40 ASTOPO[STOPCHAN%, AXISNO%<10]	
:	
N90 ASTOPO[STOPCHAN%, "Z">20.3]	
:	
N150 ASTOPO[STOPCHAN%, AXISNAME\$<1.5]	

Motion stop until basic workpiece position is reached: BSTOPA / BSTOPO

Description: Using the "BSTOPA / BSTOPO" function, it is possible to synchronize movements between channels. Depending on the **position** of one or more coordi-

nates of the **basic workpiece coordinate system BCS** in one channel, the synchronous movement on another channel is stopped and continued.

Restrictions:

- The coordinates used for synchronization must belong to a channel other than the channel to be controlled. Otherwise a self-holding lock may occur.
- The channel to be controlled must be in "Automatic" or "MDI" mode.
- If AND and OR conditions have been specified for the channel at the same time, there is a channel stop if the respective condition is met for at least one of the two functions.

One or several conditions to stop the channel can be specified for each channel to be controlled:

- **BSTOPA:**

As long as **all conditions** are fulfilled, the synchronous motion of the channel to be controlled is stopped (logical AND operation).

- **BSTOPO:**

As long as **at least one condition** is fulfilled, the synchronous motion of the channel to be controlled is stopped (logical OR operation).



If new AND and OR conditions are specified, all previous conditions in this channel become invalid.

Syntax:

AND conditions: BSTOPA[<ChannelNumber> , <Cond.1> { , <Cond.2> } { . . { , <Cond.8> } . }]		
OR conditions: BSTOPO[<ChannelNumber> , <Cond.1> { , <Cond.2> } { . . { , <Cond.8> } . }]		
With:		
<Channel number>	Number of the channel to be controlled (1..n) Integer value or variable	
<Bed.1, Bed.2...Bed.8>	Specification of 1 to 8 conditions as follows: <Coordinate><Comparison operator><Comparison value> With:	
	<Coordinate>	Related to BCS: Coordinate name or coordinate index programmed as CPL string constant or as CPL string variable.
	<Relational operator>	Operators allowed: < , ≤ , > , .
	<Comparison value>	Real value or real CPL expression. The value is determined at preparation time and remains modally effective.

Tab. 3-40: BSTOPA / BSTOPO syntax



A maximum of 4 other channels can be stopped from one channel by AND/OR conditions.

Example: Activate an AND condition for basic workpiece coordinates:

N10 BSTOPA[3,"z"<12.0,"x">15]

Channel 3 is stopped as long as the following is applicable in the controlling channel:

- Position of the basic workpiece coordinate z < 12mm AND
- Position of the basic workpiece coordinate x > 15mm

Motion stop until workpiece position is reached: WSTOPA / WSTOPO

Description: Using the "WSTOPA / WSTOPO" function, it is possible to synchronize movements between channels. Depending on the **position** of one or more coordinates of the **workpiece coordinate system WCS** in a channel, the synchronous movement in another channel is stopped and continued.

Restrictions:

- The coordinates used for synchronization must belong to a channel other than the channel to be controlled. Otherwise a self-holding lock may occur.
- The channel to be controlled must be in "Automatic" or "MDI" mode.
- If AND and OR conditions have been specified for the channel at the same time, there is a channel stop if the respective condition is met for at least one of the two functions.

One or several conditions to stop the channel can be specified for each channel to be controlled:

- **WSTOPA:**

As long as **all conditions** are fulfilled, the synchronous motion of the channel to be controlled is stopped (logical AND operation).

- **WSTOPO:**

As long as **at least one condition** is fulfilled, the synchronous motion of the channel to be controlled is stopped (logical OR operation).



If new AND and OR conditions are specified, all previous conditions in this channel become invalid.

Syntax:

AND conditions:

WSTOPA [<channel number>, <cond.1>{ , <cond.2> } { . . { , <cond.8> } . }]

OR conditions:

WSTOPO [<channel number>, <cond.1>{ , <cond.2> } { . . { , <cond.8> } . }]

With:

<Channel number>	Number of the channel to be controlled (1..n)
	Integer value or variable

<Bed.1, Bed.2...Bed.8>	Specification of 1 to 8 conditions as follows: <Coordinate><Comparison operator><Comparison value>	
	With:	
	<Coordinate>	Related to WCS: Coordinate name or coordinate index programmed as CPL string constant or as CPL string variable.
	<Relational operator>	Operators allowed: < , ≤ , > , .
	<Comparison value>	Real value or real CPL expression The value is determined at preparation time and remains modally effective.

Tab. 3-41: WSTOPA / WSTOPO syntax



A maximum of 4 other channels can be stopped from one channel by AND/OR conditions.

Example: Activate an AND condition for workpiece coordinates:

```
N10 WSTOPA[3,"Z"<12.0,"X">15]
```

Channel 3 is stopped as long as the following is applicable in the controlling channel:

The position of workpiece coordinate Z of the channel is below 12 mm AND the position of workpiece coordinate X is greater than 15 mm

Canceling stop conditions: OFFSTOPA / OFFSTOPO

Description: It clears the stop conditions for a control channel.

Syntax:

OFFSTOPA	clears all AND stop conditions
OFFSTOPO	clears all OR stop conditions

Tab. 3-42: OFFSTOPA / OFFSTOPO syntax

For the synchronization functions, the following applies:

- ASTOPO, BSTOPO, WSTOPO, OFFSTOPO are modally effective and cancel each other.
- ASTOPA, BSTOPA, WSTOPA, OFFSTOPA are modally effective and cancel each other.

Canceling the stop conditions by the end of program (M30):

At the end of a part program (M30), at channel reset or at program deselection, the stop conditions activated in this channel are cancelled implicitly.

3.19.4 Writing system date at execution time

Description Using the function "SSD / SSDE", a value is assigned to a system data by writing at runtime.

The value can be determined at different times:

- **SSD:**

The value to be assigned for the system data is determined at runtime.

Due to the evaluation at runtime, only a simple CPL expression is permitted.

- **SSDE:**

The value to be assigned to the system data is determined at preparation time (CPL interpretation time) but not assigned to the permanent CPL variable before runtime. In addition to the simple data types, it is also possible to assign strings.

It is generally recommended to use SSDE. The SSD command should only be used if the content of the assigned CPL expression must not be calculated before execution time.



The target address of the system data is **always** calculated at preparation time. That means that if the system data is specified by means of array indices, these indices should not be changed from other channels

Example:

Indices in the System Data

```
N10 SSDE[SD.Field[@6].Int = 4]
```

If the variable @6 is not set in this channel, WAIT should be programmed in front of N10 (in addition to the channel synchronization).

Syntax:

The value to be assigned is determined at runtime :	
SSD[<SystemData> = <Simple CPL Expression>]	
The value to be assigned is determined at preparation time :	
SSDE[<SystemData> = <Simple CPL Expression>]	
With:	
<System data>	System data, marked by "SD.", followed by the variable name.

<simple CPL expression>	To avoid any influence of the movement generation, only simple CPL expressions may be evaluated at runtime. A simple CPL expression is a mathematical expression consisting of permanent CPL variables, constants and the mathematical operations allowed in CPL. <i>Permitted data types are:</i> • INT • BOOL • REAL • DOUBLE
<CPL expression>	Any mathematical expression in the CPL programming language. <i>Permitted data types are:</i> • INT • BOOL • REAL • DOUBLE • CHAR arrays

Tab. 3-43: SSD/SSDE syntax

Example:

N10 SSDE[SD.Testvar=1]

A value of 1 is assigned to the system data "SD.Testvar" at runtime.

N10 SSDE[SD.Array[4].Pos=7.2*(SIN(ANG) +5)]

The value of the expression "7.2*(SIN(ANG%) +5)" is assigned to "SD.Array[4].Pos".

N10 SSDE[SD.Array[J%].Pos=4]

A value of 4 is assigned to "SD.Array[J%].Pos" at runtime.

N20 SPV[SD.MeasPos= 3.5*@MPOS! + 12.2]

The value of the expression "3.5*@MPOS! + 12.2" is determined at execution time and then assigned to "SD.MeasPos".

Special features and restrictions:

- Due to the access to the system data, the linking time of SSD and SSDE is longer than in comparable commands. It is thus recommended to use these commands thoughtfully in order not to prolong the linking time (also in case of "postlinking") unnecessarily.
- It is recommended to use SSDE. The SSD command should only be used if the content of the assigned CPL expression must not be calculated before execution time.

4 Overview on modal function groups

A number of MTX NC functions is configured in modal NC function groups. The different NC functions of a module group cancel each other. Therefore, only one function of a module group is applicable at a time.

Modal group	NC functions
Geometry	G0, G1, G2, G3, G5, G6, G33
Velocity control	G8, G9, CFD, LNU, LND, SNU, SND, S2U, S2D
Level	G16, G17, G18, G19, G20
Absolute/incremental programming	G90, G91
Feed programming	G93, G94, G95
Spindle speed programming	G96, G97
Inch/metric	G70, G71
Tool length correction	G47, G48
2.5D tool radius correction	G40, G41, G42
3D tool radius correction	G140, G141, G142
D-correction	D0, D
ED-correction	ED0, ED
2.5D tool radius correction	G40, G41, G42
Outer corners at 2.5D tool radius correction	G43, G44
Feed at 2.5D tool radius correction	G45, G46
Collision monitoring at 2.5D tool radius correction	CLN, Off**
Interpolation in the feed mode	G61, G62
Active in position window	IPS1, IPS2, IPS3
Zero point offset bank 1	G52, G53.1, G54, G55, G56, G57, G58, G59
Zero point offset bank 2	G52.2, G53.2, G54.2 G55.2, G56.2, G57.2, G58.2, G59.2
Zero point offset bank 3	G52.3, G53.3, G54.3 G55.3, G56.3, G57.3, G58.3, G59.3
Zero point offset bank 4	G52.4, G53.4, G54.4 G55.4, G56.4, G57.4, G58.4, G59.4
Zero point offset bank 5	G52.5, G53.5, G54.5 G55.5, G56.5, G57.5, G58.5, G59.5
Placement inclined plane bank 1	G151.1, G152.1, G153.1, G154.1 G155.1, G156.1, G157.1, G158.1, G159.1
Placement inclined plane bank 2	G151.2, G152.2, G153.2, G154.2 G155.2, G156.2, G157.2, G158.2, G159.2
Placement inclined plane bank 3	G151.3, G152.3, G153.3, G154.3 G155.3, G156.3, G157.3, G158.3, G159.3
Placement inclined plane bank 4	G151.4, G152.4, G153.4, G154.4 G155.4, G156.4, G157.4, G158.4, G159.4
Placement inclined plane bank 5	G151.5, G152.5, G153.5, G154.5 G155.5, G156.5, G157.5, G158.5, G159.5
Placement Workpiece position correction	BCR, Off**
Coordinate offset	TRS, Off*

Overview on modal function groups

Modal group	NC functions
Additive coordinate offset	ATR, Off**
Input tool "Contour offset"	SHT, Off**
Input help: Scaling	SCL, Off**
Input tool "Mirroring"	MIR, Off**
Input tool "Rotate"	ROT, Off**
Input tool "Pol"	PLS, Off**
Input tool "Programming coupling"	PLP, EPC, Off**
Roundings and chamfers	RND, RNE, CHF, CHL, Off**
Rounding with splines	SCO, Off**
Replace straight lines with splines, convert straight lines and circles in splines	LTS, LCTS, Off**
Override	OVD, OVE
Diameter programming	DIA, RAD
Correction assignment	G78, G79
Pole for polar coordinates	POP, Off**
Axis velocity	AVE, Off**
Axis acceleration	AAC, Off**
Path acceleration	PAC, Off**
Kv programming	KVP, Off**
Feed forward	FFW, Off**
Positioning type for endless axes	PMD, Off**
Program a reduction torque	RDT, Off**
Feed adaptation	FAD, Off**
Punching	PUN, NIB, Off**
Tangential tool guidance	TTL, Off**
Tangential tool orientation	TTO, Off*
Area monitoring	ARA, Off**
Splitting	SPLIT, Off**
Axis coupling	AXC, Off**
Selective coordinate coupling	SCC, SCCT, Off**
Compensation tables	GCT, Off**
Laser	LFP, Off**
Tangential transition	DTT, Off**
Accuracy programming	PRP, Off**
Modal subroutine	G80, G81, G82, G83, ...
Asynchronous G0-interpolation	CG0(ASY1) , CG0(ASY0)

Off**

Switch-off function by NC function programming with (0) or (), e.g. AAC (0)

Tab. 4-1:

Overview Modal NC function groups

5 NC functions with syntax according to DIN 66025 (incl. extensions)



Undesired traversing motion!

Many NC functions require the programming of axis and/or coordinate addresses. Generally, only the addresses of the respective channel axis, i.e. synchronous axes can be programmed apart from the coordinate addresses.

Programming the address of an asynchronous axis within a parameter list generally leads to a syntax error, while programming outside of a parameter list leads to an asynchronous traversing motion!

Exceptions:

- G74(Home) VA1:
The asynchronous axis VA travels to the reference point.
- GAX(VA):
The asynchronous axis VA is applied to the channel
- FsMove, FsTorque, FsReset:
The "Move to Fixed Stop" functionality also affects asynchronous axes.

5.1 Overview

The control provides multiple NC functions. In addition to the commands specified in DIN 66025, it includes important supplements in the area of G-codes and additional high-level language-similar syntax elements.

*The **NC functions** of the control include:*

- **1- and 2-digit G-codes:**

Mainly based on DIN 66025 providing appropriate supplements to its "G-code vocabulary" to

- extend function groups in DIN
(e.g. **G52**, G53, G54, ...), or
- introduce new function groups related to existing function groups.

One-digit G-codes can always be written as two-digit, i.e. with a leading zero (e.g. G0=G00, G1=G01, ...).

- **3-digit G-codes (exceptions):**

Are used if an existing function from the 2D area is applied in the same manner in the 3D area (e.g. G41, G42 → G141, G142).

- **G-codes with "." extension:**

They are for example used together with straight moves as well as zero offsets or coordinate transformations. These functions have up to five banks that can be addressed/programmed using the "." extension.

Examples:

G1.2 Straight point-to-point move in feed

G59.2 G59 Zero point offset of bank 2

- **M-functions with a defined NC function**

e.g. M0, M30

- **NC functions with high-level language syntax:**

(refer to [chapter 6 "NC functions with high-level language syntax" on page 195](#))



For a tabular overview of all NC functions, refer to [chapter 8 "Appendix" on page 597](#).

In addition, all NC functions are in the index.

Used notations: In this documentation, the following notations are used for the syntax of NC functions:

Font "Courier bold" or "Courier":

Character strings in this font have to be programmed as shown.

Example: G0 (POL)

Angle brackets < >

indicate a placeholder for an expression/parameter to be programmed. The placeholder is written in italics.

Example: <Axis1>

Curly brackets { }

indicate an optional expression/parameter.

Such syntax elements can, but do not have to be programmed.

Example: G0 { ({ POL, } {<Par1> }) }

Character "|"

separates possible parameters that cannot be used simultaneously (alternative parameters).

Example: G0 { ({ POL, } {NIPS|IPS1|IPS2|IPS3}) }

Measuring units: *Unless specified otherwise, the NC programming is based on the following measuring units:*

- Translatory positions:
[mm] with G71, [inch] with G70
- Rotary positions:
[Degrees] with rotary axes
- Translatory velocities:
[mm/min] with G71, [inch/min] with G70
- Rotary velocities:
[Degree/min]
- Translatory accelerations:
[m/s²] with G71, [1000 inch/s²] with G70
- Rotary accelerations:
[1000 degree/s²]

5.2 G-codes

5.2.1 Straight line interpolation in rapid traverse "G0"

Description Approaches a programmed position with interpolation on a straight line at maximum path velocity (= in rapid traverse).

The following applies:

- At least one axis travels at maximum velocity or acceleration. The velocity of the axes is controlled in such a manner that all axes reach the target point at the same time.
- If the function is active, the "rapid traverse active" channel interface (= interface) signal is put out.
- The function is modal and deletes G1, G2, G3, G5, G6, G12, G13, G33.
- The following distance (displacement between target and actual value) at the end of the block can be reduced to the size of certain target windows (= "Exact stop" function).

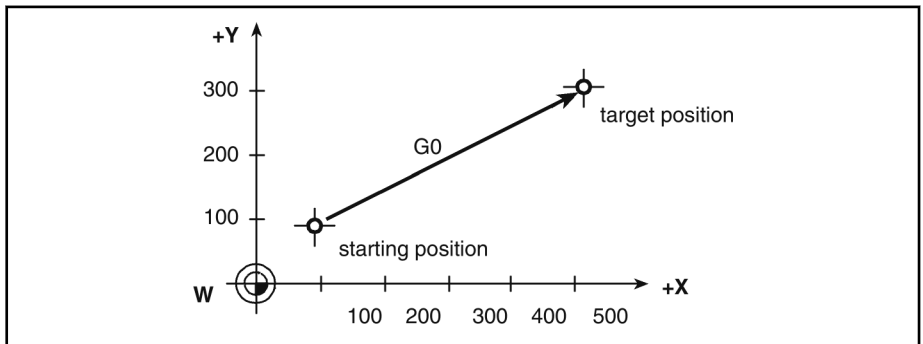


Fig. 5-1: Linear interpolation in rapid traverse

Syntax: Basic function

G0	Rapid traverse with exact stop ON and programming of the end position in Cartesian coordinates. The exact stop window that was last set for rapid traverse exact stop window size is taken into account (default = fine). At the end of the block, the control first reduces the path velocity to $v = 0$. Only when this exact stop window was reached for all axes involved, the next block is traversed.

Tab. 5-1: Syntax G0

Optional Parameters

G0 ({NIPS IPS1 IPS2 IPS3},{POL})	
with	
NIPS	Exact stop OFF Unless otherwise required, no deceleration to $v = 0$ at end of block.
IPS1	Same as G0 without parameter, however with exact stop window size, fine .
IPS2	Same as G0 without parameter, however, with exact stop window size, rough .

NC functions with syntax according to DIN 66025 (incl. extensions)

IPS3s	As G0 without parameter, but with exact stop window, infinite . That means that no positioning window is checked, but deceleration to $v = 0$ at end of block instead.
POL	Activates polar coordinate programming and sets polar angle 1 to 0 and polar angle 2 to 90 degrees. For programming in polar coordinates, refer to chapter 6.86 "Programming polar coordinates" on page 318 .

Tab. 5-2: *Optional parameters G0*

Example:

```

:
N40 X100 Y100          Initial position
N50 G0 X500 Y300       Approach target position in rapid traverse

```

Special features and restrictions:

- The function can be programmed with or without axis addresses in the same block.
- The corresponding velocity is determined using the machine parameters for the maximum axis velocity.
- The rapid traverse velocity can be limited to the reduced value set in the machine parameters using the channel interface signal "reduced rapid traverse" (qCh_RedRap).
- The rapid traverse velocity can also be limited using the function "test rapid traverse" controlled via the channel interface signal "test rapid traverse" (qCh_TestRap).
- The velocity can be modified either according to the machine parameter /CHAN/Ch[]/Path/RapTravOvrd/EnableRapTravIOvrd "Enable rapid traverse override" (7030 00014) or using the feed or the rapid traverse potentiometer.
- Changes in exact stop made with G0 (ON/OFF, size of exact stop window) apply only to rapid traverse.
- To change the size of the exact stop window in general, i.e. for rapid traverse **and** for all feed functions, use the NC functions IPS1, IPS2, IPS3.



For further information on the exact stop function, refer to G1, G61/62, IPS1, IPS2, IPS3 in this documentation as well as to the documentation "MTX Functional Description", chapter "Accuracy", section "Exact stop".

5.2.2 Linear interpolation in the feed "G1"

Description: Approaches a programmed position with interpolation on a straight line at effective feed (programmable via F-address).

The motion is coordinated in such a manner that all axes involved reach at the programmed end point simultaneously.

The following applies:

- The programmed feed value (F) acts as path feed. Thus, when moving multiple axes, each axis is thus smaller than F.
- The function is modal and deletes G0, G2, G3, G5, G6, G12, G13, G33.

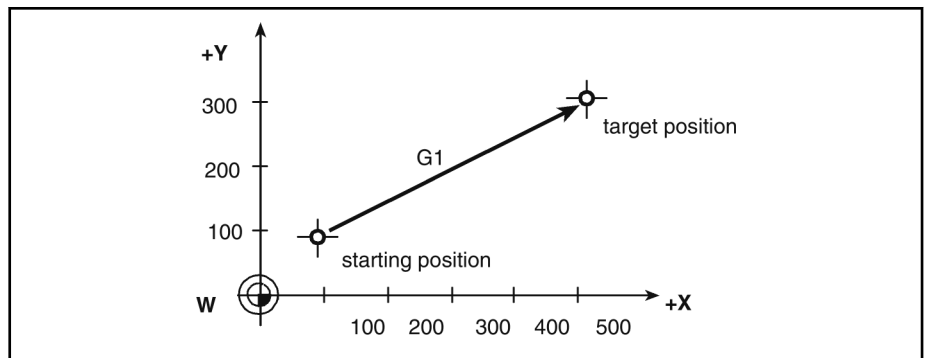


Fig. 5-2: Linear interpolation in the feed

Syntax: **Basic function**

G1	Motion according to active feed and programming the end position in Cartesian coordinates. The exact stop depends on whether G61 (exact stop ON) or G62 (exact stop OFF) is active. The exact stop window size last set for feed is taken into account (default = fine).

Tab. 5-3: Syntax G1

Optional parameters

G1 ({IPS IPS1 IPS2 IPS3},{POL}) with	
IPS	Exact stop ON irrespective of G61/G62. The exact stop window last set for feed mode is taken into account (default = fine). At the end of the block, the control first reduces the path velocity to $v = 0$. Only when this positioning window was reached for all axes involved, the next block is traversed. This activation of the exact stop applies to G1 only. For all other feed functions, G61/62 is relevant.
IPS1	As IPS, but with an exact stop window size, fine . The definition of the exact stop window size also applies for all other feed functions, except for rapid traverse (G0), while the exact stop function is only switched on for G1.
IPS2	As IPS, but an exact stop window size, rough . This definition of the exact stop window size also applies to all other feed functions, except for rapid traverse (G0), while the exact stop function is only switched on for G1.
IPS3s	As IPS, but with an exact stop window size, infinite . That means that no positioning window is checked, but deceleration to $v = 0$ at end of block instead. This definition of the exact stop window size also applies to all other feed functions, except for rapid traverse (G0), while the exact stop function is only switched on for G1.
POL	Activates polar coordinate programming and sets polar angle 1 to 0 and polar angle 2 to 90 degrees. For programming in polar coordinates, refer to chapter 6.86 "Programming polar coordinates" on page 318 .

Tab. 5-4: Optional parameters G1

Example: **Basic function:**

NC functions with syntax according to DIN 66025 (incl. extensions)

:

N40 X100 Y100 Initial position

N50 G1 X500 Y300 F100 Approach target position with a feed of F100.

Special features and restrictions:

- The function can be programmed with or without axis addresses in the same block.
- If no feed is yet active, a feed must be programmed in the same line using the F-address.
This feed remains effective until it is overwritten by a new feed value.
- The programmed path velocity can be limited using machine parameters.
- The path velocity can be modified via the feed potentiometer.
- The size of the exact stop window for all feed functions can also be changed using the NC functions G61(IPS1), G61(IPS2) and G61(IPS3).
- To change the size of the exact stop window in general, i.e. for rapid traverse **and** for all feed functions, use the NC functions IPS1, IPS2, IPS3.



For further information on the exact stop function, refer to G0, G61/62, IPS1, IPS2, IPS3 in this documentation as well as to the documentation "MTX Functional Description", chapter "Accuracy", section "Exact stop".

5.2.3 Circular/helical/helical-N interpolation "G2, G3"

General information

Description Approaches the programmed target position in the active working plane with interpolation on a circular path with the current feed (programmable via F-address).

- G2: circular path in counterclockwise rotation
- G3: circular path in clockwise rotation

Furthermore, additional axes outside the working plane can be included in the programming. Their motions are linearly interpolated.

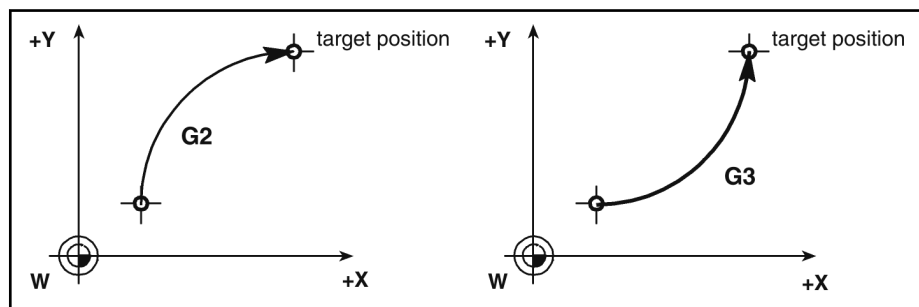


Fig. 5-3: G2 circular path in counterclockwise rotation; G3 circular path in clockwise rotation

Circular interpolation:

Involves only the axes of the active working plane.

Helical interpolation:

Involves **one more** synchronous axis in addition to the axes of the active working plane.

If this axis is positioned perpendicularly to the active working plane (e.g. the remaining axis of the workpiece coordinate system), a helical path with a constant slope is generated (helical interpolation).

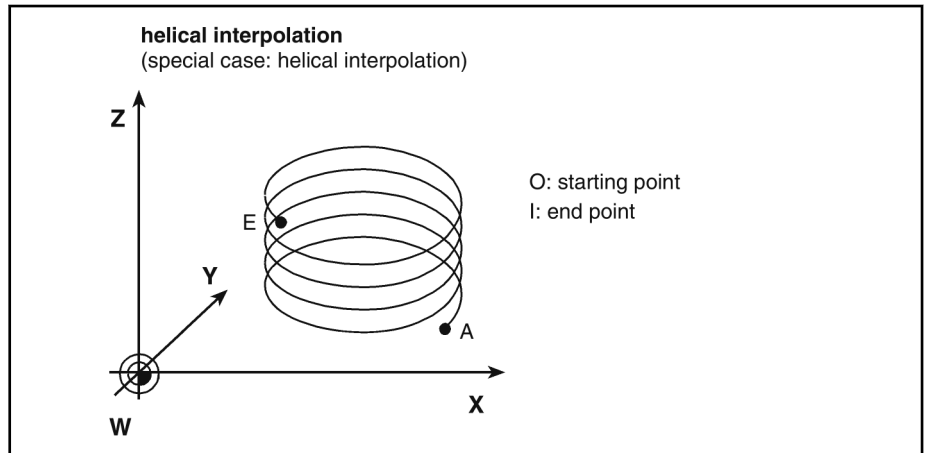


Fig. 5-4: Helical interpolation

Helical-N interpolation:

More than one additional synchronous axis (max. 6) to the axes of the active working plane is programmed.

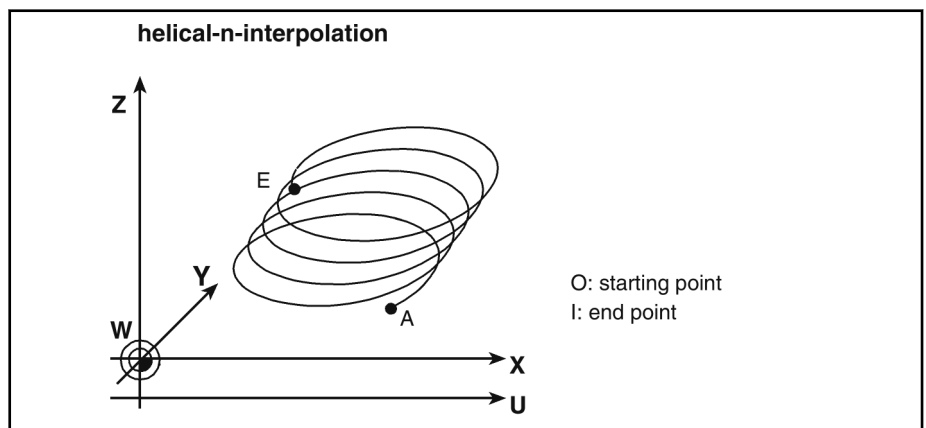


Fig. 5-5: Helical-N interpolation

The following applies:

- The axes traversing along the circular arc are clearly defined by the selected working plane (G17, G18, G19, G20).
- The programmed feed value (F) acts as path feed and normally refers to all axes to be traversed in one block. Thus, if multiple axes move, each axis is smaller than F.
- The real feed can be limited by the maximum possible radial acceleration.
- The G0, G1, G2, G3, G5, G6, G12, G13, G33 functions form a modal group and deselect themselves.
- *The circular path can be programmed using:*
 - radius programming or
 - center point programming.

Radius programming

Description: *Using the current position as starting point, specify a circular movement by programming the*

- end point of the circle and the
- radius of the circle.

The end point of the circle can be specified as an absolute or incremental position value.

Syntax: **Basic function:**

G2<EP> R<value>	Circular path in clockwise direction
G3<EP> R<value>	Circular path in counterclockwise rotation
with	
<EP>	Coordinates of the end point
<Value>	Radius of the circular path The radius has to be at least half the distance between the starting and the end point. The positive/negative sign is used to specify whether the path is to be traversed with the smaller or the larger arc if two circular paths are possible: Positive radius value: Arc ≤ 180 degree Negative radius value: Arc > 180 degrees If the radius is exactly half the distance between the starting and the end point, a semicircle results and any positive/negative sign can be used.

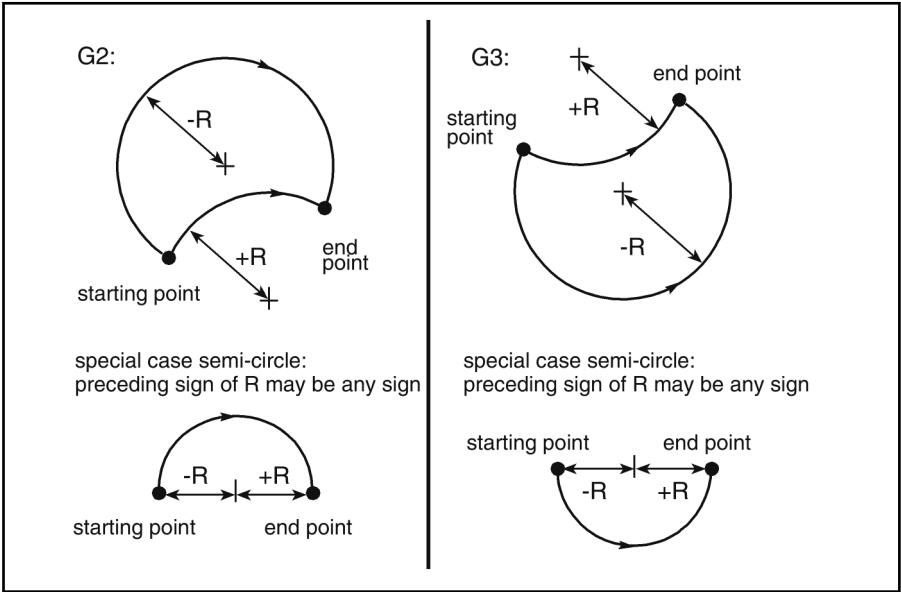


Fig. 5-6: Radius of the circular paths G2 and G3

Tab. 5-5: Syntax G2, radius programming

- Special features and restrictions:
- Full circles cannot be generated.
 - The control automatically corrects imprecise radius entries according to the tolerances set in the machine parameters.
Otherwise, a runtime error results.

Center point programming

Basic function

Description: *Using the current position as starting point, specify a circular movement by programming the*

- end point of the circle and the
- center point.

The center point of the circle can be programmed either incrementally or absolutely.

Syntax: **Basic function:**

G2<EP> <IP>	Circular path in clockwise direction
G3<EP> <IP>	Circular path in counterclockwise rotation
with	
<EP>	Coordinates of the end point If the starting and end points are identical within the circle plane, a full circle is generated automatically.
<IP>	Interpolation parameters I, J and K. For a local relative dimension programming, the distance between the circle starting point A and the circle center point M are defined for every axis. The sign results from the vector direction from A to M. Which interpolation parameter is assigned to which axis depends on the meaning of the coordinates in the machine parameters. The following applies by default: I = M(X) - A(X) for x-coordinate J = M(Y) - A(Y) for y-coordinate K = M(Z) - A(Z) for z-coordinate The center point of the circle can also be programmed in absolute coordinates using the local absolute dimension programming.

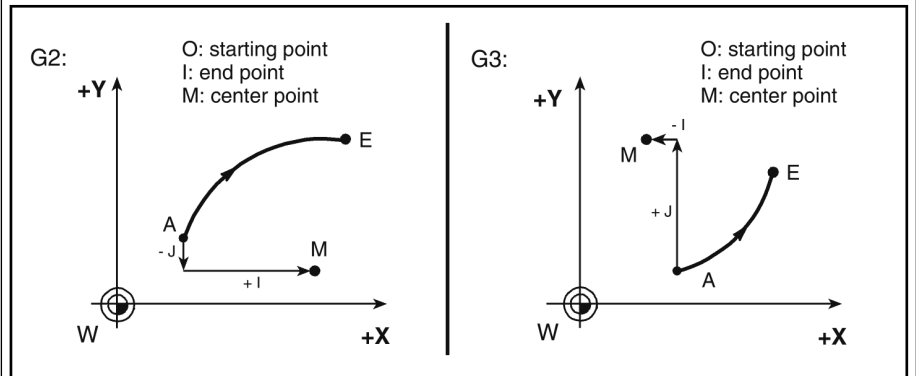


Fig. 5-7: Center point programming

Tab. 5-6: Syntax G2, center point programming

Optional parameters

Syntax:	G2/G3 (POL) <EP> R<Value>	Radius programming in polar coordinates
	G2/G3 (POL) <EP> <IP>	Center point programming in polar coordinates
	with	
	POL	Activates polar coordinate programming for the end point coordinates and sets polar angle 1 to 0 and polar angle 2 to 90 degrees. For programming in polar coordinates, refer to chapter 6.86 "Programming polar coordinates" on page 318 .

Tab. 5-7: Optional parameters G2, G3

Examples: Radius programming (Cartesian):

:

N40 G1 X10 Y10 F100

N50 G2 X38 Y20 R15

Approach initial point

Circular interpolation

Fig. 5-8: Radius programming (Cartesian)

Center point programming (Cartesian)

:

N80 G1 X100 Y100 F100

N90 G90 G17 G3 X350 Y250

I200 J-50

Approach initial position

Counter-clockwise circular interpolation in the x-/y-plane. Absolute dimension programming for end point coordinates is active.

Fig. 5-9: Center point programming (Cartesian)

Helix programming (spiral):

10 PITCH = 2	Store pitch value in a local variable.
20 TURN = 10	Store number of windings in a local variable.
N30 G0 G90 X-10 Y0 Z0	Move to start position.
N50 G2 I10 Z=IC([-PITCH])	Single winding with specified pitch.
RPT[TURN] F1000	

Special features and restrictions:

- If the starting and end points are identical, the control automatically generates a full circle.
- The control automatically corrects imprecise center point entries according to the tolerances set in the machine parameters.
To do this, it offsets the position of the center point accordingly.
- If interpolation parameters as well as a circle radius are programmed in the same block, only the circle radius is taken into account (= radius programming).
- If interpolation parameters that are not suitable for the selected plane are programmed, the control reports a runtime error.

Example: G17 G2 X5 I9 K7 (error: K does not match the x/y-plane)

5.2.4 Dwell time "G4"

Description: Stops the program execution. The subsequently programmed block is processed only after the programmed dwell time has elapsed.

The dwell time starts when the NC block before was completely processed.

Rotating spindles or traversing auxiliary axes are not stopped. Synchronous axes can reduce their lag, if necessary.

The dwell time can be programmed in:

- Seconds
- or -
- spindle revolutions.

To determine the spindle revolutions, the current actual revolutions are established cyclically from the main spindle, calculating the revolutions performed on that basis. At highly dynamic spindles, a certain deviation between the programmed spindle revolutions and the spindle revolutions actually been waited for can therefore occur in the acceleration or deceleration phases.

If the configured main spindle is an analog spindle (without speed feedback), the speed command value is used for the calculation instead of the actual speed value.

Syntax:

G4 (F<Value>)	Dwell time in seconds.
G4 (S<Value>)	Dwell time in spindle revolutions
with	
<Value>	Specification in seconds or spindle rotations If "0" is entered, the G4 block is discarded internally.

Tab. 5-8: Syntax G4

- Special features and restrictions:**
- The function has to be programmed in a separate block without path information. Only auxiliary and special functions are permitted in this block.
 - The programmed spindle revolutions refer to the main spindle configured in the machine parameters or using the "MainSp" function (refer to [chapter 6.73 "Main spindle switching MainSp, MSP" on page 294](#)).

5.2.5 Tangential circle exit"G12"

Description: Using the next programmed traversing motion, the control automatically calculates a tangential contour transition to the next traversing block for G12. Only a transition involving no reversal of direction is referred to as "tangential".



It is not possible to consecutively program several G12 blocks!

The "G0, G1, G2, G3, G5, G6, G12, G13" functions form a modal group and deselect themselves.

NOTICE

Machining marks are possible at the block transition in the case of helical/helical-N interpolation!

The calculated tangential transition refers only to the circle plane! The spatial tangent can jump at the block transition!

Syntax:

G12	<EP> [REV]
with	
<EP>	Coordinates of the circle end point
REV	Optional parameter: If programmed, a transition with reversal of direction of movement is calculated

Tab. 5-9: G12 syntax

- Special features and restrictions:**
- Programming G12 is not possible in the "manual data input" mode
 - A block with a traversing motion has to be programmed in the same program after G12.
 - The current plane must not be switched within the next block of G12
 - Following a G12 block, neither another G12 block nor a G5/G13 block must follow

5.2.6 Tangential circle entry"G5", "G13"

Description: Using the latest programmed traversing motion, the control automatically calculates a tangential circle entry for the G5 block. Only a transition involving no reversal of direction is referred to as "tangential".



The first entry tangent determines all following G5/G13 contour elements if several G5/G13 motions take place consecutively.

The "G0, G1, G2, G3, G5, G6, G12, G13, G33" functions form a modal group and deselect themselves.

NOTICE

Machining marks are possible at the block transition in the case of helical/helical-N interpolation!

The calculated tangential transition refers only to the circle plane! The spatial tangent can jump at the block transition!

Syntax:

G5 G13	<EP> [REV]
with	
<EP>	Coordinates of the circle end point
REV	Optional parameter: If programmed, a transition with reversal of direction of movement is calculated

Tab. 5-10: Syntax G5, G13

Special features and restrictions:

- Programming G5/G13 is not possible in the "Manual input" operating mode
- A block with a traversing motion has to be programmed in the same program ahead of G5/G13.
- The current plane must not be switched directly ahead of or during an active G5/G13.

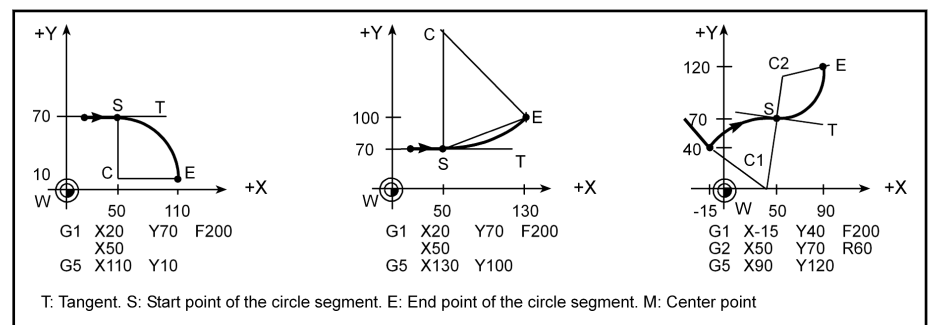
Examples:

Fig. 5-10: Examples of tangential circle entry

5.2.7 Spline programming "G6"

Overview - Spline types

Description: Compared to linear interpolation, the spline interpolation provides the same surface finish and contour precision despite its lower number of data points, since constant curves are calculated between the points.

The MTX CNC system supports the following spline types:

- **Spline type 0:**
Spline with coefficient programming
(polynomial coefficients of the CAD/CAM system)
- **Spline type 1:**
C¹-constant cubic splines with data point programming
(tangential transitions at the data points)
- **Spline type 2:**
C²-constant cubic splines with data point programming
(constant curvature transitions at the data points)

- **Spline type 3:**
B-spline with check point programming
(curve characteristics close to the data points)
- **Spline type 4:**
B-spline approximation (compressor)

Select and initialize the desired spline type with the function "SplineDef" (SPD; see [chapter 6.137 "Defining the spline type SplineDef, SPD" on page 385](#)). Then, spline programming can be activated using the function G6.



For further detailed information on the individual spline types, refer to the documentation "MTX Functional Description".

Syntax:	<table><tr><td>G6</td><td>Activating the "spline" path type.</td></tr></table>	G6	Activating the "spline" path type.
G6	Activating the "spline" path type.		

Tab. 5-11: Syntax G6

Depending on the required programming type, G6 can be programmed for the various spline types with different, modally effective parameters.

Special features and restrictions: The following functions **cannot** be programmed with splines:

- Tensor orientation
- 2D path correction G41/G42
- Punching/nibbling with path segmentation
- Chamfers and roundings
- Tangential tool guidance
- Accuracy programming
- G5 after a spline
- Canceling distance to go

Spline with coefficient programming (spline type 0)

Coordinate/axis programming

- Description:
- Each channel coordinate can be moved*
 - as spline by specifying the polynomial coefficients
 - linearly by specifying the end positions.

Syntax: Spline by specifying the polynomial coefficients:

<Coordname>(<c0>,<c1>,...,<cn>)	
	Programming individual coordinates with polynomial coefficients.

Tab. 5-12: Coordinate motion as spline

- or -

linearly by specifying the end positions:

<Coordname>(<Endpos>)	Programming the end position of individual coordinates/axes
with	
<Coordname>	Name of coordinate/axis

<c0>,<c1>,...,<cn>	Polynomial coefficient of a coordinate "n" corresponds to the spline degree defined in "SplineDef"
<Endpos>	End position of the coordinate

Tab. 5-13: "Linear" coordinate motion

Example: SplineDef(3)
G6 X(0.1,1.25,0.5,0.73) Y30 B(0.0,-1.0,0.1,-0.2)

Denominator polynomial programming

Syntax:	DN(<g _{01n} >)	Common denominator polynomial for all spline coordinates.
	with	
	<g _{01n} >	Polynomial coefficient of the denominator polynomial. "n" corresponds to the spline degree defined in "SplineDef".

Tab. 5-14: Denominator polynomial programming

Example: SplineDef(3)
G6 X(0.1,1.25,0.5,0.73) B(0.0,-1.0,0.1,-0.2) DN(1,0,1)

Orientation vector programming

Description: An active axis transformation with vector orientation ("Coord(..)") is required for this programming method.

Syntax:	O1(<o _{10111n} >)	X-component of the orientation vector
	O2(<o _{20212n} >)	Y-component of the orientation vector
	O3(<o _{30313n} >)	Z-component of the orientation vector
	with	
	<o _{10111n} >:	Spline coefficient of the x-component of the orientation vector.
	<o _{20212n} >:	Spline coefficient of the y-component of the orientation vector.
	<o _{30313n} >:	Spline coefficient of the z-component of the orientation vector. "n" corresponds always to the spline degree defined in "SplineDef".

Tab. 5-15: Orientation vector programming

Example: Program:

```

N00 ;Spline coefficients for vector orientation
001 PI=3.14159:PIH=PI/2:PIHQ=PIH*PIH:PIHC=PIHQ*PIH
N10 G1 F30000 X0 Y0 Z0 B90 C0
N20 SplineDef(3)
N30 Coord(1) ;5-axis transformation with vector orientation on
N40 G6 PL[PIH] ;PL see spline parameter length programming
N50 O1(1,0,-3/PIHQ,2/PIHC) O3(0,1,(3-PI)/PIHQ,(-2+PIH)/PIHC)
N60 O1(0,0,3/PIHQ,-2/PIHC) O3(1,0,(-3+PIH)/PIHQ,(2-PIH)/PIHC)
N70 O(0,1,0) ;normal vector orientation
N80 G1
N90 Coord(0)

```

Programming spline parameter lengths

Description: The spline parameter length is the length of the interval defined for "w", with "w" being active between 0 to w_e . The value w_e is modal and remains valid for all NC blocks until G6 is deselected.

PL has to be programmed in the first traversing block after G6: Otherwise, a runtime error occurs.

Syntax:	{ PL< w_e >}	Optional programming of the spline parameter length
	with	
	< w_e >	Any value > 0

Tab. 5-16: Programming spline parameter lengths

Example: G6 X(0.1,1.25,0.5,0.73) B(0.0,-1.0,0.1,-0.2) PL0.6
 $(X = 0.1 + 1.25 w + 0.5 w^2 + 0.73 w^3$ and
 $B = 0.0 - 1.0 w + 0.1 w^2 - 0.2 w^3$ with
w being active from 0...0.6)

Hermite spline (spline type 1)

Coordinate/axis programming

Description: Each channel coordinate can optionally be moved

- as spline by specifying the position and the first derivation
- or -
- linearly by specifying the end positions.

Syntax: Spline by specifying the position and the first derivation:

<Coordname>(<c0>,<c1>)	Programming individual coordinates with position ($\triangle c0$) and first derivation ($\triangle c1$).

Tab. 5-17: Coordinate motion as spline

- or -

linearly by specifying the end positions:

<Coordname>(<Endpos>)	Programming the end position of individual coordinates/axes.

Tab. 5-18: "Linear" coordinate motion

with	
<Coordname>	Name of coordinate/axis
<c0>,<c1>	Polynomial coefficient of a coordinate, with $c0 \triangleq$ position and $c1 \triangleq$ first derivation
<Endpos>	End position of the coordinate

Tab. 5-19: Coordinate motion

Example: SplineDef(1143)
G6 X(0.1,1.25) Y30 B(0.0,-1.0)

Programming spline parameter lengths

Description: The spline parameter length is the length of the interval defined for "w", with "w" being active between 0 to w_e . The value w_e is valid locally in the programmed block. As default value, $w_e=1$ is used.

In the starting point of the spline sequence (i.e. first traversing block after G6), it is assumed that the derivation equals zero.

Syntax:	{ PL< w_e >}	Optional programming of the spline parameter length
	with	
	< w_e >:	Any value > 0

Tab. 5-20: Programming spline parameter lengths

Example: G6 X(0.1, 1.25) B(0.0, -1.0) PL0.6
(w running from 0...0.6)

User-defined velocity profile

Description: Using the NC functions CVP and CPI, a user-defined velocity profile can be specified using the Hermite splines. The actual MTX velocity control is suspended. In case of a corresponding parameterization, the <c1> values of the coordinates correspond to the axis velocities in mm/s or inch/s, reached at the <c0> positions. Specify **F1** (scaling factor set to mm/s or inch/s) or **F60** (scaling factor set to mm/min or inch/min), depending on the machine parameter NCO/VelScaleFact/Ch[]/FactVel as F-value.

Syntax:	CVP CPI F60 SplineDef(1143)G6	Enabling the Hermite spline with user-defined velocity profile
	The numerical value programmed via the spline parameter length PL constitutes the execution duration of the block in seconds in this case.	

Tab. 5-21: User-defined velocity profile

Example: Program:

```

N00010 G0 X150 Y0 Z25
N00020 SplineDef(1143)
N00030 CPI CVP F60
N00040 X(150.0000,0.0000) Y(0.0000,0.0000) Z(24.9167,-24.9999) PL0.01
N00050 X(150.0000,0.0000) Y(0.0000,0.0000) Z(24.5003,-50.0001) PL0.01
N00060 X(150.0000,0.0000) Y(0.0000,0.0000) Z(0.5000,-50.0000) PL0.48
N00070 X(150.0000,0.0000) Y(0.0000,0.0000) Z(0.0833,-25.0001) PL0.01
N00080 X(150.0000,0.0000) Y(0.0000,0.0000) Z(0.0000,0.0001) PL0.01
N00090 X(150.0000,0.00002524) Y(0.0000,-0.00006279) Z(0.0180,9.399888) PL0.06
N00100 ...
...
N50080 X(250.0000,0.0000) Y(0.0000,0.0000) Z(70.0171,-0.00232267) PL 0.018257
N50090 G1 G8 CPI(0)

```

C1- and C2-constant cubic spline (spline type 1 and 2)

Coordinates programming

Description: Programming involves the end points of the channel coordinates.

All <Members> listed in "SplineDef" move along the spline curve. The remaining coordinates not defined in "SplineDef" move along a straight line.

Syntax:	<Coordname><Endpos>	Programming individual coordinates and their values.
	with	

NC functions with syntax according to DIN 66025 (incl. extensions)

<Coordname>	Name of the coordinate (orientation O included)
<Endpos>	End position of the coordinate

Tab. 5-22: End point programming of the channel coordinates

Example: Coordinates x, y, z and orientation coordinates phi, theta.

```
SplineDef(2203,x,y,z,phi,theta)
G6 x10 y20 phi20 theta30
x15 y22 O(1,1,2)
```

Example:

Axes X,Y,U

```
SplineDef(1213,X,Y)
G6 X10 Y10 U20 (X and Y move as spline, U moves linearly)
```

Spline parameter length

Description: The spline parameter length is calculated by the NC using the specified data points. The process (parameter setting) defined in the spline ID is used for this purpose. The spline parameter length can also be programmed if necessary:

Syntax:	{ PL<w _e >}	Optional programming of the spline parameter length if the selection of <Parameterization> is to be overwritten (see chapter 6.137 "Defining the spline type SplineDef, SPD" on page 385 , parameter<Id>).
	with	
	<w _e >	Any value > 0

Tab. 5-23: Optional programming of the spline parameter length

Starting and end boundary conditions

Refer to [chapter "Starting and end boundary conditions" on page 118](#).

B-splines (NURBS) (spline type 3)

Coordinate programming

Description: The end points of the channel coordinates (checkpoints) are programmed. All <Members> listed in "SplineDef" move along the spline curve. The remaining coordinates not defined in "SplineDef" move along a straight line.

Syntax:	<Coordname><Endpos>	Programming individual checkpoints (coordinates) and their values.
	with	
	<Coordname>	Name of the coordinate (orientation O included)
	<Endpos>	End position of the coordinate

Tab. 5-24: Programming checkpoints (coordinates/axes)

Examples: Coordinates x, y, z and orientation coordinates

```
SplineDef(3103,x,y,z,O)
G6 x10 y20 z30 O(0.1,0,1.0)
```

Axes X,Y,U

```
SplineDef (3102, X, Y) G6 X10 Y10 (X and Y move as spline, U moves linear-
U20 ly)
```

Spline parameter length

Description: The spline parameter length is automatically calculated by the NC using the specified checkpoints. The process (parameter setting = 1, 2, or 3) defined in the spline ID is used for this purpose. The spline parameter length can also be programmed if necessary:

Syntax:	{ PL<w _e >}	Optional programming of the spline parameter length if the selection of <Parameterization> is to be overwritten (see chapter 6.137 "Defining the spline type SplineDef, SPD" on page 385 , parameter<Id>).
	with	
	<w _e >	Any value > 0

Tab. 5-25: Optional programming of the spline parameter length

Spline point weight for checkpoints with B-splines

Syntax:	{ PW<w>}	Optional programming of point weights. This can be used to modify the spline in the vicinity of a checkpoint.
	with	
	<w>	Default: 1 0 < w < 1: pushes the spline away from the checkpoint w > 1: pulls the spline to the checkpoint

Tab. 5-26: Optional programming of point weights.

Example: **Coordinates x, y, z and orientation coordinates**

```
SplineDef (3103, x, y, z, O)
G6 x10 y20 z30 O(0.1, 0, 1.0) PW2.3
```

B-spline approximation (spline type 4)**Coordinate programming**

Description: Programming involves the end points of the channel coordinates. Only <Members> can be programmed.

Syntax:	<Coordname><Endpos>	Point programming of coordinates
	with	
	<Coordname>	Name of the coordinate (orientation O included).
	<Endpos>	End position of the coordinate

Tab. 5-27: Point programming

Example: In the following NC program example, the respective <Q_i> stands for a programmed point in space including orientation vector "x.. y.. z.. O(..)"

N00 SplineDef(4213)	;Defines the late G06 as b-spline approximation
N01 BAP(E0.01, OE0.05)	;Dimensional tolerance 0.01 mm, Dimensional tolerance for orientation vector 0.05 mm
..	
N100 G0 <Q ₀ >	;Starting point of the b-spline curve
N101 G06 <Q ₁ >	
N102 <Q ₂ >	
N103 <Q ₃ >	
..	
N400 <Q ₃₀₀ >	;Last point of the b-spline curve
G1	

Tolerance value of the approximation

Refer to [chapter "B-spline approximation parameters"](#) on page 119.

Starting and end boundary conditions

Refer to [chapter "Starting and end boundary conditions"](#) on page 118.

Spline parameter length

The spline parameter length is automatically determined by the NC and cannot be programmed.

Starting and end boundary conditions

With all spline types except for type 0 (coefficient programming), starting and/or end boundary conditions can be specified in NC blocks within the spline sequence.

SBC(<BC Type>{,<Values>})	Boundary condition for the starting point of a spline block of the spline sequence
EBC(<BC Type>{,<Values>})	Boundary condition for the starting point of a spline block of the spline sequence
with	

<BC Type>	0 (for spline type 4) 1 (for spline type 1, 2, 4) Specification of tangent direction at the starting and/or end point of the spline sequence. For each spline member, a value has to be specified in the <Values> list (three values for the orientation vector). 10 (for spline type 1, 2, 3, 4) The starting and end tangent points in the direction of the starting-end point connecting line of the current spline block. 11 (for spline type 1, 2, 4) With SBC: The tangent at the starting point of the current spline block points to the direction of the starting-end-point connecting line of the previous block. 2 (for spline type 1, 2) Specification of the second derivative after the spline parameter "w" at the starting or end point of the spline block. For each spline member, a value has to be specified in the <Values> list (three values for the orientation vector). 3 (only for spline type 2) The De Boor's boundary condition links the second derivatives to the first and/or last two interpolation points. <Values> is 1 in most cases. This can be programmed only at the beginning or end of the spline sequence. 4 (only for spline type 2) Periodic boundary condition: last and first point of the spline sequence are identical. SBC(4) absolutely requires EBC(4) and vice versa. The entire spline sequence has to be in the look-ahead range. Otherwise, a runtime error is created.
<Values>	Only in connection with BC type = 1, 2 and 3 Default values: 0,...,0 for BC-type = 1, 2 and 1,...,1 for BC-type = 3 No values with BC types 10, 11 and 4.

Tab. 5-28: Starting and end boundary conditions

Examples: SBC(1, 1.0, 1.0, 0.2) with SplineDef(1213,X,Y,B)

SBC(1, 1.0, 1.0, 0.2, 0.0, 0.5, 0.5) with Spline-Def(2203,x,y,z,O)

EBC(2) with SplineDef(2203,x,y,z,O)

EBC(10) in SplineDef(3203,x,y,z,O)

B-spline approximation parameters

Syntax:

BsaPar(E...,OE..)	
Short form: BAP(..)	
E (Error)	If E is missing, the value in the MP 80070030 applies Specifies the dimensional tolerance in mm or inch

OE (OrientationError)	Dimensional tolerance of the orientation vector OE is only important in case of an active vector orientation If OE is missing, the switch-on value OE=E applies

Tab. 5-29: Syntax of the B-spline approximation parameter

5.2.8 Path slope ON "G8", Path slope OFF "G9"

- Description:**
- Without the "path slope", the control performs a complete up and down slope (velocity ramp) to $v = 0$ at the beginning and end of a traversing block.

Although this reduces the contour deviation at the block transition, it requires a longer processing time.

- Using "path slope", the control attempts to generate a velocity as constant as possible being as high as the programmed feed (even at the block transition). This reduces the processing time.

The fact that the contour can be "finished" at the corners can also be an advantage for certain processing procedures (more uniform surface).

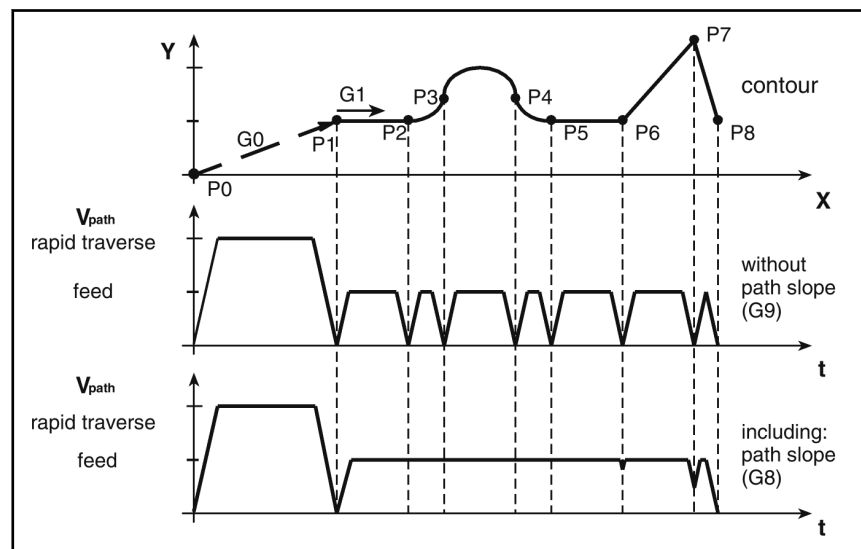


Fig. 5-11: Path slope

When determining the optimum velocity profile, the control takes into account not only the value for the jumping ability of the axes (MP 1010 00012), but also the number of subsequent program blocks (block look-ahead; MP 7060 00110 - 7060 00130), since braking within the margin of the dynamics of the machine has to be ensured at all times.

To limit contour deviations at real edges, the maximum step change must not be set at a value that is too high. On the other hand, if the maximum step change is set too low, this results in an unwanted deceleration at minor knees in the contour (nearly continuous transitions).

Smoothing the path acceleration can be entered for G8 as default setting into the machine parameter 7050 00320 (also see [chapter 6.61 "Activating the jerk limitation JerkControl, JKC" on page 281](#)).

Syntax:

G8	Path slope ON
G9	Path slope OFF

Tab. 5-30: Syntax G8, G9

Special features and restrictions:

- If the exact stop is active, the control decelerates to $v = 0$ after each block despite of the active G8.
- Auxiliary functions can reduce the effect of G8 if their execution time (including acknowledgement) is not significantly lower than the interpolation time of a block. If necessary, the traversing distance of a block has to be extended or the feed has to be reduced.
- Path slope acts only on synchronous axes (machining axes).
- The functions G8 and G9 form a modal group with the ramp functions and thus deselect each other.

5.2.9 Jerk-limited velocity control "G8(SHAPE...)", "G9(SHAPE...)", "G9(ASHAPE...)", "G9(X... , Y... , ...)", "AsynchrShapeOrder, ASO"

General description



The SHAPE function was replaced by the axis-specific jerk limitation and is not to be used anymore if possible (cf. "MTX Functional Description", "Basics of the axis-specific jerk limitation"). We recommend to have the SHAPE calculated automatically internally in the control using the axis-specific jerk limitation (i.e. JerkControl(1)). In case of a deactivated jerk limitation (i.e. JerkControl(0)), the user can also program the SHAPE. The simultaneous use of the SHAPE-function and the axis jerk limitation is not possible.

For asynchronous axes, all SHAPE data is ignored if $AX/Dr[i]/Jerk/MaxAxJerk \neq 0$ was set.

Jerk-limited velocity control smoothes jumps during path acceleration over several interpolation cycles. The number of interpolation cycles can be programmed.

This results in smooth velocity transitions (jerk limitation).

The jerk-limited velocity control can be used in the following operating modes:

- "Path operation" (in case of an active G8)
- "positioning mode" (with G9 active), and
- "Asynchronous mode"

The jerk-limited velocity control is also known as SHAPE function.

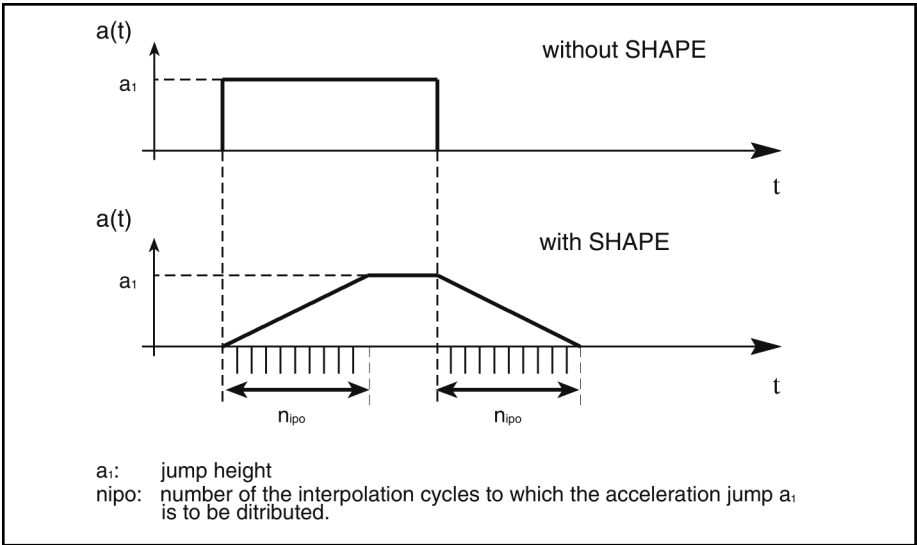


Fig. 5-12: Acceleration distribution with/without SHAPE

SHAPE for asynchronous axes

Syntax:	
AsynchShapeOrder <Value>	
Short form: ASO <Value>	
with	
<Value>	Number of interpolation cycles on which the control is to linearly distribute the acceleration jump.
	Value range: 2 ... 100 cycles; integer

Tab. 5-31: AsynchShapeOrder syntax

Examples: N10 W150 FA20000 ASO(100)

The axis jerk results from:

$$\frac{MP101000001}{100 * MP903000001 * 10^{-6}} \left[\frac{m}{s^3} \right]$$

If the information on the SHAPE order is missing in the block, the default value of parameter 1003 00008 "SHAPE order for axis" is used.

N10 W150 FA1000 AxAcc(W0.5)

The axis jerk results from:

$$\frac{0.5}{MP100300008 * MP903000001 * 10^{-6}} \left[\frac{m}{s^3} \right]$$

This value can never fall below parameter 1003 00007 "Minimum SHAPE order for axis jerk limitation". If no SHAPE order is programmed and the default and minimum SHAPE order parameters are not assigned with 0 or 1, the control moves the axis without jerk limitation. Information on the SHAPE order for asynchronous axis is effective on a block-to-block basis and refers to all programs asynchronous axes.

SHAPE for path mode - Programming

Syntax:

G8 { (SHAPE<Value>) }	
with	
<Value>	0 or 1: SHAPE for switching off path mode. 0 or 1: 2 to 100 (integer): Number of interpolation cycles on which the control is to linearly distribute the jump (SHAPE order). Without SHAPE parameter: Move with the default value for SHAPE stored in the machine parameter 7050 00320.

Tab. 5-32: G8(SHAPE) syntax



Changing the SHAPE order always leads to a block transition velocity of $v = 0$.

Example:

N30 G8 (SHAPE10)	Switch on SHAPE in contour mode and distribute the acceleration jumps on 10 IPO cycles
:	
:	
N50 G8 (SHAPE0)	Switch off SHAPE in path mode
:	
:	
N70 G8 (SHAPE1)	Switch off SHAPE in path mode
:	
:	
N90 G8	Move with the default value for SHAPE stored in the machine parameter 7050 00320

SHAPE for Positioning Mode - Programming

Syntax:

G9{(SHAPE<Value>)}	Switch on linear SHAPE for positioning mode.
with	
<Value>	Number of interpolation cycles on which the control is to linearly distribute the acceleration jump. Value range: 2 ... 100 cycles; integer
Without SHAPE parameter	Move with the default value for SHAPE stored in the machine parameter 7050 00340.
G9(SIN<Value>)	Switch on sine ² -shaped SHAPE for positioning mode
with	

NC functions with syntax according to DIN 66025 (incl. extensions)

<Value>	Number of interpolation cycles on which the control is to distribute the acceleration jump in a sine²-shaped way. The following values are valid: 5: 5 IPO cycles 10: 10 IPO cycles 15: 15 IPO cycles 20: 20 IPO cycles 40: 40 IPO cycles
G9 (ASHAPE { , TYPE <i> })	
	This assigns the corresponding axis-specific SHAPE order (number of IPO cycles) from MP 1003 00008 to every axis of the channel and makes a block-wise calculation of the resulting block-specific SHAPE order for the path.
G9 (<Par ₁ >{,<Par ₂ >,...} {,TYP<i>})	
	Program SHAPE order axis-by-axis. In case of non-programmed axes, the SHAPE order from MP 1003 00008 is used.
with	
<Par _x >	Logical axis name with the SHAPE order to be assigned to this axis (up to 100).
Example: N30 G9 (X4, Y6, Z10)	SHAPE order (x-axis) = 4 SHAPE order (y-axis) = 6 SHAPE order (z-axis) = 10
TYPE 0 or no TYPE	All axis SHAPE orders are evaluated with the ratio of the actual axis acceleration to the maximum axis acceleration. The highest SHAPE order calculated becomes the path SHAPE order (see below).
TYPE 1	The highest axis SHAPE order becomes the path SHAPE order.

Tab. 5-33: Syntax G9(SHAPE) and G9(ASHAPE)

Resulting SHAPE order

Description: The resulting path SHAPE order S_b is the maximum of the effective axis SHAPE orders S_i^{eff} of all axes in the interpolation:

$$S_b = \max \{S_1^{\text{eff}}, \dots, S_n^{\text{eff}}\}$$

For G9(...,TYPE0) or without information on the TYPE, the effective axis SHAPE orders S_i^{eff} are calculated using the programmed SHAPE orders by means of the formula:

$$S_i^{\text{eff}} = S_i^p \frac{a_i^{\text{eff}}}{a_i^{\text{max}}}$$

The following applies for (G9 ...,TYP1)

$$S_i^{\text{eff}} = S_i^p$$

In this case:

S_i^p	is the axis SHAPE order programmed with G9 (...)
a_i^{eff}	Active axis acceleration in the current NC block With linear interpolation, the value depends on the current path segment of the axis. With other types of interpolation (e.g. circular, helical), it corresponds to axis acceleration normally programmed with "AxAcc". If the functions "inclined plane" or "axis coupling" are used, effective axis acceleration is generally decreased once again as compared to the "AxAcc" value!
a_i^{max}	the maximum axis acceleration (parameter AX/Dr/Acc/FeedAxAcc (MP 1010 00001))



"AxAcc" does not change the maximum axis acceleration parameter AX/Dr/Acc/FeedAxAcc (MP 1010 00001)!

Relation between SHAPE order and jerk

Description: With the axis SHAPE orders S_i^{eff} , a maximum jerk of r_i^{max} (derivation of acceleration from time) is defined as a limit not to be exceeded in any motion.
This jerk is defined by:

$$r_i^{\text{max}} = \frac{a_i^{\text{max}}}{S_i^p T_{\text{ipo}}}$$

T_{ipo} : Cycle time of the interpolator

Example: The x-axis has a maximum acceleration (parameter AX/Dr/Acc/FeedAxAcc (MP 1010 00001)) of 10 m/s². The programmed axis SHAPE order is 5 ms and the interpolation cycle is 4 ms.

According to the formula above, a maximum jerk of 500 m/s³ is thus defined for the x-axis.

5.2.10 No plane "G16"

Description: Deactivates an active plane.

The following applies:

- If a main or secondary axis is removed from a channel (e.g. in connection with the functionality "axis transfer"), the control automatically switches off the selected plane and activates the function G16.
Circular or helical interpolation cannot be performed in this channel as long as no valid plane is selected.
- Some applications, types of machines or processing functions do not require the configuration of a plane if neither circular nor helical interpolation is necessary (e.g. channels with only one machining axis).
In parameter CHAN/Ch/Coord/Acs/ChAxClass (MP7010 00030) (coordinate significance), "0" (no working-related significance) can then be entered for each axis.
- If no plane function (G17, G18, G19, G20) was entered for a channel in switched-on state (parameter CHAN/Ch/Ini/ChResetState (MP 7060 00020) and parameter CHAN/Ch/Ini/NcResetState (MP 7060 00010)), the function G16 is automatically activated for the respective channel.

- The functions G16, G17, G18, G19 and G20 form a modal group and therefore deselect each other.

Syntax:

G16

Tab. 5-34: Syntax G16

Special features and restrictions:

At an active milling cutter path correction (G41/G42), the active plane must not be switched or deactivated.

5.2.11 Plane switching "G17, G18, G19"

Description:

Select the active plane using the plane switching function. The plane is defined by a main and a secondary coordinate, and, if necessary, an infeed axis positioned perpendicularly to this plane. Thus, G17, G18 or G19 are programmed without parameters.

For a **typical Cartesian standard machine tool**, the axes X, Y and Z span a Cartesian workpiece coordinate system (WCS). Using two coordinate axes each of the workpiece coordinate system, a total of three different basic levels can be defined.

Select the active plane using the plane switching function. The plane is defined by a main and a secondary coordinate, and, if necessary, an infeed axis positioned perpendicularly to this plane. Thus, G17, G18 or G19 are programmed without parameters.

G17, G18 and G19 specify these coordinates and the active plane as follows:

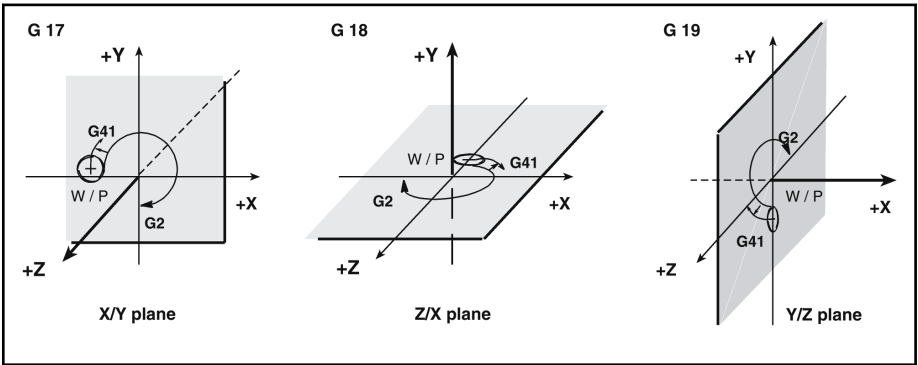


Fig. 5-13: Plane switching

	Main coordinate	Secondary coordinate	Infeed coordinate	Active plane
G17	X	Y	Z	XY
G18	Z	X	Y	ZX
G19	Y	Z	X	YZ

Tab. 5-35: Overview on the coordinates G17, G18 and G19

Syntax:

G17

G18

G19

Tab. 5-36: Syntax of G17, G18, G19

Special features and restrictions:

- At an active milling cutter path correction (G41/G42), the active plane must not be switched or deactivated.

- The following applies to the interpolation parameters I, J, K for G2/G3 and G33:
 - "I" refers to the x-axis
 - "J" refers to the y-axis
 - "K" refers to the z-axis
- The tool length correction is assigned to the active plane with G47(Act-Plane) as follows:
 - L1 of the main coordinate
 - L2 of the secondary coordinate
 - L3 of the infeed coordinate.
 This assignment is updated after the active plane has been switched.
- The functions G16, G17, G18, G19 and G20 form a modal group and therefore deselect each other.

Assignment irrespective of axis names:

Axis names can be user defined in the MTX CNC system. The axes - which the workpiece coordinate system WCS are to span - are thus selected using the machine parameter 7010 00030 (coordinate significance). **Independently** of the configured axis name, this parameter specifies which axes open the X, Y and z-coordinates of the workpiece coordinate system:

Axis with "axis significance X": X coordinate of the WCS

Axis with "axis significance Y": Y-coordinate of the WCS

Axis with "axis significance Z": Z-coordinate of the WCS

In general, the following assignment results for G17, G18 and G19:

	Main coordinate	Secondary coordinate	Infeed coordinate	Active plane
G17	X-coordinate WCS	Y-coordinate WCS	Z-coordinate WCS	XY-coordinate WCS
G18	Z-coordinate WCS	X-coordinate WCS	Y-coordinate WCS	ZX-coordinate WCS
G19	Y-coordinate WCS	Z-coordinate WCS	X-coordinate WCS	YZ-coordinate WCS

Tab. 5-37: Overview on axis name assignments

5.2.12 Advanced plane switching "G17(...), G18(...), G19(...)"

Description: In addition to the selection of the active plane, advanced plane switching defines the axes to span the current workpiece coordinate system (WCS). Thus, a workpiece coordinate system deviating from the setting in the machine parameters can be defined at runtime. Based on this coordinate system, the current plane can be selected.

NC functions with syntax according to DIN 66025 (incl. extensions)

Select the plane according to the following overview:

	Main coordinate	Secondary coordinate	Infeed coordinate	Active plane
G17	X-coordinate WCS	Y-coordinate WCS	Z-coordinate WCS	XY-coordinate WCS
G18	Z-coordinate WCS	X-coordinate WCS	Y-coordinate WCS	ZX-coordinate WCS
G19	Y-coordinate WCS	Z-coordinate WCS	X-coordinate WCS	YZ-coordinate WCS

Tab. 5-38: Overview on axis name assignments

Syntax:

G17 (<Axis1>,<Axis2>,<Axis3>)	
G18 (<Axis1>,<Axis2>,<Axis3>)	
G19 (<Axis1>,<Axis2>,<Axis3>)	
with	
<Axis1>	"X-coordinate" of the workpiece coordinate system (WCS)
<Axis2>	"Y-coordinate" of the workpiece coordinate system (WCS)
<Axis3>	"Z-coordinate" of the workpiece coordinate system (WCS)

Tab. 5-39: Syntax G17 - G19

Special features and restrictions:

- The axes can be programmed with a logical (channel-based) or physical (system-wide) axis name and have to differ from one another. Enter the axes either directly or as CPL string variable.
- The programmed axes currently have to be assigned to the channel.
- While programming, only enter the axes to be reconfigured as parameters. The corresponding coordinates of the WCS remain unchanged for all other axes.
Example: G17 (, , W)
The z-coordinate of the WCS is defined by the W axis.
The x- and y-coordinates of the WCS remain unchanged.
The active plane is opened by the x- and y- coordinates of the WCS.
- Programming an empty bracket expression restores the WCS configured in MP7010 00030 and then activates the selected plane.
Example: G17 ()
- The function must not be programmed while milling cutter path correction (G41/G42) is active.
At an active milling cutter path correction, the plane must not be switched or deactivated.
- If axis transformation is active, advanced plane switching must not be programmed, since the WCS is spanned by spatial coordinates.
- The functions G16, G17, G18, G19 and G20 form a modal group and therefore deselect each other.

5.2.13 Free selection of planes (independent of the WCS) "G20"

Description: The free selection of planes is necessary for applications in which the active plane must decoupled from the current workpiece coordinate system WCS.

This function allows to use any axes to span a coordinate system. Whether these axes also span the current WCS is of no importance.

Thus, it is also possible to settle the tool corrections (G41, G42, G47) in the main, secondary and infeed coordinates of the current plane while the workpiece transformations (placements, spanning correction) simultaneously refer to a workpiece coordinate system spanned by other axes.

Syntax:

G20 (<Axis1>,<Axis2>{,<Axis3>})	
with	
<Axis1>	Main coordinate of the plane to be spanned Is assigned to the interpolation parameter "I".
<Axis2>	Secondary coordinate of the plane to be spanned Is assigned to the interpolation parameter "J".
<Axis3>	Infeed coordinate of the plane to be spanned Can be programmed if the declaration of an infeed coordinate for tool correction "G47(ActPlane)" is required.

Tab. 5-40: Syntax G20

Special features and restrictions:

- The axes can be programmed with a logical (channel-based) or physical (system-wide) axis name.
- A milling cutter path correction G41/G42 refers to the selected plane. Therefore, the active plane can be switched only when the milling cutter path correction (G40) is not active. Otherwise, an error is reported.
- The tool length correction is assigned to the active plane with "G47(ActPlane)" as follows:
L1 of the main coordinate
L2 of the secondary coordinate
L3 of the infeed coordinate.
This assignment is updated after the active plane has been switched.
- The functions G16, G17, G18, G19 and G20 form a modal group and therefore deselect each other.

5.2.14 Contour move programming "G31"

Description: The contour move programming is a special type of contour definition allowing to almost completely apply drawing dimensions as NC parameters. The contour can consist of straight line and circle segments.

The contour move programming supports for example:

- Programming straight lines
- Straight point-to-point, three-point and four-point moves
- Connecting chamfers
- Circle programming with extended programming options compared to the standard programming
- Automatic intersection determination for incompletely defined contour segments for the transitions: straight line - straight line, circle - circle, straight line - circle or circle - straight line
- Tangential straight line connections to circles and tangential connecting circles between two arcs

Transition chamfers and roundings can optionally be programmed using the NC functions CHSL, CHLL and RNDL.

NC functions with syntax according to DIN 66025 (incl. extensions)

G31 contours always refer to the currently active plane (G17, G18, G19).

Function parameter

X, Y, Z, ...	Coordinate names (as configured)
A	Angles (syntax as for polar angle 1 of the polar coordinate programming; can be set via machine parameters)
I, J, K	Interpolation parameters for the circle center point (the circle center point is always specified as absolute for G31)
R	Circle radius
CW, CCW	Direction of rotation of the circle: Clockwise, counterclockwise
TG	Tangential transition or tangential connection
CS	Chamfer segment of a connecting chamfer
CL	Chamfer length of a connecting chamfer

Tab. 5-41: Parameter of the function "contour move programming G31"

Syntax:

Variants of the straight line programming for G31:	
X	Straight line with programmed end point (like G1)
A <Angle>	Straight line with angle programming
A <Angle> X ... or Y ... or Z ...	Straight line with angle and a programmed coordinate (straight point-to-point move)
G31 A <Angle> X ... Y ... or Z... X... or Y... Z ...	Straight line with angle specification and completely programmed end point
A <Angle> CL <Length> X	Straight line with angle; programmed length with programmed end point
A <Angle> CS <Section> X	Straight line with angle; programmed chamfer section and programmed end point
TG	Linear tangential connection (e.g. between two circles)
TG X	Tangential connection with programmed end point

Tab. 5-42: Variants of the straight line programming for G31

Variants of the circle programming for G31:	
X I CW or CCW	Standard circle
X I	Circle without specified direction of rotation
X ... or Y ... or Z ... I CW or CCW	Circle with incomplete position specification
I CW or CCW	Circle with undefined end point
I R <Circle radius> CW or CCW	Undetermined circle
R <Circle radius> TG	Tangential connecting circle
X R <Circle radius> TG CW or CCW	Circle with tangential start

Tab. 5-43: Variants of the circle programming for G31

Example:

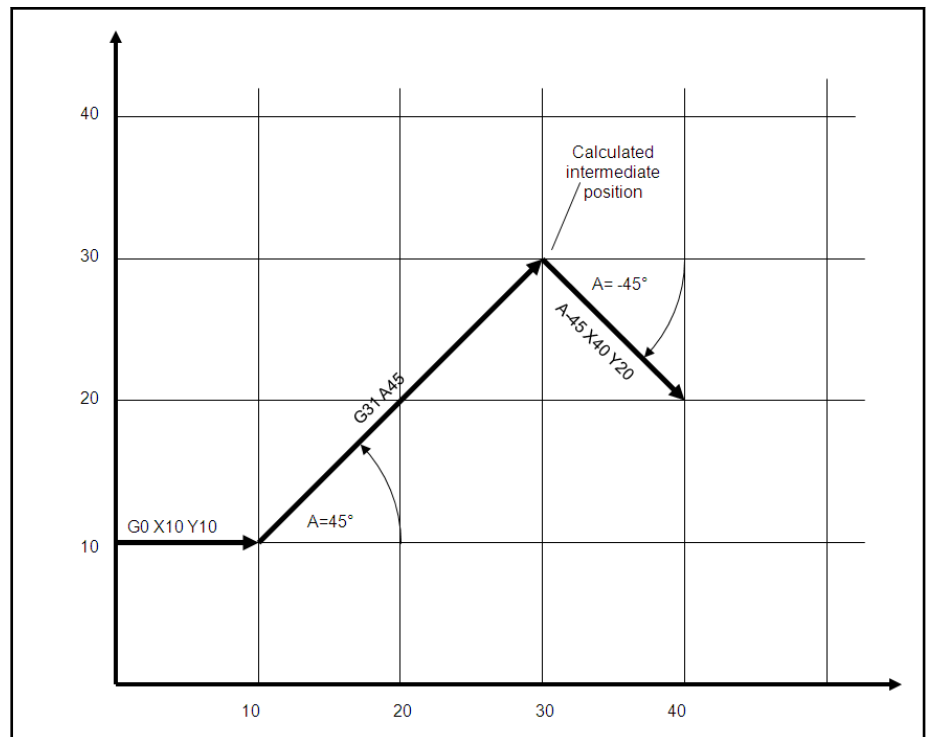


Fig. 5-14: Straight three-point move

Program:

```

N10 G0 X0 Y10
N20 G0 X10 Y10
N30 G31 A45 F1000 ; Straight three-point move
N40 A-45 X40 Y20 ; To end position X40 Y20
M30

```

Special features and restrictions:

- Circle center points are always programmed in absolute coordinates.
- Both coordinates always have to be programmed for the circle center point: Main axis coordinate and secondary axis coordinate.
- If local chamfers and roundings are programmed while a modal chamfer or rounding is active, the local chamfer or rounding overrides the modal chamfer or rounding.

5.2.15 Thread cutting "G33"

Description: "Thread cutting" allows for cutting of the following threads:

- Longitudinal thread
(cutting motion parallel to the main axis of the active plane)
- Face thread
(cutting motion parallel to the secondary axis of the active plane)
- tapered thread
(the main and secondary axes of the active plane are involved in the cutting motion)

G33 can be programmed with a speed-controlled and with a position-controlled spindle.

The cutting movement is always linked to the main spindle active on the relevant channel (refer to [chapter 6 "NC functions with high-level language syntax" on page 195](#)).

NC functions with syntax according to DIN 66025 (incl. extensions)

The feed rate of the cutting motion results from the current spindle speed and the programmed thread pitch portions (fixed, variable, refer to syntax).

Special features:

- Single and multiple threads can be machined.
- Constant and variable thread pitches can be programmed.
- Fast retract motion can be programmed.
- Sequenced threads can be manufactured



The feed potentiometer has no effect while G33 is active.



As for the circular interpolation (G2, G3), the thread cutting function also depends on the active plane (G17...G20).

Syntax:

G33 <EP> <FixedPitch> {<Var. Pitch>} {starting angle}	
with	
<EP>	Axis coordinates of main and secondary axis of the active plane. The active plane is defined by G17, G18, G19 or G20. Example: The active plane for G18 is usually defined by the axes Z (main axis) and X (secondary axis).
<fixed pitch>	Defines the path (in mm) traveled in the direction of the main or secondary axis per spindle revolution. This value is programmed by the interpolation parameter (I, J or K) valid in the corresponding active plane. For tapered threads, the thread pitch specified always has to refer to the main cutting direction. Example: For G18, the parameter K is assigned to the main axis, and I is assigned to the secondary axis. For a longitudinal thread (pitch in direction of main axis), the fixed thread pitch is programmed with the K-address.
<Var. Pitch>	Optional parameter with the address DF Defines the pitch increase/decrease per spindle revolution in mm. Programming: "DF<Value>" with <Value> in mm.
<Starting angle>	Optional parameter If the <Starting angle> is not programmed, 0 degree is assumed. The starting angle (offset) is required for multi-start threads. The interpolation parameter not assigned to the active plane is used as address. Example: The addresses I and K are assigned to the plane for G18. Therefore, the address of the starting angle is J.

Tab. 5-44: G33 syntax

Examples: Longitudinal thread

G91 G18 G8 M3 S1000	Activate relative dimension programming Activates the Z/X plane
G0 X-10	Infeed motion of the cutting tool (1)
G33 Z-50 K2	Thread cutting (2) End point: incremental by -50 mm in z-direction Fixed thread pitch: 2 mm/rev. Interpolation parameter: here K
G0 X10	Retract cutting tool (3)

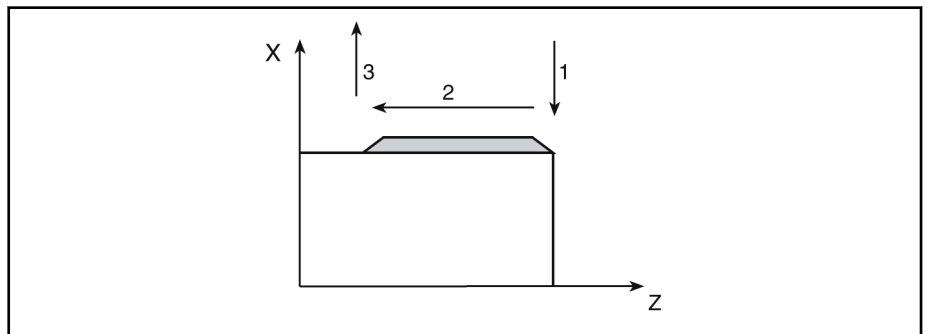


Fig. 5-15: Longitudinal thread

Face thread

G91 G18 G8 M3 S1000	Activate relative dimension programming Activates the Z/X plane
G0 Z-10	Infeed motion of the cutting tool (1)
G33 X40 I2	Thread cutting (2) End point: incremental by -40 mm in x-direction Fixed thread pitch: 2 mm/rev. Interpolation parameter: here I
G0 Z10	Retract cutting tool (3)

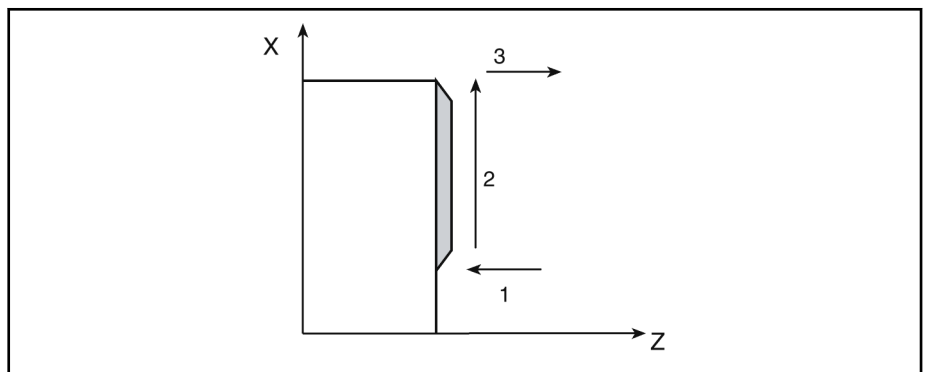


Fig. 5-16: Face thread

Tapered longitudinal thread

G91 G18 G8 M3 S1000	Activate relative dimension programming Activates the Z/X plane
G0 X-20	Infeed motion of the cutting tool (1)
G33 Z-50 X15 K2	Thread cutting (2) End point: incremental by -50 mm in z-direction and +15 mm in x-direction Fixed thread pitch: 2 mm/rev. Interpolation parameter: here I
G0 X5	Retract cutting tool (3)

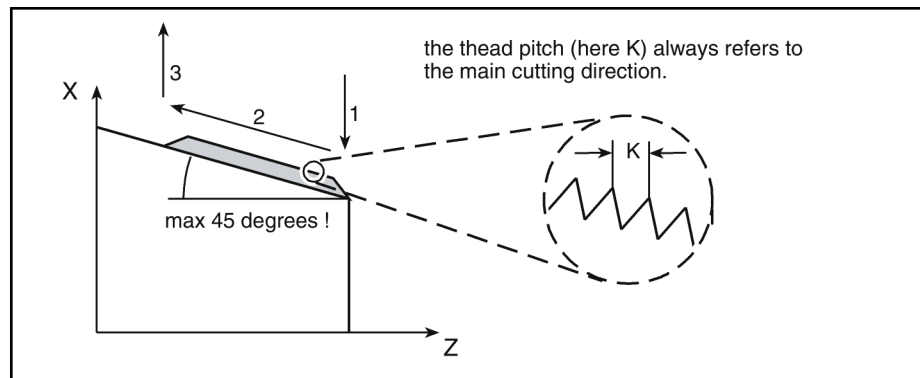


Fig. 5-17: Tapered longitudinal thread

Sequential threads

- These can be manufactured from all thread types described above.
- They are programmed by several consecutive G33 blocks.

With each G33 block programmed, the NC checks whether a subsequent G33 block has been programmed with distance (path). If this is the case, it is changed to the next block without axis standstill.

Multi-start threads

Multi-start threads are manufactured by displacing the starting angle offset (for the starting angle, refer to ["Syntax:" on page 132](#)).

A four-start thread is manufactured by four cuts displaced by 90 degrees each (0, 90, 180, 270).

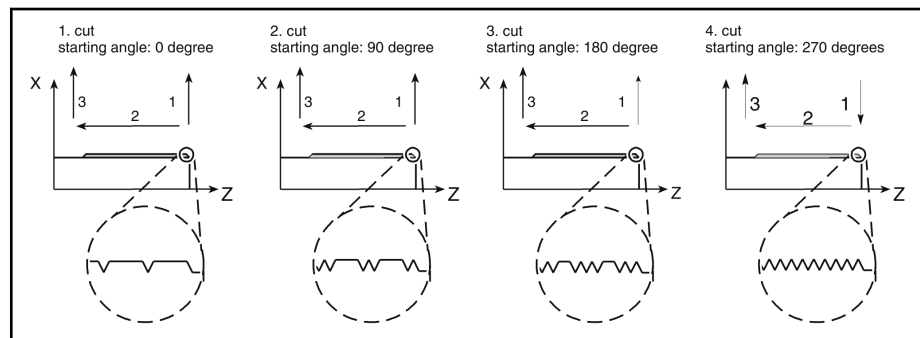


Fig. 5-18: Four-start thread

- Special features**
- At the beginning and at the end of a thread cutting manufacturing, the axes involved have to be accelerated or decelerated.
Thus, always specify a sufficiently long entry path (to accelerate the cutting axes) and leaving path (to stop).
 - The function "ThreadSet (...)"
(see [chapter 6.73 "Main spindle switching MainSp, MSP" on page 294](#)) can be used with G33.
 - Adjusting the dynamics of the cutting motion.
 - Configuring a retract motion activated via interface signal.
 - Switching the spindle mode (speed control, position control).
 - Setting a signal at the channel interface (configurable within the "active function" signals using machine parameters).

5.2.16 Milling cutter path correction "G40, G41, G42"

Description: The milling cutter path correction function moves the tool along an equidistant path parallel to the programmed path while executing a part-specific program (equidistant path = path with right-angled, constant distance from the programmed contour). The distance between the equidistant and the programmed path depends on the active radius correction value.

The following figure shows the principle:

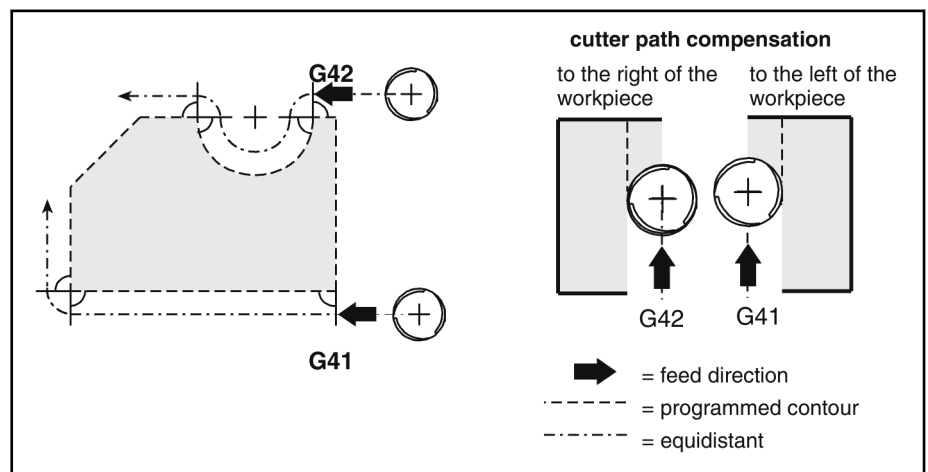


Fig. 5-19: Milling cutter path correction

Syntax:

NC functions with syntax according to DIN 66025 (incl. extensions)

G40	Milling cutter path correction OFF (switched-on state) The behavior of G40 blocks without programmed traversing motion can be set via machine parameters: • Immediate exiting of the correction build-up vertically to the previous traversing block (default setting). • The correction build-up is exited with the next programmed traversing block. If a traversing motion is programmed in the G40 block, the control deactivates the cutter path correction linearly while traveling to the end point of the programmed traversing motion. Depending on whether there is an inner or outer corner, exiting the correction build-up starts at the end perpendicular of the preceding or at the starting perpendicular of the current block. Based on the current approach for contour transitions (G43/G44), a transition path segment is inserted if the corner is on the outside. This behavior also applies for exiting the correction build-up after a G40 block without traversing motion if the correction build-up is only to be exited with the next traversing block.
G40(NOM)	Milling cutter path correction OFF Is programmed without traversing motion. The correction is only retracted with the next traversing block. It is to be considered that a following programmed incremental motion refers to the center of the tool. The tool is traversed by the given incremental motion. The resulting motion differs if the incremental traversing motion is directly programmed in one block with G40.
G40(ORTH)	Milling cutter path correction OFF Irrespective of whether there is an inner or an outer corner at the contour transition to the switch-off block, exiting the correction build-up starts at the end perpendicular of the preceding block. The correction build-up is exited linearly along the path to the end point of the programmed traversing motion.
G41	Milling cutter path correction to the left of the workpiece ON (for positive correction values seen in the direction of machining). The behavior of G41 blocks without programmed traversing motion can be set via machine parameters: • Immediate start of the correction build-up vertically to the previous traversing block (default setting). • The correction build-up is only started with the next programmed traversing block. In a G41 block with programmed traversing motion, the correction build-up is started in this block. To start the correction build-up in a traversing block, the following applies: the correction is build-up linearly on the way to the programmed end point. If there is an inner corner, the correction build-up is started at the starting perpendicular of the next programmed traversing motion. If there is an outer corner, activation of cutter path correction starts at the end perpendicular of the switch-on block. Based on the current approach for contour transitions (G43/G44), a transition path segment is inserted if necessary.
G41(ORTH)	Milling cutter path correction to the left of the workpiece ON Irrespective of whether there is an inner or outer corner at the subsequent contour transition, the correction build-up started on the starting perpendicular of the subsequent traversing blocks.

G42	Milling cutter path correction to the right of the workpiece ON otherwise identical to G41
G42(ORTH)	Milling cutter path correction to the right of the workpiece ON Otherwise as G41(ORTH)

Tab. 5-45: G40-G42 syntax



Correction values can be immediately activated or deactivated without programming a separate traversing motion. This can result in damages to the workpiece or the tool.

Thus, observe all information in this chapter!

Special features and restrictions:

- G40, G41 and G42 act modally and deselect each other.
- The behavior when starting and exiting the correction build-up is affected via machine parameters and the function parameters NOM and ORTH.
- *The following is **not allowed** with G41 or G42 active:*
 - G17 ... G20 (plane switching)
 - G63 (tapping without compensating chuck)
 - G74 (approach reference point coordinates)
 - G75 (travel to touch probe)
 - MOB (measuring on a block)
 - G76 (approach to fixed machine axis position)
 - G54.x ... G59.x (Zero offsets)
 - G154.x ... G159.x (inclined plane)
- G40, G41 and G42 do not affect any tool length correction.

5.2.17 Contour transitions for milling cutter path correction: Arc "G43", Intersection "G44" Arc

Description: Function for active milling cutter path correction (G41, G42).

*The control implements a contour transition **at the outer corners** either as an automatically generated*

- arc (G43) or as an
- intersection of the equidistant path (G44).

G43: Arc

The path gap is closed by a tangential arc with the radius "r":

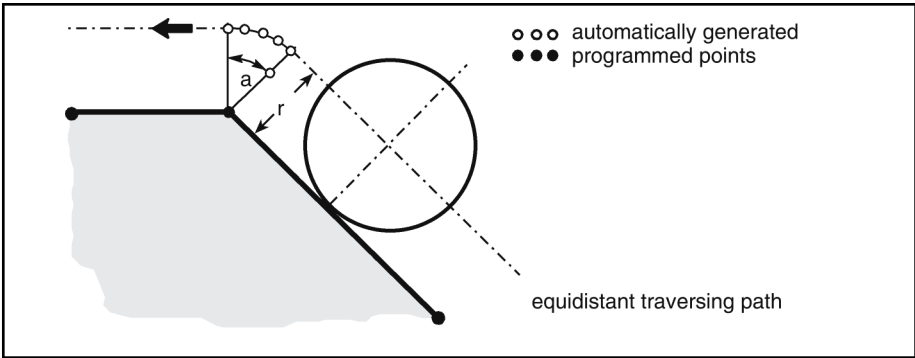


Fig. 5-20: Arc G43

G44: Intersection

The control tries to close the path gap by determining the intersection of the two equidistants.

Depending on the distance "A" between the contour corner "KE" and the intersection "S", the control proceeds as follows:

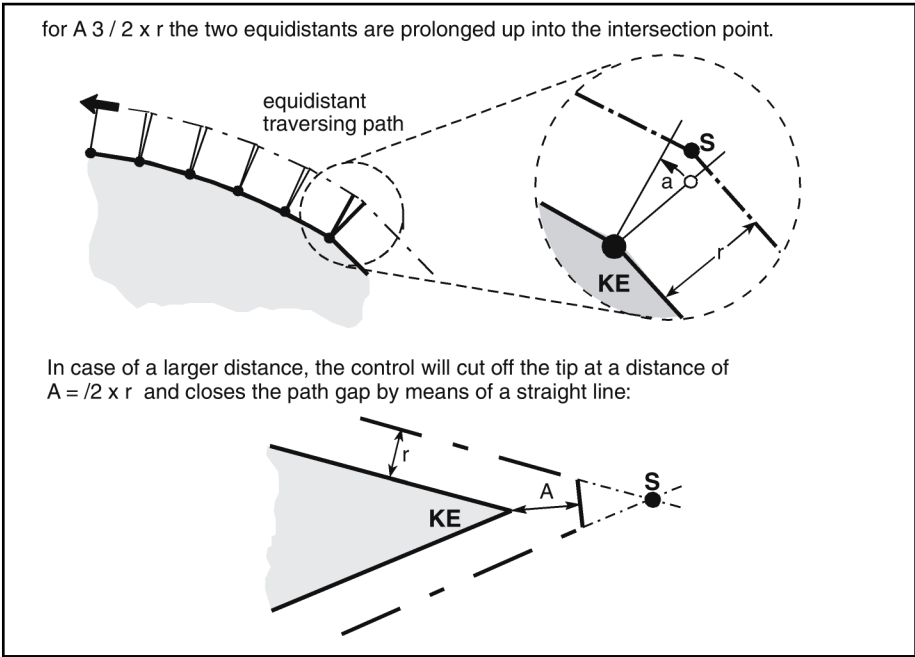


Fig. 5-21: G44 intersection



If no intersection exists, the line is closed using an arc (as for G43).

Syntax:

G43	Contour transition as arc
G44	Contour transition as intersection of the equidistants

Tab. 5-46: Syntax G43, G44

The following applies:

- G43 and G44 act modally and deselect each other.
- The switched-on state can be specified via machine parameters.

Special features and restrictions: G43/G44 are programmed without any path conditions.

5.2.18 Feed correction: Cutter contact point "G45", cutter center point "G46"

Description: Function for active milling cutter path correction (G41, G42).

At circular interpolation, define whether the control keeps the programmed feed

- at the cutter contact point (cutter cutting path) or
- on the cutter center point path

at a constant level.

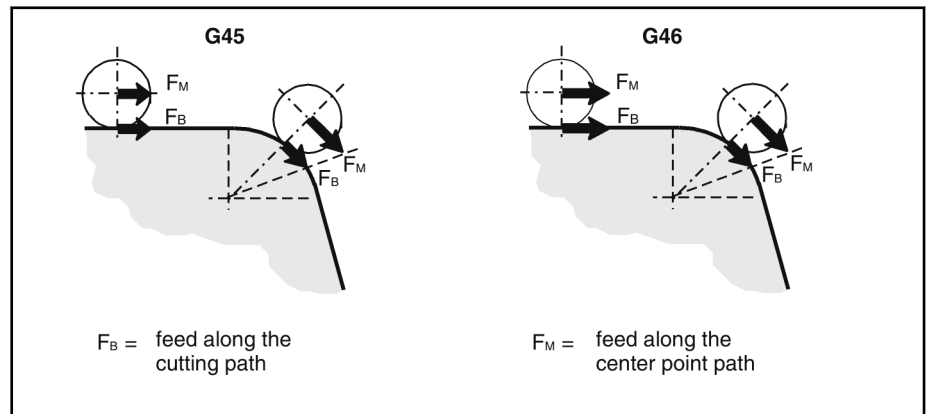


Fig. 5-22: Feed G45 and G46

Syntax:

G45	Keep the feed F_B along the cutting path at a constant level.
G46	Keep the feed F_M along the cutter center point path at a constant level.

Tab. 5-47: Syntax G45, G46

The following applies:

- G45 or G46 act modally and deselect each other
- The switched-on state can be specified via machine parameters

Special features and restrictions: Only use G45 for finish milling, since the feed rate can increase considerably along circular contours.

5.2.19 Tool length correction "G47, G48"

Description: *The function*

- Activates/deactivates tool length correction
- optionally switches the assignment of length correction values L_1 , L_2 and L_3 to individual coordinates (method of operation as for G78; see [chapter 5.2.31 "Compensation switching ON G78, Compensation switching OFF G79" on page 153](#)).

If the tool length correction is active, the following are effective:

- length correction values L_1 , L_2 and L_3 of the currently selected D-correction set (Dxx)
- plus the length correction values L_1 , L_2 and L_3 of an external tool correction (EDxx),
- the function "static tool orientation" (parameters can be set using STO; see [chapter 6.140 "Parameterizing the static tool orientation StatToolOri, STO" on page 388](#))
- the function "tool edge correction", if a tool edge has been entered in the currently selected D-correction block (Dxx) and milling cutter path cor-

NC functions with syntax according to DIN 66025 (incl. extensions)

rection G41/G42 (see [chapter 5.2.16 "Milling cutter path correction G40, G41, G42" on page 135](#)) is active.

Syntax:

G47	Tool length correction ON
G47 (ActPlane)	Assign length correction values L_1 , L_2 and L_3 of the main, secondary and normal coordinates of the active plane and then switch on the tool length correction. Remains modally effective and is automatically reconfigured at every following plane switching.
G47 ({ {-} <L1 coord> } {, { {-} <L2 coord> } {, { {-} <L3 coord> } })	Changes the assignment of length correction values L_1 , L_2 and L_3 to the individual coordinates and then activates the tool length correction.
G47 ()	Assigns the length correction values L_1 , L_2 and L_3 according to the settings in the machine parameters 7050 01300 and 7050 01310 and then activates the tool length correction.
G48	Tool length correction OFF.
with	
<Li coord>	The name of the WCS coordinate the correction L_i (with $i = 1, 2, 3$) is assigned to in the active workpiece coordinate system. If the correction L_i (with $i = 1, 2, 3$) has to have an effect on the coordinates in the tool coordinate system (TCS), the coordinate identifiers XTR, YTR and ZTR have to be used and an appropriate axis transformation has to be active. Optional negative sign: Correction is settled in negative direction. L_i correction values to which no coordinates are assigned are not taken into account.

Tab. 5-48: Syntax G47, G48

Special features and restrictions:

- Coordinates not belonging to the default channel setting are never considered in the "G47()" syntax.
- G47.. and G48 act modally and deselect each other.
- Whether the coordinate addresses programmed under G47 refer to workpiece or tool coordinates depends on the coordinate identifier used (see <coordinate i> above).
- Mixed programming such as "G47(X,,ZTR)" is not permitted.
- G47/G48 can be programmed with other path conditions, traversing information or auxiliary functions.

Examples:	G47 (X, , -Z)	The L ₁ correction is assigned to the x-axis of the workpiece coordinate system (WCS) and calculated in the positive direction. The L ₂ -correction is not settled, since it is not assigned to any coordinate. The L ₃ correction is assigned to the z-axis of the workpiece coordinate system (WCS) and calculated in the negative direction. The tool length correction is subsequently switched on.
	G47 (, , ZTR)	The L ₁ -correction and the L ₂ correction is not settled, since it is not assigned to any coordinate. The L ₃ correction is assigned to the z-axis of the workpiece coordinate system (WCS) and calculated in the positive direction. The tool length correction is subsequently switched on.

5.2.20 Programmable zero offset"G52"

Description: The programmable zero offset is used to move the machine coordinate system MCS in space by programming in the part program.

The programmable zero offset can be activated for a total of five effective zero offset banks (groups).

Offsets from different zero banks are always additive.

The programmable zero offset is a variant allowing to program offset values directly into the part program (without using a zero offset table), thus supplementing the zero offsets G54.1 to G59.5 for which the offset values have to be loaded from zero offset tables.

Together, the programmable zero offset of a particular bank and zero offsets G54.x to G59.x of the corresponding bank form a modal group. The functions of a modal group deselect each other.

The zero offset of a specific bank is deselected with G53.x. All zero offsets are deselected with G53.

Syntax:	G52.<ZO Bank> (<MachineCoordinates>) programmable ZO of <ZO Bank> ON Short form: G52 for programmable ZO of bank 1 with <ZO bank> 1...5, integer	

Tab. 5-49: G52 syntax

Deselect the programmable zero offset with

G53 ZOs for all correction banks OFF

G53.<ZO bank> ZO of <ZO bank> OFF

G54.<ZO bank> ... G59.<ZO bank> Zero offset (on table basis) of the <ZO bank> ON

The following applies:

- G52.1, G53.1, G54.1 ... G59.1
are modal and deselect each other.
G52.2, G53.2, G54.2 ... G59.2
are modal and deselect each other.
G52.3, G53.3, G54.3 ... G59.3
are modal and deselect each other.

NC functions with syntax according to DIN 66025 (incl. extensions)

G52.4, G53.4, G54.4 ... G59.4

are modal and deselect each other.

G52.5, G53.5, G54.5 ... G59.5

are modal and deselect each other.

G53 activates functions G53.1 ... G53.5 and

thus deselecting all zero offsets.

- If programmed separately, the zero offset does not cause any traversing motion. Only the current machine coordinate system (MCS) is offset.
- The programmable zero offset can be programmed together with a traversing motion. The changed ZO moves the current MCS, thus also changing the current WCS. The traversing motion programmed in the same block then refers to the new WCS.

Example: Additive effect of zero offsets from different ZO pages

Program:

N20 G52.1 (X100, Y100)	No traversing motion Zero offset activated on ACS position X100 Y100.
N30 G52.2 (X300, Y100)	No traversing motion Additive zero offset has been activated. The resulting machine zero point is now at the ACS position X400 Y200.
N40 G90 G1 F1000 X10 Y15	The axes traverse to the ACS position X410 Y215.
N50 G53.1	Zero offset bank 1 is switched off. The machine zero point is now at X300 Y100.
N60 X10 Y10	The axes traverse to the ACS position X310 Y110.
N70 G53	All zero offsets are switched off.

Special features and restrictions: The programmable zero point refers to machine coordinates.
The programmable zero offset can be programmed with other path conditions in one block.

A traversing motion can be programmed with a zero offset in one block.

5.2.21 Zero offsets (ZO) "G53", "G53.1-G59.1" to "G53.5-G59.5"

Description: ZO can be used to **offset** the machine coordinate system in space.



If a spanning correction is required, refer to the functionality "BcsCorr".

The offset distances for the machine coordinates in the channel are stored in the "zero offset tables":

- A ZO table contains 5 ZO pages (groups) with 6 zero offsets (ZO) each.
- Offsets from different ZO banks are always additive.
- Offsets within a ZO bank mutually overwrite each other.

- Offsets with channel number 0 are applied in every channel in which they are activated. However, offsets with a channel number >0 can only be applied in the specified channel. Do not use the channel number 0 in case of multiple entries of the same machine coordinates with different channel details, as all following channel-specific offsets of this coordinate are not considered in this case.

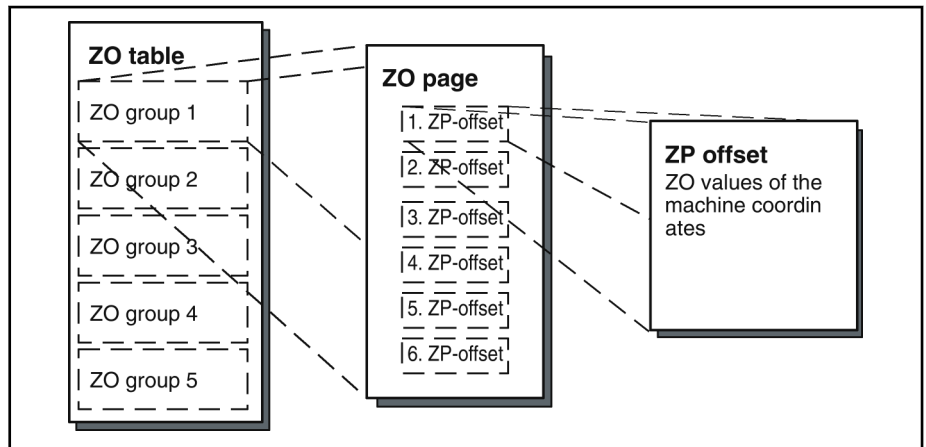


Fig. 5-23: Zero offsets (ZO)



For details about editing ZO tables, refer to the operating instructions.

Syntax: Activate the desired ZO table with "ZoTSel" (see [chapter 6.157 "Activate zero offset tables ZoTSel, ZOS" on page 415](#)).

Program the required function:

G53	all ZOs for correction banks OFF
G53.<ZO page>	All ZOs of the <ZO bank> OFF
G54.<ZO bank>	1. ZO of <ZO bank> ON
G55.<ZO bank>	2. ZO of <ZO bank> ON
G56.<ZO page>	3. ZO of <ZO bank> ON
G57.<ZO bank>	4. ZO of <ZO bank> ON
G58.<ZO bank>	5. ZO of <ZO bank> ON
G59.<ZO bank>	6. ZO of <ZO bank> ON
Short form: G54 bis G59	For the corresponding ZO of bank 1
with	
<ZO bank>	1...5, integer

Tab. 5-50: Syntax G53 - G59

The following applies:

- G52.1, G53.1, G54.1 ... G59.1
are modal and deselect each other.
G52.2, G53.2, G54.2 ... G59.2
are modal and deselect each other.
G52.3, G53.3, G54.3 ... G59.3

NC functions with syntax according to DIN 66025 (incl. extensions)

are modal and deselect each other.

G52.4, G53.4, G54.4 ... G59.4

are modal and deselect each other.

G52.5, G53.5, G54.5 ... G59.5

are modal and deselect each other.

G53 activates functions G53.1 ... G53.5 and

thus deselecting all zero offsets

- If only they are programmed, the functions cause no traversing motion. Only the current machine coordinate system is offset.
- The functions can also be programmed with other path conditions in one block. In such cases, the corresponding zero offset is activated first and then traversed to the programmed position.
- ZO offsets for a certain channel (channel > 0) are only effective in the specified channel. Thus, you have the option to activate different zero points for identical machine coordinate identifiers with one ZO table separately for each channel.

Example: The ZO table "Z01" contains the following offset values in ZO page 1:

1. ZO: X100, Y100
2. ZO: X300, Y100
3. ZO: X500, Y100
4. ZO: X100, Y450
5. ZO: X300, Y450
6. ZO: X500, Y450

Effect of the functions G54.1 to G59.1:

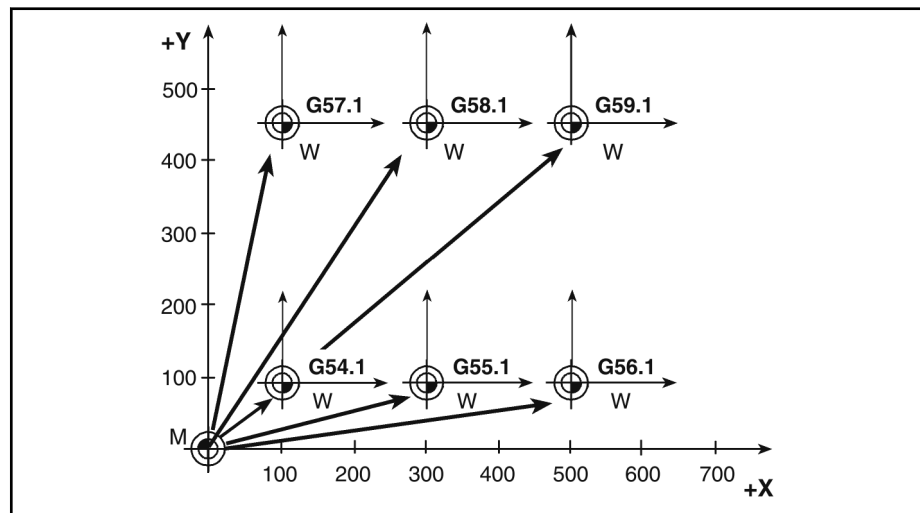


Fig. 5-24: Effect of the functions G54.1 to G59.1

Additive effect of ZOs from different ZO banks.

The ZO table "Z01" contains the following offset values:

ZO bank 1,1. ZO: X100, Y100

ZO bank 1,2. ZO: X300, Y100

ZO bank 2,4. ZO: X200, Y350

:	Current machine position 0.0
:	No ZO
:	All following coordinates are machine coordinates!
N40 ZOS(Z01)	Activate ZO table "Z01"
N50 G54.1	No traversing motion Offset to X100, Y100 activated
N60 G55.1 X...Y...	Offset to X300, Y100 activated Subsequent traversing motion
N70 G57.2 X...Y...	Offset to X500, Y450 activated Subsequent traversing motion
N80 G53	All still-active offsets OFF

5.2.22 Exact stop ON/OFF "G61, G62"

Description: Minimizes the following error at the end of a block.

During tool motion, a time offset between the target and the actual values of the individual axes occurs due to the dynamics of the machine. This "lag effect" causes a following error while chip-producing. The error size depends on the feed rate and the CO factor (axis dynamics). In case of unsteady contour transitions (corners), this following error becomes noticeable due to a corner "finishing".

The following applies:

- The following distance (displacement between target and actual values) at the end of the block can be reduced to the size of certain target windows.
- G61 only acts on feed motions. It has no effect on rapid traverse motions.
- As opposed to the exact stop function for G1, G61 (except for G0) affects all interpolation functions (i.e. also for circular/helical interpolation and spline interpolation).
- G61/G62 act modally and delete each other.

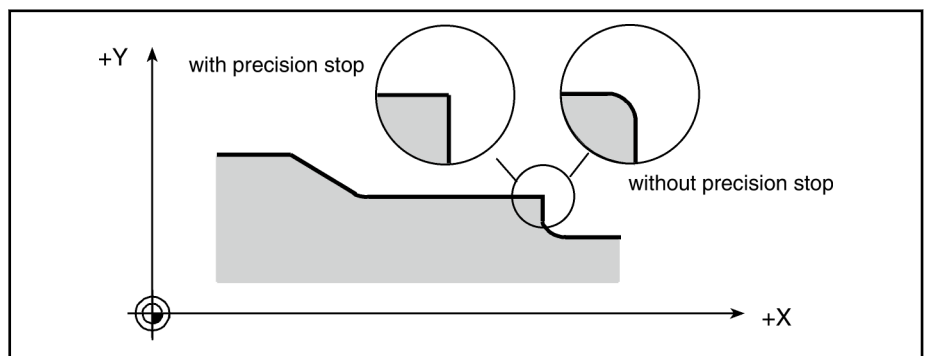


Fig. 5-25: Exact stop



When G61 is active (exact stop switched on), the control always decelerates to $v = 0$ at the end of a block.

NC functions with syntax according to DIN 66025 (incl. extensions)

Syntax: Basic function:

G61	Exact stop ON
G62	Exact stop OFF

Tab. 5-51: Syntax G61, G62

The latest activated positioning window for contour mode remains effective for "exact stop ON".

Default: Positioning window, fine

Optional parameters

To select the positioning window for the feed mode.

G61 (IPS1 IPS2 IPS3)	
with	
IPS1	Exact stop ON Wait for fine positioning window . At the end of the block, the control first reduces the path velocity to $v = 0$. Only when this positioning window was reached for all axes involved, the next block is traversed.
IPS2	Exact stop ON Wait for rough positioning window . "Inpos range 2 active" is displayed on the channel interface (also see the documentation "MTX PLC Interface"). At the end of the block, the control first reduces the path velocity to $v = 0$. Only when this positioning window was reached for all axes involved, the next block is traversed.
IPS3s	Exact stop ON Deceleration to $v = 0$ at end of block. At the end of the block, the control reduces the path velocity to $v = 0$. The next block is then traversed to a positioning window without being checked.

Tab. 5-52: Optional parameters G61 syntax



The parameters "positioning window, fine" and "positioning window, rough" can be determined in the Sercos files for phase 3.

For more information about the Sercos files, please refer to the documentation "MTX Machine Parameters" under "Sercos initialization".

Special features and restrictions:

At the latest, G61/G62 has to be programmed in the block it is to be applied.

See the following example:

```

:
N40 G61                Exact stop ON
:
N50 Y200              Positioning
:
N60 G62                Exact stop OFF
:

```

```

N70 Y0                               Positioning
:
N80 G61 Y200                         Positioning with exact stop ON
:

```

5.2.23 Tapping without compensation chuck "G63"

- Description:**
- Synchronizes linear interpolation of the drill axis with the spindle switched to C-axis mode. Thus, a compensating chuck is not required anymore, since it is used to accept the speed difference between drill axis and spindle.
 - G63 acts only in the programmed block.
 - A signal can be output at the channel interface while tapping (configurable in the "active function" signals using machine parameters).
 - Only the feed potentiometer acts while tapping.



When G63 is activated, the C-axis spindle has to be in spindle mode.



For tapping with several spindles, also see [chapter 6.147 "Spindle selection for tapping without compensation chuck TappSp, TSP" on page 399](#).

If axes have to be hidden for feed computing, see [chapter 6.43 "Feed computation: Hiding of axes FeedAd, FAD" on page 259](#).

Syntax:

G63 (M<3 4> , S<Speed> H<Pitch> , ANG<StartingAngle>) <DrillAxis><InfeedDepth> {F<Feed>}
G63.2 (M<3 4> , S<Speed> H<Pitch>) <DrillAxis><InfeedDepth> {F<Feed>}

Tab. 5-53: G63, G63.2 syntax

The starting angle can only be programmed together with G63.

For G63.2, the spindle always starts at the end position of the G63 previous block.

The thread pitch results from the ratio between path feed and speed (F/S).



Different thread pitches during drilling and retract damages the workpiece/tool!

Therefore, always program drilling and retract in such a way that the resulting thread pitch is identical!

Example: Program:

```

N20 G0 X20 Y15 Z10 F1000           Positioning
;----- Programming the speed -----
; the pitch is calculated by the control: H=F/S
N30 G63(M3,S500) Z-10 F500         Drilling (DrillAxis Z)
N40 G63(M4,S500) Z10 F500          Retract (DrillAxis Z)
N50 G0 Z20
;----- Programming the pitch -----
; the speed is calculated by the control: S=F/H
N60 G63(M3,H5) Z-10 F500           Drilling (DrillAxis Z)
N70 G63(M4,H5) Z10                 Retract (DrillAxis Z)
N80 G0 Z100
;
;----- Deep hole drilling-----
N120 G0 X20 Y15 Z10 F1000          Positioning for deep hole drilling

```

NC functions with syntax according to DIN 66025 (incl. extensions)

```

N130 G63 (M3,S500) Z-20 F500      Drilling subsection 1
N140 G63 (M4,S500) Z5 F500       Retract subsection 1
N150 G63 (M3,S500) Z-40 F500     Drilling subsection 2
N160 G63 (M4,S500) Z5 F500       Retract subsection 2
:
;----- Drilling with starting angle -----
N220 G0 X60 Y15 Z10              Positioning
N230 G63 (M3,S500,ANG60) Z-20    Drilling with starting angle 60 degrees
F500
N240 M257                        Message to PLC
N250 G63.2 (M4,S500) Z5 F500     Retract with G63.2
:

```

- Special features and restrictions:**
- The retract block is to be programmed immediately after the drilling block.
 - If no other feed rate (address F) is entered for the G63 block, the control uses the active path feed rate.
 - The addresses "M" and "S" act only in the programmed G63 block.
 - Neither "align spindle" nor "spindle stop" is required before G63.
 - Switching to C-axis mode takes place automatically.
Before the start, the control waits internally for "INPOS" of all axes involved. If an axis drifts off its INPOS range, G63 is not started.
 - After the retract block, the spindle automatically runs again in spindle mode.
 - G63 with CPL expressions:
Programming as "G63 (M[MCODE%], S[DREHZ])" is not possible.
Instead, program "G63 ([MCODE\$], S[SPEED])"

5.2.24 Inch programming "G70", Local inch programming "INCH(...)"

Description: Permits the specification of path/feed information and accelerations in **inch units**.

Syntax:	G70	Information on path, feed and acceleration are interpreted in inch.
	<Axis>=INCH (<Value>)	The path information for asynchronous linear axis is interpreted in inch.

Tab. 5-54: Syntax G70

The following applies:

- G70 acts modally and deselects G71.
- The switched-on state can be specified via machine parameters.
- G70 can also be programmed with other path conditions in one block.
- *G70 refers to:*
 - Traversing distances
 - Feeds
 - Accelerations
 - Other geometric auxiliary variables (such as the interpolation parameters I, J, K).

Example:

```

N40 G70      Starting at N40 (inclusive), all path and feed information is interpreted in inch.
:
N50 W=INCH (2)  Travel the asynchronous linear axis W to position 2 (inch).

```

5.2.25 Metric programming "G71"

Description: Specifies path/feed information and accelerations in **metric units**.

Syntax:

G71	Information on path, feed and acceleration is interpreted in metric units.

Tab. 5-55: Syntax G71

The following applies:

- G71 acts modally and deselects G70.
- The switched-on state can be specified via machine parameters.
- G71 can also be programmed with other path conditions in one block.
- *G71 refers to:*
 - Traversing distances
 - Feeds
 - Accelerations
 - Other geometric auxiliary variables (such as the interpolation parameters I, J, K).

Example:

N40 G71 Starting at N40 (inclusive), all path and feed information is interpreted in metric units.
:

5.2.26 Approaching the reference point coordinates "G74"

Description: The axes programmed in the same block as G74 traverse simultaneously to their reference position.

The traversing velocity is in rapid traverse/feed mode depending on G0/G1.

For G74, neither the reference point cams nor the markers are taken into account. G74 is a **mere positioning to the absolute axis positions** (i.e. it also applies to axes with distance-coded encoders).



Possibly active corrections are not considered during this positioning!

The following applies:

- G74 acts on a block-to-block basis and is canceled when all machine axes programmed in G74 block reached their reference point.
- When approaching the reference point with G74, the actual axis values are not set again.
- Any still active corrections, ZOs, etc., are not used in the G74 block for the programmed axes.

Syntax:

G74<AxisCoordinates>	Start "approach reference point coordinates"
with	
<AxisCoordinates>	Axis addresses have to be programmed together with a numerical value (e.g. X1 Y1 Z1). This numerical value has no effect on the reference point position. It is only required to complete the word.

Tab. 5-56: Syntax G74

NC functions with syntax according to DIN 66025 (incl. extensions)

Example:

<pre>N100 G74 X1 Y1 Z1 :</pre>	The x-, y- and z-axes start traveling to their reference positions simultaneously and reach them at the same time.

Special features and restrictions: Program G74 in a separate block together with the axes to be traversed. Auxiliary and special functions can also be programmed in the same block.

5.2.27 Approaching the reference point "G74(HOME)"

Description:

 The G74(HOME) function can also be used for asynchronous axes!

Triggers the referencing of programmed axes using the part program.

The function issues the Sercos command "drive-controlled referencing" (S-0-0148) for the drives of the programmed axes.

As a result, the drive decouples from the control and generates its positioning specifications for referencing. The drive uses the Sercos parameters S-0-0147 (referencing parameters), S-0-0041 (referencing velocity) and S-0-0042 (referencing acceleration).

 For a detailed description, refer to the documentation "MTX Functional Description".

The following applies:

- If several axes are programmed in the G74(HOME) block, these axes approach their reference points independently of each other (no path mode). Thus, the reference points are not reached at the same time.
- **Synchronous** as well as **asynchronous** axes can be programmed. Block processing is paused until all drives of the control acknowledge that their respective reference points has been reached successfully.
- The function is identical to "approach reference points" in the operation mode "Setup".

Syntax:

G74 (HOME) <AxisCoordinates>	
	Triggers the Sercos command "drive-controlled referencing" for programmed axes.
with	
<AxisCoordinates>	Axis addresses have to be programmed together with a numerical value (e.g. X1 Y1 Z1). This numerical value has no effect on the reference point position. It is only required to complete the word.

Tab. 5-57: G74(HOME) syntax

Example:

<pre>N1 G74 (HOME) X1 Y1 Z1 :</pre>	Send Sercos command "Drive-controlled referencing" to the drives of the axes X, Y and Z.

5.2.28 Traveling to probe "G75"

Description: *The NC has two G75 functions:*

- Travel against the first touch probe (G75.1 or G75)
- Travel against the second touch probe (G75.2)

Both G75 functions:

- Travel and measure, these functions create movements using the touch probe function of the drive.
- They act block-by-block or are local functions.
- They use only the linear interpolation.
- They override the module group G0, G1, G2 ... in exactly the block they are programmed in.
- They stop the block preparation automatically. WAIT does not have to be programmed explicitly.

Syntax:

G75.1 <axis coordinates> or G75 <Axis coordinates>	Activate all measuring axes of the channel reacting on touch probe 1 and moving travel axes to <axis coordinates>.
G75.2 <AxisCoordinates>	Activate all measuring axes of the channel reacting on touch probe 2 and moving travel axes to <axis coordinates>.

*Tab. 5-58: Syntax G75***Example:**

```

N100 G75.2 Y250 F500;           Travel with F500 to position Y250
110 IF SD(9)=0 THEN;           If the 2. second touch probe triggered
120 YPOS=PCSPROBE("Y");       Stores the measured value of the y-axis in
:                               YPOS
N130 MSG(CONTACT)
140 ELSE
150 SETWARN("NO CONTACT!")
N160 M0
170 WAIT
180 CLRWARN()
190 ENDIF
N200 ...

```

Special features and restrictions:

- **Commissioning:**
Before a G75 function can be programmed, it has to be commissioned. All required steps are described in the "MTX Functional Description" documentation.
- G75 has to be programmed with at least one axis coordinate (that means with at least one travel axis).
- The coordinates determine the maximum search depth up to which the touch probe had to have triggered.
- It is not allowed to program two G75 functions in the same NC block. This is checked by the NC and causes an error.
- *After a G75 block, create a suitable CPL program to*
 - check whether the touch probe has triggered;
with SD(9)
 - apply the measured values in the part program;

NC functions with syntax according to DIN 66025 (incl. extensions)

with PCSPROBE() or PROBE()

- generate the required warnings and error messages;
with SETWARN(), etc.

5.2.29 Approaching the fixed machine axis position "G76"

Description: Permits programming and linear approaching of positions in the machine coordinate system without having to separately deactivate any active corrections and transformations in the part program.

This can be necessary for example, when changing the tool, checking tools for damage, in measuring cycles or when changing pallets.

G76 acts on a block-to-block basis in the rapid traverse (G0) or in feed mode (G1) and with G93 (time programming), G94/G95 (feed programming) and the F-word.

The following functions are used for G76:

- Tool corrections (G41, G42, G47 Dxx, EDxx)
- Input helps (Mirror, Scale, Rotate, Shift)
- Zero offsets (G54.1...G59.5)
- Placements (BcsCorr, G154.1 - G159.5)
- Axis transformations based on machine coordinates,
e.g. 5-axis transformation
- Relative dimension programming (G91)
- Set program position (SetPos)

Syntax:

G76<MachineCoordinates>	
with	
<MachineCoordinates>	Machine coordinates to be approached

Tab. 5-59: G76 syntax

Special features and restrictions:

- G76 can be written together with other path conditions (e.g. G0, G1, G93, G94, G95, F-word).
- Local relative dimension programming ("IC(...)"; see [chapter 5.2.32 "Absolute dimension programming G90, Relative dimension programming G91, Local absolute dimension programming AC\(...\), Local relative dimension programming IC\(...\)"](#) on page 155) is invalid together with G76 and causes a runtime error.

5.2.30 Asynchronous subroutines: Repositioning individual coordinates "G77"

Description: The following applies in an asynchronous subroutine (Asup):

The coordinates programmed with G77 are traversed on their respective adjustment position (the mandatory coordinate position after the asynchronous subroutine). Optionally, an offset can additionally be specified to the adjustment position on which the coordinate is to be traversed in the G77 block.

The current position of the coordinate can also be specified as adjustment position.



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

G77<Coord 1><Mode><Coord n><Mode>...F<Value>	
with	
<Coord i>	Coordinate name (e.g. "x")
<Mode>	1: Traverse coordinate to adjustment position 0: Determine current coordinate position to adjustment position =IC(<Offset>): Traverse coordinate to the adjustment position value + <Offset> 2: Coordinate depends on the approaching strategy in relation to approaching the point of interruption: "Starting point"- Adjustment position = current position + starting point of block - point of interruption of block "End point"- Adjustment position = current position + end point of block - point of interruption of block "Point of interruption"- Adjustment position = current position (corresponds to mode 0)
<Value>	Feed to position at the point of return to contour.

Tab. 5-60: Syntax G77

Special features and restrictions:

- The function is intended for use in an asynchronous subroutine.
- All coordinates not programmed via G77 are automatically traveled on a straight line to the point of return to contour at the end of the asynchronous subroutine.
- Any correction changes made in an asynchronous subroutine are automatically included in the internal calculation of the required point of return to contour.
- As an alternative syntax, REPOS can be programmed instead of G77.

5.2.31 Compensation switching ON "G78", Compensation switching OFF "G79"

Description: Assigns the **length corrections** of the geometry correction functions to individual coordinates.

These are the coordinates

- of the current workpiece coordinate system (WCS) or
- of the tool coordinate system (TCS).

An assignment with respect to the coordinates of the **workpiece coordinate system (WCS)** is possible whenever the tool is aligned perpendicularly to the current working plane and its orientation with regard to the working plane remains constant during machining.

An assignment with respect to the coordinates of the **tool coordinate system (TCS)** is necessary if the orientation of the tool changes during machining, e.g. milling of freeform surfaces. An active axis transformation (e.g. 5-axis or 6-axis transformation) is necessary for this correction. The correction values are settled in the axis transformation.

NC functions with syntax according to DIN 66025 (incl. extensions)

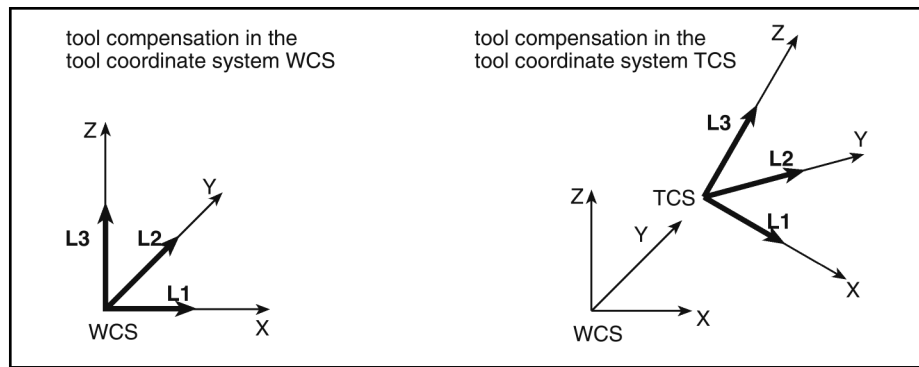


Fig. 5-26: Tool correction in the workpiece coordinate system WCS and TCS



To activate the current correction values for machining, see G47 (chapter 5.2.19 "Tool length correction G47, G48" on page 139).

Syntax:

G78 ({{-}<coordinate1>},{{-}<coordinate2>},{{-}<coordinate3>})	
	Correction switching ON
with	
<Coordinate i>	The WCS coordinate name (logical/physical axis name) to which the correction L_i (with $i = 1, 2, 3$) is to be assigned. If the correction L_i has to have an effect on coordinates in the tool coordinate system (TCS), the coordinate identifiers XTR, YTR and ZTR have to be used. Optional negative sign: Correction is settled in negative direction. L_i corrections to which no coordinates are assigned are not taken into account.
G78(ActPlane)	The corrections L_1 , L_2 and L_3 are assigned to the main, secondary and normal coordinates of the particular active plane.
G79	Correction switching OFF The settings in the machine parameters apply again.

Tab. 5-61: Syntax G78, G79

Examples:

G78(X, ,ZTR)	The L_1 compensation is assigned to the x-axis of the workpiece coordinate system (WCS), the L_3 compensation is assigned to the z-axis of the tool coordinate system (TCS). Both corrections are settled in positive direction.
G78(, , -Y)	The L_3 correction is assigned to the y-axis of the workpiece coordinate system (WCS) and calculated in the negative direction.
G78(YA,YB)	The L_1 correction is assigned to the YA axis and the L_2 correction is assigned to the YB-axis of the workpiece coordinate system (WCS). Both corrections are settled in positive direction.
G79	Corrections are switched off for all default coordinates of the channel.

Special features and restrictions:

- G78 and G79 act modally and deselect each other.
- Coordinates not belonging to the default setting of the channel are never used for G79!

- The coordinate addresses programmed under G78 can refer to work-piece coordinates or tool coordinates (see <coordinate i> above).
- G78/G79 can be programmed with other path conditions, traversing information or auxiliary functions.

5.2.32 Absolute dimension programming "G90", Relative dimension programming "G91", Local absolute dimension programming "AC(...)", Local relative dimension programming "IC(...)"

Description: Specifies whether the control is to interpret unit information for the axes and coordinates as absolute or relative (incremental) values.

- Absolute dimension information refers to the current zero point (in the program coordinate system).
- Relative dimension information refers to the latest approached position.

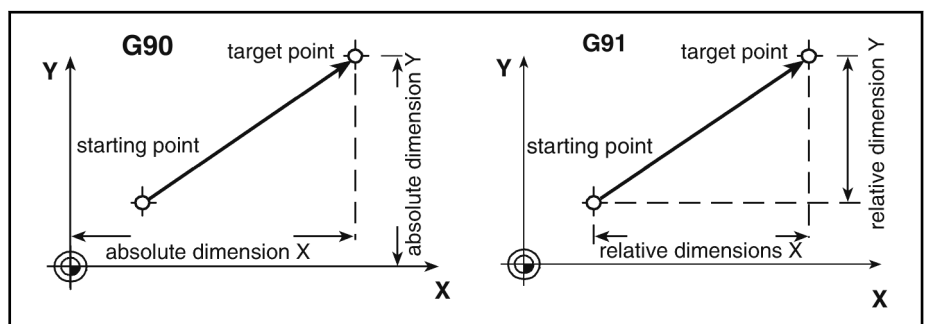


Fig. 5-27: Absolute dimension programming G90 and relative dimension programming G91

The following applies:

- G90 and G91 act modally and deselect each other.
- AC(...) and IC(...) act independently from an active G90/G91 only for the path information of the axes whose position is programmed with AC(...) or IC(...).

Syntax:

G90	Absolute dimension programming
G91	Relative dimension programming
<Axis>=AC (<Value>)	Local absolute dimension programming
<Axis>=IC (<Value>)	Local relative dimension programming
with	
<Axis>	Logical axis address
<Value>	Path information for <axis>

Tab. 5-62: Syntax G90, G91

Example:

N10 G1 G90	Absolute dimension programming ON
N20 X100 Y100	Approach coordinates X100, Y100
N30 G91	Relative dimension programming ON
N40 X100 Y100	Approach coordinates X200, Y200
N50 X=AC(50) Y50	Local absolute dimension programming for x-axis Traversing motion at the machine to X50, Y250

5.2.33 Time programming "G93"

Description: The control interprets F-values in relation to the value of the modal variable ITIM (see [chapter 6.58 "Inverse time programming InvTime, ITIM" on page 279](#)), either as machining time in seconds or as inverse time in 1/min or 1/s for the programmed contour (block duration).

By default, the period is given in seconds (that is ITIM = 0).

The unit for ITIM>0 results from

$$F = \frac{v \left[\frac{mm}{ITIM \times sec} \right]}{s[mm]} = \frac{v}{s} \left[\frac{1}{min} \right]_{ITIM=60} = \frac{v}{s} \left[\frac{1}{SEC} \right]_{ITIM=1}$$

This results in the following units for the individual ITIM values:

ITIM value	Integer; the unit of the F-word changes in relation to the ITIM value in G93 mode
0	F-value in [s]
1	F-value in [1/s]
60	F-value in [1/min]

The following applies:

The functions G93, G94 and G95 act modally and deselect each other.

 The desired switched-on state (G93 (ITIM if necessary), G94 or G95) can be specified in the machine parameters (default: G94, ITIM=0).

Syntax:	<table><tr><td>G93</td><td>Switching to time or inverse time programming</td></tr></table>	G93	Switching to time or inverse time programming
G93	Switching to time or inverse time programming		

Tab. 5-63: Syntax G93

Example:	N5 G93 G1 X30 Y20 F20	Default: ITIM not programmed or ITIM=0
	:	The programmed linear interpolation lasts 20 seconds.
	N4 ITIM60	The programmed linear interpolation takes 60/3 = 20 seconds.
	N5 G93 G1 X30 Y20 F3	
	:	
	N4 ITIM1	The programmed linear interpolation takes 1/5 = 200 ms.
	N5 G93 G1 X30 Y20 F5	
	:	

- Special features and restrictions:
- An F-value programmed with G93 remains in the internal memory when switching to G94 or G95 and becomes active again when switching back to G93.
 - After a startup or control reset, the F-word specified in the machine parameters is active (default: F0).
 - The control internally calculates the required feed based on the path length of the corresponding traversing block and the programmed machining time.

However, the resulting effective feed can be limited by the control according to the programmed contour and the maximum values of the participating axes so that the block duration increases.

- A feed override remains also effective for G93 depending on function OvrEna / OvrDis.

5.2.34 Feed programming (per minute) "G94"

Description: The control interprets F-words (see [chapter 5.4.2 "F-address F" on page 189](#) and [chapter 5.4.4 "Velocity of asynchronous axes FA" on page 190](#)) or Omega words (see [chapter 5.4.5 "Omega address \(feed\) Omega" on page 191](#)) as feed for a programmed contour.

The following applies:

- The functions G93, G94 and G95 act modally and deselect each other.
- *The programmed feed is interpreted as*
 - mm/min if G71 is active
 - inch/min if G70 is active
 - degree/min in connection with rotary axes



Machine parameters can be used to set the scaling of the feed for G70/G71.



The desired switched-on state (G93, G94 or G95) can be specified in the machine parameters (default: G94).

Syntax:

G94	Switches to feed programming

Tab. 5-64: Syntax G94

Example:

N10 G71	Metric programming ON
N20 G1 G94 X20 Y30 F200	Travel with a feed of 200 mm/min
N30 G4 (F40)	Dwell time of 40 s
N40 G70	Inch programming ON
N60 X300 Y400	Feed rate F200 (in mm/min) is effective again
N70 F100	New feed value: 100 inch/min
:	

Special features and restrictions:

- An F- or Omega value programmed during a G94 remains in the internal memory when switching to G93 or G95 and becomes active again when switching back to G94.
- After a startup or control reset, the F-word specified in the machine parameters is active (default: F0).
The Omega value is always automatically initialized to "0" after a startup.
- Due to configured maximum velocities of the participating axes, the control can limit the effective feed.
- The feed is affected by the feed override depending on OvrEna / OvrDis.

5.2.35 Absolute velocity programming "G94(LINS)" with path-proportional velocity adaptation

Description: In the default case (G94 programmed without parameter), it is interpolated the fastest possible to the new feed value if there is a newly programmed F-value. The effective acceleration values are taken into account.

In case of G94(LINS), the command value change results within the block proportionally to the distance covered. The effective acceleration values are always complied with. If possible, the new command velocity is exactly reached at the block end irrespective of the current override.

The following applies:

- The effect of G94(LINS) is modal and it is deselected by G94 [default case (see above) is applied again].
- The programmed feed does not have to be programmed together with G94(LINS) in one line.
- The programmed feed applies modally and absolutely for the following blocks.
- The unit of the programmed feed is inch/min or mm/min according to G70/G71.
- The feed is affected by the feed override depending on OvrEna / Ovr-Dis.

Syntax:

G94 (LINS)	Activate direct feed programming. The following programmed F-values result in a command velocity change proportional to the distance covered.

Tab. 5-65: Syntax G94 (LINS)

Example:

N30 G94 G1 G71 F1000 X250 Y300 :	The block is traversed with a feed value of 1000 mm/min
N50 G94(LINS) F2000 X300 Y200 :	The command path feed is increased proportionally to the distance to 2000 mm/min up to the block end
N60 X350 :	The block is traversed with a feed value of 2000 mm/min
N100 F500 X450	The command path feed is reduced proportionally to the distance to 500 mm/min up to the block end
N110 G94	G94 in default case: Changes in velocity are interpolated the fastest possible complying with the effective acceleration values

Special features and restrictions:

- The existing acceleration and deceleration limits are monitored. The resulting end velocity might thus only be reached in the next block.
- Programming a higher velocity value causes the modal value for the programmed velocity already to increase in the current block. But programming a lower velocity causes the modal value of the programmed velocity to reduce in the **following NC block**. Thus, it is ensured in both cases, that the requested velocity is each reached at the **end of the current NC block** if allowed due to the effective acceleration values.

5.2.36 Absolute velocity programming "G94(LINT)" with time-proportional velocity adaptation

Description: In the default case (G94 programmed without parameter), it is interpolated the fastest possible to the new feed value if there is a newly programmed F-value. The effective acceleration values are taken into account.

In case of G94(LINT), the command value change results within the block proportionally to the time. The effective acceleration values are always complied with. If possible, the new command velocity is exactly reached at the block end irrespective of the current override. Especially the soft velocity transitions result (can be compared to G94(DF...)).

The following applies:

- The effect of G94(LINT) is modal and it is deselected by G94 [default case (see above) is applied again].
- The programmed feed does not have to be programmed together with G94(LINT) in one line.
- The programmed feed applies modally and absolutely for the following blocks.
- The unit of the programmed feed is inch/min or mm/min according to G70/G71.
- The feed is affected by the feed override depending on OvrEna / Ovr-Dis.

Syntax:

G94 (LINT)	Activate direct feed programming. The following programmed F-values result in a command velocity change proportional to the time.

Tab. 5-66: Syntax G94 (LINT)

Example:

N30 G94 G1 G71 F1000 X250 Y300 :	The block is traversed with a feed value of 1000 mm/min
N50 G94(LINT) F2000 X300 Y200 :	The command path feed is increased proportionally to the time to 2000 mm/min up to the block end
N60 X350 :	The block is traversed with a feed value of 2000 mm/min
N100 F500 X450	The command path feed is reduced proportionally to the time to 500 mm/min up to the block end
N110 G94	G94 in default case: Changes in velocity are interpolated the fastest possible complying with the effective acceleration values

Special features and restrictions:

- The existing acceleration and deceleration limits are monitored. The resulting end velocity might thus only be reached in the next block.
- Programming a higher velocity value causes the modal value for the programmed velocity already to increase in the current block. But programming a lower velocity causes the modal value of the programmed velocity to reduce in the **following NC block**. Thus, it is ensured in both cases, that the requested velocity is each reached at the **end of the current NC block** if allowed due to the effective acceleration values.

5.2.37 Incremental velocity programming "G94(...)" with acceleration adaptation

Description: Changes the feed or speed relatively to the last actively effective value. Acceleration in the G94(...) block is adapted to ensure that the resulting velocity or speed is not reached before the end of the block. This leads to a very smooth acceleration behavior.

The following applies:

- The feed resulting at the end of the block applies modally for the following blocks.
- The unit of the incremental feed is inch/min or mm/min according to G70/G71.
- The feed is affected by the feed override depending on OvrEna / OvrDis.

Syntax:

G94 (DF<F-value>)	Program path velocity (Delta Feed) incrementally
G94 (DS1= <S-value>)	Program spindle speed (Delta Speed) incrementally for spindle 1
G94 (DF <F-value> , DS7= <S-value>)	
	Programs the path velocity and the spindle speed incrementally for spindle 7
G94 (DSA2=<Axis>, DS2=<S-value>)	
	Programs the speed of the endless axis <Axis> (Delta Speed Axis) incrementally
with	
<F-value>	Incremental path velocity Positive values increase and negative values decrease the currently effective path velocity.
<S-value>	<i>Based on the axis assignment, the S-value signifies:</i> • Incremental channel spindle speed DSi of the channel spindle with the same index (without axis specification) • Incremental speed DSi of the endless axis DSAi with the same index • Incremental coupling factor DSi of the feed-coupled endless axis DSAi with the same index (see also AFC) Increase positive values. Negative values decrease the currently effective spindle speed. Up to eight S-values (DS1 ... DS8) can be programmed at the same time. Incremental speed programming for endless axes can result in a reversal of direction.
<Axis>	Stands for an asynchronous endless axis in speed mode. Incremental S-values can be assigned at the same time for up to 8 axes (DSA1 ... DSA8).

Tab. 5-67: Syntax of G94 (F value/S value)

Example:

```

N30 G94(DF100) X250 Y300  Increases the path feed linearly by 100 mm/min
:                          up to the end of the block

N50 G94(DF-50) X300 Y200  Reduces the path feed linearly by 50 mm/min up
:                          to the end of the block

N70 G94(DS1=100) X25 Y30  Increases the speed of spindle 1 by 100 rpm up to
:                          the end of the block

                                Increases the path velocity linearly by 100
                                mm/min and the speed of spindle 2 by 150 rpm up
                                to the end of the block

N90 G94(DF100,DS7=150) X2 Y2
:

N100 B2=S(100) B3=S(50)  Starts endless axes B2 and B3 in speed mode

                                Increases the speed to 200 rpm for B2
                                Reduces the speed to 30 rpm for B3
                                up to block end

N110 G94(DSA1=B2,DS1=100,DSA2=B3,DS2=-20) X10
:

N120 B1=S(500)           Starts endless axis B1 with 500 rpm

                                Feed coupling with
                                0.5 B2 revolutions/B1 revolution

N130 AFC(FRA=B1,FA=B2,CF=0.5) B2=S(1)

                                Increase to
                                0.8 B2 rotations/B1 rotation until the end of the
                                block

N140 G94(DSA1=B2,DS1=0.3) X10 F1000

```

Special features and restrictions:

- The existing acceleration and deceleration limits are monitored. The resulting end velocity might thus only be reached in the next block.
- The calculated acceleration is only effective in the G94(...) block. If the block is cancelled, it is decelerated with the calculated acceleration.
- Absolute and incremental feed must not be programmed simultaneously in the same block.
- Programming a positive value for the velocity change causes the modal value for the programmed velocity already to increase in the current block by the respective amount. But programming a negative velocity change causes the modal value of the programmed velocity to reduce by the respective amount in the **following NC block**. The adjusted acceleration ensures in both cases that the resulting actual velocity value is always reached at the **end of the current NC block**.

- If the block is aborted before or position data is missing, the control directly activates the incremental speed values

5.2.38 Absolute velocity programming "G94(...)" with acceleration adaptation

Description: Changes the feed continuously over several blocks from the current value to the target feed. The acceleration up to the target block (new target feed or ramp completion) is adapted to ensure that the resulting velocity is only reached at the target block. This leads to a very smooth acceleration behavior.

Accordingly, the speed specifications of up to eight channel spindles can be changed continuously and path-dependently over several blocks from the current speed to the target speed.

The following applies:

- Although the function G94 is modal, the resulting feed is not modal for the following blocks.
- The unit of the target feed is inch/min or mm/min according to G70/G71.
- The feed is affected by the feed override depending on OvrEna / OvrDis. An override setting of 100% does not provide a continuous ramp in most of the cases.

Syntax:	G94 (TF<F value>)	Programs the target path speed absolutely
	G94 (TS1= <S-value>)	Programs the target spindle speed for spindle 1 absolutely
	G94 (TF <F-value> , TS7 <S-value>)	
		Programs the target path velocity and the target spindle speed for spindle 7
	G94 (TB)	Ramp completion for target feed and target speeds
	with	
	<F-value>	Target path velocity
	<S-value>	Target spindle speed

Tab. 5-68: Syntax of G94 (TF value/TS value)

Example:	N10 G94 F1000 M3 S1=1000	Constant path feed and constant speed.
	:	
	N30 G94 (TF1500) X50 Y30	Increase the path feed linearly from F1000 to F1500 up to block N50.
	:	
	N50 G94 (TF500, TS1=2000) X70 Y50	Decrease the path feed linearly from F1500 to F500 up to block N70. At the same time, the speed of spindle 1 is continuously increased from 1000 rpm to 2000 rpm up to block N70.
	:	
	N70 G94 (TB) X90 Y70	Starting from the target block, the feed is F500 and the speed of spindle 1 is 2000 rpm.
	:	

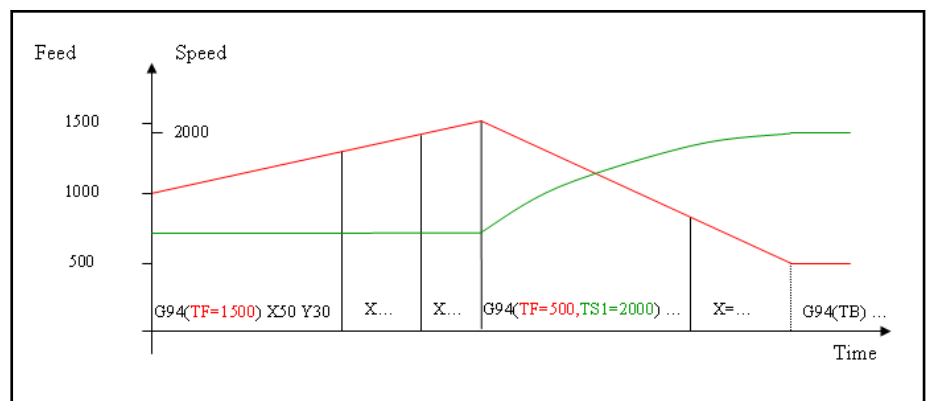


Fig. 5-28: Continuous acceleration and deceleration

Special features and restrictions:

- The existing acceleration and deceleration limits are monitored. The resulting end velocity might thus only be reached in the next block.
- The calculated acceleration is only effective in the G94(...) block. If the block is aborted, it is decelerated with the maximum possible acceleration.
- In a target feed sequence (TF-TB), do not program an incremental feed (DF) nor a modal feed (F word).
- In a target speed sequence (TS-TB), you must neither program an incremental speed (DS) nor a modal spindle speed (S word).
- The number of maximum possible target feed/target speed blocks is limited by the block preview parameters
NCO/LookAh/Ch[k]/MaxBlk/MaxBlkMiscFunc and
NCO/LookAh/Ch[k]/PercSplit/PercMiscFunc.
- Target speed specifications can only be specified for direct spindle programming (G97).

5.2.39 Feed programming (per revolution) "G95"

Description: The control interprets F-words (see [chapter 5.4.2 "F-address F" on page 189](#)) as feed/revolution. This is required in connection with the main spindle.

The following applies:

- The functions G93, G94 and G95 act modally and deselect each other.
- *The programmed feed is interpreted as*
 - mm/rev if G71 is active
 - inch/rev if G70 is active



- Machine parameters can be used to set the scaling of the feed for G70/G71.
- The desired switched-on state (G93, G94 or G95) can be specified in the machine parameters (default: G94).
- For the definition of the main spindle, see MP 7020 00010 or the function "MainSp".

Syntax:

G95	Switching to rotary feed programming

Tab. 5-69: Syntax G95

NC functions with syntax according to DIN 66025 (incl. extensions)

Example:	N05 G71	Metric programming ON
	N10 S200 M4	Spindle speed 200 rpm, counterclockwise
	N20 G1 G95 X20 Z30 F0.2	Traveling with a feed of 0.2 mm/rev
	N30 G4 (S20)	Dwell time of 20 revolutions
	N40 G70	Inch programming ON
	N60 X300 Z40	Feed F0.2 (in mm/rev) is effective again
	N70 F0.1	New feed = 0.1 inch/rev
	:	

- Special features and restrictions:**
- G95 requires a rotating main spindle.
 - The effective feed is influenced by both the spindle and the feed potentiometer.
 - Due to configured maximum velocities of the participating axes, the control can limit the effective feed.
 - An F-value programmed during G95 remains in internal memory when switching to G93 or G94 and becomes active again when switching back to G95.
 - After a startup or control reset, the F-word specified in the machine parameters is active (default: F0).

5.2.40 Incremental velocity programming "G95(...)" with acceleration adjustment

Description: Acceleration within the G95(...) block is adapted to ensure that the resulting velocity or speed is not reached before the end of the block. This leads to a very smooth acceleration behavior.

The following applies:

- The feed resulting at the end of the block applies modally for the following blocks.
- The unit of the incremental feed is inch/rev or mm/rev according to G70/G71.
- The feed is affected by the feed override depending on OvrEna / OvrDis.

Syntax:	G95 (DF<F-value>)	Program path velocity (Delta Feed) incrementally
	G95 (DS1=<S-value>)	Program spindle speed (Delta Speed) incrementally for spindle 1
	G95 (DF<F-value>, DS7=<S-value>)	
		Programs the path velocity and the spindle speed incrementally for spindle 7
	G95 (DSA2=<Axis>, DS2=<S-value>)	
		Programs the speed of an endless axis (Delta Speed Axis) incrementally
	with	

<F-value>	Incremental path velocity in mm/rev or inch/rev. Positive values increase and negative values decrease the currently effective path velocity.
<S-value>	Based on the axis assignment, the S-value signifies: • Incremental speed DSi of the channel spindle with the same index (without axis specification) • Incremental speed DSi of the endless axis DSAi with the same index • Incremental coupling factor DSi of the feed-coupled endless axis DSAi with the same index (see also AFC) Positive values increase and negative values decrease the currently effective speed. Up to eight S-values (DS1 ... DS8) can be programmed at the same time. Incremental speed programming for endless axes can result in a reversal of direction.
<Axis>	Stands for an asynchronous endless axis in speed mode. Incremental S-values can be assigned at the same time for up to 8 axes (DSA1 ... DSA8).

Tab. 5-70: Syntax of the incremental velocity programming with G95

Example:

```

N10 G95 (DF=-0.5) G1 X100 ;Decreasing feed degressively by 0.5 mm/rev to
                           ;up to the end of the block

...

N20 M3 S1=500             ;Starting channel spindle S1 with 500 rpm
N30 G95 (DS1=200) X10     ;Increasing speed S1 to 700 rpm up to block end
F0.2                      ;

...

N70 B2=S(100) B3=S(50)   ;Starting endless axes B2 and B3
                           ;Start speed mode

                           ;Increasing speed to 200 rpm for B2
                           ;Reducing speed to 30 rpm for B3
                           ;up to block end

N80 G95 (DSA1=B2, DS1=100, DSA2=B3, DS2=-20) X10

...

N100 B1=S(500)           ;Starting endless axis B1 with 500 rpm
N110 G1 G95 (FRA=B1) F0.3 ;Feed reference: 0.3 mm/B1 revolution

                           ;Feed coupling: 0.5 B2 revolutions/
                           ;B1 revolution

N120 AFC (FA=B2, CF=0.5) B2=S(1)

```

NC functions with syntax according to DIN 66025 (incl. extensions)

;Increasing F to 0.4 mm/B1 revolution and

;CF to 0.7 B2 revolutions/B1 revolution

; up to block end

N130 G95 (DF=0.1, DSA1=B2, DS1=0.2) X10

- Special features and restrictions:**
- The existing acceleration and deceleration limits are monitored. The resulting end velocity might thus only be reached in the next block.
 - A constant acceleration in the G95(DF...) block is only achieved with an override of 100%.
 - The SPLIT command forms the programmed feed ramp in each block segment.
 - If the block is aborted before time or position data is missing, the control directly activates the incremental speed values.

5.2.41 Feed programming per rotary axis revolution "G95(FRA=<Axis>)"

Description: Establishes the feed reference in [mm or inch/360°] to a system rotary axis or for linear endless axes [mm or inch/modulo interval] and changes the feed in relation to the last actively acting value. The rotary axis reference remains modal.

The following applies:

- After switching on, the main spindle reference is established with G95.
- The function G95 with the programmed reference axis is modal.
- The feed is affected by the feed override depending on OvrEna / OvrDis.

Syntax:

G95 (FRA=<SystemAxisName>)	
	Establishing feed reference to the endless axis
G95 (FRA=MSP)	Establishing feed reference to the main spindle (MainSp)

Tab. 5-71: Syntax of feed programming per rotary axis revolution G95

Example:

N10 Z3=S(1000)	;Z3-speed specification 1000 rpm
N30 G95 (FRA=Z3) F1.0	;Feed reference to the rotary axis Z3
N40 G1 X100	;Feed: 1000 mm/min at 100% OVR
N50 G95 (FRA=MSP)	;Feed with regard to the active master spindle

- Special features and restrictions:**
- An endless axis or the current reference spindle (MainSp) can only be used as reference axis (FRA).
 - The reference axis must not be a channel axis.

5.2.42 Constant cutting velocity "G96", Direct speed programming "G97"

Description: *During turning, the control interprets S words as*

- Target cutting velocity of the tool (G96) or
- Speed of the workpiece rotary axis (G97).

At a given spindle speed, the cutting velocity depends on the distance between the cutting tool and the workpiece rotary axis.

In order to compensate this, the G96 function automatically changes the speed of the rotary axis depending on the distance between the cutting tool and the workpiece rotary axis:

- for G71:

$$S_{act} [\text{min}^{-1}] = \frac{S_{G96} \times 1000}{2 \times \pi \times \left| \text{position (coordinate, reference system) tool compensation} \right| \left[\frac{\text{m} \times 1000}{\text{min} \times \text{mm}} \right]}$$

Fig. 5-29: G71

- For G70:

$$\left[\frac{\text{feet} \times 12}{\text{min} \times \text{inch}} \right]$$

Fig. 5-30: G70

By default, the NC calculates the distance in the machine coordinate system (MCS reference system):

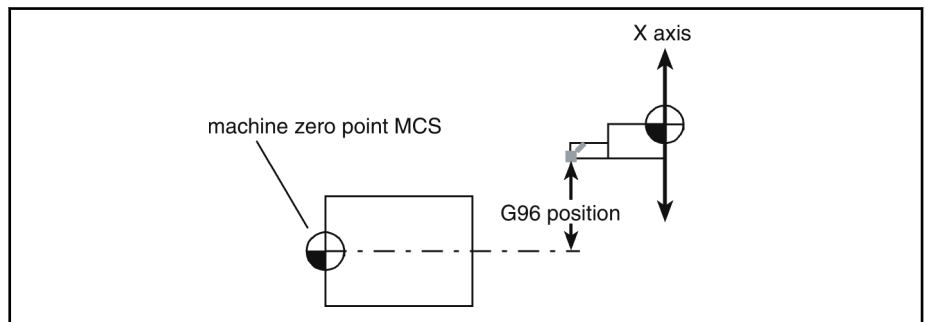


Fig. 5-31: Reference system MCS

If the spindle has to be positioned between different tool holder points, G96 can include corresponding zero offsets spanning a local coordinate system (LCS reference system):

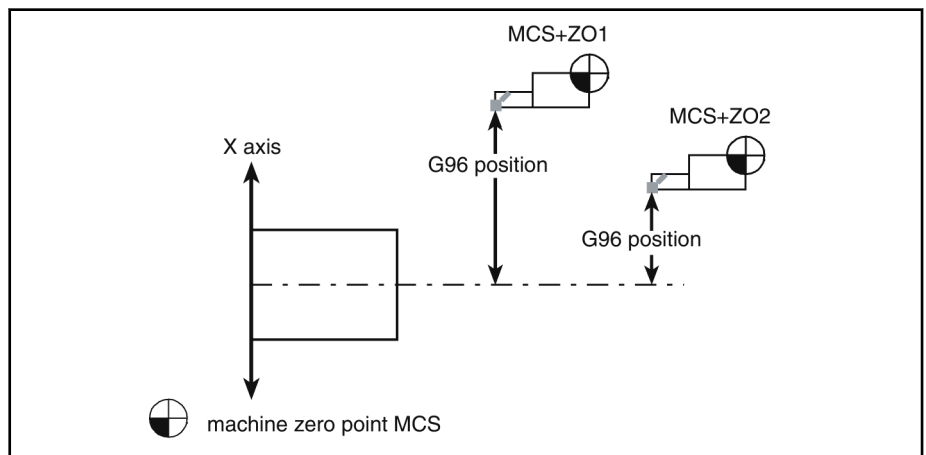


Fig. 5-32: Reference system LCS

In case of tools that are not arranged axis-parallel to the spindle, offsets and rotations (placements) of the workpiece zero point can additionally used (PCS reference system):

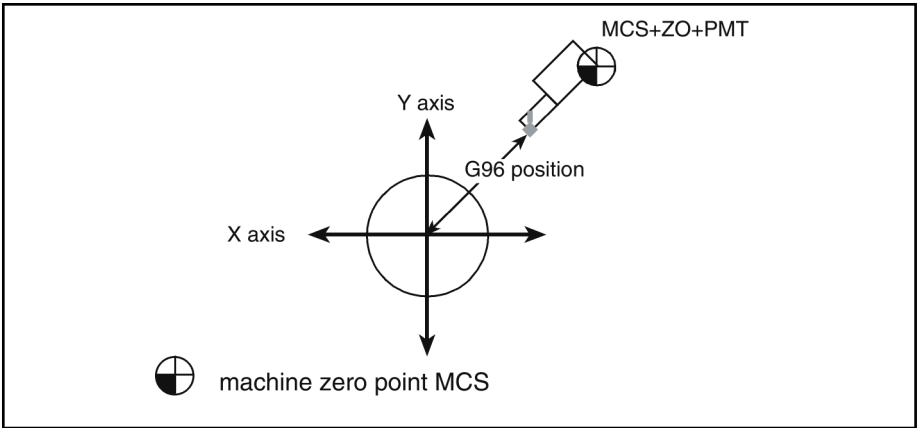


Fig. 5-33: Reference point PCS

The cutting velocity at the workpiece is thus no longer affected by the distance between the cutting tool and the rotary axis of the workpiece.

If this behavior is not required or desired, use G97 instead. The speed of the rotary axis of the workpiece is then exclusively determined by the programmed S-word.

The following applies:

- The reference axis for the distance between the cutting tip and the rotary axis of the workpiece is defined by the machine parameter 7010 00110 (default value).
- The reference axis can be reprogrammed at runtime.
- The reference coordinate system (point of action) for the reference axis is preset using the machine parameter 7010 00120.

The following can be selected:

- PCS:
Position in the program coordinate system
- LCS:
Position in the local machine coordinate system (displaced by the ZO)
- MCS:
Position in machine coordinate system (default)
- The point of action of the reference axis can be changed at runtime.
- The programmed cutting velocity is interpreted as
 - m/min if G71 is active
 - feet/min if G70 is active (1 feet = 12 inch).
- In addition to the spindle override, the functions for speed limitation (SMin, SMax; see chapter 6.117 "Speed limitation SMin, SMN, SMax, SMX" on page 363) are also effective.

Syntax:	G96 { ({ <ReferenceAxis> { , <PointOfAction> } }) } ... S<i>=<V>
	S words of spindles that are programmed in the G96 block are interpreted as command cutting velocity at the tool during turning. All other spindles drop back to direct speed programming!

G96	The latest programmed settings are applied. If no settings were made, the default values apply.
G96 () ...	The default values for the reference axis and the point of action are activated again.
G97 ... S<i>= <Speed>	
	Only the spindles whose S words are programmed in the G97 block will revert to direct speed programming. For these spindles, programmed S words are interpreted as the speed of the rotary axis of the workpiece.
G97	All spindles drop back to direct speed programming.
with	
<Reference axis>	Physical or logical name of the reference axis. Programming remains until a new value is programmed or the default value is activated.
<Points of action>	Coordinate system of the reference axis: PCS, LCS or MCS Programming remains until a new value is programmed or the default value is activated.
<i>	Spindle index. 1: first spindle; 2: second spindle, etc.
<V>	Cutting velocity of the relevant spindle in m/min or feet/min.
<Speed>	Speed of the relevant spindle in rpm.

Tab. 5-72: Syntax G96, G97

Example: (Two spindles are configured in the system):

N10 G71 S1=500 S2=500	Defines command revolutions of both spindles for the first time in the program.
:	
N60 G96 S1=50	Activates G96 for the first spindle with a cutting velocity of 50 m/min.
:	
:	2. Spindle still running on speed programming
:	G96 active
N100 S1=30	Reduces the cutting velocity of the first spindle to 30 m/min.
:	
:	2. spindle still running on speed programming
:	G96 active

NC functions with syntax according to DIN 66025 (incl. extensions)

N140 G96 (Y, LCS) S2=100	The reference axis is Y and the point of action is the local machine coordinate system LCS.
:	
:	Activates a cutting velocity of 100 m/min for the second spindle
:	
:	1. spindle falls back to speed programming
:	G96 active
:	
N180 G97	All spindles revert to direct speed programming, if this has not already happened.
:	
:	G97 active

- Special features and restrictions:**
- Several spindles can be switched together in the same block by programming the corresponding S words in sequence (for example: G96 S1=100 S2=1000).
 - To change the cutting velocity of a spindle that has already been switched to G96 for the remainder of program execution, just reprogram the S word of the appropriate spindle.
 - The currently effective cutting velocity of a spindle remains in internal memory after switching to G97. It is activated again as soon as the respective spindle is "switched back" to G96.
 - When switching from G96 to G97, the control uses the current speed as the new target speed value for all spindles whose S word is not programmed in the G97 block.
 - A desired gear shift has to be executed before G96 is activated.
 - When G96 is active, the function "automatic gear stage switching" does not switch any gear stage.
 - If G96 is active, the control checks whether the reference axis "affects" the active point of action. If this is not the case (e.g. if the axis exits the channel), a runtime error is reported.

5.2.43 3D Tool radius correction "G140, G141, G142"

Description: The function moves a rotation-symmetric tool to the left or right of the programmed path, guiding its point of action to the programmed path in active feed.

In case of pure orientation motions, the current position of the point of action remains fix and the TCP moves.

At outer corners, the path is closed using an automatically generated intermediate block (arc). An intersection is calculated at inner corners.

The control considers for the correction:

- The tool radius "r" for the active tool edge (ED, DCT) and the active D-correction (D, DcTSel, DCS, DCT)
- An optional plunging depth "d" (see syntax for `INSDEP`)
- an optional allowance for the path that acts additively to the tool radius (see syntax for `COFFS`)
- An optional change rate of the allowance (see syntax for `DCOFFS`)

The following figure shows the principle:

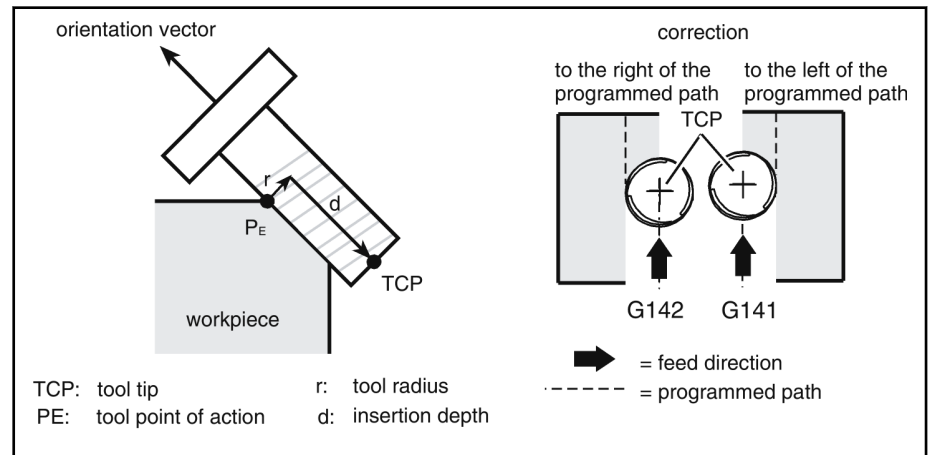


Fig. 5-34: 3D tool radius correction

A change in the plunging depth (**INSDEP**), the allowance (**COFFS**), the tool length or the milling tool radius performed immediately and smoothly (using splines) when G141/G142 is active.

The change rate (**DCOFFS**) of the allowance is only effective if G141/G142 is active, but not in the correction build-up blocks. Changes in the change rate are applied immediately. During the traversing block, the allowance changes linearly from its starting value at the beginning of the block to the end value at the block end given by $\langle \text{Start value} \rangle + \langle \text{Length of programmed path} \rangle * \langle \text{Change rate of allowance} \rangle$. In the next block, the end value becomes the starting value unless it is overwritten by programming **COFFS** in the next block.



For detailed information on this function, refer to the documentation "MTX Functional Description".



Correction values can be immediately activated or deactivated without programming a separate traversing motion. This can result in damages to the workpiece or the tool.

Thus, observe all information in this chapter and in the documentation "MTX Functional Description".

NC functions with syntax according to DIN 66025 (incl. extensions)

Syntax:	G140 3D tool radius correction OFF (switched-on state after control startup) The correction (radius, plunging depth) is reduced in an interpolating manner, if necessary with a programmed traversing motion (using 3rd-degree splines). Effective tool length correction is not influenced by G140.
	G141 3D tool radius correction to the left of the path ON (as seen in processing direction and with positive correction values). Traversing motions or orientation motions can be programmed in the same block. The radius and the plunging depth are retracted if one of the active spatial coordinates is programmed in the same block. <i>Otherwise, the control builds up the correction</i> - only with the next traversing block if G140 was previously active - immediately (in active feed) if G141/G142 was active before.
	G142 3D tool radius correction to the right of the path ON. Otherwise identical to G141
	INSDEP<ET> Defines the plunging depth <ET> Acts modally Switched-on state after control startup: 0
	COFFS<KA> Defines the allowance values <CA> Acts modally and additive to the tool radius "r" Switched-on state after control startup: 0
	DCOFFS<DA> defines the change rate <DA> of the allowance. Acts modally and changes the allowance continuously along the path Switched-on state after control startup: 0

Tab. 5-73: Syntax G140 - G142

Special features and restrictions:

- G140, G141 and G142 act modally and deselect each other
- Start and exit correction build-up on any contour (G2, G6, etc.)
- *The following is **not allowed** for an active G141 or G142:*
 - G17 ... G20 (plane switching)
 - G70, G71 (inch/metric switching)
 - G63 (tapping without compensating chuck)
 - G74 (approach reference point coordinates)
 - G75 (travel to touch probe)
 - MOB (measuring on a block)
 - G76 (approach to fixed machine axis position)
 - SetPos (set program position)
 - G54.x ... G59.x (Zero offsets)
 - G154.x ... G159.x (inclined plane)

- G40...G42 (Radius correction)
- coord() (switching ON / OFF of axis transformation)
- Transfers of axes being part of the current spatial coordinates via axis/coordinate transformation.
- If G141 or G142 is active, a collision check is not carried out.
- If the program ends without M30, G140 has to be active at this time. The current plunging depth, allowance and its change rate are retained.
- G140 is automatically activated after a control reset, a system reset or M30. The current plunging depth, allowance values and its change rate are set to 0.

5.2.44 Placement: Inclined plane "G151 - G159.5"

Description: The placement "inclined plane" can shift **and** orient the workpiece coordinate system anywhere in space. The inclined plane affects the coordinates with the meanings "X", "Y" and "Z" in the corresponding channel.



The axis names "X", "Y" and "Z" used in the following refer to the axes with the **meaning** X, Y and Z.

Since there are three degrees of freedom for orientation, every orientation can be represented by three consecutive basic rotations:

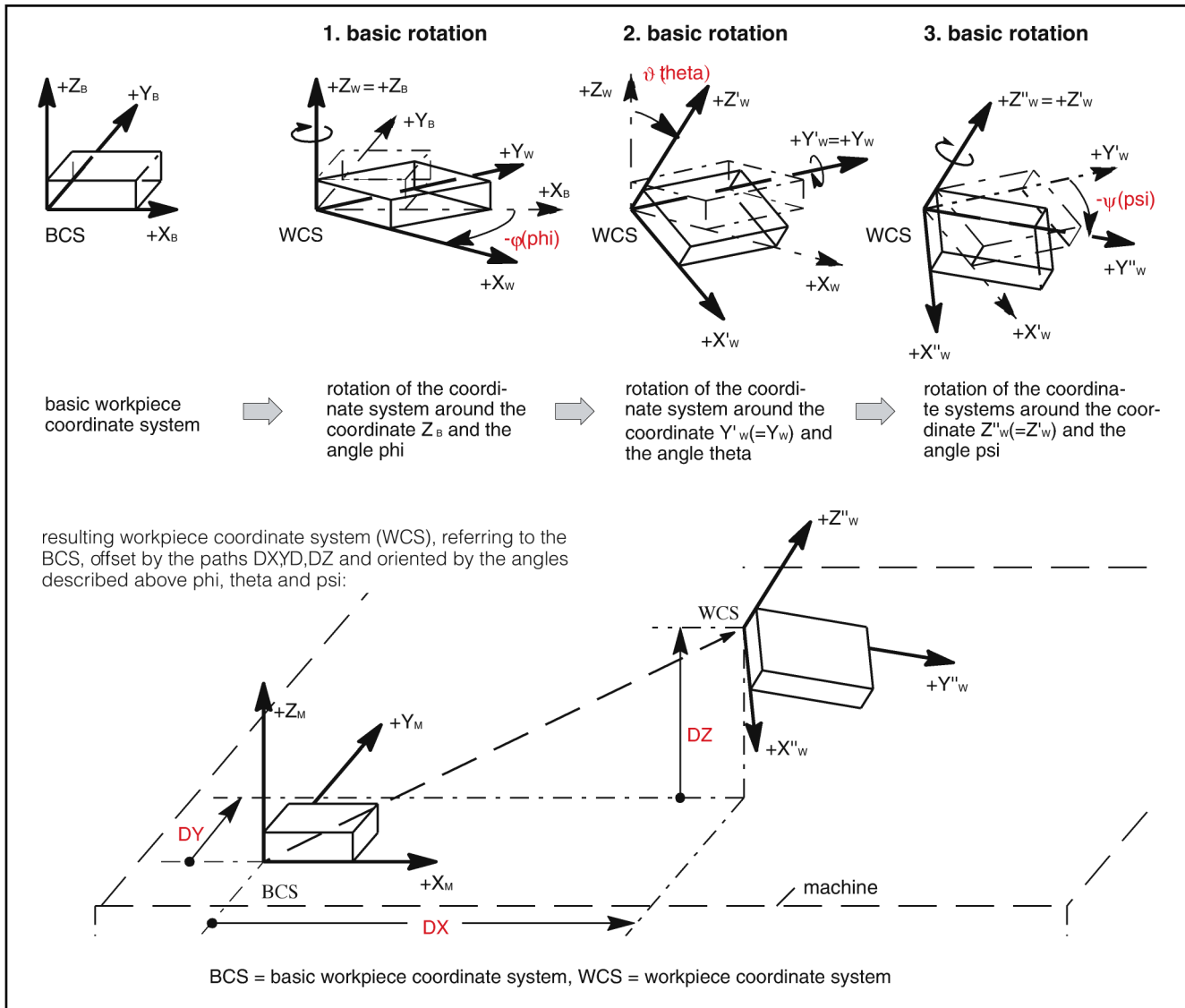


Fig. 5-35: "Inclined plane" placement

The resulting functionality of the "inclined plane" is affected by

- five banks (1 to 5); each acts additively with the other.
Thus, up to five workpiece coordinate systems based on each other can be created.
- Eight alternative sets per bank.
Each set contains all required offset and orientation data for a workpiece coordinate system.
Within a bank, only one single set can be active at a specified time (i.e. the sets of a bank deselect one another).

The inclined plane is additive to the Placement "Workpiece position correction" and it is thus in the "sequence of settlement" behind the workpiece position correction:

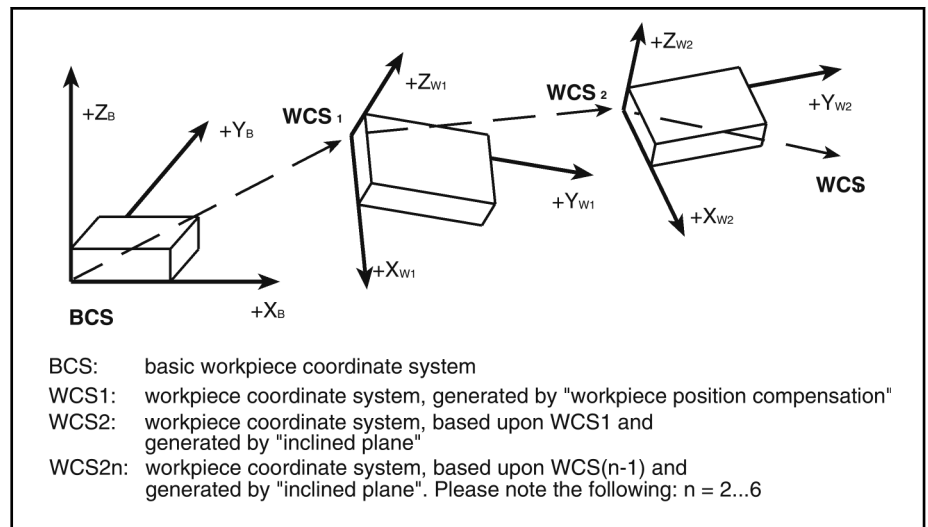


Fig. 5-36: Placement "Workpiece position correction"

Syntax: A "set" is referred to as all data required for the offset and orientation of a workpiece coordinate system.

Sets can be operated as follows:

- Implicitly, the sets can be enabled according to the current tool orientation in the part program (G151 - G151.5)
- The sets can be programmed as function parameters in the part program (G152 - G152.5)
- The user can also specify them in a placement table (G154 - G159.5) (refer to [chapter 6.85 "Activating placement tables PmTsel, PMS" on page 317](#))

A placement table can contain up to 30 sets (five banks, each with six sets).

Implicit activation in the part program (G151 - G151.5):

If the inclined plane is perpendicular to the tool, the position and orientation of the new workpiece coordinate system is defined directly via the TCP and the tool orientation.

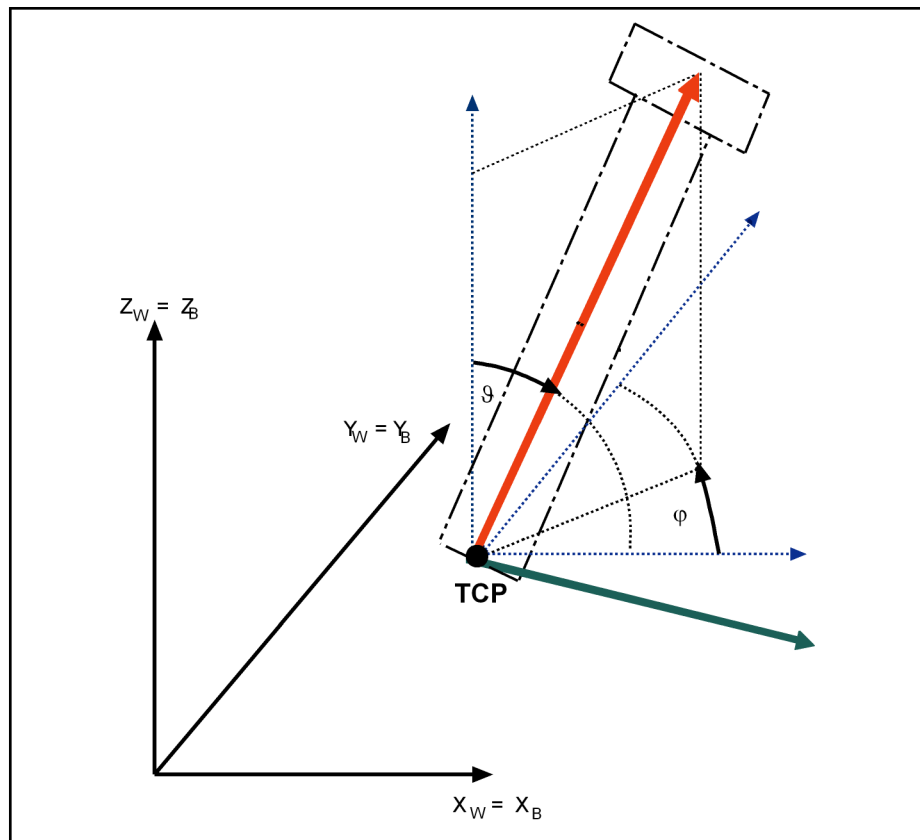


Fig. 5-37: *Inclined plane vertical to tool*

After the inclined plane perpendicular to the tool has been activated, the origin of the new workpiece coordinate system is positioned in the TCP of the tool.

The new Z_w axis is arranged along the longitudinal axis of the tool.

The new y_w coordinate is arranged in parallel to the plane that was defined by the previous X_B and y_B basic coordinates.

As per mathematical definition, the new X_w axis results from the vector product of the two remaining coordinates $X_w = Y_w \times Z_w$.

The figure below shows the configuration after the inclined plane perpendicular to the tool has been activated

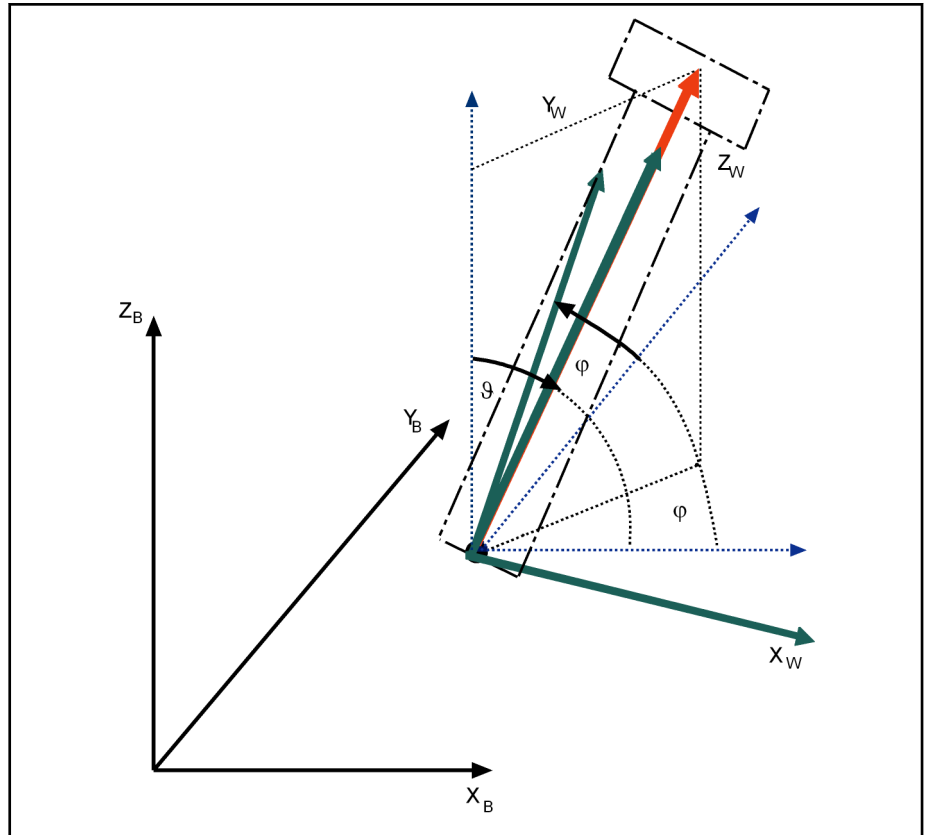


Fig. 5-38: Configuring after activation of the inclined plane perpendicular to the tool

You can also program the angle of rotation psi to rotate the orientation of the coordinate system about the new Z_W coordinate.

<code>G151.<Bank>{(<Angle3>)}</code>	Inclined plane of the desired bank ON The offset values result from the current position of the tool top (TCP) in the current program coordinate system (PCS). The angles of rotation result directly from the current tool orientation. If the active axis transformation is not orientable, the values of the Euler angles are set to 0 each.
Short form: G151	corresponds to G151.1
<code>G153.<Bank></code>	Inclined plane of the programmed bank OFF
G153	Inclined plane OFF (switch off all banks).
with	
<Bank>	Number of the desired bank Value range: 1 to 5
<Angle3>	Angle of rotation psi about the new Z_W coordinate Value range: $0 \leq \text{<Angle3>} \leq 360$ degrees

Tab. 5-74: Syntax G151, G153

Programming directly in the program line as parameter (G152 - G152.5):

NC functions with syntax according to DIN 66025 (incl. extensions)

In case of direct programming of the inclined plane with G152 - G152.5, the position and orientation of the new workpiece coordinate system is directly programmed by the position parameters of the function.

G152. <Bank> ({<X _W Offset>}{, {<Y _W Offset>}{, {<Z _W Offset>}{, {<Angle1>}{, {<Angle2>}{, {<Angle3>}}}}}}))	
	The inclined plane of the desired bank ON, with programmed offset/orientation data.
Short form: G152	corresponds to G152.1
G153. <Bank>	Inclined plane of the programmed bank OFF
G153	Inclined plane OFF (switch off all banks).
with	
<Bank>	Number of the desired bank Value range: 1 to 5
<X _W offset>	Offset value in the direction of the main coordinate
<Y _W offset>	Offset value in the direction of the secondary coordinate
<Z _W offset>	Offset value in direction of the normal coordinate
<Angle1>	Rotating angle along the z-coordinate Range of values $0 \leq \text{<Angle1>} < 360$ degrees
<Angle2>	Angle of rotation along the new Y'-coordinate Range of values $0 \leq \text{<Angle2>} < 180$ degrees
<Angle3>	Rotating angle along the new Z''-coordinate Range of values $0 \leq \text{<Angle3>} < 360$ degrees

Tab. 5-75: Syntax G152, G153

Programming together with placement tables:

Activate the desired placement table (see [chapter 6.85 "Activating placement tables PmTSEL, PMS" on page 317](#)).

Program the required function:

G154. <Bank>	Inclined plane of the programmed Bank Set 1 ON
G155. <Bank>	Inclined plane of the programmed Bank Set 2 ON
G156. <Bank>	Inclined plane of the programmed Bank Set 3 ON
G157. <Bank>	Inclined plane of the programmed Bank Set 4 ON
G158. <Bank>	Inclined plane of the programmed Bank Set 5 ON
G159. <Bank>	Inclined plane of the programmed Bank Set 6 ON
Short form: G152 - G159	corresponds to G154.1 - G159.1
G153. <Bank>	Inclined plane of the programmed bank OFF
G153	Inclined plane OFF (switch off all banks).

with	
<Bank>	Number of the desired bank Value range: 1 to 5

Tab. 5-76: Syntax of G153 - G159

Example: Boring of tapered holes with G151

```

:
N05 Coord(1)
N10 G1 x0 y0 z100 phi90 theta1 F2000
N15 G151
N20 P STC_G81(SL90, DT80, RL100) F500
N25 G153
N30 phi90 theta10
N35 G151
N40 P STC_G81(SL90, DT80, RL100) F500
N45 G153
N50 theta45
N55 G151
N60 P STC_G81(SL90, DT80, RL100) F500
N65 G153
N70 phi0
N75 G151
N80 P STC_G81(SL90, DT80, RL100) F500
N85 G153
N90 G2 I100 J0 ROTAX(0,0,1) O(-360)
N95 G151
N100 P STC_G81(SL90, DT80, RL100)
F500
N105 G153
N110 G90G1
N120 Coord(0)
M30

```

Axis transformation 1 is activated in the program. Subsequently, the tool is positioned and oriented at various points in space. For each of these positions, a placement, perpendicular to the tool is activated and subsequently, a boring is executed using the MTX standard boring cycle. Boring depth (DT), Safety plane (SL) and retract height (RL) are transmitted to the subroutine as parameter.

NC functions with syntax according to DIN 66025 (incl. extensions)

Example: Direct placement programming with G152

```

:
N40 G152.1(100,0,0,90)    Program placement for Bank 1
:                        No traversing motion
:                        The zero point is offset by 100 mm in the X direc-
:                        tion and the WCS is rotated by 90 degrees about
:                        the z-coordinate.
:
N180 G153                Switch off all banks (switch off inclined plane com-
                        pletely)

```

Example: Activate placement using the table: G154 - G159.5

```

N40 PMS(Tab1)            Activate placement table "Tab1"
N50 G154.1               Switch on bank 1 with set 1
:                        No traversing motion
N90 G154.2 X1 Y2 Z3      Switch on bank 2 with set 1
:                        (is additive to bank 1)
:                        In the resulting WCS, approach the programmed posi-
:                        tion P(1,2,3).
N120 G153.2              Switch off bank 2 (bank 1 remains effective)
:
N180 G153                Switch off all banks (switch off inclined plane com-
                        pletely)

```

- Special features and restrictions:**
- If an axis transformation is active, the inclined plane must not be programmed in the same block with the traversing motions.
 - The switched-on state and the behavior in case of a control reset are configured in the machine parameters 7060 00010 and 7060 00020.
 - All functions are modal within each bank (1 - 5) and deselect each other.
 - Activating and deactivating the inclined plane interrupts the block look-ahead. Thus, it must not be programmed when the milling cutter path correction is active (G41/G42, see [chapter 5.2.16 "Milling cutter path correction G40, G41, G42" on page 135](#)).
 - The function of the "inclined plane perpendicular to the tool" is called without this call being checked for plausibility. Thus, the user is responsible to call the function from the correct placement bank.
 - To activate an inclined plane placement in a channel, at least two axes with an axis significance ("X", "Y", "Z") have to be available in a channel.

5.3 M-codes

5.3.1 Interrupt program (program stop) "M0, M00"

- Description:**
- Interrupts the NC program,
 - Stops machine motions after block execution and
 - outputs the channel-based interface signal "program stop M0".

The current channel state switches to "NC ready".

Program execution is continued by a new "NC start".



See M1/M01 to interrupt programs depending on the channel-based interface signal "optional stop".

Syntax: M0 or M00

Special features and restrictions: "Program stop" can be programmed with other NC functions in the same block. Only after all other programmed functions have been executed, the "program stop" is applied.

5.3.2 Conditional program abort (cond. program halt) "M1, M01"

Description:

- Aborts the NC program and
- stops machine motions after executing the block if the channel-based interface signal "optional stop" is active.

The current channel state switches to "NC ready".

Program execution is continued by a new "NC start".

Syntax: M1 or M01

Tab. 5-77: M1 syntax

Special features and restrictions: The "conditional program stop" can be programmed with other NC functions in the same block. Only after all other programmed functions have been executed, the "conditional program stop" is applied.

5.3.3 Exiting the program (program end) "M2, M02, M30"

Description: Completes a program.

The following applies for a program that is a subroutine:

- the NC issues the auxiliary function (M2, M02 or M30)
- The NC returns to the calling program
- The NC processes the calling program



Modal states that have been changed in the subroutine are not reset!

The following applies for a program that is a main program:

- The NC sets the channel-based interface signal "program end M30"
- The NC cancels the channel-based interface signal "program running"
- The NC deselects an active "automatic gear selection" (but the current gear stage remains selected)
- The NC switches to the "direct speed programming" (G97)
- The NC activates all states defined in machine parameter 7060 00020 "default states" for an "M30" event
- The NC returns to the start of the main program
- The NC waits for the next "NC start"

Program execution is restarted upon "NC start".

Syntax: M2 or M02 or M30

Tab. 5-78: M2 syntax

Special features and restrictions: Program the function in a separate program line.



Undefined switch-on states can cause damage!

If certain states or functions are required after exiting a main program, it is **absolutely necessary** that the parameters of the init string of the "M30" event are set correctly in MP 7060 00020!

The init string has to contain all functions setting the NC to the required/desired state after the end of a main program. In this context, remember that modally acting functions remain active even after the end of the program!



For detailed information about the switch-on states, see the documentation "MTX Functional Description".

5.3.4 Spindle clockwise rotation "M3, M103, M203", Spindle clockwise rotation and coolant ON "M13, M113, M213"

Description:



The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

To determine whether the functions described actually apply to your machine, contact your system administrator!

- Starts clockwise spindle rotation, from a "tool-to-workpiece" perspective.
- Cancels a position control activated by "orient/position spindle".
- Reserves the corresponding spindle(s) for the current channel.



The spindle motion starts only if an S- or SSPG-word greater than 0 was programmed for the relevant spindle/spindle group in a previous or in the same block (see [chapter 5.4.6 "Programming spindle speed S, SSPG" on page 191](#)).

Syntax:

M3	Applies to the first spindle group
M103	Applies to the first spindle
M203	acts on the second spindle
M13	acts on 1. spindle group In addition, coolant ON
M113	Applies to the first spindle In addition, coolant ON
M213	acts on the second spindle In addition, coolant ON

Tab. 5-79: Syntax of M3, M13, M103, M113, M203 and M213

Special features and restrictions:

- The respective function automatically reserves the corresponding spindle(s) for the current channel. Use of reserved spindle(s) by another channel is permitted only if the function "spindle stop" (see [chapter 5.3.6 "Spindle stop M5, M105, M205" on page 184](#)) or "SpAdmin" (SPA, see [chapter 6.120 "Enabling/transferring reserved spindle SpAdmin, SPA" on page 366](#)) is used for the corresponding spindle(s).

- The respective function is active until a new motion state is programmed for the same spindle(s) (e.g. another direction of rotation, with/without coolant, "spindle stop" or "orient spindle").
- Competing jobs between single spindles and spindle groups programmed in the same block lead to a runtime error.
(Example M3 and M104 are not permitted in the same block.)
- After the gear shifting, the previously programmed direction of rotation of a spindle is automatically restored.

5.3.5 Spindle counterclockwise rotation "M4, M104, M204", Spindle counterclockwise rotation and coolant ON "M14, M114, M214"

Description:



The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

To determine whether the functions described actually apply to your machine, contact your system administrator!

- Starts counterclockwise spindle rotation, from a "tool-to-workpiece" perspective.
- Cancels a position control activated by "align/position spindle".
- Reserves the corresponding spindle(s) for the current channel.



The spindle motion starts only if an S- or SSPG-word greater than 0 was programmed for the relevant spindle/spindle group in a previous or in the same block (see [chapter 5.4.6 "Programming spindle speed S, SSPG" on page 191](#)).

Syntax:

M4	Applies to the first spindle group
M104	Applies to the first spindle
M204	acts on the second spindle
M14	acts on the first spindle group In addition, coolant ON
M114	Applies to the first spindle In addition, coolant ON
M214	acts on the second spindle In addition, coolant ON

Tab. 5-80: Syntax of M4, M14, M104, M114, M204 and M214

Special features and restrictions:

- The respective function automatically reserves the corresponding spindle(s) for the current channel. Use of reserved spindle(s) by another channel is permitted only if the function "spindle stop" (see [chapter 5.3.6 "Spindle stop M5, M105, M205" on page 184](#)) or "SpAdmin" (SPA, see [chapter 6.120 "Enabling/transferring reserved spindle SpAdmin, SPA" on page 366](#)) is used for the corresponding spindle(s).
- The respective function is active until a new motion state is programmed for the same spindle(s) (e.g. another direction of rotation, with/without coolant, "spindle stop" or "orient spindle").

- Competing jobs between single spindles and spindle groups programmed in the same block lead to a runtime error.
(Example M3 and M104 are not permitted in the same block.)
- After the gear shifting, the previously programmed direction of rotation of a spindle is automatically restored.

5.3.6 Spindle stop "M5, M105, M205"

Description:



The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

If you cannot ensure that the functions described actually apply to your machine, contact your system administrator.

Syntax:

- Stops (a) spindle(s)
- Cancels any possibly active reservation of the given spindle(s) by the active channel.
- Cancels a position control activated by "orient/position spindle".

M5	Acts on spindle group 1
M105	Applies to the first spindle
M205	acts on the second spindle

Tab. 5-81: Syntax of M5, M105 and M205

Special features and restrictions:

- The respective function is active until a new motion state is programmed for the same spindle(s) (e.g. "spindle, counter-clockwise/clockwise rotation" or "Orient spindle").
- Competing jobs between single spindles and spindle groups programmed in the same block lead to a runtime error.
(Example M3 and M105 are not permitted in the same block.)

5.3.7 Orienting the spindle/positioning the spindle "M19, M119, M219"

Description:



The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

If you cannot ensure that the functions described actually apply to your machine, contact your system administrator.

The spindle is reserved for the current channel and positioned in a definable position using position control.

Positioning occurs

- in case of a standstill:
as defined in the parameter S-0-0154 of the drive.
- in case of an active motion of rotation:
while the direction of rotation is maintained.

Syntax:

M19	Spindle group 1: Position all spindles involved to their reference angle (= 0 degree)
M119	Position the first spindle to its reference angle (= 0 degree)
M219	Position the second spindle to its reference angle (= 0 degree)
M19S<Angle>	Spindle group 1: Position all involved spindles to <angle>
M119 S1=<Angle>	Position first spindle to <Angle>
M219 S2=<Angle>	Position second spindle to <Angle>
<Angle>	Desired absolute spindle position in degrees Value range: $0^\circ \leq \text{Spindle position} < 360^\circ$ If a different value is programmed for the position, it is automatically converted to the specified interval. If the spindle is already in the specified position, no motion is executed.

*Tab. 5-82: Syntax of M19, M119 and M219***Examples:**

N60 M19 :	All spindles of the first spindle group position to 0 degree.
N70 M219 :	Only the second spindle positions to its reference angle.
N80 M19 S180 :	All spindles of the first spindle group position to 180 degrees.
N90 M119 S1=370 :	1: The second spindle positions to 10 degrees
N95 M19 S1=10 S2=20 :	If the first and second spindle are assigned to the first spindle group: First spindle positions to 10 2: The second spindle positions to 20 degrees All other spindles of the first spindle group position to their reference angle.

Special features and restrictions:

- Spindle positioning is only supported for Sercos spindles.
- The respective function must not be programmed in the same block as a competing spindle function ("spindle, counterclockwise/clockwise rotation", "spindle stop").
- If the S word is required, it must be programmed in the same block.
- After the positioning, the spindle remains in position control. The position control is only automatically cancelled via "spindle, counterclockwise/clockwise rotation" or "spindle stop".
- Competing jobs between single spindles and spindle groups programmed in the same block lead to a runtime error.
(Example M3 and M119 are not permitted in the same block.)

- Set the interface signal "qSp_SValueSD" and the position is assigned via the system date "SysSpCmdData[n]/OriPos". The programmed specification is ignored.

5.3.8 Automatic gear selection "M40, M140, M240"

Description:



The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

If you cannot ensure that the functions described actually apply to your machine, contact your system administrator.

The control automatically selects the suitable gear stage from the ones available with regard to the active speed.

If the speed ranges of individual levels overlap, the control always selects the next lower level (with the higher motor speed).

Syntax:

M40	Automatic gear stage selection for spindle group 1 ON.
M140	Enable automatic gear stage selection for first spindle.
M240	Enable automatic gear stage selection for second spindle.

Tab. 5-83: Syntax of M40, M140 and M240

Special features and restrictions:

- Programming a speed of "0" does not cause a change in the current gear stage.
 - M40, M41-M44, M48, M140, M141-M144 and M148 are modal and deselect each other.
 - M240, M241-M244 and M248 are modal and deselect each other.
 - M30 deselects an active automatic gear selection (but the current gear stage remains selected).
 - Gear switching is only supported for Sercos spindles.
- For spindles without Sercos interface, only the auxiliary function is forwarded to the PLC.

5.3.9 Manual gear selection "M41...44, M141...144, M241...244"

Description:



The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

If you cannot ensure that the functions described actually apply to your machine, contact your system administrator.

Selects the corresponding gear stage.

If a speed is programmed outside the corresponding gear speed range, the control outputs the minimum or maximum speed of the relevant gear stage.

Syntax:

M4<Stage>	Activates the gear stage <Stage> for spindle group 1
M14<Stage>	Activates the gear stage for the first spindle.
M24<Range>	Activates the gear stage <stage> for the second spindle
with	
<Stage>	Input range: 1 ... 4

Tab. 5-84: Syntax of M4, M14 and M24

Special features and restrictions:

- M40, M41-M44, M48, M140, M141-M144 and M148 are modal and deselect each other.
- M240, M241-M244 and M248 are modal and deselect each other.
- Gear switching is only supported for Sercos spindles.
For spindles without Sercos interface, only the auxiliary function is forwarded to the PLC.

5.3.10 Gear idle selection "M48, M148, M248"

Description:

The described syntax applies only to the default setting of the machine parameters 1040 001xx and 1040 002xx. The syntax can be freely configured and thus be different on your machine!

Therefore, the documentation provided by the MTB always has priority.

If you cannot ensure that the functions described actually apply to your machine, contact your system administrator.

Disengages the gear stage.

The gear is subsequently in idle mode.

Syntax:

M48	Disengages gear stage for spindle group 1
M148	Disengage gear stage for first spindle
M248	Disengage gear stage for second spindle

Tab. 5-85: Syntax of M48, M148 and M248

Special features and restrictions:

- M40, M41-M44, M48, M140, M141-M144 and M148 are modal and deselect each other.
- M240, M241-M244 and M248 are modal and deselect each other.
- Gear switching is only supported for Sercos spindles.
For spindles without Sercos interface, only the auxiliary function is forwarded to the PLC.

5.4 Feed and speed programming

5.4.1 Speed setting or modulo speed setting "<Axis>=S<Value>"

Description: An asynchronous endless axis is accelerated to the specified velocity (revolutions per minute or modulo intervals per minute). The velocity specification

remains active until it is replaced by a new specification or canceled by a positioning motion. The velocity sign determines the direction of rotation or the direction of motion. When the required velocity is reached, the interface signal iAx_ProgSpReach = 1 is set.

The flying positioning motion from the velocity mode stops the axis at the next possible position specified in the direction of motion. The positioning logic of the axis and the positioning attributes (DC, ACN etc.) are not applied. A feed specification with FA=<Feed> can accelerate the positioning, but is not able to decelerate it.

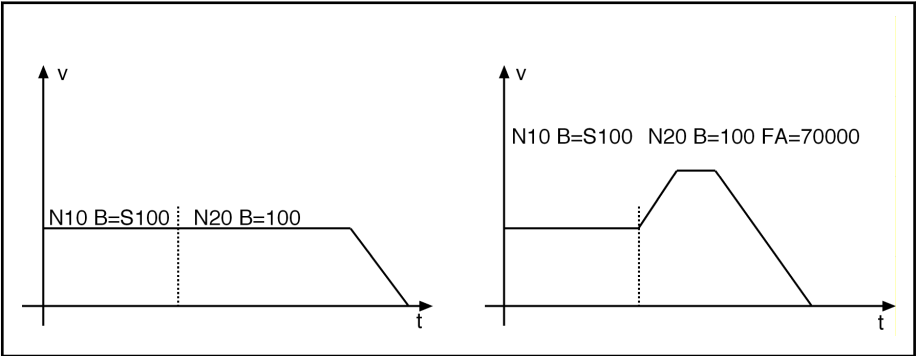


Fig. 5-39: Flying positioning

Syntax:

<AxisAddress>=S<Value><Value>	
with	
<Value>	Velocity in rpm or modulo intervals/min.

Tab. 5-86: Syntax S attribute

Example:	N100 B=S1000	Accelerate B-axis to 1000 rpm and move constantly
	N200 B=-600	Decelerate B-axis and accelerate in opposite direction to 600 rpm.
	N300 B=100	Positioning of the B-axis to the absolute position 100° with a speed with an increased positioning speed (corresponds to 1000 rpm)
	N300 B=100 FA=360000	
	N400 W=S100	Accelerating linear W-axis with a modulo range of 20 mm to 100 modulo ranges/MIN (200 mm/min) and moving them constantly.

- Special features and restrictions:
- The resulting velocity depends on the axis-specific override. The maximum velocity is limited by the maximum admissible axis velocity (AX/Dr[i]/Vel/MaxVel/) and by half the modulo range (S-0-0103) per NC cycle time (SCS/ScsIf[1..2]/ScsCycTime).
 - The programming attributes ACN, ACP, DC or IC of a positioning motion are ignored in case of an active speed interpolation.
 - System reset cancels the velocity interpolation with the admissible axis acceleration.
 - A jerk limitation (ASO) can only be activated at the beginning of the speed interpolation. In case of speed modifications or a flying positioning, the specification is ignored.

5.4.2 F-address "F"

Description: Depending on the currently active G function G93, G94, G95, the control interprets F-addresses as

- Interpolation time in seconds
(see G93, [chapter 5.2.33 "Time programming G93" on page 156](#))
- Feed in mm/min or inch/min
(see G94, [chapter 5.2.34 "Feed programming \(per minute\) G94" on page 157](#))
- Feed in mm/rev
(see G95, [chapter 5.2.39 "Feed programming \(per revolution\) G95" on page 163](#))

The following applies:

F is applied modally for G94 and G95.



The last F-value may change after a startup, control reset or reset!

After the events listed above, the F-value defined in the machine parameters 7060 00020 or 7060 00010 is applied (default value: F0). Whether G93, G94 or G95 is to be active is also stored there (default value: G94).

Therefore, ensure that the required feed is always programmed before the machining!

Syntax:

F<Value>	
with	
<Value>	Depending on the active G function, it is interpreted as interpolation time, feed or dwell time.

Tab. 5-87: Syntax F<value>

Special features and restrictions: The programmed path velocity can be suppressed using the "Test feed" function.

This is controlled via the channel interface signal "qCh_TestFeed".

5.4.3 F2-address "F2"

Description In addition to the presently available F-value (default feed), a second feed F2 (Feed2) is introduced for G94 mode. The value of Feed2 can be lower or higher than the default feed. The velocity limitation as well as the velocity profile preparation (look-ahead) are each considered based on the maximum value of the two F-values.

Feed2 is activated and deactivated by the PLC via the output signal 13.2 (qCh_Feedrate2) of the channel interface. Feed2 is active with "qCh_Feedrate2 = 1". The change in feed is applied immediately, i.e., the path velocity is immediately changed to the new velocity, i.e., already in the active block.

Using the PLC input signal 13.4 (iCh_Feedrate2) of the channel interface, the NC report to the PLC that Feed2 is active.

If G94 is active, the control interprets the F2-addresses "F2" as

- feed in mm/min or inch/min (see G94, [chapter 5.2.34 "Feed programming \(per minute\) G94" on page 157](#))

The following applies:

NC functions with syntax according to DIN 66025 (incl. extensions)

If G94 is active, F2 has a modal effect.



The latest current F2-value does not change after a control reset or reset!

Therefore, ensure that the required feed is always programmed before the machining!

Syntax:

F2=<value>	The syntax is "F2=". A blank is not allowed between "F2" and "=".
with	
Value	Is interpreted as feed if G94 is active

Tab. 5-88: Fig. 5.115 : Syntax F2=<value>

Example:

N10 G1 G94 X200 Z300 F200	Feed of synchronous axes:
F2=1000	1000 mm/min if qCh_Feedrate2=1 and 200 mm/min if qCh_Feedrate2=0

Special features and restrictions:

The programmed Feed2 can be suppressed using the "Test feed" function. This is controlled via the channel interface signal "qCh_TestFeed".

As is the case with the default feed, the feed override is also applied at Feed2. An independent override value (value between 0 and 100%) can be added to Feed2 via the system data SD.SysChCmd[...].Feedrate2Ovr. The effect is multiplicative. By default, SD.SysChCmd[...].Feedrate2Ovr is initialized to 100 and exists for each channel.

Switching between the two feed values (F ↔ F2) also has the same effect as a change in override.

5.4.4 Velocity of asynchronous axes "FA"

Description:

All traversing motions of asynchronous axes programmed in the FA-block are traversed with the programmed velocity instead of rapid traverse.

To define the unit for interpretation of the asynchronous feed of linear axes, use the configuration parameter "NCO/CorrUnit/AsynchrAxUnit": mm/min or inch/min.

Switch the unit locally for a particular block using the programming attributes MM(...) and INCH(...).

For rotary axes, the unit of the programmed FA -feed is interpreted in degree/min irrespective of the FA-feed set in the configuration and irrespective of the programming attribute.



Sporadically, machine damage in case of incorrect programming.

The specified velocity is only applied in the current FA-block.

Programming asynchronous axes without reprogramming the FA word in a subsequent block will let the axes travel in rapid mode again.

Syntax:

FA<Value>	
with	
<Value>	Desired velocity

Tab. 5-89: FA syntax

Example:	N10 G1 G94 X200 Z300 F200	Feed of synchronous axes: 200 mm/min
	N11 UA400 VA140 FA250	The asynchronous axes UA and VA traverse at 250 mm/min.
	N12 UA0 WA10	The asynchronous axes UA and WA traverse in rapid traverse again.

5.4.5 Omega address (feed) "Omega"

Description: If only the axes hidden for feed computing are moved in a block (see "FeedAd", [chapter 6.43 "Feed computation: Hiding of axes FeedAd, FAD" on page 259](#)), their **feed** can be set via the "Omega" address.

Syntax:

Omega<Value>	
with	
<Value>	Desired feed

Tab. 5-90: Omega syntax

5.4.6 Programming spindle speed "S, SSPG"

Description: An active G97 defines the desired speed of

- a single spindle (S...) or
- a complete spindle group (SSPG...).



The spindle reaches the programmed speed only if a spindle rotation command (e.g. clockwise rotation: M3; counterclockwise rotation: M4) is present.

Syntax:

S<Number>=<Value>	Programs the speed for a single spindle.
SSPG<Group>=<Value>	Programs the speed for a complete group of spindles.
S<Value>	Abbreviated notation to program the speed of the first spindle. Applies only to the first spindle if it is not assigned to any group of spindles via MP 1040 00002. Otherwise, the complete spindle group in which the first spindle is contained is programmed with this value.
with	
<Number>	Number of the spindle (spindle index) Input range: 1 up to the number of the defined spindles (defined via MP1040 00001) Integer
<Group>	Number of the spindle group Input range: 1 ... 4
<Value>	Desired spindle speed (default unit: rpm) Input value: ≥ 0

Tab. 5-91: Syntax of S and SSPG

NC functions with syntax according to DIN 66025 (incl. extensions)

- Special features and restrictions:**
- When G96 is active, an S word is interpreted as the cutting velocity.
For the syntax, see [chapter 5.2.42 "Constant cutting velocity G96, Direct speed programming G97" on page 166](#).
 - In conjunction with the function "align spindle", an S word is interpreted as the positioning angle.
For the syntax, see [chapter 5.3.7 "Orienting the spindle/positioning the spindle M19, M119, M219" on page 184](#).
 - The speeds of several spindles/spindle groups can be programmed in the same block.
 - The resulting speed depends on the spindle-specific override.
 - *The resulting speed can be limited by*
 - the functions "SMin" or "SMax"
(refer to [chapter 6.117 "Speed limitation SMin, SMN, SMax, SMX" on page 363](#))
 - Gear stage limit values (MP 1040 00011, 1040 00012)
 - The maximum permitted speed of a group of coupled spindles (synchronously running spindles) depends on the NC cycle time.
The following applies: $S_{max} [rpm] = 14400 / (MP\ 9030\ 00001 [ms])$
 - The set speed is applied until it is overwritten by a new speed specification for the same spindle(s).
After the control startup, S=0 applies.
 - By setting the interface signal "qSp_SValueSD", the speed is specified via the system data "SysSpCmdData[n]/Speed". The programmed specifications via S, Sx=, SSPGx are ignored.

Example:	N10 G97	Activates speed programming
	N20 SSPG1=1000	Speed of all spindles of the first spindle group to 1000 rpm
	:	
	N50 S1=2000 S2=60	Speed of the first spindle to 2000 rpm
	:	Speed of the second spindle to 60 rpm
	N80 S3=2000	Speed of the third spindle auf 2000 U/min
	:	
	N90 S1500	Speed of the first spindle to 1500 rpm
	:	If the first spindle is assigned to a spindle group, the speed specification applies to the entire spindle group.

5.5 Tool correction

5.5.1 D-correction "D"

Description: The D-correction retrieves the tool correction values stored in XML-D-correction tables in the MTX CNC system. A correction table can contain a maximum of 99 data blocks.

Each data block contains the following correction values:

- Three tool lengths L1, L2, L3,
- Tool edge radius RAD,
- Tool edge position ORI.

D-corrections are equally suitable for drilling, milling, lathing and angle-head tools. With a total of three offset values (L1, L2 and L3), constant three-dimensional tool offsets for a tool as well as parallel length corrections for up to three different tools can be performed.

Activate a D-correction table with the function "DcTsel" (DCS); then select a correction set from the maximum of 99 data blocks with the NC command "D".

The correction values are applied if either the tool correction G47 (tool length and tool edge position) or milling cutter path correction G41/G42 or G141/G142 (tool radius) is active. They are additively superimposed by the actively selected correction values of external tool correction ED.

The following applies:

- The preselected tool set has a modal effect. A new programming deletes the currently preselected tool set.
- A D-correction can be programmed in the same block as other path conditions, traversing motions and auxiliary functions.
- The tool correction is only settled after the corresponding NC function was activated: G47, G41, G42, G141, G142.

Syntax:

D<No. tool set>	Preselects the tool correction block from the active D-correction table
D0	Deselects the tool correction set without selecting a new one
with	
<no. tool record>	Number of the correction data block Input range: 1... 99

Tab. 5-92: Syntax of D and D0

Example:

```
N10 D7      Preselect correction values for tool set 7
:           :
```

5.5.2 ED compensation "ED"

Description: External tool correction (ED compensation) calls compensation values for a maximum of 16 tool edges. The PLC or the part program with the CPL command "DCT" can be used to write on the correction values via the program function block "MT_TCorr".

Each of the 16 data blocks contains the following correction values:

- Three tool lengths L1, L2, L3,
- Tool edge radius RAD,
- Tool edge position ORI

ED compensation is equally suitable for drilling, milling, lathing and cross-staff tools. With a total of three offset values (L1, L2 and L3), constant three-dimensional tool offsets for a tool as well as parallel length corrections for up to three different tools can be performed.

Use the NC command "ED" to select a correction set from the maximum of 16 data blocks.

The correction values are applied if either the tool correction G47 (tool length and tool edge position) or milling cutter path correction G41/G42 or G141/

G142 (tool radius) is active. They are additively superimposed by the actively selected correction values of the D-correction.

The following applies:

- The preselected tool edge has a modal effect. New programming deletes the preselected tool edge that was previously active.
- An ED-correction can be programmed in the same block as other path conditions, traversing motions or auxiliary functions.
- The tool correction is only settled after the corresponding NC function was activated:
G47, G41, G42, G141, G142.
- The preselected tool edge can be output at the PLC interface of the corresponding channel:
iCh_ActFunc1..24

Syntax:	ED<No. tool edge>	Preselects the correction set
	ED0	Deselects the correction set without selecting a new one.
	with	
	<No. tool edge>	Number of the tool edge Input range: 1... 16

Tab. 5-93: ED syntax

Example:	N10 ED7	Preselects correction values for tool edge 7
	:	

6 NC functions with high-level language syntax



Undesired traversing motion!

Many NC functions require the programming of axis and/or coordinate addresses. Generally, only the addresses of the respective channel axis, i.e. synchronous axes can be programmed apart from the coordinate addresses.

Programming the address of an asynchronous axis within a parameter list generally leads to a syntax error, while programming outside of a parameter list leads to an asynchronous traversing motion!

Exceptions:

- G74(Home) VA1:
The asynchronous axis VA travels to the reference point
- GAX(VA):
The asynchronous axis VA is applied to the channel
- FsMove, FsTorque, FsReset:
The "Move to Fixed Stop" functionality also affects asynchronous axes

6.1 Overview

The control provides multiple NC functions. In addition to the commands specified in DIN 66025, these functions include important supplements in the field of G-codes and additional general language-type syntax elements that are described in this chapter.

NC functions with high-level language syntax:

With NC functions with general language syntax, the meaning of the function can generally be derived from the name (e.g. "scale": scaling).

There are the following variants of equal importance:

- Long form with no case sensitivity.
To improve the readability of this documentation, the first letter of the composite partial words is capitalized. Example: "KvProg"
- Short form, consisting of three capital letters (four capital letters in exceptional cases).

Example:

Short form of KvProg: KVP

The corresponding short form is also shown in the process data display of the active modal NC functions.

The following syntax descriptions show both variants.



For a tabular overview of all NC functions, refer to [chapter 8 "Appendix" on page 597](#).

In addition, all NC functions are in the index.

Used notations:

In this documentation, the following notations are used for the syntax of NC functions:

Font "Courier bold" or "Courier":

Character strings in this font have to be programmed as shown below.

Example: G0 (POL)

Angle brackets < >

indicate a placeholder for an expression/parameter to be programmed. The placeholder is written in italics.

Example: <Axis1>

Curly brackets { }

indicate an optional expression/parameter.

Such syntax elements can, but do not have to be programmed.

Example: G0 { ({ POL, }{<Par1>}) }

Character "|"

separates possible parameters that cannot be used simultaneously (alternative parameters).

Example: G0 { ({ POL, }{NIPS | IPS1 | IPS2 | IPS3 }) }

6.2 Programming effective radius factors "ActRadFact, ARF"

6.2.1 Effect

Feed calculation requires calculation of the path length of the programmed blocks. In case of linear axes/coordinates, this is the geometric distance ($\sqrt{L_1^2 + L_2^2 + L_3^2 \dots}$). In case of rotary axes/coordinates, an additional factor can be defined by which the distance is multiplied to modify the effect of the axis on the feed ($\sqrt{L_1^2 + L_2^2 + L_3^2 + (R_1 \cdot F_1)^2 + (R_2 \cdot F_2)^2 \dots}$). If this factor is equal to 1, 1°=1mm applies. This corresponds to an effective radius of 57.3 mm.

Example The linear axis travels 100 mm, the rotary axis travels 100 degrees, and both axes travel with the programmed feed/root(2) in [mm/min] and [degrees/min], since both travel paths are considered to be of equal length.

An effective radius factor can be defined by the machine parameters "Channel-specific scaling factors for rotary axis velocities" (NCO/VelScaleFact/Ch[k]/FactRotInch und NCO/VelScaleFact/Ch[k]/FactRotMetr).

An **additional** factor can be specified in the part program via "ActRadFact".

In addition to the effective radius factors for axes and coordinates, it is also possible to specify an effective radius factor for the orientation, provided this is supported by the active transformation.

6.2.2 Programming

Syntax:

ActRadFact (<AxisName> <Factor> {, <AxisName> <Factor>}...)	
Short form: ARF (. .)	
with	
<Axis name>	Name of a rotary axis, a coordinate or an "O" (orientation)
<Factor>	The effective radius factor applicable from this following block (factor >= 0)
ActRadFact ()	All effective radius factors of the channel are reset to 1.

Tab. 6-1: ActRadFact syntax

Special features and restrictions:

- Effective radius factors may not be negative.

- After system start, all effective radius factors of ActRadFact are set to 1. The factors configured in the machine parameters are applied again.
- If an axis is removed from the channel, the programmed effective radius factor is lost.
- If the effective radius of an axis/coordinate is equal to 0, motions only containing parts of this axis/coordinate may not be programmed.
- If the active transformation is changed, the effective radius factors of equal coordinates are applied. All other effective radius factors are reset to 1.

Example:

G0 X0 A0	Starting position
N010 G1 G91 F1000 X100 A100	X → 707[mm/ min] A → 707[degree/ min]
N020 ARF(A0.5)	A-portion is divided in halves
N030 X100 A100	X → 894[mm/ min] A → 894[degree / min]
N040 ARF(A2)	A-portion is duplicated
N050 X100 A100	X → 447[mm/ min] A → 447[degree/ min]
N060 ARF(A10)	A-portion is multiplied tenfold
N070 X100 A100	X → 99[mm/ min] A → 99[degree/ min]
N080 ARF(A0)	A-portion is hidden
N090 X100 A100	X → 1000[mm/min] A → 1000[degree/min]
N100 ARF()	Effective radius factor for A is reset to 1

6.3 Adjusting modal NC functions "Adjust, ADJ"

Description: Using the adjustment function "Adjust", modal NC functions within the adjustment programs can be comfortably adjusted after block pre-run. The syntax of a modal NC function is provided as parameter to the adjustment function. The control enables the NC function in the respective modal group in front of the target block of the block pre-run.

Syntax:

ADJ (<NCF Name>)	
<NCF name>	Syntax of a modal NC function of the group to be adjusted.

Tab. 6-2: ADJ syntax

Example:

ADJ(G0)

The modally effective geometry function (G0, G1, G2, G3...) before the target block becomes active and thus the NC function group is adjusted.

The function-specific parameters for the NC functions listed in the following are adjusted simultaneously:

Modal group	NC functions	Supported parameters
Interpolation functions	G0	Polar programming and exact stop window (NIPS, IPS<n>)
	G1	Polar programming and exact stop window (IPS<n>)
	G2, G3	Polar programming
Path slope	G8	Path shape
	G9	Path shape, axis shape

NC functions with high-level language syntax

Modal group	NC functions	Supported parameters
Level	G17, G18, G19, G20	Axis assignment
Tool length correction	G47	Length assignment
Zero point offset 1	G52.1	Programmable ZO 1 ⁾
	G53.1	ZO 1 off ⁾
	G54.1,...,G59.1	Active ZO table bank 1 ⁾
...		
Zero point offset 5	G52.5	Programmable ZO 5 ⁾
	G53.5	ZO 5 off ⁾
	G54.5,...,G59.5	Active ZO table bank 5 ⁾
Placements: "Inclined plane" 1	G151.1	Tool orientation ^{**)}
	G152.1	Programmable inclined plane 1 ^{**)}
	G153.1	Inclined plane 1 off ^{**)}
	G154.1,...,G159.1	Active inclined plane; table bank 1 ^{**)}
...		
Placements: "Inclined plane" 5	G151.5	Tool orientation 5 ^{**)}
	G152.5	Programmable inclined plane 5 ^{**)}
	G153.5	Inclined plane 5 off ^{**)}
	G154.5,...,G159.5	Active inclined plane; table bank 5 ^{**)}
Exact stop	G61	Exact stop window (IPS<n>)
Tool length assignment	G78	ActPlane or length assignment
Path feed	G93	F-value
	G94	F-value, F2-value
	G95	F-value, reference axis (FRA)
Spindle programming	G96	Reference axis, spindle assignment
	G97	Spindle assignment
D-correction	D0	D-correction off ^{***)}
	D1,...,D99	Active D-correction ^{***)}
Axis acceleration	AAC	Programmed axis accelerations
Axis jerk	AJK	Programmed axis jerks
Additive program coordinate offset	ATR	Program zero point coordinates
Maximum axis velocity	AVE	List of axis velocities
Channel axis coupling	AXC	The channel master axes of the block pre-run and the corresponding slave axis coupling relations and tables. Prerequisites: The target block must not contain any coupling commands. The coupling has to be disabled.

Modal group	NC functions	Supported parameters
Workpiece position	BCR	Workpiece zero point coordinates, angle
Chamfer/roundings	CHL CHS RND RNE	Chamfer length, chamfer segment Radius, epsilon
Automatic corner detection	CLD	Angle, angle factor, maximum point distance, distance factor, accuracy limit
Milling path collision detection	CLN	Monitoring type, preview area
Axis transformation	COORD	Axis transformations 1 and 2 with transformation options
Radius/diameter programming	DIA RAD	Diameter coordinates
Channel spindles and spindle groups	DSP, GSP, RSP SPGn, SPGALL	Channel spindles and channel group assignment (corresponds to SPADJSP)
Block transition without reduction of velocity	DTT	Transition angle
Feed masking axes	FAD	List of masking axes
Feed coordinate group	FDG FDGT	List of path coordinates, feed group type
Straightness and angle error compensation	GCT	-
Active axis jerk	JAL	List of path-relevant axis jerk axes
Jerk limitation	JKC	-
Input tool "Mirroring"	MIR	Mirroring axes
Main spindle	MSP	Spindle
Path acceleration	PAC	Path acceleration, path deceleration
Pole definition	PLS	Mirroring/turning point
Rotary axis positioning mode	PMD	Positioning mode
Pole definition	POP	Pole coordinates
Accuracy programming	PRP	Radius contour error (EPS), lagging (DIST)
Maximum radial acceleration	RAC	Acceleration
Input help: Turning	ROT	Angle
Input help: Scaling	SCL	Axis factors
Input help: Offset	SHT	Contour zero point coordinates
Spline definition	SPD	Spline type, spline coordinates, starting conditions (SBC), end conditions (EBC), spline parameter length (PL)
Geometric splitting	SPT	Mode, length

NC functions with high-level language syntax

Modal group	NC functions	Supported parameters
Program coordinate offset	TRS	Program zero point coordinates
Tapping spindle	TSP	Spindle

Tab. 6-3: Adjustable modal parameterizable NC functions

- *) Implicit ZO table selection (ZOS)
- **) Implicit placement table selection (PMS)
- ***) Implicit D-correction table selection (DCS)

- Special features and restrictions:**
- If necessary, the adjustment of modal parameterizable NC functions not mentioned above has to be performed using pre-run system data in the adjustment program (InitAdjPrg) or using the action block.
 - Each bank of a zero point offset or an inclined plane has to be adjusted separately: ADJ(G54.1), ADJ(G54.2), etc.
 - The tool radius correction G41/G42 becomes only active in the adjusted state with the approaching block after the adjustment. During the adjustment, the internal effect is suppressed. The correction is not calculated.

6.4 Programmable distance-dependent signal

6.4.1 General information

Before reaching a programmed mark, the Advance Distance Signal (ADS) signal sets a signal. The signal properties are specified by the system date / SysAdvDistSig/Sig[].



For more information, refer to the "MTX Functional Description Special Functions" in the chapter "Programmable distance-dependent signal".

6.4.2 Activate search for a mark "ADSENA"

Description: Advance-Distance-Signal-ENable (ADSENA)

ADSENA applies the current content of the " /SysAdvDistSig/Sig[]" system date and activates the search for a programmed mark (see ADSLAB).

Syntax:

ADSENA (<Signal> {,<Signal> ...})	
with	
<Signal>	1 .. 8: Number of a signal

Tab. 6-4: Syntax ADSENA

Example:

SignalConfig

```

N00 WAIT
001 SD.SysAdvDistSig.Unit = 0 : REM mm
002 REM
003 REM Configure Signal 3
004 SD.SysAdvDistSig.Sig[3].DistOn = 1.2 : REM 1.2 mm
005 SD.SysAdvDistSig.Sig[3].DistOff = 0.0 : REM 0.0 mm
006 SD.SysAdvDistSig.Sig[3].TimeOn = 0.0 : REM 0.0 msec
007 SD.SysAdvDistSig.Sig[3].TimeOff = 0.0 : REM 0.0 msec
008 SD.SysAdvDistSig.Sig[3].Duration = 40 : REM 40 msec
009 SD.SysAdvDistSig.Sig[3].DistCalc = 0 : REM calc. dist. in active plane
010 SD.SysAdvDistSig.Sig[3].Type = 0 : REM iCh_Custom2
011 SD.SysAdvDistSig.Sig[3].Index = 2 : REM iCh_Custom2

```

PartProg

```

...
N20 ADSENA(3) ; Activate look-ahead for label ADSLAB(3)
N21 G1 X... Y... F...
N22 G2 X... Y...
N23 G1 X... Y...
...
N50 G1 X... Y... ADSENA(3) ; Set signal iCh_Custom2 1.2 mm before end of block
N51 G1 X... Y...
...
N80 ADSDIS(3) ; Deactivate look-ahead for label ADSENA(3)
...

```

Special features: Control reset und M30 disable the function.

6.4.3 Deactivate the search for a mark "ADSENA"

Description: Advance-Distance-Signal-ENABLE (ADSENA)

ADSENA deactivates the search for a programmed mark (see ADSLAB).

Syntax:

ADSDIS (<Signal> {,<Signal> ...})	
with	
<Signal>	1 .. 8: Number of a signal
ADSDIS ()	Disables the search for all marks.

Tab. 6-5: Syntax ADSDIS

Example: See "Activate search for a mark "ADSENA"".

Special features:

- ADSDIS does not exit the signal output.
- Switching signals cannot be deleted using the ADSCLR(<Signal>) function.

6.4.4 Set mark "ADSLAB"

Description: Advance-Distance-Signal-ENABLE (ADSENA)

ADSLAB is the searched for mark.

Syntax:

ADSLAB (<Signal> {,<Signal> ...})	
with	
<Signal>	1 .. 8: The signal "DistOn" + "TimeOn" is set in front of the mark.
	-1 .. -8: The signal "DistOn" + "TimeOff" is reset in front of the mark.

Tab. 6-6: Syntax ADSLAB

Example: See "Activate search for a mark "ADSENA"" and "Reset switching signal"ADSCLR"".

6.4.5 Reset switching signal "ADSCLR"

Description: Advance-Distance-Signal-CLEAR (ADSCLR)

Resetting switching signals.

Syntax:

ADSCLR (<Signal> {,<Signal> ...})	
with	

NC functions with high-level language syntax

<Signal>	1 .. 8: Number of a signal
ADSCLR ()	Resets all switching signals.

Tab. 6-7: Syntax ADSCLR

Example: *SignalConfig*

```

N00 WAIT
001 SD.SysAdvDistSig.Unit = 0 : REM mm
002 REM
003 REM Configure Signal 3
004 SD.SysAdvDistSig.Sig[3].DistOn = 1.2 : REM 1.2 mm
005 SD.SysAdvDistSig.Sig[3].DistOff = 0.0 : REM 0.0 mm
006 SD.SysAdvDistSig.Sig[3].TimeOn = 0.0 : REM 0.0 msec
007 SD.SysAdvDistSig.Sig[3].TimeOff = 0.0 : REM 0.0 msec
008 SD.SysAdvDistSig.Sig[3].Duration = -1 : REM switching signal
009 SD.SysAdvDistSig.Sig[3].DistCalc = 0 : REM calc. dist. in active plane
010 SD.SysAdvDistSig.Sig[3].Type = 0 : REM iCh_Custom2
011 SD.SysAdvDistSig.Sig[3].Index = 2 : REM iCh_Custom2

```

PartProg

```

...
N20 ADSENA(3) ; Activate look-ahead for label ADSLAB(3)
N21 G1 X... Y... F...
N22 G2 X... Y...
N23 G1 X... Y...
...
...
N50 G1 X... Y... ADSLAB(3) ; Set signal iCh_Custom2 1.2 mm before end of block
N51 G1 X... Y...
N52 G1 X... Y... ADSLAB(-3) ; Reset signal at end of block
...
N80 ADSDIS(3) ; Deactivate look-ahead for label ADSLAB(3)
...

```

"Duration = -1" defines signal 3 as switching signal.

If the part program "PartProg" is canceled by a reset in block N51, signal 3 and thus iCh_Custom2 remains set. If the ADSCLR () command is contained in the init string or the init program, the signals is reset by a system reset.

6.5 Extending the retrace forward program block "AdvanceAppend, ADVA"

Description: For reverse operation, a forward program block is generated for each part program block in the retrace area Retrace(1)..Retrace(0). The function "AdvanceAppend" inserts the specified NC functions at the end of the current forward program block.

Syntax:

AdvanceAppend (<NCF>...<NCF>)	
Short form: ADVA (...)	
with	
<NCF>...<NCF>	List of NC functions

Tab. 6-8: Syntax AdvanceAppend (ADVA)



Contour changes result in switching errors in reverse operation.
CPL functions and subroutine calls are not allowed.

Example:	Part program:	Forward program:	Note:
	N100 X100 M10	N1 X=100	Auxiliary functions are not applied
	N200 X0 ADVA(M125 M126)	N2 X=0 M125 M126	In the forward program, M125 and M126 is to be output to the PLC.



For checking purposes, save the modified forward program in a file (see [chapter 6.105 "Retrace, RETRACE" on page 347](#)).

6.6 Replacing the retrace forward program block "AdvanceSubstitute, ADVS"

Description: For reverse operation, a forward program block is generated for each part program block in the retrace area Retrace(1)..Retrace(0). The "AdvanceSubstitute" function replaces the current forward program block by the specified NC functions.

Syntax:

AdvanceSubstitute(<NCF>...<NCF>)	
Short form: ADVS (...)	
with	
<NCF>...<NCF>	List of NC functions

Tab. 6-9: Syntax AdvanceSubstitute (ADVS)



Contour changes result in switching errors in reverse operation. CPL functions and subroutine calls are not allowed.

Example:	Part program:	Forward program:	Note:
	N5 X0 F10 ADVS(X0 F100)	N1 X0 F100	Allow a higher feed in the forward program. Always insert the programmed end point coordinates!



For checking purposes, save the modified forward program in a file (see [chapter 6.105 "Retrace, RETRACE" on page 347](#)).

6.7 Area monitoring "area, ARA"

Description: It defines, activates or deactivates up to 32 rectangular, two-dimensional dead area or workspaces with axis-parallel boundaries and monitors their intersections with linear and circular traversing motions. Monitoring exclusively takes place in the active plane and concentrates only on machine and axis coordinates. The ("Area","ARA") function is not intended for monitoring of workpiece coordinates.



If the workpiece coordinates are to be monitored, e.g. in case of an active tool length correction, coordinate rotation or scaling, the monitoring function "Range monitoring with system data" described in the "MTX 14VRS Functional Description - Special Functions" has to be used.

The described function is independent of the ("Area", "ARA") function!

- **Dead areas:**

They may neither be crossed nor touched during a traversing motion (incl. their boundaries).

- **Workspaces:**

They must not be exited – incl. their boundaries – during a traversing motion.



Default data of the areas 1-10 is defined using the machine parameter group MP 8002.

If the optional system date SysARACfg is defined, the machine parameter group MP 8002 is inactive: The default data of all areas has to be configured/pre-assigned in SysARACfg.

Static areas:

By default, areas are statically machine-fixed. A detected area violation results in an error and in a cancelled program. Only active areas are monitored (-> monitoring status).

For static areas, define the (optional) system date SysARACfg. All pre-assignments can be done in the system date. Thus, the configuration via the machine parameters is not required anymore.

Dynamic areas:

Dynamic areas can become active or inactive (released) during program processing, e.g. by mechanic retracing.

Dynamic areas are monitored analogously to the static areas.

Only active areas are monitored (-> monitoring status).

The following (optional) system data are relevant for the dynamic areas and have to be defined:

- SysARACfg
- SysARA
- SysARAAadm

The property "dynamic" (of an area) is an optional configurable property that can only be specified in the SysARACfg.DynamicArea[<i>] system data for each area <i>.

0: static area

1: dynamic area, can only be switched ineffectively if no other dynamic area is ineffective

2: dynamic area, can only be switched ineffectively if no other dynamic area is ineffective

The PLC has to specify via the SysARA.PlcAreaDisabled[<i>] system date at the time of execution if the dynamic area is active:

0: Range is active

1: Range is inactive/released

A detected area violation of a dynamic area does not result in a cancelled program but in a motion stop before the critical traversing motion. A warning is output (warning can be suppressed with SysARAAdm.MsgDisable_5511=1). The traversing motion is stopped until the affected range is released and once the PLC reported a message.

The state "Area violated" is signaled at the point of execution of the critical NC block by an identifier in the system data (SysARA.AreaViolationOccured[<i>]=1). After the affected area has been enabled (inactively) in the application (PLC->System data interface SysARA.PlcAreaDisabled[<i>]=1), the critical traversing motion is executed and the state "Area violated" is reset (SysARA.AreaViolationOccured[<i>]=0).

If SysARA.PlcAreaDisabled[<i>]=1 is detected before reaching the critical traversing motion, the motion is executed without communication.

In order to be able to stop at a boundary, the velocity is decelerated, if required.

It is recommended to select an extension of the area to avoid a collision in case of axis oscillation.

Syntax: **Activating, deactivating or releasing one or all monitoring areas at once.**

Area (<BNo> , <Sta>)	
Short form: ARA (...)	
with	
<BNo>	Area number Integer. Input range: -1, 1...32 -1: Activates/deactivates all areas
<Sta>	Desired monitoring status 0: Deactivates monitoring 1: Activates monitoring -1: Deactivate and release monitoring

Tab. 6-10: Syntax 1 Area (ARA)

Defining, activating, deactivating and release one single monitoring area.

Area (<BNo> , <Sta>{ , <Mod> , {<P1> } , {<P2> } , {<D1> } , {<D2> } , {<Ax1> } , {<Ax2> } })	
Short form: ARA (...)	
with	
<BNo>	Area number. Integer. Input range: 1...32.
<Sta>	Desired monitoring status 0: Disable monitoring of <BNo> range 1: Enable monitoring of <BNo> range -1: Disable monitoring of <BNo> range and release <BNo> range
<Mod>	0: Range <BNo> is not used 1: Range <BNo> is a dead area 2: Range <BNo> is the working range

NC functions with high-level language syntax

<P1>	Position in the machine coordinate system Specifies the position values of the center point of the area, referring to the first system axis involved in the area (defined using MP 8002 00001 (optional: SysARACfg.Pos1) or programmed with Ax1), fixed. See the following example The same programming unit as for the axis coordinates
<P2>	Position in the machine coordinate system Specifies the position values of the center point of the area, referring to the second system axis involved in the area (defined using MP 8002 00002 (optional: SysARACfg.Pos2) or programmed with Ax2), fixed. Refer to the following example. The same programming unit as for the axis coordinates
<D1>	Specifies the length of the area, referring to the first system axis involved in the area (defined using MP 8002 00001 (optional: SysARACfg.Ext1) or programmed with Ax1), fixed. Refer to the following example. The same programming unit as for the axis coordinates.
<D2>	Specifies the area length, referring to the second system axis involved in the area (defined using MP 8002 00002 (optional: SysARACfg.Ext2) or programmed with Ax2), fixed. Refer to the following example. The same programming unit as for the axis coordinates
<Ax1>	Specifies the first system axis of the area (optional: SysARACfg.AreaSysAxisNoDim1) System axis number
<Ax2>	Specifies the second system axis of the area (optional: SysARACfg.AreaSysAxisNoDim2) System axis number

Tab. 6-11: Syntax 2 area (ARA)



Non-programmed values are retained if they were already entered during ongoing program execution.

If the values have not been specified during the program run, the control used the corresponding values of the machine parameter group 8002 for the areas 1-10, the values of the areas 11-32 are not assigned (optional: Pre-assignment SysARACfg).

Example:

N100 Area(4,0,,100,200)	Deactivate area 4 and sets the center point of the area to the given machine coordinates (100,200). The area lengths remain unchanged.
N120 Area(12,1,2,100,200,,1,3)	Activate the area 12 with the type working range and set the area center point to the specified machine coordinates (100, 200). The area lengths remain unchanged. First system axis of the area: 1. axis Second system axis of the area: 3. axis
N150 Area(12,-1)	Deactivate area 12 and activate it for other channels

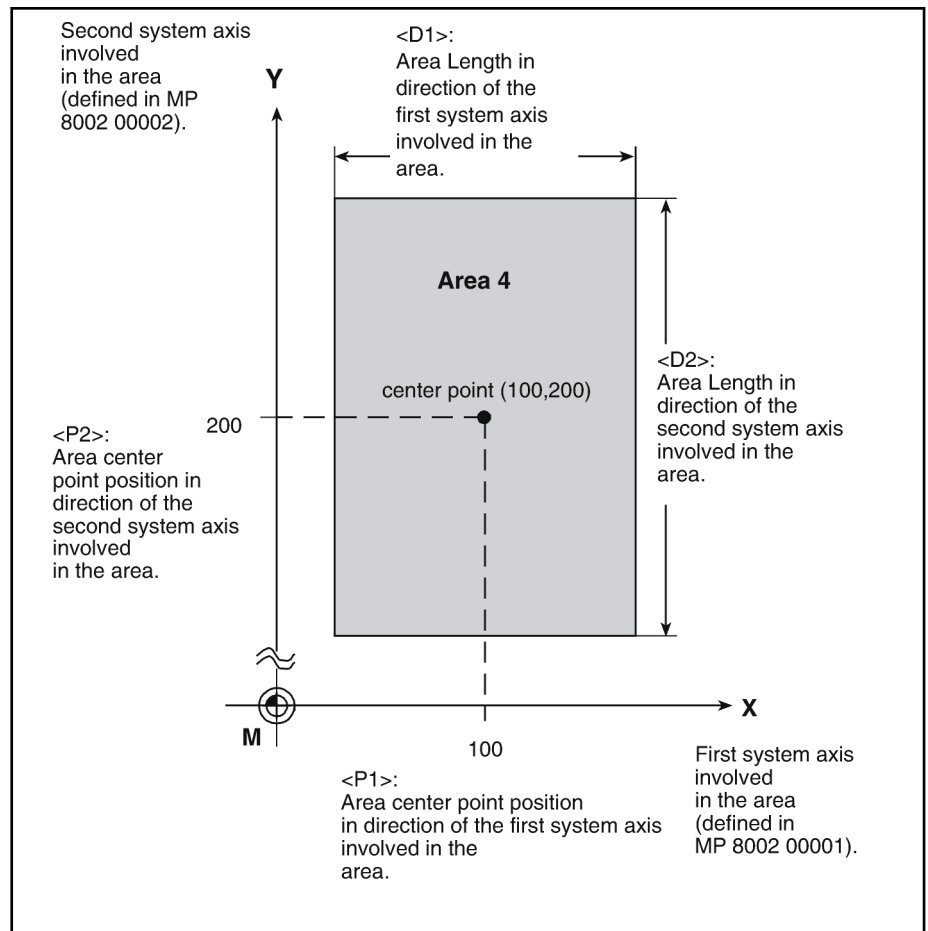


Fig. 6-1: Example of area (ARA)

Special features and restrictions:

- Only axis and machine coordinates are monitored. Workpiece coordinates cannot be monitored.
- The following applies for the areas 1-10: All system axes involved in the individual areas have to be defined in MP 8002 00001 and MP 8002 00002 or have to be programmed with Ax1 and Ax2.
- The following applies to the areas 11-32: All system axes that are part of the individual areas have to be programmed using Ax1 and Ax2.
- To influence the areas 1-10 using "Area", MP 8002 00032 has to be set accordingly.
- The function requires referenced axes.
- When an area is activated, the corresponding axes have to be in the current channel.
- Influencing an area via "Area" only has an in the channel in which "Area" is programmed. If the axes of an area are transferred to another channel, this area is provided with the default values of the target channel as preset in the machine parameter group 8002 and the area is deactivated. Any values programmed via "Area" in the source channel are not transferred.
- Each are is unique and only exists once per system and can only be modified by a channel. If an area is to be used by another channel, the area has to be activated first by the first channel
- In jog mode: Axes moved via the handwheel are not monitored.

NC functions with high-level language syntax

- In jog mode: Only one of the axes spanning a dead area may be jogged at a time.
- Area violations in jog mode result in a warning. The corresponding axis does not move and can only be jogged in opposite direction.
- If the dimensions of an area are within an active plane, linear motions as well as circular motions are monitored.
- If the dimensions of an area are outside an active plane, only linear motions are monitored for this area.
- If the dimensions of an area are partially outside of the active plane, only linear motions are monitored for this area. Circular motions cannot be monitored and result in a warning message (this warning can be suppressed: SysARAAdm.MsgDisable_3732=1 - the system data is verified before a warning is output).
- Linear traversing motions whose involved axes are in the monitored channel are monitored.
- Circular traversing motions are monitored if the relevant monitoring area is in the active plane with both axis dimensions.
- If both dimensions of a monitoring area are outside the active plane during a circular motion, this area is not monitored. If only one dimension of the area is in the active plane, a warning message is displayed reporting that the motion is not monitored.

System data for dynamic protection areas

The system data of the protection areas are volatile The configuration data is preassigned using the SDDatAra.xml file.

How to define the system data for the protection areas:

Copy the "feprom/schemas/sdara.xsd" file to "usrfep/schemas/sdara.xsd" and integrate them in an active definition file (e.g. usrfe/SDDefMtb.xml).

```
<Variable Dimension="32" Storage="volatile">
```

```
<Name>SysARA</Name>
```

```
<Type>ARA_t</Type>
```

Creating the ARA system data:

```
...
```

```
</Variable>
```

```
<Variable Dimension="32" Storage="volatile">
```

```
<Name>SysARACfg</Name>
```

```
<Type>ARACfg_t</Type>
```

```
...
```

```
</Variable>
```

```
<Variable Storage="volatile">
```

```
<Name>SysARAAdm</Name>
```

```
<Type>ARAAdm_t</Type>
```

```
....
```

```
</Variable>
```

Templates of these definitions are stored in feprom/SDDefOem.xml

6.8 Asynchronous subroutines

6.8.1 Asynchronous subroutines: log off "ASPCLR"

Description: Logs off an asynchronous subroutine in the current channel.
Logged-off subroutines can be neither activated nor deactivated.



To log on logged-off asynchronous subroutines, refer to the function "ASPSET", [chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211](#).



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

ASPCLR (<Up-No>)	
with	
<SR no.>	Subroutine number Input range: 1...8. Integer

Tab. 6-12: ASPCLR syntax

- Special features and restrictions:**
- The corresponding subroutine has to be logged on in the target channel (see ASPSET, [chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211](#)).
 - To temporarily deactivate a subroutine, refer to the function "ASPDIS", [chapter 6.8.2 "Asynchronous subroutines: switch off ASPDIS" on page 209](#).

6.8.2 Asynchronous subroutines: switch off "ASPDIS"

Description: Deactivates an asynchronous subroutine in the current channel.
A deactivated subroutine is not called if the relevant event occurs.



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

ASPDIS (<Up-No> {,<IrUp-No>})	
with	
<SR no.>	Subroutine number Input range: 1...8. Integer
<IrUp-No>	Interruptible asynchronous subroutine (optional): Not specified: - Locking interruption of the part program Number of the interruptible asynchronous subroutine: Input range: 1..8. Integer - Locking interruption of the asynchronous subroutine <IrUp-No>

Tab. 6-13: ASPDIS syntax

- Special features and restrictions:**
- The corresponding subroutine has to be logged on in the target channel (see ASPSET, [chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211](#)).
 - To activate a deactivated subroutine, refer to the function "ASPENA", [chapter 6.8.3 "Asynchronous subroutines: switch on ASPENA" on page 210](#).

6.8.3 Asynchronous subroutines: switch on "ASPENA"

Description: Activates an asynchronous subroutine in the current channel.
Only activated subroutines can be called when the relevant event occurs.



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

ASPENA (<Up-No> {,<IrUp-No>})	
with	
<SR no.>	Subroutine number Input range: 1...8. Integer
<IrUp-No>	Interruptible asynchronous subroutine (optional): Not specified: - Unlocking interruption of the part program Number of the interruptible asynchronous subroutine: Input range: 1..8. Integer - Unlocking interruption of the asynchronous subroutine <IrUp-No>

Tab. 6-14: ASPENA syntax

- Special features and restrictions:**
- The corresponding subroutine has to be logged on in the target channel (see ASPSET, [chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211](#)).
 - To deactivate a subroutine, refer to the function "ASPDIS", [chapter 6.8.2 "Asynchronous subroutines: switch off ASPDIS" on page 209](#).

6.8.4 Asynchronous subroutines: Define point of return to contour "ASPRTP"

Description: *Defines where the control, after an asynchronous subroutine of a traversing block that can sporadically be interrupted is to position to:*

- Starting point
- End point
- Point of interruption

If no traversing block was active at the time of interruption, the control always positions on the last active coordinates.



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

ASPRTP (<Up-No> , <Point>)	
with	

<SR no.>	Subroutine number Input range: -1; 1...8. integer -1 : Define the desired point of return to contour for all asynchronous sub-routines of the current channel.
<Point>	Desired point of return to contour: 1: Starting point 2: End point 3: Interruption point

Tab. 6-15: ASP RTP syntax

Special features and restrictions:

- The set desired point of return to contour is cleared by a reset or M30.
- Any correction changes made in an asynchronous subroutine are automatically included in the internal calculation of the required point of return to contour.
- The "REPOSTP" function (see [chapter 6.104.2 "Defining point of return to contour in asynchronous subroutine REPOSTP" on page 346](#)) can be used to temporarily suppress the defined point of return to contour in the asynchronous subroutine.

6.8.5 Asynchronous subroutines: register "ASPSET"

Description: Logs on an asynchronous subroutine in the current channel and activates it (for activation, see also the function "ASPENA", [chapter 6.8.3 "Asynchronous subroutines: switch on ASPENA" on page 210](#)).

Only logged on and activated subroutines can be used.



To log off logged-on asynchronous subroutines, refer to the function "ASPCLR", [chapter 6.8.1 "Asynchronous subroutines: log off ASPCLR" on page 209](#).



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

ASPSET (<Up-No> , <Up-Name>{, <Flags>})	
with	
<SR no.>	Subroutine number Input range: 1...8. Integer

<SR name>	Subroutine name With or without path specification If the path is missing, the file is searched according to the parameter NCO/FileOrg/SrchPathSubProg (MP 3080 00001) (search path for subroutines).
<Flags>	Switch that can be used to affect the behavior after the call: 00: neither linking nor automatic start. 10: linking, but no automatic start. 01: no linking, but automatic start. 11: linking and automatic start.

Tab. 6-16: ASPSET syntax

Example:

```

:
N30 ASPSET(1,AS- Logging on of the program ASUP1 as the first asynchronous
UP1,10)          subroutine in the active channel. After its call (e.g. via the in-
:                terface signal), this subroutine is – if required – automatically
                linked and explicitly started via NC start.

```

6.8.6 Asynchronous subroutines: triggering via the program "ASPSTA"

Description: It calls an asynchronous subroutine program-controlled in any channel.



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

ASPSTA (<Up no.>{, <Channel No.>})	
with	
<SR no.>	Subroutine number Input range: 1...8. Integer
<Channel number>	Target channel in which <SR no.> is to be called If not programmed, <Up-No> is called in the current channel

Tab. 6-17: ASPRTP syntax

Special features and restrictions:

- Asynchronous subroutines must not be nested.
- The called subroutine has to be logged on in the target channel (see ASPSET, [chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211](#)).
- The called subroutine must not be deactivated in the target channel (see ASPDIS, [chapter 6.8.2 "Asynchronous subroutines: switch off ASPDIS" on page 209](#) or ASPENA, [chapter 6.8.3 "Asynchronous subroutines: switch on ASPENA" on page 210](#)).

6.9 Assigning logical axis name "AssLogName, ALN"

Description: Assigns a new logical axis name to a synchronous axis in the calling channel. The old logical axis name becomes invalid.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

AssLogName (<PAN> <PAI> <LAN>, <LANnew> { , <PAN> <PAI> <LAN>, <LANnew>}...)	
Short form: ALN (. .)	
with	
<PAN>	Physical axis name (system axis name) Specifies the axis to be renamed in the current channel.
<PAI>	Physical axis index (system axis index) Same effect as <PAN>
<LAN>	Logical axis name (channel axis name) Same effect as <PAN>
<LANnew>	New logical axis name The axis specified via <PAN>, <PAI> or <LAN> is provided with the logical name <LANnew> in the current channel. <LANnew> has to be defined either in parameter CHAN/Ch/Coord/Acs/ChAx/ChAxName (MP 7010 00010) (channel axis name) or in parameter CHAN/Ch/Coord/Acs/OptAxName/OptChAxName (MP 7010 00020) (optional axis name).

Tab. 6-18: AssLogName (ALN) syntax

Example:

```

:
N030  ALN (YP, X, 3, Y, B, Z)
:

```

The system axis YP is assigned the channel axis name (logical name) X, the third system axis is assigned the channel axis names Y. The channel axis B is assigned the channel axis name (logical names) Z.
Subsequent programming of B generates a runtime error.

Special features and restrictions:

- An axis to be renamed has to be at standstill. If this is not the case, the control generates an error message and aborts the program.
- Axis positions in the same block always have to be programmed according to the expression "AssLogName(...)".

6.10 Calibrating axis kinematics

6.10.1 Backward transformation "ATBWD"

Description: Used to convert coordinates of the workpiece coordinate system (WCS) into real axis positions (ACS). This procedure is called "backward transformation".

Syntax:

AtBwd (<AxCoord>, <Coord>)
with

<Ax-Coord>	Permanent, global or local CPL array. Type: Double Dimension: At least the number of axes in the current channel. After a subroutine call, the individual variables of the array contain the resulting real axis positions in the ACS.
<Coord>	Permanent, global or local CPL array. Type: Double Dimension: At least the number of channel coordinates of the coordinate system generated by axis transformation. The variables of the array have to contain the workpiece coordinates of all coordinates in the WCS involved in the axis transformation.

Tab. 6-19: *AtBwd syntax*

Special features and restrictions: The backward transformation is performed for the axis transformation active at the time of the call. The required lengths and angle parameters are obtained from the machine parameters.

6.10.2 Optimizing parameters "ATCAL"

Description: In combination with the "calibrating axis kinematics" function, this function is used to optimize length and angle parameters specific to axis kinematics.

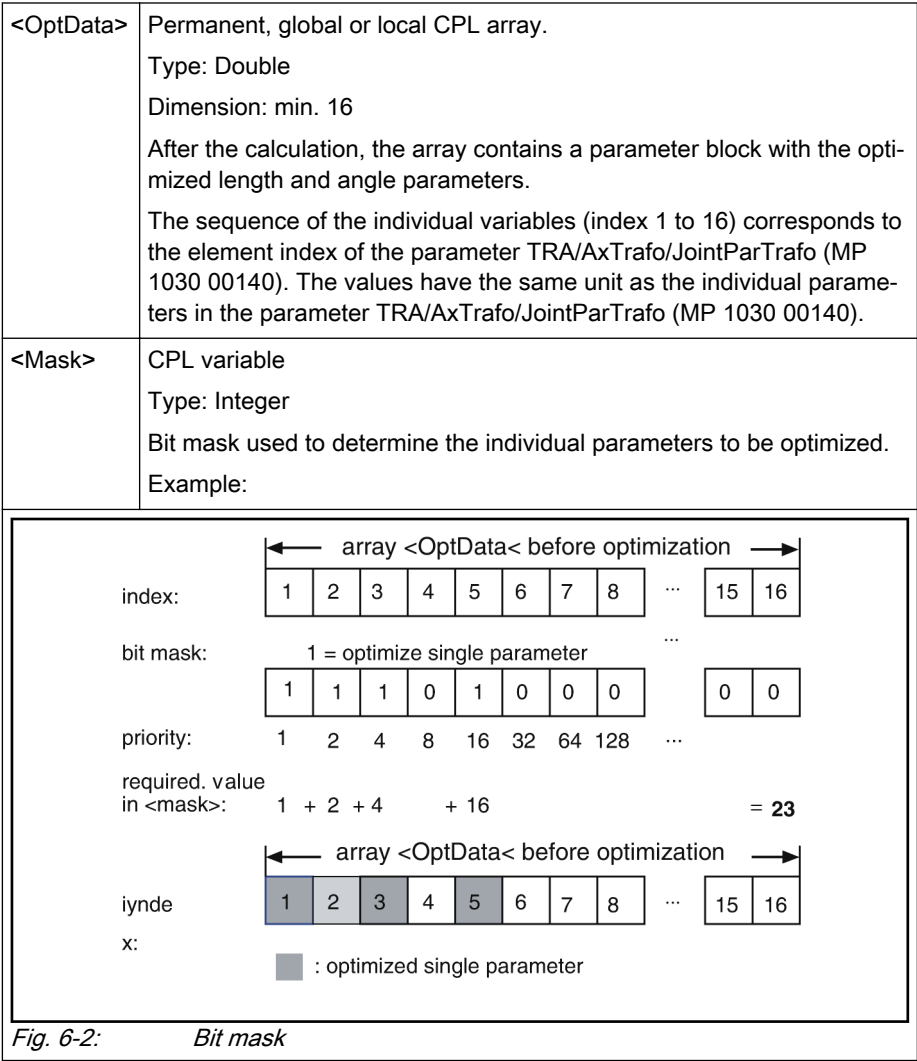
Such data varies with the respective axis transformation type and is contained in the parameter TRA/AxTrafo/JointParTrafo (MP 1030 00140) for every axis transformation. As a rule, it has to be read using the function "ATGET" (see [chapter 6.10.4 "Reading parameter from NC ATGET" on page 217](#)) before optimization and returned to the NC after optimization using the function "ATPUT" (see [chapter 6.10.5 "Writing parameters to the NC ATPUT" on page 218](#)).



For comprehensive information on the calibration of axis kinematics, refer to the documentation "MTX Functional Description".

Syntax:

ATCAL (<File>, <OptData>, <Mask>{, <Info>}{, <NumIt>})	
with	
<File>	Name of the calibration file with or without path specification. If the path is missing, the file is searched according to the parameter NCO/FileOrg/SrchPathSubProg (MP 3080 00001) (search path for sub-routines). <File> contains data required for the optimization. For further information, refer to the documentation "MTX Functional Description".



NC functions with high-level language syntax

<Info>	Permanent, global or local CPL array. Type: Double. Dimension: min 4 After optimization, it contains the following data: <Info>[1] Value of the criterion function (quadratic deviation) before the optimization. Unit: mm <Info>[2] Value of the criterion function (quadratic deviation) after the optimization. Unit: mm <Info>[3] Maximum deviation before the optimization (in mm) <Info>[4] Maximum deviation after the optimization (in mm)
<NumIt>	Maximum number of iteration steps for the optimization. If this is not entered or has a value of "-1", the iteration is only terminated after the deviation of two successively calculated parameter blocks drops below a sufficiently low internal NC threshold (a relevant deviation no longer exists). If "1" is entered, this is a linear computation of adjustment.

Tab. 6-20: ATCAL syntax

Example:	01 DIM PAR! (16)	Dimensions a local CPL array with 16 arrays of the type "Double" (for the optimized length/angle parameters).
	02 DIM GA! (4)	Dimensions a local CPL array with four arrays of the type "Double" (for criterion and deviation).
	03 MASK%=2+4+32	Optimization of individual parameters with the array indices 2, 3 and 6.
	N4 ATCAL (CL.TXT, [PAR!], [MASK%], [GA!], -1)	Start optimization

6.10.3 Forward transformation "ATFWD"

Description: Together with the "calibrating axis kinematics" function, this function is used to convert real axis positions into the coordinates of a coordinate system generated by axis transformation. This procedure is also called "forward transformation".



For comprehensive information on the calibration of axis kinematics, refer to the documentation "MTX Functional Description".

Syntax:

ATFWD (<Coord>, <Axcoord>{, <ParData>})
with

<Coord>	Permanent, global or local CPL array. Type: Double Dimension: At least the number of channel coordinates of the coordinate system generated by axis transformation. The individual variables of the array contain the resulting channel coordinates in the transformed coordinate system after conversion.
<Ax-Coord>	Permanent, global or local CPL array. Type: Double Dimension: At least the number of axes in the current channel. The variables of the array have to contain the real axis positions of all the channel axes participating in the axis transformation.
<ParData>	Permanent, global or local CPL array. Type: Double Dimension: min. 16 The array has to contain a parameter block with all length and angle parameters for an axis transformation. The sequence of the individual variables (index 1 to 16) corresponds to the element index of MP 1030 00140. The values have to have the same unit like the individual parameters in the machine parameter 1030 00140. If this is not programmed, the parameter block of the currently active axis transformation is used. If two axis transformations are active in the control at the same time (AT1, AT2; refer to the function "Coord" (chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245), AT2 data is used.

Tab. 6-21: *ATFWD syntax*

Special features and restrictions: If <ParData> was programmed and no axis transformation is active, a run-time error message is displayed.

6.10.4 Reading parameter from NC "ATGET"

Description: In combination with the "calibrating axis kinematics" function, this function is used to read axis-specific transformation machine parameters.

This data is different for each type of axis transformation. As a rule, it has to be read before the optimization (see function "ATCAL", [chapter 6.10.2 "Optimizing parameters ATCAL" on page 214](#)) and returned to the NC after the optimization using the function ATPUT (see [chapter 6.10.5 "Writing parameters to the NC ATPUT" on page 218](#)).



For comprehensive information on the calibration of axis kinematics, refer to the documentation "MTX Functional Description".

Syntax:

AtGet (<ParData>{, <AxTrafoNo>}{,<MachParNo>})
with

NC functions with high-level language syntax

<ParData>	Permanent, global or local CPL array. Type: Double Dimension: Minimum of 8 (MachParNo = 1030 00130) or Minimum of 16 (MachParNo = 1030 00140). After the execution of the function, the array comprises the currently valid values of the machine parameter "MachParNo" of the axis transformation "AxTrafoNo". The sequence of the individual variables (index 1,2,...) corresponds to the element index of the respective machine parameter. The values have the same unit as the corresponding machine parameter.
<AxTrafoNo>	Number of the axis transformation whose parameters are to be read. If the number is not programmed, the data of the currently active axis transformation is read. If two axis transformations are active in the control at the same time (AT1, AT2; refer to the function "Coord" (chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245), AT2 data is used.
<MachParNo>	Number of the machine parameter which is read. If the number is not programmed, the default value 1030 00140 is assumed, i.e. the length and angle parameters are retrieved.

Tab. 6-22: AtGet syntax

Example:

01 DIM LENPARAM! (16)	Local CPL array with 16 elements for length and angle parameters.
01 DIM ZEROPOS! (8)	Local CPL array with eight elements for the reference positions of the channel axes.
	The following values are read in:
N1 ATGET (LENPARAM!)	MP 1030 00140 of the current axis transformation
N1 ATGET (LENPARAM!, 3)	MP 1030 00140 of axis transformation 3
N1 ATGET (LENPARAM!, 2, 103000140)	MP 1030 00140 of axis transformation 2
N1 ATGET (ZEROPOS!,, 103000130)	MP 1030 00130 of the current axis transformation
N1 ATGET (ZEROPOS!, 3, 103000130)	MP 1030 00130 of axis transformation 3

Special features and restrictions:

- Valid values for MachParNo are 1030 00130 (axis positions of the reference position) and 1030 00140 (length and angle parameters).
- The "ATGET" command causes a runtime error if "AxTrafoNo" is not programmed and no axis transformation (COORD(0.2) and COORD(0.1)) is active.

6.10.5 Writing parameters to the NC "ATPUT"

Description: In combination with the "calibrating axis kinematics" function, this function is used to overwrite axis transformation-specific machine parameters.

This data is different for each type of axis transformation. Generally, it has to be read before the optimization (refer to the function "ATCAL", [chapter 6.10.2 "Optimizing parameters ATCAL" on page 214](#)) (refer to the function "ATGET", [chapter 6.10.4 "Reading parameter from NC ATGET" on page 217](#)) and returned to the NC after the optimization using the function "ATPUT".



For comprehensive information on the calibration of axis kinematics, refer to the documentation "MTX Functional Description".

Syntax:

<code>AtPut (<ParData>{, <AxTrafoNo>},{<MachParNo>})</code>	
with	
<ParData>	Permanent, global or local CPL array. Type: Double Dimension: min. 8 (MachParNo = 1030 00130) or min. 16 (MachParNo = 1030 00140). The array has to comprise a complete parameter block of the machine parameter "MachParNo" of the "AxTrafoNo" axis transformation. The sequence of the individual variables (index 1,2,...) corresponds to the element index of the respective machine parameter. The values have the same unit as the corresponding machine parameter.
<AxTrafoNo>	Number of the axis transformation on whose parameters is to be written. If the number is not programmed, the data of the currently active axis transformation is overwritten. If two axis transformations are active in the control at the same time (AT1, AT2; see function "Coord" (chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245), AT2 data is modified.
<MachParNo>	Number of the machine parameter which is overwritten. If the number is not programmed, the default value 1030 00140 is assumed, i.e. the length and angle parameters are changed.

Tab. 6-23: *AtPut* syntax

Example:

01 DIM LENPARAM! (16)	Local CPL array with 16 elements for length and angle parameters.
01 DIM ZEROPOS! (8)	Local CPL array with eight elements for the reference positions of the channel axes.
01 LENPARAM! (1) = 123.456	Value assignment
:	
01 ZEROPOS! (8) = 1.0	The following values are overwritten:
N1 ATPUT (LENPARAM!)	MP 1030 00140 of the current axis transformation
N1 ATPUT (LENPARAM!, 3)	MP 1030 00140 of axis transformation 3
N1 ATPUT (LENPARAM!, 2,103000140)	MP 1030 00140 of axis transformation 2

```

N1 ATPUT (ZEROPOS!,,      MP 1030 00130 of the current axis transformation
103000130)

N1 ATPUT (ZEROPOS!,      MP 1030 00130 of axis transformation 3
3,103000130)
    
```

- Special features and restrictions:**
- Valid values for MachParNo are 1030 00130 (axis positions of the reference position) and 1030 00140 (length and angle parameters).
 - The "ATPUT" command causes a runtime error if AxTrafoNo is not present and no axis transformation (COORD(0.2) and COORD(0.1)) is active.
 - Executing the function "ATPUT" requires an NC user level that permits the overwriting of machine parameters.
 - The written parameter block becomes active only after a system reset and reprogramming of COORD (<AxTrafoNo>) .

6.11 Executing active auxiliary functions of all groups "AUXFUNC"

Description: Executes the currently active channel-specific and channel-comprehensive auxiliary functions of all existing auxiliary function groups.

The function is relevant in connection with block pre-run (e.g. after processing abort) to restore all auxiliary function states at a certain part program position.

For example, if a part program is prematurely terminated during machining, it is possible to use the block pre-run to restart processing at the block that was being processed at the time of interruption.

Although the part program is restarted at the beginning using the block pre-run, machining on the machine starts at the defined program block.

Since the control neither interpolates nor provides programmed auxiliary functions during block pre-run, all auxiliary function states have to be restored at the end of the block pre-run.



For comprehensive information on the use and parameterization of auxiliary functions, refer to the documentation "MTX Functional Description".

Syntax:

AUXFUNC

Tab. 6-24: AUXFUNC syntax

Example:

```

N100 T102      Selects tool T102

N110 M6        Change tool T102

:

N150 M3        1. spindle/spindle group activation
:              Clockwise rotation
    
```

In N160 cancelled program:

- T102 and M3 were active.
- T102 is changed. The tool position is not changed.

The program is then reselected and runs via block pre-run up to and including N150 without traversing motions. Programmed auxiliary functions are activated, but not output.

Before N160 is processed, AUXFUNC is used now to execute the currently active channel-specific and channel-comprehensive auxiliary functions of all existing auxiliary function groups.

In the example, AUXFUNC causes:

- Activates tool number T102 (tool is still changed).
- 1. spindle/spindle group activation with clockwise rotation

6.12 Changing maximum axis acceleration "AxAcc, AAC", save maximum axis acceleration "AxAccSave, AAS"

- Description:**
- **AxAcc:**
Changes the upper limits of maximum axis accelerations. The function overrides the maximum axis acceleration values from the machine parameters with the programmed values.
 - **AxAccSave:**
Temporarily saves the current maximum acceleration values of all axes in an internal memory.
This internal memory is pre-initialized with the values from the machine parameters whenever the program is selected.

Syntax:

AxAccSave	
Short form: AAS	Temporarily saves the current axis acceleration values.
AxAcc or AxAcc (1)	Reactivates axis accelerations saved earlier using "AxAccSave".
AxAcc (<values> { , <reference> })	
	Changes the maximum axis accelerations
AxAcc () or AxAcc (0)	Activates axis accelerations again from the machine parameters.
Short form: AAC (. .)	
with	
<Values>	Axis name and acceleration value Data for several axes are separated by a comma. Depending on the active measuring unit (G71/G70), the control interprets the programmed data as "1000 Inch/s ² " or "m/s ² ".
<Reference>	Not programmed: • RapidAxAcc > 0.000: The feed acceleration is changed. • RapidAxAcc = 0.000: The acceleration values for feed and rapid traverse mode are to absolute. RAP: The rapid traverse acceleration is changed. ALL: The feed and rapid traverse acceleration are changed.

Tab. 6-25: Syntax of AxAcc, AAC

Example: Initial situation:

NC functions with high-level language syntax

The axes X to Z are pre-assigned with a feed acceleration of 8m/s² and a rapid traverse acceleration of 10 m/s² in the machine parameters.

:	
N40	Max. feed acceleration for x-axis: 1.0 m/s ²
AAC (X1.0, Z2.1)	Max. feed acceleration for z-axis: 2.1 m/s ²
	The max. feed acceleration of the y-axis remains unchanged (8.0 m/s ²).
	The rapid traverse acceleration values remain unchanged on 10 m/s ² .
:	
N80 AxAccSave	Temporarily buffers all currently active maximum acceleration values.
N90 AAC ()	Activates again the values from the machine parameters:
	Max. feed acceleration values (X,Y,Z): 8 m/s ²
	Max. rapid traverse acceleration values (X,Y,Z): 10 m/s ²
:	
N150	Max. feed and rapid traverse acceleration of the y-axis: 5.0 m/s ²
AAC (Y5, ALL)	
	The max. accelerations of the x-axis and the z-axis remain unchanged.
:	
N200 AAC (1)	Activates again the values saved with "AxAccSave":
	Max. feed acceleration for x-axis: 1.0 m/s ²
	Max. feed acceleration for y-axis: 8.0 m/s ²
	Max. feed acceleration for z-axis: 2.1 m/s ²
	All rapid traverse acceleration values are set to 10.0 m/s ² .

- Special features and restrictions:**
- If there are no separate acceleration values set for the rapid traverse operation (AX/Dr[i]/Acc/RapidAxAcc = 0) and the parameter RAP is not programmed, the AxAcc/AAC function changes the acceleration values to absolute for feed and rapid traverse operation. In this case, the behavior is compatible to the MTX version V11 or older.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.13 Loading axis error correction table "AxCompTab"

Description: The NC function allows exchanging the axis error correction tables (LSEC und CCOMP).

If there is a basic table for the axis error correction for an axis and it is identified as LOADABLE, an alternative correction table can be loaded with the "AxCompTab" function.



For further information on the compensation of mechanical inaccuracies using the cross compensation (CCOMP) and the lead spindle screw error compensation (LSEC), refer to the documentation "MTX Functional Description".

Syntax: Activating a reloadable table:

AxCompTab (<CAX> , <Table>)	
with	
<CAX>	Compensated axis Name or number of a system axis.
<Table>	Name (incl. path specification) of a CCOMP or LSEC table

Tab. 6-26: AxCompTab syntax - reloadable table

Activating a basic table:

AxCompTab (<CAX> , <RAX>)	
with	
<CAX>	Compensated axis Name or number of a system axis.
<RAX>	Reference axis Name or number of a system axis. CAX and RAX are equal: LSEC table CAX and RAX are different: CCOMP table

Tab. 6-27: AxComp tab syntax - basic table

Activating all basic tables:

Activating all basic tables	
AxCompTab (<CAX>)	
with	
<CAX>	Compensated axis Name or number of a system axis.

Tab. 6-28: AxComp table syntax - all basic tables

Example: The basic table "cnc03r02.tab" contains only the keywords to enable the reloadable tables and not the correction values.

The table "Reload0302.tab" is only activated via the NC function "AxCompTab" if required.

The system axis 2 has the name YA and axis 3 the name ZA.

Basic CCOMP table: cnc03r02.tab

```
#STUP
LOADABLE
EMPTY
#ESTUP
```

Loadable CCOMP table: Reload0302.tab

#CONF

CAX 3 ; ZA is the compensated axis

RAX 2 ; YA is the reference axis

#ECONF

-20000.0 50000.0 ; Starting value -20mm, step width 50 mm

7.0 7.0 ; Compensation for -20 mm

9.0 9.0 ; Compensation for 30 mm

9.8 9.8 ; Compensation for 80 mm

9.0 9.0 ; Compensation for 130 mm

7.0 7.0 ; Compensation for 180 mm

4.0 4.0 ; Compensation for 230 mm

0.0 0.0 ; Compensation for 280 mm

:

N010 AxCompTab(ZA, Re-
load0302.tab)

The CCOMP table Reload0302.tab becomes ac-
tive.

:

N200 AxCompTab(ZA, YA)

The CCOMP table cnc03r02.tab becomes active

:

Special features and restrictions:

- The tables of axes of the individual channel and of asynchronous axes can be exchanged.
- The name of the active table is stored in a component of the system data "SysAxActData[]".
- An axis motion takes place when changing the table!
When loading a table, the motion is limited to a maximum of 2 mm. If a greater motion is required, a runtime error is output.
When returning to the basic table(s), the motion is not limited.
- The "AxComTab" function is **not executed in the block pre-run**.

6.14 Axis coupling "AxCouple, AXC"

Description: The axis coupling is used to create a specific relationship between the motion of a synchronous "master axis" and of one or more (max. 7) synchronous "slave axes".

If the slave axes are not in the coupling position when the coupling is activated, they are moved to this position by an internally generated linear traversing motion. The active path feed and the override potentiometer are applied.

If the master axis travels, all the slave axes move automatically according to their own defined relationship to the master axis. The combination of the master and all slave axes involved is also called a "group of coupled axes".

Possible relationships between the master and a slave axis:

- The command positions of the master axis are transformed using an **constant offset** to the corresponding command positions of the slave axis (see formula 1).

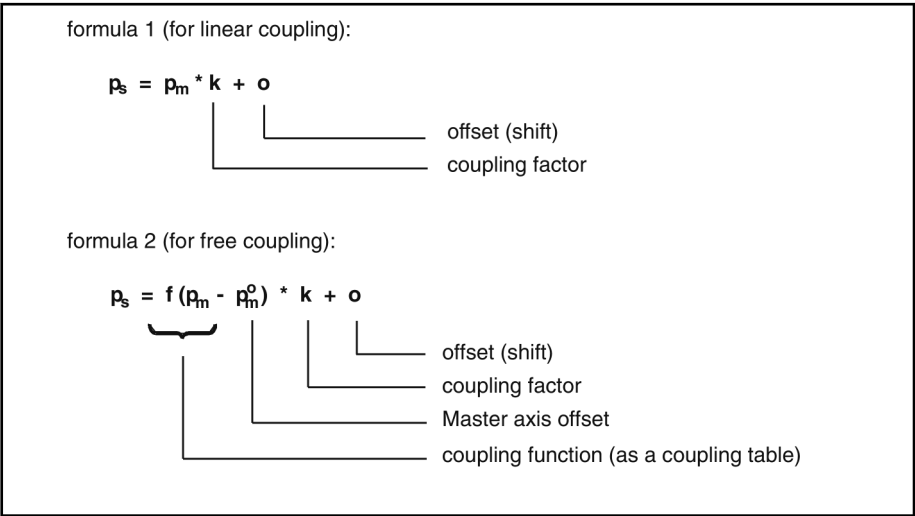
This can be used to offset the slave axis against the master axis by any constant path in positive or negative travel direction.

- The command positions of the master axis are transformed using an **constant coupling factor** to the corresponding command positions of the slave axis (see formula 1).

This can be used to move the slave axis proportionally against the master axis.

- The target positions of the master axis are transformed using a (coupling) table to **any** corresponding target position of the slave axis. Here, support points that provide the corresponding slave axis position for a master axis position and – if required – a master axis offset are stored to the table (see formula 2).

The control can determine positions between individual data points using the cubic spline function.



NC functions with high-level language syntax

<M>	Channel axis name of master axis
<Variant>	0 generates a new group of axes 1 changes the group of axes (adds new slave axes or changes coupling name) -1 deletes the slave axis or completely cancels the group of coupled axes
<S>	Channel axis name of slave axis If <S> cannot be differentiated from the programming of an axis/coordinate with a value, an equal sign must be added in between "(" and "...<S>= (...)" See also the following example.
<SO>	<i>Offset of the slave axis:</i> • Coupling distance in mm or inches • ACT for a coupling at the current position If this is not programmed, <SO> = 0.
<SF>	Coupling factor of the slave axis. If this is not programmed, <SF> = 1.
<MO>	Master axis offset Relevant only in case of "free coupling" (see formula 2 above). If this is not programmed, <MO> = 0.
<Tab>	Name of the coupling table of the corresponding slave axis. Relevant only in case of "free coupling" (see formula 2 above).

Tab. 6-29: Syntax of AxCouple

Examples:

N100 AXC (C0, A () , B ())	Create axis group C: Master axis A/B: Slave axes Both slave axes run with "linear" coupling. The default values "coupling distance 0.0" and "coupling factor 1.0" are applied.
N100 AXC (C0, A (ACT) , B (ACT, 2))	Create axis group C: Master axis A/B: Slave axes Both slave axes run with "linear" coupling. A and B are coupled to C without approaching motion. B is operated with coupling factor 2.

```

N100 AXC(C0,B(,
0.5,,T_B))
:
N150 AXC(C1,A(4.5,2))
:
N200 AXC(C-1,A())
:
N200 AXC(C-1)
:

```

Create axis group
C: Master axis
B: Slave axis
Slave B runs with "any" coupling.

Extending a group of coupled axes
A is added as slave axis.
Slave axis A runs with "linear" coupling.
Slave axis B still runs with "any" coupling.

Slave axis A is removed from axis group c.

The entire axis group c is deleted.

Another example:

Master axis X slave axis Y2

The slave axis has the ambiguous axis name Y2 which does not differ from programming the y-axis with the value 2.

AXC must be programmed as follows:

```
AXC(X,Y2=(...))
```



The function triggers a traversing motion of all the slave axes programmed in this block!

They travel to their specific coupling position (master value) defined by the position of the master axis and the coupling characteristics.

Special features and restrictions:

- All axes participating in a group of coupled axes must be synchronous axes, at least during the axis coupling.
Asynchronous axes or Hirth axes as slave axes a group of coupled axes are not admissible.
- Programming a traversing motion of slave axes is not permitted; this leads to an error message.
- A slave axis cannot simultaneously be a master axis in another group of coupled axes.
- The end of program does not automatically disband a group of coupled axes.
- If the master axis is a modulo axis, the slave must also be a modulo axis in the case of a linear coupling relationship.
- To allow axis-by-axis traveling to the reference point, the group of coupled axes has to be opened.
- The permitted traversing range of the master axis can be reduced by coupled slave axes, (e.g. if the slave axis reaches its end ranges more quickly than the master axis or if the traversing range of the slave axis is less than that of the master axis).
- The maximum dynamics of the "weakest" axis determines the maximum dynamics of the complete group of coupled axes.

- Locking axes when a group of coupled axes is active is prohibited.
- Axes coupled in test mode must be uncoupled before the test mode is switched off.

6.15 Axis feed coupling "AxisFeedCouple, AFC"

Description: Derives the feed for an asynchronous axis from the velocity of a reference axis or a reference spindle. For example, this function allows specifying a feed per revolution (G95) for asynchronous axes. The coupling is effective both in positioning mode and speed mode. The resulting feed is calculated as follows:

$$\text{Feed}[\text{AsynchronousAxis}] = |\text{Feed}[\text{ReferenceAxis}]| * \text{Coupling factor}$$

The feed can additionally be scaled via the axis override.

Syntax:

AxisFeedCouple, AXISFEEDCOUPLE	
	Activate/deactivate feed coupling
Short form: AFC (. . .)	
AFC (FRA=<ReferenceAxis>, FA=<FeedAxis>, CF=<Factor>)	
Feed coupling parameters:	
	Feed coupling with system axis reference
AFC (FRA=<Axis>, FA=<Axis>, CF=<Factor>)	
	Feed coupling with channel spindle reference n=1..8
AFC (FRA=Sn, FA=<Axis>, CF=<Factor>)	
	Feed coupling with system spindle reference nn=00..32
AFC (FRA=<System spindle>, FA=<Axis>, CF=<Factor>)	
G95	Feed coupling with current G95 reference
AFC (FA=<Axis>, CF=<Factor>)	
AFC (FA=<Axis>)	Disable the feed coupling for <axis>
AFC ()	Deactivate feed couplings of all system axes
	The control continues an active axis interpolation with the programmed feed (S-/FA-value) or the parameterized feed (max. axis velocity).
with	
FRA=<Reference axis>	The reference axis can be a system axis, a channel spindle (S1..S8) or a system spindle. If there is no parameter, the control stores the active G95 reference spindle or the G95 reference axis.

FA=<Feed axis>	Asynchronous linear, rotary or endless axis
CF=<Factor> or CF=S(<Factor>)	Scales the reference axis velocity to the requested feed rate. The reference axis velocity for rotary axes and spindles is in °/min and for linear axes in mm/min. The following applies to combined rotary/spindle linear axes: 1 °/min corresponds to 1 mm/min. If an endless reference axis is the slave axis of a system axis coupling, a "modified" speed can be taken into account for a uniform feed per revolution by means of the "S (<Factor>)" attribute. Positioning axis or table portions of the system axis coupling are not contained in the "modified" speed. <Factor> must be greater than 0. The feed coupling factor can be dynamically changed with the "G94/G95(DSi=) Incremental velocity programming" functions.

Tab. 6-30: Syntax AxisFeedCouple, AFC



If feed coupling is deactivated, axes can perform unintentional motions!

Feed axes coupled to a reference axis at standstill can be moved again by deactivating the coupling.

Complete asynchronous traversing motions with the system home position or stop endless axes in speed mode by specifying S(0) before deactivating the coupling.

Example:

```

;Gear shaping:
;Z3 stroke axis, C1 tool axis, feed gear FG [mm/Z3 revolution]
;One Z3 revolution corresponds to the double stroke (DS) of the shaping tool
;Tool diameter 100 mm
;Feed gear 0.2 mm/DS
1 PI=3.141593 : DIA=100 : FW=0.2
;Coupling factor=FW/C1 circumference

1 FC=FW/
(PI*DIA)

;Feed coupling for C1 axis
N20 AFC (FRA=Z3, FA=C1, CF=[FC])
N30 C1=S(-1) ;Start feed motion: C1=S(-(Z3*FC))
...

; feed per revolution for asynchronous linear axis VA
N50 G95 F0.5 ;Activate channel feed per revolution

;Feed per revolution for asynchronous
;VA axis: 1 mm/rev

```

NC functions with high-level language syntax

```
N60 VA=10 AFC (FA=VA, CF=[1/360])
```

```
...
```

```
; Feed per revolution of gear axis C2=SAC(B2,X,Z)
```

```
;VA feed 1 mm/C2 revolution,
```

```
;with B2 portion but without X and Z portions
```

```
N80 VA=0 AFC (FRA=C2, FA=VA, CF=S ([1/360]))
```

```
...
```

```
;Feed coupling with path-dependent speed difference
```

```
N100 G1
```

```
G95 (FRA=B1) ;G95 with 1.5 mm/B1 revolution
```

```
F1.5
```

```
;B2 with B1 speed in opposite
```

```
;direction (1:1 coupling)
```

```
N110 AFC (FA=B2, CF=1.0) B2=S (-1)
```

```
;Increases B2 speed until block end
```

```
; (1:2 coupling)
```

```
N120 G95 (DSA1=B2, DS=1.0) X10
```

```
...
```

```
;Deactivates coupling for single axis
```

```
;Stops C1 axis and
```

```
N150 C1=S (0)
```

```
AFC (FA=C1) ;deactivates feed coupling
```

```
...
```

```
;Deactivates feed coupling of all system axes
```

```
;Deactivates B2 and VA feed coupling,
```

```
N170 AFC () ;Changes B2 speed to 1 rpm
```

Special features and restrictions:

- A simultaneously programmed asynchronous feed (FA) limits the maximum coupling feed.
- If the feed coupling is deactivated in speed mode or during positioning, the programmed S-value or FA-value is activated. The motion continues at different feed.
- The feed coupling remains modally after the motion. If it is reprogrammed, the axis travels with the existing coupling.

- If an endless axis is operated in speed mode and feed coupling is activated, the control only evaluates the sign of the S-value for the direction of rotation. The speed is determined by the coupling specified.
- Enabling or disabling the feed coupling does not start or stop the feed axis, but switches the flying feed with the allowed axis acceleration.
- The actual velocity of reference spindles and the command velocity of reference system axes are used for coupling.
- Feed coupling does not ensure positional synchronism of the coupled axes.
- Feed coupling has no effect in setup mode.
- The channel feed switchover (G93/G94/G95) does not have any effect on active axis feed couplings.

6.16 Changing the maximum axis jerk "perform, AJK", save maximum axis jerk "AxJerkSave, AVS"

- Description:**
- **AxJerk:**
Changes the upper limits of maximum axis jerks. The function overrides the maximum axis jerk values from the machine parameters with the programmed values.
 - **AxJerkSave:**
Temporarily saves the current maximum jerk values of all axes to an internal memory.
This internal memory is pre-initialized with the values from the machine parameters whenever the program is selected.

Syntax:

AxJerkSave Short form: AJS	Temporarily buffers the current jerk values.
AxAcc or Ax-Acc (1)	Activates again jerk values saved earlier using "AxJerkSave".
AxJerk (<Values> {,<Reference>})	
	Modify maximum axis jerk values
AxJerk () or Ax-Jerk (0)	Activates again axis jerk values from the machine parameters.
Short form: AJK (. .)	
with	

NC functions with high-level language syntax

<Values>	Axis name and jerk values Data for several axes are separated by a comma. Depending on the active measuring unit (G71/G70), the control interprets the programmed data as "1000 inch/s³ or "m/s³".
<Reference>	<i>Not programmed:</i> • RapidAxJerk > 0.000: The feed jerk is changed. • RapidAxJerk = 0.000: The jerk values for feed and rapid traverse mode are changed to absolute. RAP: The rapid traverse jerk is changed. ALL: The feed and rapid traverse jerk are changed.

Tab. 6-31: Syntax AxJerk, AJK

Example: Initial situation:

The axes X to Z are pre-assigned with a feed jerk of 40 m/s³ and a rapid traverse jerk of 60 m/s³ in the machine parameters.

```

:
N10 JKC (1)           Enable jerk limitation
:
N40
AJK (X30.0, Z20.0)    Max. feed jerk for x-axis: 30.0 m/s3
                      Max. feed jerk for z-axis: 20.0 m/s3
:
                      The max. feed jerk of the y-axis remains unchanged (40
                      m/s3).
                      The rapid traverse jerk values remain unchanged on 60
                      m/s3.
N80 AxJerkSave        Temporarily buffers all currently active maximum jerk val-
                      ues.
N90 AJK ()            Activates again the values from the machine parameters:
:
                      Max. feed jerk values (X,Y,Z): 40.0 m/s3
                      Max. rapid traverse jerk values (X,Y,Z): 60.0 m/s3
N150 AJK (Y35, ALL)   Max. feed and rapid traverse jerk of the y-axis: 35.0 m/s3
:
N200 AJK (1)          Activates again the values saved with "AxJerkSave":
                      Max. feed jerk for x-axis: 30.0 m/s3
                      Max. feed jerk of the y-axis: 40.0 m/s3
                      Max. feed jerk for z-axis: 20.0 m/s3
                      All rapid traverse jerk values are set to 60.0 m/s3.

```

- Special features and restrictions:**
- If there are no separate jerk values set for the rapid traverse operation (AX/Dr[i]/Acc/RapidAxJerk = 0) and the parameter RAP is not programmed, the AxJerk/AJK function changes the jerk values to absolute for feed and rapid traverse operation. In this case, the behavior is compatible to the MTX version V11 or older.

- The jerk limit values have only an effect in case of an enabled jerk limitation (JerkControl/ JKC).
- If both a physical and a logical axis of the active channel exist under the same name in the system, the jerk of the logical axis is always influenced.

6.17 Switching off the C-axis mode for spindles "AxisToSpindle, ATS"

Description: Switches a spindle that is in the C-axis mode (see [chapter 6.132 "Enabling C-axis mode for spindles SpindleToAxis, STA" on page 380](#)) to the spindle mode.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

AxisToSpindle (<PAN> <PAI>{, <PAN> <PAI>}...)

Short form: ATS (. .)

with

<PAN>	Physical axis name Specifies the spindle that is to be switched from the C-axis mode back to spindle mode.

<PAI>	Physical axis index Same effect as <PAN>

Tab. 6-32: Syntax of AxisToSpindle

Example:

```

:
N030 ATS (CH)   The physical axis CH (i.e. the spindle with the name CH during
:               axis operation) is switched to spindle operation.

```

Special features and restrictions:

- A given axis must be at a standstill and must not belong to an axis group.
If this is not the case, the control generates an error message and aborts the program.
- Axis positions in the same block must always be programmed according to the expression AxisToSpindle(...).

6.18 Switching off the C-axis mode for spindles and wait "AxisToSpindleWait, ATSW"

Description: Switches a spindle that is in the C-axis mode to the spindle mode and waits until the spindle is active (see [chapter 6.132 "Enabling C-axis mode for spindles SpindleToAxis, STA" on page 380](#) and [chapter 6.133 "Enabling C-axis mode for spindles and wait SpindleToAxisWait, STAW" on page 381](#)).



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

AxisToSpindle (<PAN> <PAI>{, <PAN> <PAI>}...)

Short form: ATSW (. .)

with	
<PAN>	Physical axis name Specifies the spindle that is to be switched from the C-axis mode back to spindle mode.
<PAI>	Physical axis index Same effect as <PAN>

Tab. 6-33: Syntax of AxisToSpindle

Example:

```

:
N030 ATSW (CH)    The physical axis CH (i.e. the spindle with the name CH during
                  axis operation) is switched to spindle operation.
N040 M3 S1000     The spindle is started.
:

```

Special features and restrictions:

- A given axis must not belong to an axis group.
If this is not the case, the control generates an error message and aborts the program.
- Axis positions in the same block must always be programmed according to the expression AxisToSpindleWait(...).

6.19 Replacing axes "AxisReplace, AXR "

Description: After the block pre-run in the "Target adjustment program", this parameter allows entering the part program with a configuration differing from the calculated axis configuration, by replacing axes with each other that are similar in their function.

Syntax:

AXR(<KAN1>, <PAN2>)	The axis with the channel axis name <CHAN1> is removed from the channel, the axis with the physical axis name <PA2> is added to the channel and receives the channel axis name <CHAN1>.

Tab. 6-34: Syntax of AxisToSpindle

Example:

AXR(A,A2)

The axis with the channel axis name A is removed from the channel, the axis with the physical axis name A2 is added to the channel and receives the channel axis name A.

Special features and restrictions:

- Axes can only replace each other if they are of the same type.
- Axes having a machining function cannot be replaced.
- A channel axis name must be available and programmed for the axis to be replaced
- The Replace Axes function can only be programmed in the "Target adjustment program" and an Init adjustment program must be configured at the same time.
- The axis configuration must be adjusted before AXR is called.
- Only one AXR command may be programmed per NC line.
- After axes have been replaced, it is no longer allowed to program a WAX, RAX, GAX, RLN, or ALN command.
- No more than a total of 4 axes may be replaced.

6.20 Changing maximum axis velocity "AxVel, AVE", Save maximum axis velocity "AxVelSave, AVS"

Description: AxVel:

This temporarily changes the upper limits of the axis velocities. The function overrides the maximum axis velocities values from the machine parameters with the programmed values.

AxVelSave:

Saves the current maximum velocities values of all axes to an internal memory. This internal memory is pre-initialized with the values from the machine parameters whenever the program is selected.

Syntax:

AxVelSave	Saves the currently active maximum axis velocity values.
Short form: AVS	
AxVel (<Values>)	Change maximum axis velocity
AxVel () or AxVel (0)	
	Reactivates the maximum axis velocity from the machine parameters.
AxVel (1)	Activates the maximum axis velocity saved by AxVelSave.
AxVelSave	Saves the currently active maximum axis velocity values.
Short form: AVE (. .)	
with	
<Values>	Axis name and velocity value

Tab. 6-35: Syntax of AxVel, AxVelSave

Data for several axes are separated by a comma.

Depending on the active measurement unit (G71/G70), the control interprets the programmed data for linear axes as "inches/min" or "mm/min".

Rotary axes are always specified in "1000 degrees/min".



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

Example:

Initial situation:

The default value for axes X to Z in the machine parameters is 50000 mm/s.

:

N40

max. velocity X-axis: 10000.0 mm/s

AVE (X10000, Z30000)

max. velocity Z-axis: 30000.0 mm/min

0)

The max. velocity of the y-axis remains unchanged (50000.0 mm/min).

:

N80 AVS

Temporarily buffers all currently active maximum velocity values.

:

N90 AVE () Reactivates the values from the machine parameters.
:
N200 AVE Reactivates the values saved by means of "AVS/AxVel-Save":
:
 max. velocity X-axis: 10000.0 mm/s
 max. velocity Z-axis: 30000.0 mm/min
 max. velocity Y-axis: 50000.0 mm/min

6.21 Parameters of the B-spline approximation "BsaPar, BAP"

Description: Setting parameters of the B-spline approximation.

Syntax:

BsaPar (E.., OE..)	
E	Radius of the tolerance margin Switch-on value: Parameter SPL/BSplineApproxTol (MP 8007 00030)
OE	Tolerance value for the orientation vector Switch-on value: OE = E
BsaPar (0) or BsaPar ()	Resetting B-spline approximation to switch-on values
Short form: BAP	

Tab. 6-36: BAP syntax

Example: Setting parameters of the B-spline approximation

```
SPD(4233,x,y,z,O)      ;It defines a spline type activated later on with G6
BAP(E0.02,OE0.1)      ;It sets the parameters of the B-spline approximation:
                        ;Tolerance 0.02 mm
                        ;Tolerance orientation 0.1 mm
G6 x.. y.. z.. phi.. theta.. ;B-spline approximation (spline 4233) on
...
BAP()                  ;Switching back to switch-on values:
                        ; Tolerance from the parameter SPL/BSplineApprox-
                        ; Tol (MP 8007 00030)
...
G1 x.. y.. z.. phi.. theta.. ;B-spline approximation off
```

6.22 Placement: Workpiece position correction "BcsCorr, BCR"

Description: Serves as setting-up correction.

The placement "workpiece position correction" can shift **and** orient the workpiece coordinate system anywhere in space. The workpiece position correction affects the coordinates with the meanings "X", "Y" and "Z" in the corresponding channel.



The axis names "X", "Y" and "Z" used in the following refer to the axes with the **meaning** X, Y and Z.

The setup effort is significantly reduced because the workpiece position is measured after spanning and taken into account using workpiece position correction.

Since there are 3 degrees of freedom for orientation, every orientation can be represented by 3 consecutive basic rotations. For the sake of readability, the following figure shows only the basic rotation about the z-coordinate:

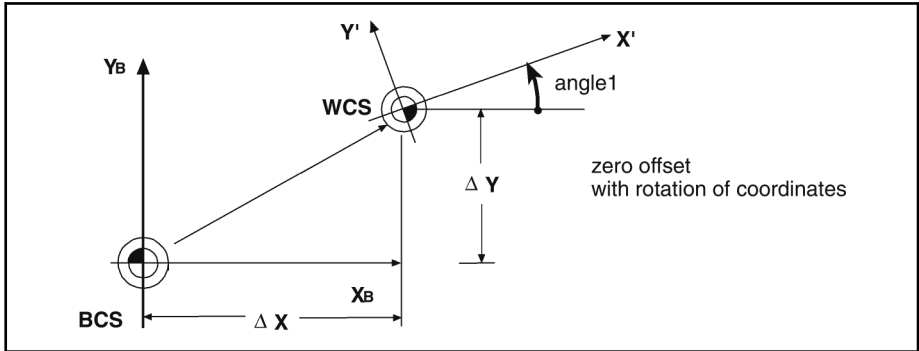


Fig. 6-4: Basic rotation around the z-coordinate



The placement "inclined plane" (see [chapter 5.2.44 "Placement: Inclined plane G151 - G159.5" on page 173](#)) is identical in terms of function. A representation of all 3 basic rotations can also be found there.

Additional placements (e.g. inclined plane) have an additive effect. In the "Chain of calculation" is located in front of the inclined plane:

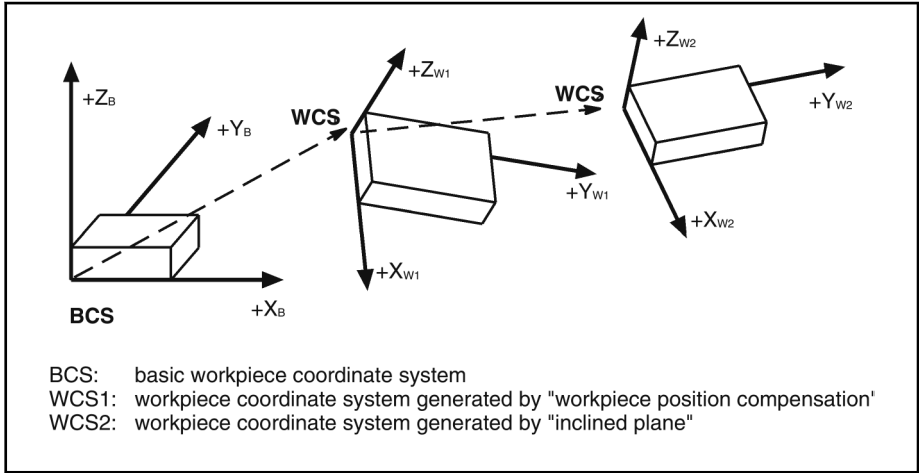


Fig. 6-5: Chain of calculation

Syntax:

<code>BcsCorr ({<X_w-Offset>},{<Y_w-Offset>},{<ZW-Offset>},{<Angle1>}{, {<angle2>},{<angle3>}}})</code>	
	Workpiece position correction ON
<code>BcsCorr ()</code> or <code>BcsCorr (0)</code>	Workpiece position correction OFF
Short form: <code>BCR (. .)</code>	

with	
<X _W offset>	Offset value in the direction of the primary coordinate X
<Y _W offset>	Offset value in the direction of the secondary coordinate Y
<Z _W offset>	Offset value in direction of the normal coordinate Z
<Angle1>	Rotating angle along the z-coordinate Range of values $0 \leq \text{<Angle1>} < 360$ degrees
<Angle2>	Angle of rotation along the new Y'-coordinate Range of values: $0 \leq \text{<Angle2>} < 180$ degrees
<Angle3>	Rotating angle along the new Z"-coordinate Range of values $0 \leq \text{<Angle3>} < 360$ degrees

Tab. 6-37: Syntax of BcsCorr (BCR)

Example:	N70 G40	Milling cutter path correction OFF
	N80	Workpiece position correction ON
	BCR(50,300,10, 1.23)	The zero point of the new workpiece coordinate system is located in the BCS on X50 Y300 and Z10. In relation to the BCS, the X and y-coordinate axes of the new workpiece coordinate systems are rotated counter-clockwise by 1.23 degrees around the new z-coordinate axis.
	:	
	:	
	N200 BCR()	Workpiece position correction OFF

- Special features and restrictions:**
- Activating and deactivating the workpiece position correction interrupts the block look-ahead; therefore, it must not be programmed when cutter path correction is active (G41/G42, see [chapter 5.2.16 "Milling cutter path correction G40, G41, G42" on page 135](#)).
 - The switched-on state and the behavior in case of a control reset are configured in the machine parameters 7060 00010 and 7060 00020.

6.23 Next block monitoring with buffered NC block input "ChkNxtBlk, CNB"

Description: With buffered NC block input, the ChkNxtBlk function can be used for monitoring the data flow.

It is checked whether the buffered NC block input is able to provide the next block in time. If this is not the case, a runtime error is output.



For information on the OPC interface see the documentation "MTX OPC Communication".

Syntax:	ChkNxtBlk(1)	Activates the next block monitoring.
	ChkNxtBlk(0) or ChkNxtBlk()	Deactivates the next block monitoring.
	Short form: CNB(. . .)	

Tab. 6-38: Syntax ChkNxtBlk (CNB)

- Special features and restrictions:** The function is deactivated by a reset.

6.24 Chamfer programming "ChLength", "CHL", "CHLL", "ChSection", "CHS", "CHSL"

Description: The "chamfer programming" function inserts a transition phase between two consecutive NC blocks of the type "straight line" or "circle," the length of which can be specified as absolute **chamfer length** or as **length of the chamfer segment**. The chamfer is generated within the active working plane.

The "Chamfer programming" function is provided either modally (ChLength, CHL, ChSection, CHS) or block by block (CHLL, CHSL). When programming block by block, a local feed can be defined used to machine the chamfer.

The following chamfer transitions are possible:

- **Chamfer between two abutting straight lines**

The chamfer is at right angle to the bisector between the neighboring path segments. The length of the chamfer is automatically corrected (reduced) when there is no intersection with the neighboring programmed path segments.

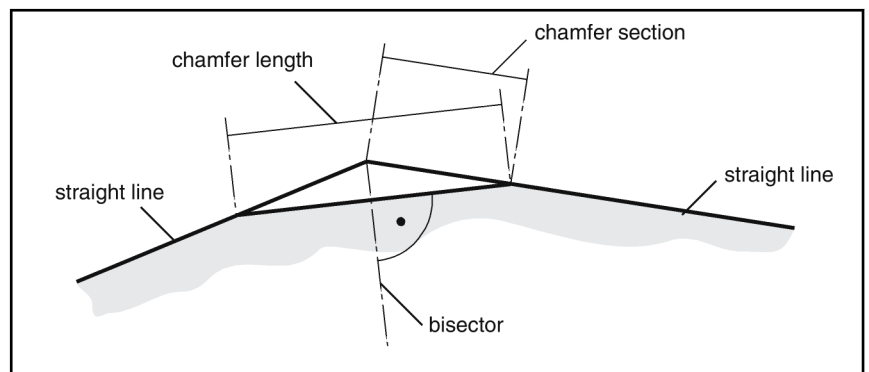


Fig. 6-6: Chamfer between two abutting straight lines

- **Chamfer between two abutting circle segments**

In case of contour transitions involving circle segments, the dimensions of the chamfers refer to the respective end or starting tangent of the path segments involved in the contour transition. The actual resulting chamfer length is strongly dependent on, among other things, the radii of the circles involved and thus deviates more or less from the programmed dimensions.

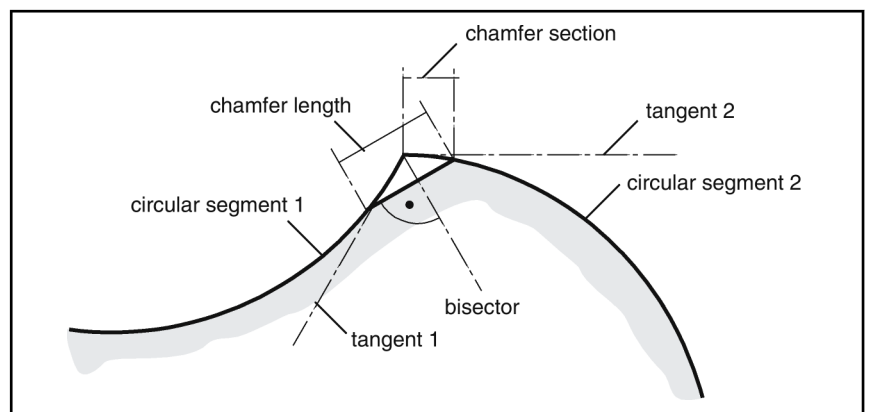


Fig. 6-7: Chamfer between two abutting circle segments

NC functions with high-level language syntax

Syntax:

ChLength (<ChamferLength>)	Chamfer programming ON Specifies the desired <ChamferLength> in mm (G71) or inches (G70).
ChLength() or ChLength(0)	Chamfer programming OFF
Short form: CHL(. .)	
ChSection (<ChamferSegment>)	
	Chamfer programming ON Specifies the desired <ChamferSegment> in mm (G71) or inches (G70).
ChSection() or ChSection(0)	Chamfer programming OFF
Short form: CHS(. .)	

Tab. 6-39: of ChLength (CHL) syntax and ChSection (CHS) syntax

CHLL (<Length> {, <LocFeed> })	Block by block chamfer programming with chamfer length specification
CHSL (<Section> {, <LocFeed> })	Block by block chamfer programming with chamfer segment specification
with	
<Length>	Specifies the desired <ChamferLength> in mm (G71) or inches (G70).
<LocFeed>	Local feed used to travel the transitional chamfer.

Tab. 6-40: CHLL syntax and CHSL syntax

Special features and restrictions:

- The functions "ChLength", "ChSection", "RoundEps" and "Rounding" **act modally** and deselect each other.
- The functions "CHLL" and "CHSL" **act locally** (block by block).
- If either a chamfer or a rounding is modally active and if a local chamfer is programmed at the same time, the local function cross-fades the modal function block by block.
- The chamfer exclusively refers to the active working plane (G17, G18, G19, G20). If additional axes are involved in the motion, the chamfers remain unaffected.
As the coordinates of the programmed traversing blocks of the axes within the working plane are manipulated by the chamfers, but the values for the axes outside the working plane remain unchanged, the direction of straight lines in space can change for example.
- The function does only have an effect in the operating area of the "processing" under "sequential block", "single block" or "single step".
Since "program block" acts like manual data input, chamfer programming is not effective.
- The switch on/off response as well as the response at control reset is determined by the init strings in machine parameters 7060 00010 and 7060 00020 .

- If the plane, a zero offset or an axis transformation is switched between two subsequent blocks, no chamfer is generated.

6.25 Automatic detection of corners and lines "CLD"

Description: Depending on programmed parameters, the "automatic detection of corners and lines" function generates corners and straight lines within spline sequences. If a knee angle is greater than the programmed limit angle, a programmed knee is retained, e.g. the angle is not rounded by the spline curve. If the point distance is greater than the programmed limit distance in a block, a linear spline is generated. In addition, angle and distance factors may be defined (only spline type 4). If an angle or a distance exceeds its average value in the look-ahead range, the function also generates a knee or a straight line. Furthermore, a corner tolerance value can be specified. If a rounding spline with this tolerance can be inserted at a block transition so that at least the half of one of the two sides remains, the distance condition for this spline is met. The function applies only if G6 (spline interpolation) is active. It is effective with all spline types except spline coefficient programming.

Syntax: The automatic detection of corners and lines is enabled using the **CornerLineDetection** syntax

CornerLineDetection(ANG<a>,AFACT<fa>,DIST<d>,DFACT<fd>)	
The parameters have the following meaning:	
ANG<a>	"a" is the maximum permissible knee angle in the contour. Knee angles greater than a fulfill the angle condition.
AFACT<fa>	Spline type 4 only: "fa" is the angle factor Knee angles exceeding the average knee angle by the factor "fa" fulfill the angle condition.
DIST<d>	"d" is the maximum permissible point distance. Distances greater than "d" fulfill the distance condition.
DFACT<fd>	Spline type 4 only: "fd" is the distance factor Point distances exceeding the point distance by the factor "fd" fulfill the distance condition.
E<e>	"e" is a contour accuracy value If a rounding spline with this tolerance can be applied at a corner so that at least the half of one of the two sides remains, the distance condition for this spline is met.

Tab. 6-41: Syntax of CornerLineDetection

Parameters can be omitted individually. The DIST, DFACT and E as well as the ANG and AFACT conditions are each linked by a logical "OR". After programming CLD with at least one parameter, the function is modally switched on and is active as soon as G6 is active (exception: spline coefficient programming). The parameters can be changed at any time by reprogramming.

The short form is CLD(...)

It is switched off by programming without parameters CornerLineDetection().

Example: c2-constant cubic spline

```

SPD(2203,x,y,z,O) CLD(ANG20,AFACT5,DIST5)
...
G6 x.. y.. z.. phi.. theta..      ;spline 2203 on; absolute distance condition active;
                                   ; absolute and relative angle condition active

x.. y.. z.. O(..)
...
x.. y.. z.. O() CLD(DIST8)        ;Spline off
G1                                ;deactivates angle conditions,
                                   ;new absolute distance condition

...
G6                                ;Spline on, the last CLD(DIST8) is valid

```

6.26 Collision monitoring "Collision, CLN"

Description: *Allows collision monitoring of milling cutter path correction G41/G42 to be*

- switched on or off and
- adapted to the application in terms of the look-ahead range and the behavior in case of a collision.

The following applies:

- Collision monitoring is only applied if the milling cutter path correction G41/G42 is active, even if the correction radius value is "0".
- If the current radius correction value does not permit machining of individual contour elements, the control tries to modify the related path to avoid damage to the contour.
- During the course of the contour, the collision monitor takes only the coordinates of the active working plane into account. If collisions are prevented by the program, e.g. by changing the tool infeed depth, collision monitoring is nevertheless activated within the current look-ahead range.

In such cases, collision monitoring can temporarily be switched off in the affected machining section.



For detailed information on the "collision monitoring" function, refer to the documentation "MTX Functional Description".



We recommend to enter the desired switch-on behavior of collision monitoring in MP 7060 00010 / MP 7060 00020.

Syntax:

Collision(1) or CLN(1) or CLN	Collision monitoring ON Retains the current behavior in the case of a collision. If the behavior has not been programmed or entered in MP 7060 00010 / MP 7060 00020 until now, the behavior corresponds to that of Collision (CollErr 0). CLN ("CLN without brackets") corresponds to CLN(1) and/or Collision(1).
Collision() or Collision(0)	Collision monitoring OFF
Collision(DEF)	Specifies the presetting for the look-ahead range of collision monitoring to 2 blocks.
Collision(CollErr<Art>)	
	Activates the collision monitor and specifies the behavior in the case of a collision.
Collision(DLA<blocks>)	Specifies the preset for the look-ahead range of collision monitoring. It is applied the next time that G41/G42 is programmed.
Collision(LA<blocks>)	Temporarily changes the look-ahead range until the next time that G41/G42 is programmed.
Short form: CLN(...)	
with	
<Art>	Specifies the behavior of the control when a collision is detected: 0: neither runtime errors nor warnings are sent. 1: runtime error output Machining is terminated. 2: Output warning Machining is not terminated.
<Blocks>	Specifies the size (number of blocks) of the look-ahead range. Integer value. Recommended look-ahead range: 1 to 10 blocks.

Tab. 6-42: Collision (CLN) syntax

Example:

```

:
N100 CLN(DLA      Presetting for the look-ahead range of collision monitoring
5)                starting at the next G41/G42: 5 blocks.

:
N110 G41 D10      Milling cutter path correction to the left of the workpiece
N120 X10          Travel forwards

```

NC functions with high-level language syntax

```

N130 X20
N140 X30
N150 G42      Switch milling cutter path correction to the right of the work-
:             piece.
              The look-ahead of collision monitoring is terminated as of block
              150 and then restarted.

N160 X20      Travel backwards
N170 X10
N180 X0
N190 G40      Milling cutter path correction OFF
:             Collision monitoring thus becomes inactive, but it is not deacti-
              vated!

```

Special features and restrictions:

- The presetting for the look-ahead range is 2 blocks.
- The maximum look-ahead range depends on the machine parameters 7060 00110 to 7060 00130.
- To travel backwards without triggering the collision monitoring function when milling cutter path correction has been activated for the contour, the active correction direction must be replaced starting at the point of reversal (G42 is programmed when G41 is active and G41 is programmed when G42 is active).

The control thus automatically terminates the look-ahead range of the collision monitoring function in a G41 or G42 block and restarts the look-ahead function immediately after that.

6.27 G0 interpolation configuration "ConfG0, CG0"

Description: The G0 interpolation is configured in an asynchronous way. The axes remain synchronous; however, the motion is executed block-by-block asynchronously. I.e., they move at their respective dynamical limit values to the programmed end position. The axes reach their end position at different points in time.

The NC block is switched if all axes reached their programmed end position. Depending on the currently active IPS or NIPS, reaching the end position relates to the actual or target position.

The function comprises an NC-internal WAIT functionality.

Syntax:

ConfG0(ASY<status>)	The status may have the value "1" or "0".
ConfG0(ASY1)	enable asynchronous G0 interpolation
ConfG0(ASY0)	disable asynchronous G0 interpolation
Short form CG0	

The asynchronous interpolation takes effect in all subsequent G0 blocks until CG0(ASY0) is programmed, i.e. it remains active until it is deactivated.

Example:

```

: enable asynchronous G0 interpolation
N10 CG0(ASY1) G0 Approach position (100,100,50) asynchronously, wait
N20 XA100 YA100 ZA50 until the weakest axis reached its position
N30 XA200 YA300 ZA55 Approach position (200,300,55) asynchronously, wait
N40 CG0(ASY0) until the weakest axis reached its position
N50 XA400 YA450 ZA60 disable asynchronous G0 interpolation
: Travel to position (400,450,60) synchronously

```

- The motion is always carried out in an axis-specific form. With an active axis transformation at the point of contact 1 or 2, the axis end position corresponding to the coordinates is calculated internally and traveled to in an asynchronous form, i.e. there is no motion in coordinates.
- The function causes a runtime error if a system axis coupling is active.
- The function is not retrace-compatible.

6.28 Selecting the axis transformation "Coord, CRD"

Description: Activates or deactivates axis transformations that are configured in machine parameter group 1030.



The function is required, for example, in connection with spatial coordinate programming (see documentation "MTX Functional Description") or possibly when calibrating axis kinematics (see [chapter 6.10.4 "Reading parameter from NC ATGET" on page 217](#), [chapter 6.10.5 "Writing parameters to the NC ATPUT" on page 218](#) and [chapter 6.10.3 "Forward transformation ATFWD" on page 216](#)).

An axis transformation must only be active at a maximum of 2 points of action:

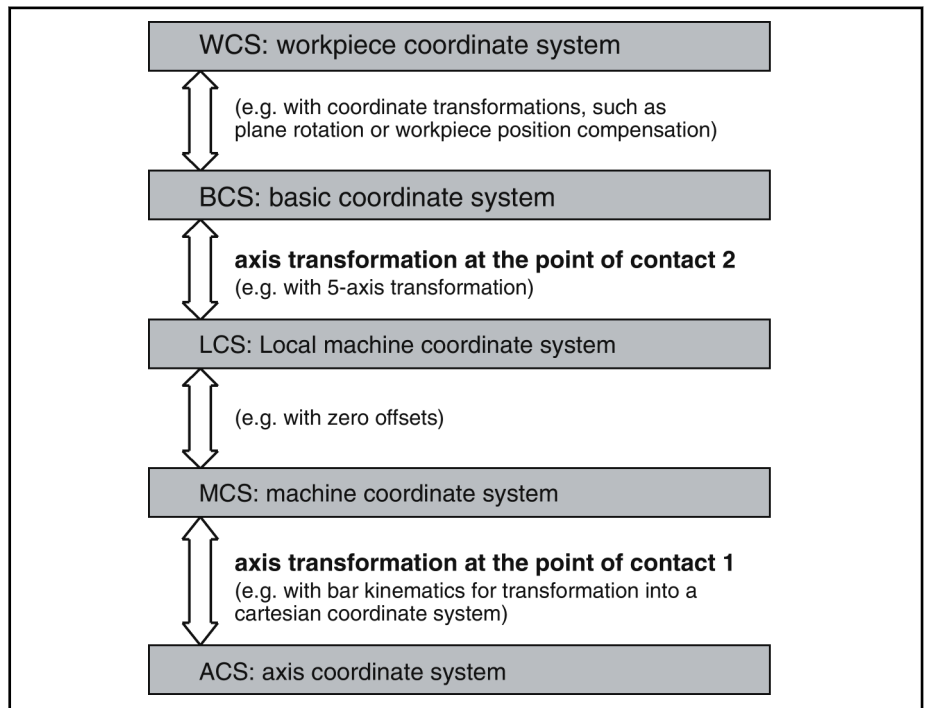


Fig. 6-8: Axis transformation



For every available axis transformation, the point of action at which this transformation is applied, has already been specified internally.

Syntax:

<code>Coord(<AxTrafoNo>)</code> or <code>Coord (<[SDTrfName]>)</code>	Axis transformation ON
<code>Coord()</code> or <code>Coord(0)</code>	Axis transformation at point of action 2 OFF
<code>Coord(0,<point of action>)</code>	Axis transformation on the selected point of action OFF
Short form: <code>CRD (...)</code>	
with	
<code><AxTrafoNo></code>	Axis transformation number Input range: 1.. 20 integer The assignment of the AxTrafoNo and the axis transformation type are stored to MP 1030 00110.
<code><[SDTrfName]></code>	Name of the system data (in brackets) of the SysAxTrafo_t type containing transformation data to be activated. The axis transformation type is directly contained in the system data.
<code><Points of action></code>	Point of action of the axis transformation to be switched off Input: 1 or 2

Tab. 6-43: Syntax of Coord (CRD)

Special features and restrictions:

- It is possible to switch directly between different axis transformations at point of action 2. Previous deactivation is not necessary.
- An axis transformation at point of action 1 may be activated only if no axis transformation is active at point of action 2.



For information on the available axis transformations, refer to the documentation "MTX Functional Description".

6.29 Sharpening internal corners "Cornering, COR"

Description: The function "Sharpening of internal corners" allows the sharpening of internal corners during milling with conic tools.

To deburr and sharpen land-shaped contours with a triangular cross section, a conic form cutter with a corresponding angle of aperture is used and machined with both edges of the land during an active tool radius correction "G41" or "G42". Result: A sharp tool cutting edge at the tip of the land. Residual material can, however, be found at internal corners (caused by the edge radius at the tip of the tool). Consequently, no sharp tool cutting edge can be machined in this area.

To machine the residual material, the function "Cornering, COR" retraces the tool center point to the tip of the corner and back on a calculated contour, by taking the contour before and after the corner into consideration. The tip of the tool is lifted so that the tool is engaged with the contour before and after the contour at the height of the land tip. At the corner point, the (assumed) tool tip is positioned directly on the land tip.

Contours before and after the corner can consists of several, arbitrary linear and/or circular NC blocks. However, it is important that the number of relevant blocks before and after the corner are within the selected look-ahead range of the function "Collision monitoring CLN". Furthermore, the collision monitoring has to be active. The same applies for the function "Circular arc at contour transition G43".

Syntax:

Cornering([CAN<can>] [, MAN<man>] [, FF<ff>])	
with	
<can>	Cone ANgle: Opening angle of the tool edge in degrees. Default value: 90°
<man>	Min ANgle: Minimum angle of adjacent blocks at the block transition to sharpen one edge. Default value: 10° This parameter is only of relevance to "single corners". A "single corner" is a corner that is solely defined by the two adjacent contour elements. In case of collisions detected by the collision monitoring, the parameterized minimum angle is of no relevance as a collision does not necessarily contain a geometric corner. If a collision is detected, sharpening always takes place.
<ff>	Feed Factor: Feed factor with which the programmed feed is scaled in case of sharpening of corners. Default value: 1 Value range: 0 < <ff> 1
Cornering(0)	Deactivate sharpening of internal corners.

Tab. 6-44: Syntax cornering

Example:

:	Activate circular arc at contour transition
N10 G43	Activate collision monitoring; monitoring range is set to 80 blocks
N20 CLN(DLA80)	
N30 CON(CAN87.8, MAN15, FF0.2)	Activate and parameterize corner sharpening: Opening angle 87.8°, minimum angle 15°, feed factor 20%
N40 G42 ED1	Activate the tool radius corrections and apply tool data from correction block ED1
:	

Special features and restrictions:

- The function sharpening of corners is modal.
- Precondition for corner sharpening: an active "collision detection CLN."
- Sharpening of corners only works in case of an active function "circular arc at contour transition G43."
- The correction radius of the tool cannot change within the corner sharpening process.

6.30 Spline coupling table "CoupleSplineTab, CST"

Description: In axis or coordinate coupling, the coupling function is stored in the form of support point pairs in a coupling table. To calculate the positions of the slave axis between the data points, block preparation generates a spline table.

NC functions with high-level language syntax

Spline tables are created when the coupling syntax is being interpreted. They are stored as a file to the link table directory.

Spline tables are generated automatically for the channel axis coupling (AXC, SCCT).

CoupleSplineTab(...) can also be used to create a spline table in defined form:

- **CST(TAB(<TabName>, 1))**
Explicitly requires new creation of the spline table.
- **CST(TAB(<TabName>, 0))**
Forces such creation if no spline table exists or if it is older than the coupling table.
- **CST(TAB(<TabName>{,0|1}),LOAD)**
Loads the Spline table additionally into the system table memory of the system axis coupling.
- **CST(TAB(<TabName>{,0|1}),UNLOAD)**
Removes the spline table from the system table memory of the system axis coupling.

With *CoupleSplineTab(...)* , a spline table can be created even without an existing group of coupled axes (e.g. in Manual Data Input mode).

For the application of tables in the system axis coupling, also refer to the manual "MTX Functional Description", Chapter "System axis coupling".

The name of the spline table is generated from the name of the currently active coupling table by attaching the extension ".s"; for example if the name of the coupling table is **curve.fct**, the name of the spline table is **curve.fct.s**.



The coupling table is searched in the current search path. The search path is set in machine parameter 3080 00001.

The default link table directory is /usr/lnk. However, it may also be freely defined using machine parameter 3080 00004.

Syntax:

CoupleSplineTab(TAB (<TabName>{,<1 0>}) { , LOAD UNLOAD } { , <Location>})	
	Generate spline table
Short form: CST (...) CST	
with	
<TabName>	Name of the coupling table that is searched in the current search path and for which a spline table is created.
<1 0>	Optional: 0: The spline table is only created if it does not yet exist or if it is older than the coupling table. (Default) 1: Regenerates the spline table.

LOAD UNLOAD	Optionally only for system axis coupling: LOAD: The Spline table is loaded into the channel-comprehensive table memory SysAxCoupleTab for the usage of the system axis coupling. If there is no location specification, the entry "<TabName>" is used or - if not available - the first free entry. UNLOAD: The Spline table is removed from the SysAxCoupleTab table memory. If there is not location specification, the entry <TabName> is searched and used.
<Position>	Optionally only for system axis coupling: Loads or removes the table of the system table memory location SysAxCoupleTab[<Location>] At location-oriented sharing (UNLOAD), a specified table name is ignored. The location specification is only effective together with LOAD/ UNLOAD.

Tab. 6-45: Syntax of CoupleSplineTab (CST)

Example:

CST (TAB (curve.fct))	Creates the spline table /<LinkDirectory>/ curve.fct.s if required.
CST (TAB (curve.fct, 1))	Creates the spline table /<LinkDirectory>/ curve.fct.s irrespective of the date or existence.
CST (TAB (curve.fct) , LOAD)	Creates the spline table and loads it into the system table memory SysAxCoupleTab(system data) if necessary.
CST (TAB (curve.fct) , UNLOAD)	Remove the table from the system table memory.
CST (TAB (curve.fct) , LOAD, 5)	Loads the table curve.fct into the system table memory location SysAxCoupleTab[5]
CST (, UNLOAD, PLACE%)	Remove the table from the system table memory location SysAxCoupleTab[PLACE%]

6.31 Curve parameter interpolation "CurveParamInterpolation, CPI"

Description: For contour elements with curve parameters– i.e. spline types 0 to 4 and the rounding splines (syntax SCO) – curve parameter interpolation can be selected instead of path length interpolation. The channel coordinates do not follow the path synchronously to the distance s , but synchronously to the curve parameter w . The curve parameter w runs from $w = 0$ and $w = W_e$. With splines, W_e is the spline parameter length. It is programmed with PL or calculated automatically and internally, depending on the selected spline parameterizing.

Syntax:	CurveParamInterpolation(1)	
		Switch on curve parameter interpolation.
	CurveParamInterpolation(0)	
		Switch back to path length interpolation.
	Short form: CPI	

Tab. 6-46: Syntax of CoupleSplineTab (CST)

Example:

```

N10 SplineDef(2203)
N20 G1 F123 X0 Y0      ;Feed is 123 mm/min
N30 CPI(1)             ;Switching on the curve parameter interpolation
N40 G06 X10 Y10 PL5    ;Block duration is 5/123 min(*)
N50 X20 Y15 PL4 F88    ;Block duration is 4/88 min(*)
N60 G1 CPI(0)          ;Return to path length interpolation

```

(*) except for acceleration and deceleration processes in this block

Special features and restrictions: Path lengths interpolation is active after the control startup.

6.32 Customer-specific velocity profile"CVP"

Description: The **CVP** disables the internal velocity control MTX (G8, G9). The user has the option to specify a user-defined velocity profile sequence using a Hermite spline (spline type 1).

Syntax:	CVP	Disable velocity control of MTX
	The CVP function is deselected by programming G8 or G9.	

Tab. 6-47: Syntax CVP

Special features and restrictions: The function can only be used in connection with Hermite splines. The G8, G9 functions and CVP constitute a modal group and deselect themselves.

6.33 Activating D-correction tables "DcTsel, DCS"

Description: Activates a D-correction table (geometry correction table, GEO table). D-correction tables are stored as XML files in the file system of the control.

Syntax:	DcTsel ({<path>}<FileName>)	
	Short form: DCS (. .)	
	with	

<Path>	Optional path specification about the directory the <FileName> is stored in. If there is no specification, it is searched in the "/database" path. If file name cannot be found, the control uses the search path for sub-routines to search for <FileName> in other directories.
<FileName>	File name of the D-correction table, incl. file extensions. Tables with standard names (DC <Number>.dct) can directly be activated using this number, e.g. DcTsel(7) activates the table DC7.dct.

Tab. 6-48: Syntax of DcTsel (DCS)



Refer to the Control Operating Instructions for information on how to create and edit D-correction tables.

Example:

```

:
N030 DCS (geotab.dct) First searches for D-correction table "geotab.dct" in
:                      the directory "/database" and then, if it is not there, in
                      the search path for subroutines. The first D-correction
                      table found with the name "geotab.dct" is activated.

N130 DCS (/mnt/ge.dct) Searches and activates the D-correction table "ge.dct"
:                      in the directory "/mnt". If it is not found there, an error
                      message is displayed.

```

6.34 Applying axis settings from MP "DefAxis, DAX"

Description: Cross-channel activation of the default axis configuration according to MP 1003 00002.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

DefAxis
Short form: DAX

Tab. 6-49: DefAxis syntax, DAX

Special features and restrictions: If one of the axes involved has not been enabled yet, a runtime error is caused.



Since it often happens that several channels take part in the axis transfer, we recommend to enter the function in MP 7060 00020 behind the key word "#SysRes".

This way, the default axis configuration can always be regenerated together with the "system reset".

6.35 Resetting channel spindles to MP "DefSpindle, DSP"

Description: Activates the default spindle setting in accordance with MP 7020 00020.

Syntax: DefSpindle
Short form: DSP

Special features and restrictions: If the channel-specific integer auxiliary functions CSP1 to CSP8 are defined, the PLC is informed about the current channel spindle assignment.

6.36 Block transition without deceleration "DefTangTrans, DTT, DTT"

Description: The function "Block transition without deceleration" limits the impact of the maximum step change on wide transition angles. By appropriate parameterization, the behavior can be defined so as to take major knees in the contour into account in detail, whereas minor contour knees are considered as quasi continuous and the path velocity is not reduced. At a real machine, this can result to a blending of the programmed contour.

Syntax:	DefTangTrans (<angle>)
	Short form: DTT (<angle>)
	with
<Angle>	If the change of the ratio of an axis in the standardized direction vector is greater than $\sin(\text{<angle>})$ at the contour transition, the maximum step change for the contour transition in question is taken into consideration. In case of changes smaller than $\sin()$, the maximum step change is ignored and no deceleration takes place at the contour transition (provided that the brake path available in the scope of the set block look-ahead is sufficient). Value range: 0 to 50 degrees
DefTangTrans({0})	The limit angle entered in machine parameter 7030 00310 becomes active.

Tab. 6-50: Syntax of DefTangTrans (DTT)

- Special features and restrictions:**
- When exact stop is active, the velocity at each block transition is decelerated to $v = 0$.
 - After reset, the setting from the startup string is applied.
 - When the "DefTangTrans" function is used, always consider the velocity at which the system is to travel. As the limit values configured within the machine parameter are temporarily ignored, servo errors may occur above all with high velocities and great transition angles!

6.37 Diameter programming "DiaProg, DIA", Radius programming "RadProg, RAD"

Description: Coordinate information on transverse axes (on lathes; usually x-coordinates) can optionally be interpreted as diameter or radius. In this way, any dimensional information that may exist can be transferred directly to the part program without conversion.

If "diameter programming" is activated, the diameter symbol is placed in front of the axis displays of the transverse axis for the workpiece position, distance to go, end position and program value.

The machine position, actual axis value and lagging are always displayed as radius values.



Incorrect interpretation of dimension information possible! "DIA" affects only the entered/configured diameter coordinates. Diameter programming has no influence on circular interpolation parameters I, J, K.

Always ensure that only suitable dimension information is programmed.

Syntax:

DIA {({ <Coord1> {, . . . , <Coord8>}})}	
	Diameter programming for up to 8 linear axes/coordinates ON. Diameter programming is switched off for all coordinates that are not entered.
with	
<Coordi>	A maximum of 8 linear axes/coordinates (i = 1...8) whose distances are to be evaluated as diameter information.
DIA	Restores the last RAD status. After the control startup, the default values of the machine parameters apply.
DIA ()	The default values are activated. If no diameter coordinates are defined in the machine parameters, an axis of classification X becomes a diameter axis. If this axis does not exist either, the control reports a runtime error.
RAD	Radius programming for all coordinates ON

Tab. 6-51: Syntax of DiaProg (DIA) and RadProg (RAD)

Example:

N10 DIA ()	Activates configured diameter coordinates
N20 DIA (Y1, W2)	The given coordinates Y1 and W1 become diameter coordinates and all non-programmed coordinates become radius coordinates.
N30 RAD	Diameter programming is deactivated for all coordinates.
N40 DIA :	Coordinates Y1 and W1 become diameter coordinates (last RAD status).

Special features and restrictions:

- The functions are modal and deselect each other mutually.
- If a diameter coordinate is incorporated in an axis transformation, the NC switches off diameter programming for this axis. However, if axis transformation is switched on, coordinates can be switched to diameter programming.
- The following applies to the operation modes "manual input of data" and "processing":
With dimensions for circle center coordinates, tool lengths and zero offsets, the coordinates of the transverse axes are always interpreted as the radius value.
- The following applies to the operation modes "manual handwheel operation" and "jog mode":
The axis interface signal "diameter incr. step" (qAx_JogDia) can be used to switch between diameter and radius programming.

6.38 Delayed response to drive errors "DERDelay"

Description To avoid machine damage, it is required for some applications that the NC keeps the drives under control for a short period and performs a coordinate standstill of the machine axes when drive errors occur.

The "DERDelay" function can activate/deactivate the delayed error reaction for drives labeled as "programmable" in "MAIN/Dr[i]/DelayedErrorReaction" (MP 1001 00012).



For further information, refer to the chapter "Delayed NC error reaction for non-fatal drive errors" in the "MTX Functional Description".

Syntax

DERDelay(1)	Activates the delayed error reaction for all drives labeled as programmable in "MAIN/Dr[i]/DelayedErrorReaction" (MP 1001 00012).
DERDelay() or DERDelay(0)	Deactivates the delayed error reaction for all drives labeled as programmable in "MAIN/Dr[i]/DelayedErrorReaction" (MP 1001 00012).
DERDelay({ON(<DriveList>)},{OFF(<DriveList>)})	
	Activates/deactivates the delayed error reaction for the programmed drives.
with	
<Drive list>	List of system drive numbers or system axis names separated by commas.

Tab. 6-52: DERDelay syntax

Example N10 DERDelay(ON(X,B)) ;Activate x- and b-axis

...

N200 DERDelay(OFF(X,B)) ;Deactivate x- and b-axis

Special features and restrictions

- The NC function can only be used for drives labeled as "programmable" in the machine parameter "MAIN/Dr[i]/DelayedErrorReaction" (MP 1001 00012).
- Only the master axis is programmed for gantry groups.
- A system reset deactivates the delayed error reaction enabled by the "DERDelay" function.

6.39 Dimension of the working plane "DimPlane, DimP"

Description: Enables the calculation of the minimum and maximum values in all three coordinate systems Acs (axis coordinates), Mcs0 (machine coordinates (without ZO)) and Bcs (basic coordinates). The extreme values are determined individually for each coordinate. Thus, the dimension of the working planes are established.

Syntax:

DimPlane(1) or DimPlane	Enable calculation of "dimension of the working plane"
DimPlane(0) or DimPlane()	Disable calculation of "dimension of the working plane"
Short form: DimP(..)	

*Tab. 6-53: DimPlane syntax***Example:****Program:**

```

:
N100 DimPlane(1)      ;Calculation "Dimension of working plane" on
:
N500 G1 x11 y22       ;programmed traversing motions
:
N900 DimPlane(1)      ;Calculation "Dimension of working plane" of
N1000 G0              ;target block (e.g. for block search)
:

```

All extreme values are determined between block N100 and N900.

Special features and restrictions:

- The extreme values are internally stored on the volatile channel-specific system date "/SysDimPlane". The following structural elements are defined in the schema "sddimp.xsd_"

General data (SysDimPlane):

Structural element	Description
MinAcs	Minimum values of the channel axes
MaxAcs	Maximum values of the channel axes
MinMcs0	Minimum values of the machine coordinates in the channel (without ZO)
MaxMcs0	Maximum values of the machine coordinates in the channel (without ZO)
MinBcs	Minimum values of the basic coordinates in the channel
MaxBcs	Maximum values of the basic coordinates in the channel

Tab. 6-54: Overview on structural elements SysDimPlane (general data)

- The "/SysDimPlane" system date has to be created by the user. Templates are available in the control (see the "MTX Functional Description - Special Functions" documentation in the chapter "Dimension of the working plane").
- After values are zeroed after the control startup.
- Following the activation, (i.e. programming "DimPlane(1)"), all values are initialized: Minimum values = infinite / maximum values = -infinite.
- The calculated values are pending on the system date until the next activation.
- The rough interpolation between "DimPlane(1)" and "DimPlane(0)" is active during the computation of the block pre-run. This, however, slows down the block pre-run but the NC can determine the minimum and maximum values without a traversing motion.

6.40 Distance control for digitalization "DistCtrl, DCR"

Description: This function keeps the distance between the surface scanned and the measuring device (e.g. laser) constant during digitizing. This is to ensure that the available working range of the measuring device is not exceeded.



For detailed information on this function, refer to the documentation "MTX Functional Description".

Syntax:

DistCtrl l(1) or DistCtrl	Starts distance control and applies the current distance between the surface and the measuring device as the reference value. The configuration data defined via the machine parameter 7050 007xx is active.	
DistCtrl l() or DistCtrl l(0)	Deactivates axis distance control, stores the current correction value and stops axis motion. If this is programmed in the same block as a traversing motion, the axis distance control is not disabled before the motion has been completed.	
DistCtrl l(<fct>)	Suppresses certain function-specific configuration data in the machine parameters.	
Short form: DCR (...)		
with		
<Fct>	DcAxis (<Axis>,) <Corr> Overrides MP 7050 00702 Short form: DCA (...)	
	<Axis>	Name or number of the channel axis for which axis distance control is to be activated.
	<Corr>	Direction of motion in which the correction values are to be included in the calculation: +1 or 1: In positive direction of motion -1 : In the negative direction of motion
<Fct>	DcFilter (<Time>) Overrides MP 7050 00730. Short form: DCF (...)	
	<Time>	Filter parameterization for smoothening the sensor values. 0: Filter off >0: Filter on, smoothing time in ms

<Fct>	DcLimit ({<Vel>}, {<Acc>}) Overrides MP 7050 00740 or MP 7050 00741. Short form: DCL (...)	
	<Vel>	Maximum change velocity of the correction value. Overrides MP 7050 00740 Value input using the unit mm/min or inches/min, as required by the active unit of measurement (G71, G70).
	<Acc>	Maximum steepness (acceleration) of the correction value. Overrides MP 7050 00741 Value input using the unit m/s ² or 1000 inch/s ² , as required by the active unit of measurement (G71, G70).
<Fct>	DcMon ({<collision>}, {<hole>}) Overrides MP 7050 00750 or MP 7050 00751. Short form: DCM (...)	
	<Collision>	Tolerance band for collision detection Overrides MP 7050 00750 Value input using the unit mm or inch, as required by the active unit of measurement (G71, G70). 0: collision detection OFF
	<hole>	Tolerance band for hole detection Overrides MP 7050 00751 Value input using the unit mm or inch, as required by the active unit of measurement (G71, G70). 0: hole detection OFF
DcBreak	Interrupts axis distance control. The current correction value remains active.	
Short form: DCB		
DcCont	Resumes axis distance control after an interruption using DCB. The NC controls the deviation from the reference value as fast as possible.	
Short form: DCC		

Tab. 6-55: Syntax of DistCtrl (DCR)



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.41 Triggering and exiting diagnostic recording "DREC_START", "DREC_STOP"

Description: Using the NC commands "DREC_START" and "DREC_STOP", a diagnostics recording (axis oscilloscope) can be triggered and/or exited, controlled via the part program.

If the recording is to be started with a delay, this can be achieved by programming G4 <Delay> in the same NC block.

Syntax:

DREC_START	Triggering start triggers for a diagnostics recording from a part program.
DREC_STOP	Exit diagnostic recording.

*Tab. 6-56: Syntax DREC_START, DREC_STOP***Example:**

```

N50 DREC_START G4 F5 ; Trigger diagnostic recording with a delay time of 5
                        seconds.

...

N100 DREC_STOP ; Exit diagnostic recording.

```

6.42 End position coupling "EndPosCouple, EPC"

Description: Couples two currently synchronous coordinates within a channel according to the following relationship:

Δ Slave coordinate = coupling factor * Δ master coordinate.

To calculate the resulting slave coordinate, the position values of the master coordinates are always automatically converted into incremental paths.

The control uses the above ratio to calculate the required end position of the slave coordinate for every NC block in which a position value of the master coordinate is programmed.

In relation to other NC functions, the slave coordinate behaves as if its travel path in the part program had been written especially for it using local relative dimension programming (IC). This means, for example, that the current feed affects the **resulting path motion**.

Syntax:

EndPosCouple (<M-coord>, <S-coord>, <Factor>)	
	End position coupling ON
EndPosCouple () or EndPosCouple (0)	End position coupling OFF
Short form: EPC (...)	
with	
<M-coord>	Master coordinate name
<S coord>	Slave coordinate name
<Factor>	Ratio of slave travel distance to master travel distance.

*Tab. 6-57: Syntax of EndPosCouple (EPC)***Example:**

```

N10 G18 G0 Z0      Active plane: z, x
                   with rapid traverse to z=0

N20 G1 F1000 Z3 X1  Positioning to P1

N30 EPC (Z, X, 1)   Switch on end position coupling
:                  Master coordinate: Z;
:                  Slave coordinate: X
:                  Slave travel distance = master travel distance

N40 Z4              Positioning to P2

N50 X1              Positioning to P3

```

N60 Z5	Positioning to P4
N70 X3	Positioning to P5
N80 Z0	Positioning to P6
N90 EPC ()	End position coupling OFF
:	

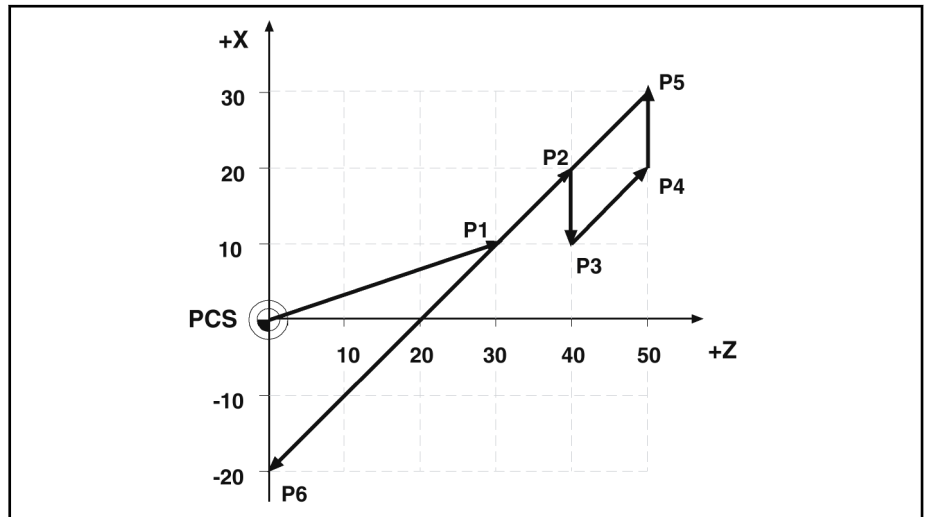


Fig. 6-9: Example of EndPosCouple (EPC)

Special features and restrictions:

- End position coupling cannot be used together with endless coordinates (coordinates whose values are automatically transformed into the defined range of values when they exceed their range limits).
- Polar coordinate programming is not permitted if end position coupling is activated.
- Master and slave coordinates must not be programmed together in one NC block if end position coupling is active. However, programming only slave coordinates in an NC block is permitted.
- Neither the master nor the slave coordinate must exit the channel if end position coupling is active. An axis exchange and changes to the active axis transformation are not permitted.
- Even if diameter programming is switched on for the master coordinate (see [chapter 6.37 "Diameter programming DiaProg, DIA, Radius programming RadProg, RAD"](#) on page 252), the corresponding radius value is used for the coupling.

6.43 Feed computation: Hiding of axes "FeedAd, FAD"

Description: Hides coordinates from feed computation.

Removes all programmed coordinates or the axes defined in MP 1003 00020 from feed computation

The removed coordinates are moved synchronously.

As a result, the real path velocity can increase in comparison with the programmed F-value.

NC functions with high-level language syntax

Syntax:	
FeedAd(1) or FeedAd	Removes axes defined in machine parameter 1003 00020 from feed computation.
FeedAd(<Coordinates>)	Removes programmed coordinates from feed computation.
FeedAd(0) or FeedAd()	Takes all coordinates into account for feed computing.
with <Coordinates>	List with coordinate name incl. orientation coordinates e.g.: X, Y, Z, ..., x, y, z, phi, theta, O
Short form: FAD(. .)	

Tab. 6-58: Syntax of FeedAd (FAD)

Example: (y-axis entered in MP 1003 00020)

```

:
N100 G94 G0 X0 Y0      Positions to P(0;0) in rapid traverse
N110 FeedAd()
N120 X100 Y100 F100    Positioning to P(100;100)
:                      Progr. path velocity: 100 mm/min
:                      Real path velocity: 100 mm/min
N150 FeedAd            Hide Y (via MP)
N160 X200 Y200 F100    Positioning to P(200;200)
:                      Progr. path velocity: 100 mm/min
:                      Real path velocity: 141.42 mm/min
N190 FeedAd(X)         Hide X
X300 Y300 F100         Positioning to P(300;300)
:                      Progr. path velocity: 100 mm/min
:                      Real path velocity: 141.42 mm/min

```

Special features and restrictions: If, in a block, only coordinates hidden for feed computing are moved, their feed can be set the instead of the "F" address via the "Omega" address.
(refer to [chapter 5.4.5 "Omega address \(feed\) Omega" on page 191](#)) is opened.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.44 Feed forward"FeedForward, FFW"

Description: This function reduces system-related lagging by appropriately correcting the command interpolator values in the drive. The result is an increased contour accuracy or "operation without velocity errors".



The "feed forward" function is integrated in the drive according to the manufacturer's specifications and is only activated/deactivated by the displayed command syntax of the part program.

For a detailed description of the "feed forward" function, refer to the drive documentation.



The possibility for activating the feed forward must have been released for the corresponding axes using the machine parameter 1003 00009.

Syntax:

FeedForward(1) or FeedForward	Activates the feed forward for all axes released according to MP 1003 00009 (the relevant drives are switched to secondary mode 1).
FeedForward(<Addr><value>,...)	Activates/deactivates the feed forward for the programmed axes (effect depends on <value>).
FeedForward() or FeedForward(0)	Deactivates feed forward for all axes. (All relevant drives are switched to their main modes.)
Short form: FFW(. .)	
with	
<Addr>	Physical or logical axis address
<Value>	Path information for <addr.> 0: Deactivates feed forward unequal to 0: Activates feed forward

Tab. 6-59: Syntax of FeedForward (FFW)

Example:

N10 FFW	Activates the feed forward for all axes of the channel released by the parameter AX/Dr/FeedFwd/AxFeedFwd (MP 1003 00009)
:	
N50 FFW(Z0)	Disable feed forward for z
:	(Switch Z to main operation mode)
N90 FFW()	Disable feed forward of all axes of the channel
:	(Switch to main operation mode)

Special features and restrictions:

- When the feed forward is deactivated, all the axes of the channel are switched to their main mode.
- The parameters of the feed forward control function can be set in the drive only by writing the relevant drive parameters.

"Writeld3" is available as a function for writing Sercos parameters through the part program (see [chapter 6.156 "Writing Sercos parameters with extended Sercos ID number Writeld3, WID3" on page 413](#)).



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.45 Feed forward for asynchronous axes "FFWAxis, FFWA"

Description: This function reduces system-related lagging by appropriately correcting the command interpolator values in the drive. The result is an increased contour accuracy or "operation without velocity errors".



The "feed forward" function is integrated in the drive according to the manufacturer's specifications and is only activated/deactivated by the displayed command syntax of the part program.

For a detailed description of the "feed forward" function, refer to the drive documentation.



The possibility for activating the feed forward must have been released for the corresponding axes using the machine parameter AxFeedFwd (MP 1003 00009).

Syntax:

FFWAxis ({ON (<AxisList>) }, OFF (<AxisList>))	
Short form: FFWA (..)	
	Enables activation/deactivation of the feed forward of axes of the individual channel and asynchronous axes (incl. asynchronous C-axes).
with	
<AxisList>	List of system axis names and system axis numbers separated by comma.

Tab. 6-60: Syntax FeedforwardAxis (FFWA)

or alternatively

FFWAxis (1)	Enables activation/deactivation of the feed forward of axes of the individual channel and asynchronous axes (i.e., all axes enabled by MP 1003 00009).
FFWAxis () or FFWAxis (0)	Enables activation/deactivation of the feed forward of axes of the individual channel and asynchronous axes.

Tab. 6-61: Syntax FeedforwardAxis (FFWA)

Example:

N10 FFWA (ON (B, AS10, AS12))	Activate feed forward for channel axis B and for the asynchronous axes AS10, AS12
:	
N20 FFWA (OFF (AS10, AS12))	Deactivate feed forward for the asynchronous axes AS10, AS12
:	
N80 FFWA (1)	Activate feed forward of all channel axes and all asynchronous axes
:	
N90 FFWA (0)	Deactivate feed forward of all channel axes and all asynchronous axes
:	

Special features and restrictions:

- Gantry slaves are only influenced by programming the master.
- Switching only affects the axis mode in case of spindles/C-axes.

After switching "Spindle -> C-axis", the axis is in state position interface (without feed forward).

- The parameters of the feed forward control function can be set in the drive only by writing the relevant drive parameters.
- The feed forward can only be switched at standstill.

This means that the part program at the start of the FFWA block has to wait for standstill of all programmed asynchronous axes.

6.46 Feed computation: Feed group"FeedGrp, FDG"

Description: Determines the feed computing coordinates.

The programmed path feed refers to the programmed coordinates.

All non-programmed coordinates are moved synchronously.

As a result, the real path velocity can increase in comparison with the programmed F-value.

Syntax:

FeedGrp (1) or FeedGrp	Activates the default feed group. The path feed refers to the coordinates with X, Y, Z machining function.
FeedGrp (<Coordinates>)	Activate the feed group mentioned by name. The path feed refers to the programmed coordinates.
FeedGrp (0) or FeedGrp ()	No programmed feed group is active.
with <Coordinates>	List with coordinate name incl. orientation coordinates e.g.: X, Y, Z, ..., x, y, z, phi, theta, O phi, theta and O are equal in their effect.
Short form: FDG (. .)	

Tab. 6-62: Syntax of FeedGrp (FDG)

Example:

FeedGrp (X, Y, Z)

Coordinates X, Y and Z are feed computing.

FeedGrp (x, y, z, O)

Coordinates x, y, z and the orientation vector O (and therefore the orientation coordinates phi and theta) are feed computing.

Example:

```

:
N100 G94 G0 X0 C0
N110 FeedGrp (X, Y)      Coordinates X and Y are feed computing.
N120 X100 C50 F100      The X-motion is performed with F100. C is moved syn-
                           chronously.
N130 C100 F100           No XY motion.
:                         The C-motion is performed with the F-value.
N200 FeedGrp (1)         The default feed group (here: X, Y, Z) becomes active.
```

NC functions with high-level language syntax

```

N210 X200 C150 F100 The X-motion is performed with F100. C is moved syn-
                    chronously.
N220 C200 F100      No XYZ motion.
:                  The C-motion is performed with the F-value.
N300 FeedGrp(C)     Coordinate C is feed computing.
N310 X300 C250 F100 The C-motion is performed with F100. X is moved syn-
                    chronously.
N320 C300 F100      The C-motion is performed with F100.
:

```

- Special features and restrictions:**
- If the only coordinates that are moved are coordinates which do not belong to the feed group, the programmed feed refers to these coordinates.
 - If the default feed group FeedGrp(1) is active, the feed group is automatically adjusted when the extended level switchover (G17(...), G18(...), G19(...)) is programmed and when an axis transformation (Coord(...)) is switched over.
 - If a named feed group is active, axes temporarily removed from the channel are added again to the feed group after their return to the channel.

6.47 Feed computation: Feed group type"FeedGrp, FDGT"

Description: If the feed group FeedGrp / FDG is activated, the path velocity profile can be manipulated on block switching.

Syntax:

FeedGrpType (SMOOTH) or FeedGrpType (1)	The MTX calculates a modified path feed based on the feed F which is programmed for the FeedGrp members.
FeedGrp (STRICT) or FeedGrpType (0)	The programmed feed F directly refers to the members of the feed group. The remaining coordinates are moved synchronously.
Short form: FDGT (. .)	

Tab. 6-63: Syntax of FeedGrpType (FDGT)

Example: A feed group is active. The programmed feed refers to the members of the feed group.

FeedGrpType(SMOOTH)

The internal path feed is adjusted such that the coordinates programmed with FeedGrp move with the programmed feed. The following velocities result in the program passage:

Program:

```

N10 G0 X0 V0
N20 FeedGrp(X) FeedGrpType(SMOOTH)
N20 G1 F1000 X10 V20 ; Fcont = 2236.068, Fx = 1000, Fv = 2000
N30 X20 V100 ; Fcont = 8062.258, Fx = 1000, Fv = 8000

```

The path feed Fcont starts increasing from 2236 to 8062 at the transition from block N20 to block N30. This leads to a relatively smooth change in velocity of Fv.

FeedGrpType(STRICT)

The following velocities result in the program passage:

Program:

```
N10 G0 X0 V0
N20 FeedGrp(X) FeedGrpType(STRICT)
N20 G1 F1000 X10 V20 ; Fcont = 1000, Fx = 1000, Fv = 2000
N30 X20 V100 ; Fcont = 1000, Fx = 1000, Fv = 8000
```

The path feed Fcont does not change at the transition from block N20 to block N30. The acceleration margin of the V-axis limits the effective path velocity on block switching.

Special features and restrictions: SMOOTH is set as default variant during control startup.

6.48 Flying measurement "FlyMeas, FME"

Description: *The "Flying Measurement" function (FME):*

- supplements the travel command
- allows measuring while travelling - therefore, "flying measurement"
- uses the touch probe function of the drive
- Acts on a block-to-block basis or is a local function
- (or the travel command) uses the interpolation type of the active modal group G0, G1, G2 ...
- does not stop the following block preparation. WAIT is to be programmed explicitly if required

Syntax:

FME (MpiAxis <i>) <TravelCommand>	
or	
FlyMeas (MpiAxis <i>) <TravelCommand>	
„MpiAxis“	Is always to be programmed before the index <i>.
<i>	System axis index of the measurement axis
<Travel command>	Contains the (axis) coordinates for travelling.

Tab. 6-64: Syntax FME (FlyMeas)

Example:

```
:
N100 IME(MpiAxis 1);           Initialization of the measurement axis
:
N110 G0 X0 Y0;                 Travel to X0, Y0
N120 FME(MpiAxis 1) G1 X20 Y20; FME when travelling on X20, Y20
:
----- or -----
:
N100 IME(MpiAxis 1)
:
N110 G1 X0 Y0
N120 FME(MpiAxis 1) X20 Y20;   G1 is modally effective
:
```

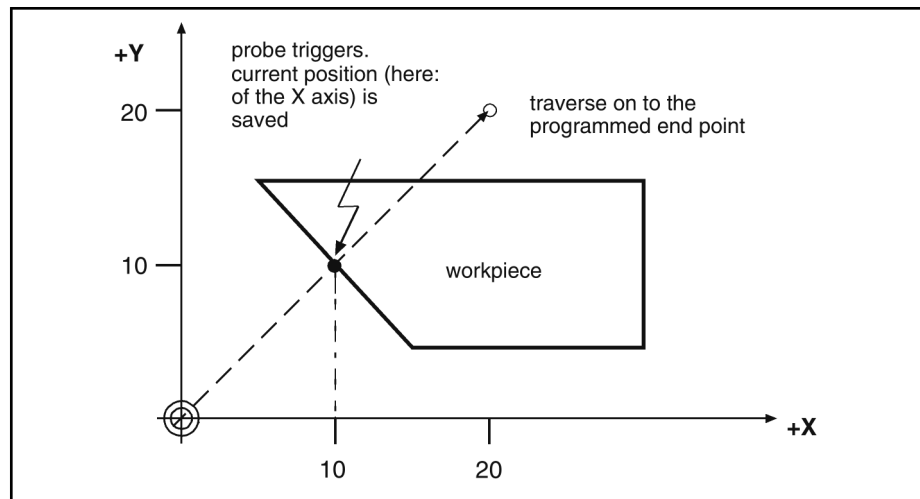


Fig. 6-10: Example of FME (FlyMeas)

Special features and restrictions:

- **Commissioning:**
Before an FME function can be programmed or initialized with IME, "Flying measurement" (FME) must be operated. All required steps are described in the "MTX Functional Description" documentation.
- Before a measurement axis is programmed first with an FME, it is to be initialized with an IME.
- After an FME block, the user has to create a suitable CPL program to transfer the measured value into the part program (with PCSPROBE() or PROBE()).
- The measured value does not are immediately applied in the part program. This is not executed before the total number of blocks for look ahead in the NNCO/LookAh/Ch[k]/NofBlkPrep path.
The specified block number can be limited with the BlkNmb (BNB) function. A WAIT is to be programmed explicitly if required.
- FME can also be used for linear endless axes if the programmed coordinates have a positive sign. Backward traveling using the touch probe (programming negative coordinates) does not provide unique values.

6.49 Measurement at fixed stop "FsProbe, FSP"

Description: While the control travels all the programmed synchronous axes using linear interpolation and the given feed to the programmed end point, the current torque is monitored for a selected axis.

If the torque of this axis exceeds a configurable limit value during the motion, the following actions occur in the control:

- Storing the actual position
- Reducing the path motion velocity to $V = 0$ with the maximum permitted deceleration.
- Clearing of the distance to go
- Deletion of "FsProbe" ("FsProbe" on a block-to-block basis).



"FsProbe" should only be used in combination with a CPL program for analysis.

Syntax:

FsProbe (MfsAxis<i>) <Coordinates> <Feed>	
	"Measurement at fixed stop" for measurement axis <i> ON. Traveling programmed coordinates with linear interpolation.
or	
FsProbe (MfsAxis (<k>{<threshold>})) <Coordinates> <Feed>	
	"Measurement at fixed stop" for measurement axis <k> ON. If <threshold> is programmed, this value is used for the measurement axis <k>. Traveling programmed coordinates with linear interpolation.
Short form: FSP (..)	
with	
<i>	System axis number of the measurement axis
or	
<k>	System axis name or number of the measurement axis
<Threshold>	Torque threshold Input value: in % of maximum torque If <threshold> is not programmed, the machine parameter 1003 00031 (fixed stop torque limit value) is active.
<Coordinates>	Position to be approached while measuring.
<Feed>	Desired path feed Limited by the MP 1005 00030 (maximum feed for "move to fixed stop") and MP 1005 00002 (maximum axis velocity and rapid traverse velocity).

Tab. 6-65: Syntax of FsProbe (FSP)

Example:

N100 FSP(MfsAxis(1,30)) X100 F500	Activates "Measurement at fixed stop" for the first physical axis and approach position X100 with F500. Sets the torque threshold to 30% of the maximum torque.
110 IF SD(9)=0 THEN	Query whether torque threshold was exceeded.
120 XPOS=PROBE(1) :	Save position in switching torque of the first axis (x-axis) in the XPOS variables.
N130 (MSG, CONTACT)	
140 GOTO N180	
150 ENDIF	
N160 (MSG, NO CONTACT)	
N170 M0	Program HALT
N180 ...	

Special features and restrictions:

- The following functions are not permitted in the "FsProbe" block:
 - G75 (Move to touch probe),
 - MOB (measuring on a block)

- InitMeas/FlyMeas (flying measurement),
- RedTorque (torque reduction)
- FsMove/FsReset/FsTorque (move to fixed stop).
- FsProbe must be programmed together with at least one coordinate. Its value represents the maximum search depth before which the "fixed stop" must have been reached.
- FsProbe implicitly pauses the block preparation of the subsequent blocks. Therefore, WAIT does not have to be programmed.
- Evaluation of whether the "fixed stop" has been reached, continuation of program (after an error), safety monitoring, generation of error messages, etc. must be implemented by the CPL program.
- SD(9) can be used to query whether the fixed stop was reached.
- The fixed stop position can be requested with PCSPROBE or PROBE.

6.50 Moving to fixed stop "FsMove, FSM", "FsTorque, FST", "FsReset, FSR"

Description:



The "move to fixed stop" function can also be used for asynchronous axes!

The overall functionality "move to fixed stop" comprises the following sub-functions:

1. "Torque reduction fixed stop": FsTorque, FST

Sets the maximum torque that the drive can have after switching on "move to fixed stop".

The programmed value only has an effect for the first following "Traversing to fixed stop".

If "FsTorque" is not used, the parameter MPS/Dr/MaxTorqPoStop (MP 1003 00031) is effective.

2. "Move to fixed stop": FsMove, FSM

Starts motion in the direction of the fixed stop, taking into account the maximum permitted torque.

If the max. permitted torque is reached (see point 1.) during this motion, the control triggers the following sequence of actions:

- Sending the axis interface signal "Fixed stop reached".
- Reducing the path motion velocity to $v=0$ with the maximum permitted deceleration.
- Setting the command position to:
Actual position + 0.1 mm (or actual position + 0.1 degree)
- Monitoring of the axis position for:
Position at fixed stop + parameter MPS/Dr/MaxDevPoStop (MP 1003 00032) ("monitoring window fixed stop in mm or degrees")
- Stopping the specified torque on the affected drive.
- Continues processing of the part program.

The control generates an error message if no fixed stop has been reached at the programmed end of the path (specified torque threshold is reached; see "FsTorque").

"FsMove" remains effective beyond the "FsMove" block and is switched off only by "FsReset".

3. "Reset fixed stop": FsReset, FSR

Releases the axes and, if necessary, moves away from the fixed stop.

- If synchronous and/or asynchronous axes have been programmed in the "FsReset" block, the control travels all axes to the programmed end points at specified feed. The maximum permitted torque of the individual axes applies to the motion away from the stop.
- If no axes have been programmed in the "FsReset" block, only the synchronous axes are released. In this case, asynchronous axes for which "move to fixed stop" is still active can only be released by interface signal "reset fixed stop".

Syntax:

FsTor- que(<Addr><Tor >)	Activates the max. admissible torque <Tor> for the <Addr> axis. Synchronous and asynchronous axes are permitted.
Short form: FST (. .)	
FsMove . .	Activates move to fixed stop.
<Coord-Syn> <Feed> <Coord asy> <Feed asy>	
	Moves synchronous and asynchronous axes to the programmed end positions.
Short form: FSM . .	
FsReset . . <Coord-Syn> <Feed> <Coord asy> <Feed asy>	
	Deactivates move to fixed stop. Moves synchronous and asynchronous axes to the programmed end positions.
Short form: FSR . .	
with	
<Addr>	Physical or logical axis address; asynchronous axes are also possible.
<Mom>	Maximum torque of the relevant axis standstill torque in %. Value range: 0 to 500 %.
<Coord-Syn>	Desired end point positions of synchronous axes in the axis coordinate system (e.g. "X100 Y100 Z100"). The axes traverse each without considering <Feed> and the parameter MPS/Dr/MaxVelAccStop (MP 1010 00030) (Maximum acceleration "move to fixed stop").

NC functions with high-level language syntax

<Feed>	Desired feed for synchronous axes Programming by F-address, limited by the parameter MPS/Dr/MaxVelPoStop (MP 1005 00030) (maximum feed "move to fixed stop") and the parameter AX/Dr/Vel/MaxVel (MP 1005 00002) (maximum axis and rapid velocity). The feed has an individual effect on each involved synchronous axis.
<Coord asy>	Requested end point positions in the axis coordinate system of asynchronous axes. The coordinates are each approached individually considering <FeedAsy> and the parameter MPS/Dr/MaxVelAccStop (MP 1010 00030) (Maximum acceleration "move to fixed stop").
<Feed asy>	Desired feed of asynchronous axes. Programming by "FA", limited by the parameter MPS/Dr/MaxVel-PoStop (MP 1005 00030) (maximum feed "move to fixed stop") and the parameter AX/Dr/Vel/MaxVel (MP 1005 00002) (maximum axis and rapid velocity). The feed has an individual effect on each involved asynchronous axis.

Tab. 6-66: Syntax of FsMove (FSM), FsTorque (FST) and FsReset (FSR)

Example program for moving to fixed stop

```

N100 FST(X20,Y20)      ; Restrict torque to 20% for X and Y
N110 FSM F200 X100 Y=IC(20) ; Activate moving to fixed stop for X and Y
;                      ; X travels along the axis position 100, Y travels incrementally
;                      ; Both axes each travel with the feed value 200
;
;                      ; Further part program processing
;
;
N500 FSR               ; Deactivate move to fixed stop and
;                      ; release all synchronous axes

```

- Special features and restrictions:**
- *The following functions are not permitted in the "FsReset" block and while "move to fixed stop" is active:*
 - G75 (travel to touch probe)
 - MOB (measuring on a block)
 - InitMeas/FlyMeas (flying measurement)
 - Torque reduction via PLC interface
 - FsProbe (measuring at fixed stop).
 - FsMove must be programmed together with at least one endpoint coordinate. Its value represents the maximum search depth before which the "fixed stop" must have been reached.



The axis positions programmed with FsMove and FsReset are always interpreted as absolute position values irrespective of G90/G91! If an incremental traversing distance is to be specified for an axis, program the programming attribute IC.



Programmed axis positions are positions in the axis coordinate system (ACS) and not positions in the workpiece coordinate system (WCS)!



All motions programmed in connection with "Travel to fixed stop" are executed asynchronously. Thus, the velocities of the axes are always scaled with the current override position irrespective of OVD/OVE!

6.51 Straightness and angle error compensation "GeoComp, GCT"

Description: Activates the straightness and angle error compensation to compensate for geometrical errors which are caused by inaccuracies of the mechanic system and cannot be recorded by the encoders.



For comprehensive information on the use and parameterization of the "straightness and angle error compensation", refer to the documentation "MTX Functional Description-Special Functions".

Syntax:

GeoComp(...)	
Short form: GCT (. .)	
GCT (1)	Switch on straightness and angle error compensation
GCT (0)	Switch off straightness and angle error compensation

Tab. 6-67: GCT syntax

Special features and restrictions:

- The user saves the compensation files to an XML schema.
- These files are stored in the root directory or in the USRFEP of the control. The file name is freely selectable, the file extension must be "gct".
- GCT enables or disables the straightness and angle error compensation for all channels.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.52 Applying the axis "GetAxis, GAX"

Description: Applies an asynchronous axis to the calling channel. This turns an asynchronous axis into a synchronous axis.

The axis can then be programmed in the current channel using its physical or logical axis name.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

GetAxis (<PAN> <PAI>, {<LAN>}{, <PAN> <PAI>, {<LAN>}}...)	
Short form: GAX (. .)	
with	
<PAN>	Physical axis name Specifies the axis that is to be transferred to the current channel.

<PAI>	Physical axis index Same effect as <PAN>
<LAN>	Logical axis name If this is programmed, the axis to be integrated into the current channel will be given the logical name <LAN>. <LAN> has to be defined either in the parameter CHAN/Ch/Coord/Acs/ChAx/ChAxName (MP 7010 00010) (channel axis name) or in the parameter CHAN/Ch/Coord/Acs/OptAxName/OptChAxName (MP 7010 00020) (optional axis name).

Tab. 6-68: Syntax of GetAxis (GAX)

Example:

```

:
N030          Physical axes YP and ZP are integrated in the receiving chan-
GAX (YP, , ZP, Z) nel.
:             While YP is also addressed as YP in the calling channel, ZP
              receives the address Z in the calling channel.

```

Special features and restrictions:

- An axis to be integrated has to be at standstill.
If this is not the case, the control generates an error message and aborts the program.
- Contrary to function "WaitAxis" (see [chapter 6.155 "Applying the axis, wait, if required WaitAxis, WAX" on page 412](#)) -
- Axis positions in the same block must always be programmed after GetAxis(...) and may be programmed only if axis transformation is not active.
- Axes to be transferred must not be participating in any active monitoring area (see [chapter 6.7 "Area monitoring area, ARA" on page 203](#)).

6.53 Creating channel spindles "GetSpindle, GSP"

Description: Applies a system spindle to the calling channel. This turns a system spindle into a channel spindle. Following this, the spindle can be programmed in the current channel via its logical names and its logical auxiliary functions.

Syntax:

GetSpindle (<SysSpNo> <SysSpName> , <ChanSpNo> <ChanSpName> , { ... })	
Transfers a system spindle as logical spindle into the channel (channel spindle).	
with	
<SysSpNo>	1..32
<SysSpName>	System spindle name
<ChanSpNo>	1..8
<ChanSpName>	S1 .. S8
Short form: GSP	

Tab. 6-69: Syntax of GetSpindle (GSP)

Example:

```

...
N50 GSP (7, 2)      applies the 7th system spindle as second channel
                    spindle

```

N60 M203 S2=500	starts the second channel spindle
...	
N110 GSP(SSP12,S2)	applies the 12th system spindle as second channel spindle

Special features and restrictions:

- A system spindle can be logically present in several channels.
- GetSpindle does not cause a runtime error if there exists already a channel spindle with the specified number.
Exception:
The spindle to be replaced is operated at constant cutting velocity (G96).
- If the channel-specific integer auxiliary functions CSP1 to CSP8 are defined, the PLC is informed about the current channel spindle assignment (e.g., system spindle 12 is assigned to CSP2 12, i.e., ChannelSpindle2).

6.54 Flying block transition via high-speed signal "HsBlkSwitch, HSB"

Description: Permits a premature block switching using the high-speed inputs or the customer inputs of the channel interface of the control. In this way, a **linear** traversing motion can be prematurely terminated, depending on external events.



Modification of the programmed path!

If the programmed end points of the "HsBlkSwitch" block and the block following it do not lie on a straight line, an external event always leads to a modification of the path that cannot be precisely predicted!



The high-speed signals can be configured using machine parameters.

The following applies to block switching on-the-fly:

- deletion without distance to go in the operation modes "Continuous block", "Single block" and "Single step",
- deletion with distance to go in the operation modes "program block" and "manual data input",
- **usually without** axis standstill
the block is exited at the current velocity (for exceptions, see "Special features and restrictions"),
- no check for maximum possible axis jump capability as well as for
- the effect that depends on the current operation mode (Automatic/Single/Program block, Single step, Manual data input). See "special features and restrictions".

Syntax:

HsBlkSwitch (HS<x>=<y>) or
HsBlkSwitch (CI<x>=<y>)

Block switching on-the-fly

Short form: HSB (. .)

with

NC functions with high-level language syntax

<x>	Number of the high-speed signal or of the customer input (qCh_Custom<x>). Value range: 1...8, integer.
<y>	Logical signal status required for a block switching. 0: Low 1: High

Tab. 6-70: Syntax of HsBlkSwitch (HSB)

Example: Event-related feed

The traveled feed is to be reduced, depending on an external event, on a straight path.

Three NC blocks in which different feed information is entered are required for this. If deceleration to $v=0$ is not to occur at the block switching, block switching on-the-fly is required.

Remember that due to the nature of the task (... "on a straight path" ...) all programmed endpoints must lie on the same straight line.



The endpoints of a following block must not be identical to those of the preceding block. Otherwise a traversing motion is not carried out in the following block!



The programmed distance traveled of a following block influences the maximum possible feed at the block transition. If the distance traveled is too short, the path velocity can automatically be reduced!

:

N20 G0 X0 Y0

Approach initial position

N30 HSB(HS1=1) G1 X100
Y10 F500Moves with F500 until high-speed signal 1 =
"high" or X100 Y10 is reached.N40 HSB(HS1=0) X110 Y11
F100Traverses with F100 until high-speed signal 1 =
"low" or X110 Y11 is reached.

N50 X120 Y12 F500

Traverse distance to go until X120 Y12 with F500

:

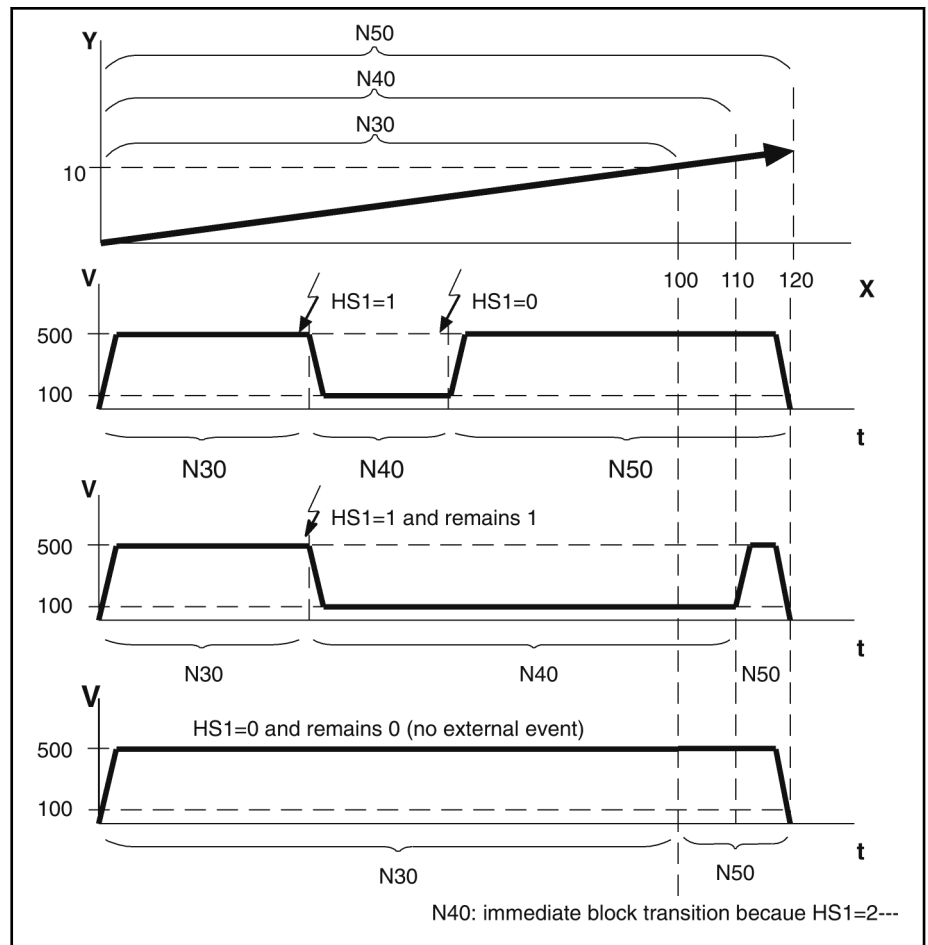


Fig. 6-11: Example HsBlkSwitch (HSB)

Special features and restrictions:

- The function requires a linear traversing motion in both the "HsBlkSwitch" block and the following one.
- The function has to be written with path information. It can be written together with other path conditions.
- In the operation modes "Continuous block", "Single block" and "Single step", the endpoints of the axes that are not programmed in the following block are transferred from the prematurely terminated block.
- *"Block switching on-the-fly" with an axis standstill takes place in the following cases:*
 - Contour knee > 90° between "HsBlkSwitch" block and the following block.
 - Exact stop is active, i.e. G0(IPS...) or G1(IPS...).
 - Due to additional programming, the following block starts with v=0 (e.g. if CO programming or feed forward is active).
 - "Jerk-limited velocity guidance" (G8(SHAPE...), G9(SHAPE...) etc.) is active.
 - Operating mode "single block", "single step", "program block" or "manual data input" is active.

6.55 Block switching with stop via high-speed signal "HsBlkSwitch(..,HSSTOP=..), HSB(..,HSSTOP=..)"

Description: Permits a premature block switching using the high-speed inputs or the customer inputs of the channel interface of the control. In this way, a **linear** traversing motion can be prematurely terminated, depending on external events.



Modification of the programmed path!

If the programmed end points of the "HsBlkSwitch" block and the block following do not lie on a straight line, the occurrence of an external event always leads to a modification of the path that cannot be precisely predicted!



The high-speed signals can be configured using machine parameters.

The following applies to block switching with stop:

- Delete with distance to go in all operation modes
- Always brake to v=0;
either with velocity jump or downslope
Decelerating to v=0 occurs at the end of the block even if the external event did not occur.

Syntax:

HsBlkSwitch (HS<x>=<y>, HSSTOP=<z>) or HsBlkSwitch (CI<x>=<y>, HSSTOP=<z>)	
	Block switching with stop
Short form: HSB (. . , HSSTOP= . .)	
with	
<x>	Number of the high-speed signal or of the customer input (qCh_Custom<x>). Value range: 1...8, integer
<y>	Logical signal status required for a block switching. 0: Low 1: High
<z>	Deceleration type if the event occurs: 0: Ramp down to v=0 with max. deceleration -1: Velocity jump to V = 0.

Tab. 6-71: Syntax of HsBlkSwitch (HSSTOP)

Example: Event-related stop of traversing motion:

```

:
N20 G1 X0 Y0 F1000           Approach initial position

```

<pre>N30 HSB(HS1=1, HSSTOP=-1) X10 F10</pre>	Traverses x-axis with F10 until high-speed signal 1 = "High" or X10 is reached. An event triggers a jump to v=0 and deletion with distance to go.
<pre>N40 HSB(HS2=1, HSSTOP=0) Y100 F200</pre>	Traverses y-axis with F200 until high-speed signal 2 = "High" or Y100 is reached. An event triggers downgrading to v=0 with the max. deceleration and deletion with distance to go.

- Special features and restrictions:**
- The function requires a linear traversing motion in both the "HsBlkSwitch" block and the following one.
 - The function has to be written with path information. It can be written together with other path conditions.
 - If "jerk-limited velocity guidance"(G8(SHAPE...), G9(SHAPE...), etc.) is active, **no** velocity jump occurs during HSSTOP=-1.

6.56 Online correction in workpiece coordinates "HWOC", "HWOCDIS"

Description: *The "online correction" in workpiece coordinates, the following is "overlapped" by a correction:*

- current positions or orientations in the workpiece coordinate system of a channel, or
- current positions of the tool length axis in the TCS z-direction

The control derives the extent of the correction from a connected handwheel (normal case) or drive or from the value of a CPL variable.

For detailed information on this function, refer to the documentation "MTX Functional Description".

Syntax:

HWOC ({ CHAN<channel no> } , CRDNO <Coord no>,{STEP<incr>})	
	Online correction ON
HWOCDIS { (CHAN<channel no>) }	
	Online correction OFF. The current correction extent remains in internal storage.
HWOC () or HWOC (0)	Online correction OF clearing of the relevant correction values.
with	
<Channel-No>	Channel number in which the online correction is activated/deactivated. If CHAN <channel no> is not programmed, the number of the current channel is applied.

NC functions with high-level language syntax

<Coordno>	Coordinate on which the online correction is to have an effect. Range of values: 1 to 9 and 103 Integer 1 to 8: Coordinate number in the given channel 9 or 103: TCS coordinate
<Incr>	Specification of the desired increment step per 1 incr. Effective only if MP 7050 00926 = 0.

Tab. 6-72: Syntax of HWOC, HWOC DIS

Special features and restrictions:

- A coordinate position generated by online correction is not analyzed for possible travel beyond the software limit switches.
Therefore, end limit monitoring on the drive-end should be activated when using online correction.
- Online correction is not possible in the operation modes "Manual" (jog mode) and "Manual traveling to reference point".
- G76 approaches a position that is offset by the current correction value.
- The CPL functions PPOS and PCSPROBE do not take the correction value of the online correction into account.
- G75 measures the real actual position (incl. online correction).
- "FsProbe" measures the real actual position (incl. online correction).



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.57 Initialization "Flying measurement" "InitMeas, IME"

Description: The function "Initialize flying measurement" (IME):

- Initializes the touch probe function of the drive for the "Flying measurement" (FME) function
- Acts on a block-to-block basis or is a local function
- Has no effect on the interpolation type or on the active modal group G0, G1, G2 ...

Syntax:

IME (MpiAxis <i>) or InitMeas (MpiAxis <i>)	
"MpiAxis"	Is always to be programmed before the index <i>.
<i>	System axis index of the measurement axis

Tab. 6-73: Syntax IME (InitMeas)

Special features and restrictions:

- Commissioning:
Before an IME function can be programmed, "Flying measurement" has to be operated. All required steps are described in the "MTX Functional Description" documentation, chapter "Measuring functions".
- IME (InitMeas) has to be called before the FME (FlyMeas) function is called.

6.58 Inverse time programming "InvTime, ITIM"

Description: ITIM is only effective in G93 mode. The unit of the programmed F-word is defined by means of ITIM (see [chapter 5.2.33 "Time programming G93" on page 156](#)).

Syntax:

ITIM <Value>	
with	
<Value>	Integer; the unit of the F-word changes in relation to the <value> in G93 mode
0	F-value in [s]
1	F-value in [1/s]
60	F-value in [1/min]

Tab. 6-74: Syntax of inverse time programming

The unit for ITIM>0 results from

$$F = \frac{v \left[\frac{mm}{ITIM \times sec} \right]}{s[mm]} = \frac{v \left[\frac{1}{min} \right]}{s[min]}_{ITIM=60} = \frac{v \left[\frac{1}{sec} \right]}{s[sec]}_{ITIM=1}$$

Special features and restrictions: ITIM can be set in the Init string.

The default value is "0" (time programming in seconds).

If the time base is switched by means of ITIM, a new F-value must be programmed.

6.59 Selecting the exact stop window "IPS1, IPS2, IPS3"

Description: The NC commands IPS1, IPS2, IPS3 allow to preset the size of the exact stop window in general, which is applied for rapid traverse and feed mode.

IPS1	Fine exact stop window size
IPS2	Rough exact stop window size
IPS3s	Infinite exact stop window size

Tab. 6-75: Exact stop windows that can be set with IPS1, IPS2, IPS3

If exact stop is active for the currently activated NC function from the modal group {G0, G1, G2, G3, G5, G6, G33}, the above presetting is applied immediately.

Syntax:

IPS1	Fine exact stop window size: At the end of the block, the control first reduces the path velocity to $V = 0$. Only when this exact stop window was reached for all axes involved, the next block is traversed.
IPS2	Rough exact stop window size: otherwise same behavior as for IPS1
IPS3	Infinite exact stop window size: At the end of the block, the control only reduces the path velocity to $V = 0$. The exact stop window is not checked.

Tab. 6-76: *Effect of IPS1, IPS2, IPS3 if exact stop is active for the currently activated NC function from the modal group {G0, G1, G2, G3, G5, G6, G33}*

Special Features and Restriction:

If exact stop is not active for the currently activated NC function from the modal group {G0, G1, G2, G3, G5, G6, G33}, IPS1, IPS2, IPS3 can be used to preset the exact stop window size for rapid traverse and feed mode. But this setting is not applied.

Apart from reprogramming IPS1, IPS2 and IPS3, the size of the exact stop window can be changed as follows:

- G0(IPS1), G0(IPS2), G0(IPS3) (effective for rapid traverse only)
- G1(IPS1), G1(IPS2), G1(IPS3), G61(IPS1), G61(IPS2), G61(IPS3) (effective for feed mode only)



For further information on the exact stop function, refer to G0, G1, G61/62 in this documentation and to the documentation "MTX Functional Description", chapter "Accuracy", section "Exact Stop".

6.60 Determining system axes to specify the optimum SHAPE Filter in Case of Enabled Jerk Limitation "JerkAxlst, JAL"

Description: Modifies or determines the list of the possible system axes which are considered when calculating the SHAPE order of the active jerk limitation.

This NC function can also be used as "machine parameter".

For more detailed information, refer to the chapter "Axis-specific Jerk Limitation" in the documentation "MTX Functional Description".

Syntax:

Absolute programming in the part program:	
JerkAxlst() or JerkAxlst(DEF)	Enable default from CHAN/Ch[k]/Ini/NcResetState
JerkAxlst(<AbsList>)	
	Select programmed system axes
Relative programming in the part program:	
JerkAxlst({DEF}{, -, <MinusList>}{, +, <PlusList>})	
	Changes the system axis list
Short form: JAL(. . .)	
with	

DEF	Default from CHAN/Ch[k]/Ini/NcResetState
<AbsList> <MinusList> <AbsList>	List of system axis names separated by comma.
Absolute programming in CHAN/Ch[k]/Ini/NcResetState:	
JerkAxList (<AbsList>)	
	Select programmed system axes
Relative programming in CHAN/Ch[k]/Ini/NcResetState:	
JerkAxList (-, <MinusList>)	
	Remove programmed system axes
Short form: JAL (..)	
with	
<AbsList> <MinusList>	List of system axis names separated by comma.

Tab. 6-77: Syntax JerkAxList, JAL

Example: Initial situation:

The first channel controls a machining unit with three linear axes (XA, YA, ZA) and a rotary table (CA). The VA axis is temporarily included in the channel.

The rotary table (CA) is only used to rotate the workpiece and is therefore irrelevant for the processing.

The VA is relevant for the processing.

Enter JerkAxList(-,CA) in CHAN/Ch[k]/Ini/NcResetState (MP 7060 00010).

Alternatively, the following variant can also be used:

JerkAxList(XA,YA,ZA,VA)

Special features and restrictions:

- This function acts like a machine parameter if entered in the CHAN/Ch[k]/Ini/NcResetState (MP 7060 00010).
Absolute Programming:
JerkAxList(<SysAchsList>)
Relative Programming:
JerkAxList(- , <SysAchsList>)
- If this function is not programmed, all axes being a channel member or becoming one when the axis is transferred are used when calculating a SHAPE filter in case of an enabled jerk limitation.

6.61 Activating the jerk limitation "JerkControl, JKC"

Description: Enables jerk limitation in the channel.



For detailed information on this function, refer to the documentation "MTX Functional Description".

Syntax:

JerkControl or JerkControl (1)	Enable jerk limitation
JerkControl () or JerkControl (0)	Disable jerk limitation
Short form: JKC (..)	

Tab. 6-78: Syntax JerkControl, JKC

Example:

```
N10 JKC (1)      Enable jerk limitation
:
N90 JKC (0)      Disable jerk limitation.
```

6.62 Jog mode in workpiece coordinates "JogWCSSelect"

Description: For the operation mode "setup mode, jogging in workpiece coordinates", this function defines which coordinate is to be jogged or traveled using the hand-wheel.

The following coordinates can be selected for jogging:

- all coordinates of individual machine axes (pseudo coordinates) if no axis transformation is active.
- all linear and orientation coordinates (related to the active WCS) if an orientable axis transformation is active.
- the TCS coordinate Z if an orientable axis transformation is active.

The setting in MP 7050 01010 determines whether the feed and step width (for incremental jogging) can be derived from the settings of an existing machine axis or whether they have to be explicitly programmed.



For detailed information on this function, refer to the documentation "MTX Functional Description".

Syntax:

JogWCSSelect{JWSCHAN<ChannelNo>}JWSCOORD<CoordNo> { JWSFEED<F-value>{JWSSTEP<Incr>}}	
with	
<Channel-No>	Channel number in which a coordinate is to be jogged. Default: Number of the channel in which the function is programmed.
<Coordno>	Coordinate selection Value range: 1 to 8 and 103 Integer 1 to 8: Number of the coordinate to be jogged 103: TCS coordinate is to be jogged.

<F-value>	Jog velocity Entered value in the unit mm/min, inches/min or degrees/min, depending on active unit of measurement (G71, G70). Effective only if MP 7050 01010 = 0.
<Incr>	Selects incremental jogging and specification of the desired step width in increments. Effective only if MP 7050 01010 = 0.

Tab. 6-79: Syntax of JogWCSSelect

Special features and restrictions: The function is designed for use in a special NC program for coordinate selection (see also MP 7050 01110).

For detailed information, refer to the documentation "MTX Functional Description".

6.63 Disabling the velocity limit due to the sum of radial and tangential acceleration "KinLimAcc, KLA"

Description: The KLA function suppresses the provision of an acceleration reserve for override-conditioned changes in velocity.

When providing such reserves, the programmed velocity values can sporadically not be reached when specifying a customer-specific velocity profile using Hermite splines and CVP.

Syntax:

KinLimAcc (1)	Disabling the velocity limit due to the sum of radial and tangential acceleration.
KinLimAcc (0)	Reactivating the velocity limit.
Short form: KLA	

Tab. 6-80: Syntax KinLimAcc

Special features and restrictions: The KLA function acts in connection with CVP and Hermite splines (spline type 1).

6.64 Disabling kinematic NC block splitting "KinSplitDisable, KSD"

Description: For the time-optimized processing, NC blocks are split into several blocks, if necessary, in order to allow for better adjustment of the path feed to the different path-dependent maximum velocities.

This behavior may be undesired as, for example, a programmed NC block is to be traveled with slow, however, constant velocity. For this purposes, kinematic block splitting can be disabled.

Syntax:

KinSplitDisable (1)	Deactivates kinematic NC block splitting
KinSplitDisable (0)	Activates kinematic NC block splitting
Short form: KSD	

Tab. 6-81: KinSplitDisable syntax

Special features and restrictions: Block splitting is active after the control startup

6.65 CO programming "CvProg, CVP"

Description: The function enables a program-controlled change of the flow KV values of individual axes. This can be used for temporary increases of the rigidity of axes (e.g. for milling a bore).

After the control startup, the NC (among other things) reads the current CO values of all coupled drives and saves them.

During CO programming, the control loads the programmed CO values into the relevant drives (parameter S-0-0104).

When CO programming is switched off, the control transfers the previously saved CO values back to the drives, thus restoring the initial state.

The following applies:

- $CO = (\text{path velocity in m/min}) / (\text{lagging in mm})$
- Decelerating to $v=0$ is performed before each block containing a CO switch, since the CO value in the drive should only be switched at stand-still.
- After having triggered the command for CO switching, the control internally waits for the acknowledgement by all participating drives.
- CO switching occurs immediately before any traversing motion that may also be programmed in the same block.

Example:

```
:
N10 G1 F1000
:
N40 X40           Positioning with default CO value
N50 KVP (X2) X50  Transmit CO value 2 to axis X and subsequently traverse
:
```

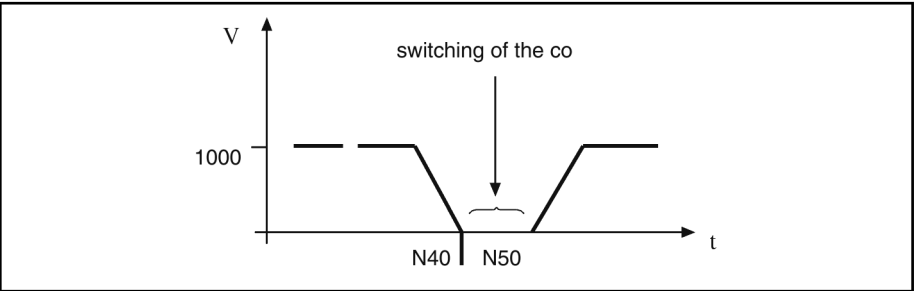


Fig. 6-12: Example 1 KvProg (KVP)

Syntax:	KvProg (<Axis1>{,<Axis2>}{,...})	
		CO programming ON
	KvProg ({0})	CO programming OFF
	Short form: KVP (. .)	

with	
<Axis>	Physical (system-wide) or logical (channel-based) axis designation, incl. the desired CO value. Max. programmable CO value: 655.35

Tab. 6-82: Syntax of KvProg (KVP)

Example:

```

:           Home position: A flow coefficient value of 1.0 is active in all
           drives.

N10 G0 X0 Y0   Positioning with CO value = 1.0
Z100

N20           Transfers the CO value 2.1 to the drive of physical axis Z.
KVP(Z2.1)

:

N30 G1 Z0      Positioning with KV value = 2.1

N40 KVP()      Deactivates CO programming
Z100           The NC automatically reloads the CO value 1.0 to all the drives.

:           Then the z-axis travels.

```

Special features and restrictions: Only axes that are currently assigned to the channel may be programmed.

6.66 Lag compensation "LCP"

Description: The "LCP" (LagCompensation) function acts on the basis of the system axis. In the NC function, the axes must be addressed by their system axis names.

Syntax: **Activation of groups:**

N100 LCP (AbsMode,<Slave1>,<OnOff>,<Slave2>,<OnOff>...)	
with	
AbsMode	If "AbsMode" has been programmed, the additional axes are controlled around their zero positions. For this purpose, MLD draws the additional axis/axes to the 0 position when compensation is switched on. If "AbsMode" has not been programmed, no synchronization motion is performed. The additional axes are controlled around their actual position.
	<Slave i> System axis name(s) of the additional axis (axes)
	<OnOff> 0: Deactivate coupling 1: Activate coupling

Tab. 6-83: LCP syntax

By means of this call, one or more additional axis/axes is/are coupled to (or decoupled from) their master(s) defined in the configuration.

Deactivation of all groups:

```
N200 LCP()
```



For the actual description of the function, refer to the documentation "MTX Functional Description".

6.67 Dividing the traversing block: Length of path segments "LEN"

Description: It divides a programmed traversing block into several partial paths of equal length.



Effective only in conjunction with the functions "punch" (see page [chapter 6.94 "Punching Punch, PUN" on page 335](#)) and "nibble" (see page [chapter 6.80 "Nibbling Nibble, NIB" on page 303](#)).

Syntax:

LEN=<value>	
with	
<Value>	For linear blocks: Length of path segments For circular blocks: Arc length The same programming unit as for the axis coordinates <value> does not have to be an integral divisor of the programmed length of path. Internally, the NC automatically computes an effective LEN value that is less than or equal to the programmed LEN value with the effect that the effective path segments are always integral divisors of the programmed length of path.

Tab. 6-84: LEN syntax

- Example:**
- G90 is active (absolute dimension programming)
 - Active plane: X/Y
 - Current position: X=0, Y=0, C=0
 - Punching/nibbling is switched off

:

N20 X100 Y100 LEN=15 Divide the following traversing blocks into equal path segments with a maximum length of 15 mm each. LEN is not yet in effect because punching/nibbling is switched off.

N30 X200 Y200 C180 Punching ON
Punch (1) LEN is in effect

The path length is split into 10 block segments.

Resulting stroke positions (X,Y,C):

P1 (110,110,18)

P2 (120,120,36)

:

P10 (200,200,180).

N40 Y290 C210

The path length (90 mm) is split into 6 block segments. Resulting stroke positions (X,Y,C):

P11 (200,215,185)

P12 (200,230,190)

:

P16 (200,290,210).

N50 Punch ()

Punching OFF

:

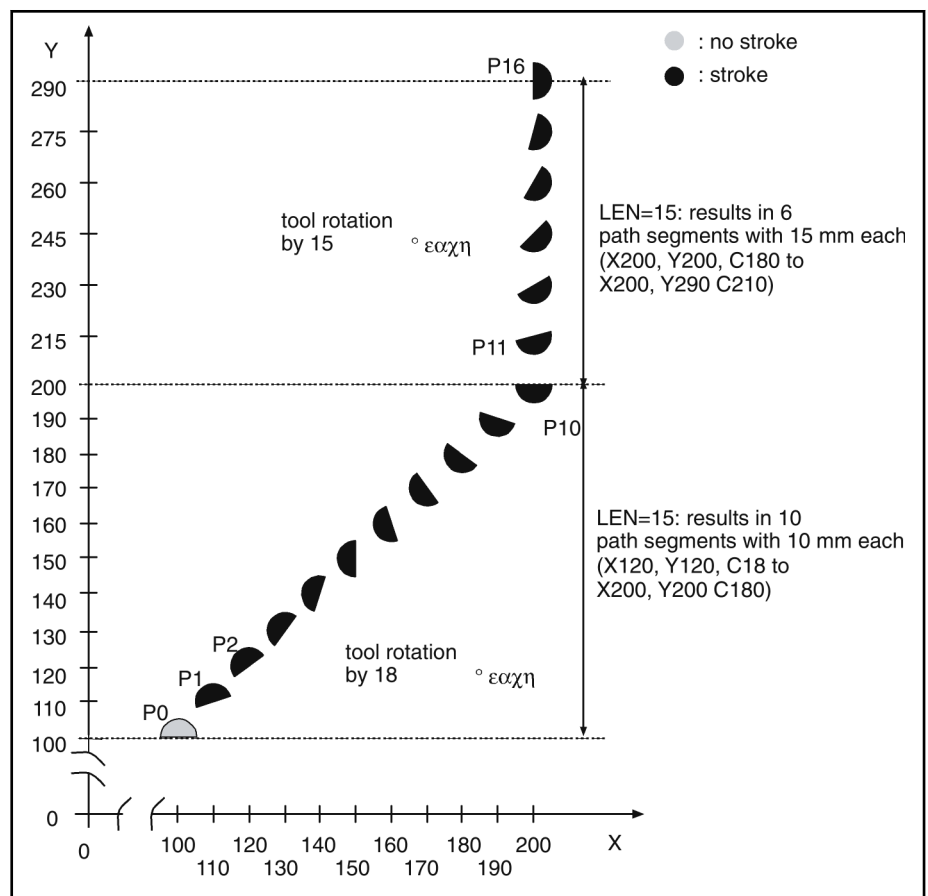


Fig. 6-13: Example LEN

Special features and restrictions:

- The motion from one stroke to the next is always linear in the case of divided circular blocks.
- LEN works modally as long as function Punching/Nibbling is active. However, it can be overridden on a block-to-block basis by NUM (see [chapter 6.81 "Dividing the traversing block: Number of path segments NUM" on page 305](#)).
- Programming LEN is possible at any time when Punching/Nibbling (MP 8001 00010) has been enabled. However, block splitting begins only after Punching/Nibbling has been activated.

6.68 Path velocity-dependent laser power control "LFP", "LFCnf, LFC"

Description: Controls the power of a laser using an analog voltage signal (0...10 V), depending on the current path velocity v_{path} .



For detailed information on this function, refer to the documentation "MTX Functional Description".

Syntax:

NC functions with high-level language syntax

LFP (1) or LFP	Laser power control ON with current parameter settings.
LFP ({LL (. . .)})	Laser power control ON with additional parameter settings as described under LFCnt.
LFP (0)	Laser power control OFF
LFCnf({LL(<Volta>,<Vmin>)}, {UL (<Volta>,<Vmax>)}, {PL (<Mode>)}, {CD (<Coord 1>{,<Coord n>}...)} ,)	Parameterize laser power control The relevant coordinates are selected by means of PL(..) or CD(..).
Short form: LFC (. .)	
with	
<Volta>	Voltage in % of the maximum output voltage (10 V) In conjunction with <Vmin>: is output in case of path velocities lower than <Vmin>. In conjunction with <Vmax>: is output in case of path velocities higher than <Vmax>.
<Vmin>	Lower path velocity limit Entered value in the unit mm/min, inches/min or degrees/min, depending on active unit of measurement (G71, G70).
<Vmax>	Upper path velocity limit Entered value in the unit mm/min, inches/min or degrees/min, depending on active unit of measurement (G71, G70).
<Mode>	Defines which coordinates are to be used to determine v_{path} : APL all coordinates of the current plane. ASP all coordinates of the current space. CFD coordinates according to MP 7050 00820.
<Coord x>	Defines which coordinates are to be used to determine v_{path} . Input values: Name of the participating spatial coordinates (if axis transformation is active) or logical axis name (pseudo; if axis transformation is not active).

Tab. 6-85: Syntax of LFP, LFCnf and LFC

Example:

LFP (LL (10,100), UL (60,700))	Laser power control ON Additional parameter settings: output voltage to 10% (=1V) if the path velocity drops below 100 mm/min, and 60% (=6V) if the path velocity increases beyond 700 mm/min.
LFP (0)	Laser power control OFF
LFCnf (PL (APL))	Parameterization only: The path velocity is derived from the path motion in the active plane.

LFConf (CD(X , Z))

Parameterization only:

The path velocity is derived from the motion of the X and z-coordinates.

LFConf (UL(80,500))

Parameterization only:

Limit output voltage to 80% (=8V) if the path velocity rises above 500 mm/min.

Special features and restrictions:

- The available analog outputs limit the number of channels that may use this function.
- The voltage signal for the laser control drops to 0 V in the following cases:
 - An error occurs (runtime error, diagnostics class 1 error)
 - The "drive in operation" signal of a drive participating in the path is deactivated (no release, drive off)
 - The "feed hold" signal becomes active.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.69 Linear to spline switching "LinearToSpline, LTS"

Description: The function changes a programmed linear interpolation G1 to a spline interpolation programming G6.

The following applies:

- Together with LCTS, the function forms an NC function group.
- The function is modal and deletes LCTS
- LTS() and LCTS() have the same effect

Syntax:

LinearToSpline(1)	Enable The following G1 commands are changed to G6.
LinearToSpline(0)	Disable All following G1 commands remain G1.
Short form: LTS	

Tab. 6-86: Syntax



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

Example: The following two program sections have the same effect.

- N10 SplineDef(2203,X,Y,Z) LTS(1)
N20 G1
- N10 SplineDef(2203,X,Y,Z)
N20 G6

6.70 Linear-circular-spline-conversion"LCTS"

Description: The function converts a contour programmed with straight lines and circles into a spline sequence.

Each motion specified with G1 (straight line interpolation with feed) and with G2/G3/G5 (circle/helical interpolation) is split up into a straight move. The number of points depends on the specified tolerance (BAP(E<Tol>). The generated points are subsequently connected by splines.

Programmed as well as MTX-internally generated straight lines and circles (roundings, chamfers, inserted blocks of the milling path correction) are converted.

Local, motion-generating NC functions (e.g. G74, G75, G76) are exempt from the conversion.

The following applies:

- Together with LTS, the function forms an NC function group
- The function is modal and deletes LTS
- LCTS() and LTS() have the same effect



The function is described in detail in the "MTX Functional Description - Extension" documentation.

Syntax:

LCTS	Enable All subsequent G1, G2, G3 motions are converted to G6 motions.
LCTS ()	Disable The conversion is disabled.

Tab. 6-87: LCTS syntax

Example:

The following two program sections have the same effect.

- N10 LCTS BAP(E0.005)
N20 G1 ..
N30 G2 ..
N40 G1 ..
- N10SPD(4233) BAP(E0.005)
N20 G6 ..
N30.1 G6 .. ; 1st part of the circle
N30.2 G6 .. ; 2nd part of the circle
..
N30.x G6 .. ; last part of the circle
N40 G6 ..

6.71 Zeroing the modulo axis "LinModZp, LMZ"

Description:

"LinModZp" can be used to define the current position of a linear modulo axis as a new program zero point.

Afterwards, the axis display jumps to the value "0". An offset value is control-internally calculated for the linear modulo axis. This offset value is added to all following programmed axis positions.

Syntax:

LinModZp{ (<LinModAxis>) }

Short form: LMZ (..)

with	
<LinModAxis>	Optional parameter: System axis index, system axis name or channel axis name of the linear modulo axis to be zeroed If no axis is specified, the zero point is set again for all linear modulo axes of the channel.

Tab. 6-88: Syntax of LinModZp (LMZ)

Example: LinModZp (LMZ)

N50 LMZ (1) X20	Defines the current position of the linear endless axis with the system axis index 1 as new program zero point. Subsequently move to the position X20.
N90 LMZ (X) X-43.7	Defines the current position of the x-axis as new program zero point. Then, move the x-axis to the position -43.7.
N120 LMZ G91 X7.3	Reset all linear modulo axes in the channel to zero and subsequently traverse the x-axis into positive direction by 7.3 mm.

Special features and restrictions:

- The traversing range of a linear modulo axis is restricted by the limit values "**minus modulo value**" and "**plus modulo value**" related to the current zero point of the axis.
- Programming a position outside the traversing range defined by the limit values, causes a runtime error.
- A linear modulo axis can be traversed absolutely or incrementally within these current limit values.
- Using the LMZ command, the current axis position is applied as new zero point and the traversing range is moved correspondingly.
- By setting the zero point again permanently, a linear modulo axis can be endlessly traversed into one direction. As the command positions towards the drive are modulo-calculated, no position overflow results.
- Even though the linear modulo axis is a modulo axis at the driving end, from the point of view of the program, it responds like an ordinary linear axis with modulo values effective like end switches. The only feature is that the zero point and thus the traversing range can be moved by the LMZ command.
- A traversing motion can be programmed together with the LMZ command in one line. In this case, the LMZ command does not interrupt a motion sequence.



The modulo value is defined in the drive using the Sercos parameter S-0-0103 when the Sercos is started up. A changed modulo value is applied only after the Sercos is restarted!

6.72 Velocity profiles (ramp functions)

Description: The function allows the definition of machining sections that are to be executed by means of separate velocity profiles.

The following "individual blocks" are available for this purpose:

- 1 constant feed interpolator
- 3 acceleration interpolators and 3 deceleration interpolators
(each with a linear, sinusoidal and sine²-shaped velocity profile)

All functions described below work modally, forming a modal group with G8 and G9, and deselect each other.

The following applies:

1. **Acceleration interpolators "LinUpFeed, LNU", "SinUpFeed, SNU", "Sin2UpFeed, S2U"**

Starting from velocity V_0 (velocity at the beginning of the block), the control accelerates across the entire programmed path length to a target velocity (V_1).

Depending on the programmed function, this is performed with a linear, sinusoidal or sine²-shaped velocity increase.

The target velocity V_1 is reached together with the programmed end point, and results from the programmed feed as a function of the current override value.

It is limited by the

- maximum path acceleration and
- the maximum permitted path velocity.

Both quantities are calculated by the control specifically for each path segment and each NC block; a 1-block look-ahead is performed in connection with the maximum permitted velocity. This prevents a violation of the maximum axis velocities in the next block.



If V_1 is not greater than V_0 , the call of the acceleration interpolator in the current block is ignored.

Behavior in the event of override changes:

- Increasing the override results in the recalculation of the acceleration ramp.
- An override reduction to final values below the start velocity V_0 has the following consequences:
 - The reduction results in a recalculation of the braking ramp in the case of "LinUpFeed"; this lasts until the programmed end point.
 - The reduction is ignored in case of "SinUpFeed" and "Sin2UpFeed".

2. **Effect constant feed interpolator "ConstFeed, CFD"**

The control tries to reach the programmed target velocity, taking the maximum permitted path velocity and the current override value into account.

Velocity changes during override changes are executed with the permitted accelerations in each case.

3. **Deceleration interpolators "LinDownFeed, LND", "SinDownFeed, SND", "Sin2DownFeed, S2D"**

Starting from velocity V_0 (velocity at the beginning of the block), the control brakes across the entire programmed path length to a standstill ($V_1=0$).

Depending on the programmed function, this is performed with a linear, sinusoidal or sine²-shaped velocity decrease.

Override changes have no effect, except for the following:

If the override was set to 0% in the previous block and the target velocity of 0 was reached afterwards exactly at the block switching to the deceleration interpolator, the control maintains the deceleration interpolator until the override is increased to a value > 0!

The velocity is increased by one acceleration step (depending on the permitted path acceleration).

The control calculates the necessary deceleration ramp on the basis of the velocity value resulting from this function. Afterwards, the actual override value is not applied until the end of the block.

Syntax:

LinUpFeed Short form: LNU	Activates acceleration interpolator with linear increase in velocity.
SinUpFeed Short form: SNU	Activates acceleration interpolator with sine-shaped increase in velocity.
Sin2UpFeed Short form: S2U	Activates acceleration interpolator with sine ² -shaped increase in velocity.
ConstFeed Short form: CFD	Activate constant feed interpolator
LinDownFeed Short form: LND	Activates acceleration interpolator with linear velocity decrease.
SinDownFeed Short form: SND	Activates acceleration interpolator with sine-shaped velocity decrease.
Sin2DownFeed Short form: S2D	Activates acceleration interpolator with sine ² -shaped velocity decrease.

Tab. 6-89: Syntax of LNU, SNU, S2U, CFD, LND, SND and S2D

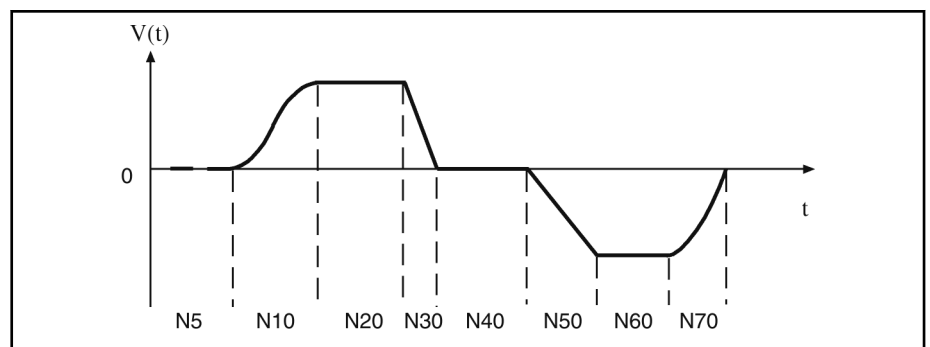
Example: Velocity profile for an oscillation cycle of the U axis

Fig. 6-14: Velocity profile for an oscillation cycle of the U axis

N5 G0 U10	U axis is traveled to starting position (U=10mm)
N10 S2U U17 F500	Sine ² -shaped acceleration up to position U=17 Command feed at end point: F=500 mm/min
N20 CFD U23	Constant feed until position U=23
N30 LND U29	Linear braking until position U=29 End velocity: 0 mm/min
N40 G4 F0.5	Dwell time at the point of reversal

NC functions with high-level language syntax

N50 LNU U20	Linear acceleration until position U=20
N60 CFD U17	Constant feed until position U=17
N70 SND U10	Sine-shaped deceleration until the position U=10
	End velocity: 0 mm/min

Special features and restrictions:

- The desired end point coordinate must always be entered in the programmed block.
- All ramp functions can be used only in the "Automatic/Continuous block" operation mode. Other operation modes (Manual data input, Single block, Single step, or Program block) lead to a runtime error.
- When using braking interpolators in connection with extremely short travel paths, excessive acceleration is possible, which may lead to a servo error.

Therefore, observe the maximum possible machine dynamics already when creating the part program.

- Auxiliary functions or functions such as "Exact stop" are not permitted while constant feed and acceleration interpolators are active. The use of these functions can result in sudden velocity drops!

The following functions are prohibited:

G0, G4, "KvProg", G63, G33, G61, "G1(IPS)", G75, "MOB", "G74(HOME)", "HsBlkSwitch", "Writeld".

6.73 Main spindle switching "MainSp, MSP"

Description: Other than the setting in MP 7020 00010, it defines the given spindle as the main spindle in the current channel.

Thus, it can be determined in the part program to which spindles the following functions should apply:

- G33 (thread cutting)
- G95 (feed programming in mm/rev), and
- G4 (dwell time)

Syntax:

MainSp (<Num>) or MainSp (<Spindle>)	
Short form: MSP...	
with	
<Num>	Number of the spindle (spindle index) Input range: -1; 1...8 Integer -1: main spindle according to MP 7020 00010
<Spindle>	Same as <Num>, or spindle name (e.g. S1)

Tab. 6-90: Syntax of MainSp (MSP)

6.74 Input tool: Mirroring "Mirror(...), MIR(...)"

Description: The "mirroring" function is one of the input tools.

The control machines a programmed contour or, for instance, a borehole pattern in a mirror-inverted way.

Mirroring always refers to the current reflection point (see function "PoleSet" (chapter 6.87 "Defining the reflection/rotation point PoleSet, PLS" on page 322). If this was not explicitly programmed, the current program zero point is the reflection point.



Mirroring is an input tool; as a result, it does not change the current program coordinate system. An input tool merely provides another input method for the program coordinates.

Mirroring can also be used together with scaling and rotating.

The following applies:

- The function acts modally. It remains active until it is disabled.
- It may be programmed in the same block as other path conditions and input tools.

Syntax:

<code>Mirror(<Axis 1>{, <Axis n>1})</code>	
	Activates mirroring for the given axes.
<code>Mirror(<Axis 1>0{, <Axis n>0})</code>	
	Deactivates mirroring for the given axes.
<code>Mirror(0)</code> or <code>Mirror()</code>	Deactivates mirroring for all the axes in the channel. Approached axis positions are retained until they are reprogrammed.
Short form: <code>MIR(. .)</code>	
with	
<code><Axis1>, <Axes></code>	Axes addresses (e.g. X) in conjunction with a value of "1" are activated by the function: all subsequently programmed path commands of the corresponding axes (e.g. X100) are multiplied internally by a value of "-1". Mirroring is not applied before the next traversing information.

Tab. 6-91: Syntax of `Mirror(...)`, `MIR(...)`

Examples: **Mirroring**

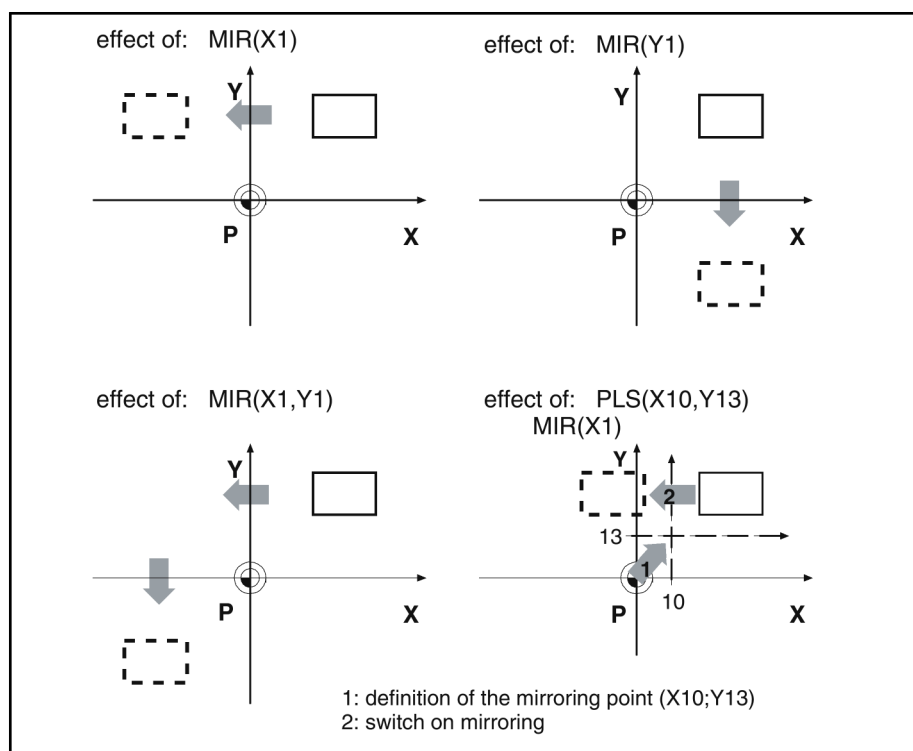


Fig. 6-15: Mirroring examples

Special features and restrictions:

- The function takes into account the interpolation parameters in the case of circular interpolation.
 - It influences the programmable contour shift.
- Refer to [chapter 6.116 "Programmed contour shift Shift, SHT" on page 363](#).
- *It does **not** influence:*
 - Zero offsets (G54-G59.5; see [chapter 5.2.21 "Zero offsets \(ZO\) G53, G53.1-G59.1 to G53.5-G59.5" on page 142](#))
 - Program coordinate offsets (Trans or ATrans; see [chapter 6.151 "Program coordinate offset Trans, TRS, Additive Program Coordinate Offset ATrans, ATR" on page 405](#)),
 - Set program position ("SetPos"; see [chapter 6.115 "Setting program position SetPos, SPS" on page 362](#)),
 - Approach reference point coordinates (G74; see [chapter 5.2.26 "Approaching the reference point coordinates G74" on page 149](#)),
 - Approach to fixed machine axis position (G76; see [chapter 5.2.29 "Approaching the fixed machine axis position G76" on page 152](#))
 - Milling tool radius and tool length correction values.

Mirroring an orientation vector:

An orientation vector is only mirrored on a component-by-component basis using the following function syntax.

Any scaling that may be active or a mirror/rotation point has no effect on the result.

Syntax:

Mirror(O(<Sx>,<Sy>,<Sz>))	
	Activates/deactivates mirroring for the given components of an orientation vector.
Short form: MIR(. . .)	
with	
<Sx>,<Sy>,<Sz>	Reflection factors for the individual vector components. 0: No mirroring 1: Mirroring

Tab. 6-92: Mirror syntax

The following applies to the orientation vector:

- The polar coordinates φ and ϑ themselves cannot be mirrored.
- Programming of MIR(phi1, theta1) is not permitted.

6.75 Measuring on a block "MOB"

Description: MOB is function that acts block-by-block and which can be used to measure a workpiece position.

MOB is programmed together with a traversing motion (G1, G6, TcsMove, ...).

Syntax:

MOB	
	Measuring with probe 1
MOB(MP<Touch probe>)	
	Measuring with probe 1 or 2
with	
<Touch probe>	Selects a probe (1 or 2)

Tab. 6-93: Syntax of NC function MOB

Example: MOB as option for G75.2

```

..
N100 MOB(MP2) G1 Y250 F100          ;MOB with measuring probe 2
0110 IF SD(9) = 0 THEN
0120   YPOS! = PCSPROBE("Y")
0130 ELSE
0140   SETWARN("NO CONTACT")
N150   M0
0160 ENDIF
..

```

Measuring in tool direction

```

N010 CRD(2)                          ;Activate an axis transformation
..
N100 G1 z50 F1000
N110 G1 x50 y70 F1000
..
N200 MOB TCM( , , -20) F100          ;MOB with traversing in TCSz direction

```

```
0210 IF SD(9) = 0 THEN
0220   ZPOS! = PCSPROBE("z")
0230 ELSE
0240   SETWARN("NO CONTACT")
0250   M0
0260 ENDIF
..
```

- Special features and restrictions:**
- Commissioning the probe is described in the manual "MTX Functional Description - Extension" in chapter "Measuring on a block (MOB)".
 - MOB has to be programmed together with a traversing motion.

6.76 Measuring on a contour (MOC)

Description: The NC function `MOC` can be used to measure along any path desired. The measuring path consists of motion-generating blocks and is defined by one `MOC` with parameters and one without parentheses.



We recommend to write the switch-off of the measurement (`MOC()`) to the "state after channel reset" (machine parameter `CHAN/Ch[k]/Ini/ChResetState`). Otherwise, the measuring path would not be completed, above all not with the reset reaction (`RE3`).

Once the measurement event occurs, the measured positions are written to the system datum `/SysMeasPos`. Additionally, CPL functions of `G75` (such as `PROBE`, `SD(9)` ...) can be used to query the measured positions in the part program.



If CPL functions are used for reading the measured values, it is mandatory to set a `WAIT` command between the end of the measuring path and the CPL functions.

If a motion is programmed in the starting block of the measurement, the measurement is started on-the-fly (i.e., without any intermediate stop). The same applies to the end of the measuring path.

Only one `MOC` command is active at a time. A measurement can be completed by programming another measurement.



Multiple measurements cannot be taken. This means that, within a measuring path, the `MOC` logic is activated only until the first measurement event occurs and is then switched off. A further measurement can only be taken, if `MOC` is programmed once more.

Syntax:

```
MOC (MP<Probe>|CI<CustomerInput>|HS<HighSpeed>{, ES<Edge>}, RE<Reaction>{, AS<Asup>}{, ER<Error>}{, <ChannelAxisName1>{, <ChannelAxisName2> ...}})
```

with

<Touch probe>	Selects a probe (1 or 2) as signal source.
<Customer input>	Selects a customer input in the channel interface (1...8) as signal source.
<High-speed>	Selects a high-speed input (1...8) as signal source.

<Edge>	Selects the falling edge (1), the rising edge (2) or both (3) edges as measurement triggers. If a customer input or a High-Speed input has been selected as signal course, edge selection is a mandatory parameter. If a touch probe is used, selection of the edge is optional. If the touch probe is not programmed, the configured edge (see MTX Machine parameters MEAS/Dr[i]/Probe1Edge, MEAS/Dr[i]/Probe2Edge, and MPEDGE, chapter 6.79 "Changing the touch probe edge configuration (MPEDGE)" on page 302) can be used as a trigger for MOC.
<Response>	One of the following reactions can be triggered by the measurement result: 1. Switching the path velocity 2. Triggering an asynchronous subroutine 3. Resetting the channel 4. No reaction (flying measurement) 5. Skipping the remaining measuring path For a detailed description of possible reactions, refer to the manual "MTX Functional Description", chapter "Measuring on a contour".
<Asup>	Number of a logged-in asynchronous subroutine to be executed after the measurement event (only with RE2). The asynchronous subroutine must have been logged in to the system before MOC is called (e.g., with ASPSET, see chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211).
<Error>	Reaction to measurement events that have not occurred along the measuring path. Potential reactions are: • 0: No message (default setting) • 1: Output warning • 2: Output error message
<ChannelAxisNameX>	Axis names of all channel axes to be measured This data is only relevant if a touch probe is used for measurement. If axis data is not specified, the measurement is carried out with all axes of the channel which are configured for the touch probe (see documentation "MTX Machine Parameter", MEAS/Dr[i]/EnablProbeFunc).
MOC ()	
	End of measuring path The measurement is switched off and, if the measurement event did not occur, a message is generated depending on ER.

Tab. 6-94: Syntax of the NC function MOC

Example:

N10 ASPSET(3, SAFE- POS01, 11)	; Log in asynchronous subroutine "SAFEPOS01" to the system as number 3.
N20 MOC(CI4, ES1, RE2, AS3, X, Z)	; If customer input 4 delivers a falling edge, Asup 3 (SAFEPOS01) is executed and axes X and Z are measured

NC functions with high-level language syntax

N30 X... Y... Z...	; Measurement is in progress
N40 X... Y... Z...	; Measurement is in progress
N50 X... Y... Z...	; Measurement is in progress
N60 X... Y... Z...	; Measurement is in progress
N70 MOC()	; Measurement is completed (the touch probe is deactivated)

Special features and restrictions:

- Only one signal source (touch probe/customer input/high-speed input) may be selected.
- If reaction 2 (triggering an asynchronous subroutine) is used for input of a buffered NC block, the block buffer is cleared after completion of the asynchronous subroutine. In this case, the measuring path specified is only to be travelled until the measurement event occurs.
- Measuring axes may not be removed from the channel along the measuring path.
- The position measured in the program coordinate system (PCS) is supplied only if all axes of the channel were measured. This is based on the (axis) configuration active at the moment of measurement.
- It is not allowed to program a `G75`, `MOB` or `FlyMeas` command on a measuring path. It is neither allowed to reconfigure the measuring edges with `MPEDGE`.

Special features of reaction RE5 (skipping the remaining measuring path):

- It is not allowed to change zero point offsets, placements and transformations along the measuring path.
- M-functions in skipped blocks are applied; modal functions are switched over.
- CPL functions in skipped blocks are processed completely.
- If blocks are skipped, asynchronous subroutines cannot be started.
- If the block of the measurement distance end does not contain any programmed motion, the NC travels to the last position of the measuring path in this block using the motion function active at that point.



There is a traversing motion in the measuring path end block (`MOC()`) after skipped blocks, even if such a motion has not been explicitly programmed.

It is useful to program a traversing motion (incl. motion function) to an absolute and safe position in the measuring path end block. It must be ensured that axes can move to this position from any point of the measuring path without collision.



We recommend to incorporate the end of the measuring path (`MOC()`) into the "state after channel reset" (machine parameter `CHAN/Ch[k]/Ini/ChResetState`) because, otherwise, runtime errors may occur during the reset if an error occurred while blocks were being skipped.

Special features if a touch probe is used:

- It is not allowed to program 3 (both edges) for edge selection.
- The measuring edges of all measuring axes must be configured identically and correspond to the programmed value of `ES`.

- The measurement only involves axes the channel axis names of which are specified. If channel axis names were not programmed, all channel axes that are configured for use with the touch probe are measured (machine parameter `MEAS/Dr[i]/EnablProbeFunc`).
- If a measuring axis is not configured for use with the touch probe (machine parameter `MEAS/Dr[i]/EnablProbeFunc`), a runtime error is generated.

6.77 Modulo calculation for incrementally traveling endless axes "Modulo, MOD"

Description: If endless axes are traveled incrementally, modulo calculation is suppressed in the control. This is to ensure that the defined travel path is really traveled. Permanent modulo calculation is only used for the drive and the display values target values. For the axis values in the control, modulo calculation is only carried out with the next absolute position information.

If an endless axis is incrementally traveled in the same direction over and over again, it finally reaches the internal limit of the displayable values. This is acknowledged by the control by means of a limit switch error.

If the sum of the incremental motions exceeds this limit value, it is necessary to reset the internal axis memories to the range of the modulo in a coordinated way. For this purpose, the "modulo" function is used.

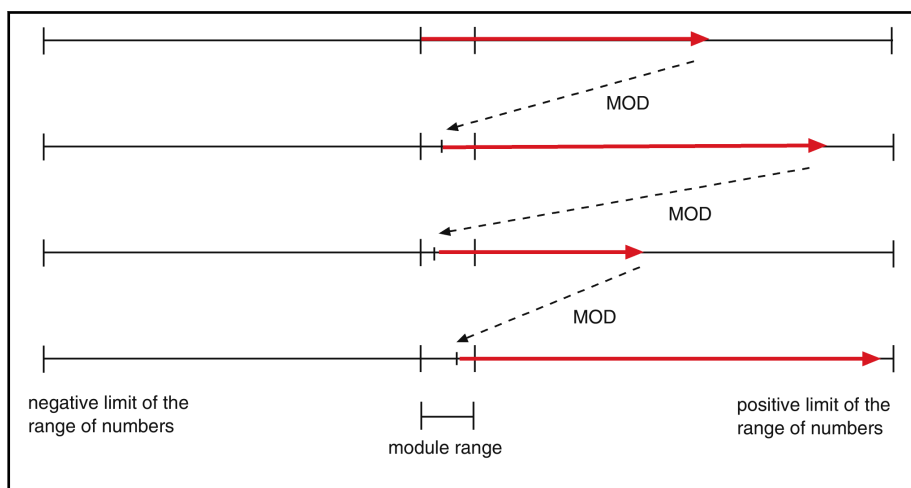


Fig. 6-16: Modulo range

Syntax: Modulo calculation for all endless axes of the channel:

Modulo
Short form: MOD

Tab. 6-95: MOD syntax - for all endless axes

Modulo calculation for individual endless axes of the channel:

Modulo(<AN1>{,<AN1>{, ...}})	
Short form: MOD (. . .)	
with	
<ANx>	being the axis name of the modulo axis (channel or system axis name)

Tab. 6-96: MOD syntax - for individual endless axes



The modulo command may be programmed together in one block with any traversing motion. The modulo command does not lead to a sudden velocity drop.

The module command can also be used for rotary axes if they have been configured for modulo operation (for drive).

Example:
:
N030 MOD(A,B) A270 B90 F1000
:

6.78 Modulo offset for endless axes"ModuloShift, MODS"

Description: Facilitates the offset of the program position of a modulo or rotary axis with configured module operation by a specified number of rotations.

Syntax:

MODS () { , ES1 ES2 }	
with	
MP1 MP2	Selects probe 1 or 2 for changing the configuration
ES1 ES2	Selects the falling (ES1) or the rising (ES2) edge for the touch probe. If neither ES1 nor ES2 is programmed, MPEDGE resets the edge selection for the touch probe back to the configuration setting.
MPEDGE ()	
	Resets the edges of all touch probes back to the configuration setting.

Tab. 6-97: Syntax of the NC function MPEDGE

Example:
:
N030 MODS(A1,B-2)
:

6.79 Changing the touch probe edge configuration (MPEDGE)

Description: By the following machine parameters, it is defined at which edge (rising or falling) of the touch probe input the drive should trigger a measurement for an individual drive:

MEAS/Dr[i]/Probe1Edge and

MEAS/Dr[i]/Probe2Edge

This configuration can be overridden with the NC function MPEDGE. The new edges are active for all functions using a touch probe (G75, MOC ...).

Syntax:

MPEDGE (MP1 MP2{ , ES1 ES2 }	
with	
MP1 MP2	Selects probe 1 or 2 for changing the configuration
ES1 ES2	Selects the falling (ES1) or the rising (ES2) edge for the touch probe. If neither ES1 nor ES2 is programmed, MPEDGE resets the edge selection for the touch probe back to the configuration setting.
MPEDGE ()	
	Resets the edges of all touch probes back to the configuration setting.

Tab. 6-98: Syntax of the NC function MPEDGE

Example:

N100 MPEDGE(MP2, ES2)	; Probe 2 reacts to rising edge
...	
N200 MPEDGE(MP1)	; Edge of probe 1 back to configuration parameter
...	
N300 MPEDGE()	; All touch probes reset to configuration parameter

Special features and restrictions:

- MPEDGE stops at the start of the block (path velocity is zero).
- It is not allowed to change the configuration along the measuring path of an MOC.
- It is not possible to trigger a measurement at both edges.

6.80 Nibbling "Nibble, NIB"

Description: Switches the "Nibbling" function on or off.

If nibbling is activated, no stroke is released in the following cases:

- at the end of every path segment programmed or generated by the "NUM" (see [chapter 6.81 "Dividing the traversing block: Number of path segments NUM" on page 305](#)) or "LEN" (see [chapter 6.67 "Dividing the traversing block: Length of path segments LEN" on page 286](#)) function, and
- at the beginning of the first partial path if no traversing motion was programmed in the active plane or nibbling was switched off in the previous block.

The subsequent traversing motion always starts before the stroke is finished.



Functions to influence the stroke release time:

- PtDefault
(refer to [chapter 6.92.2 "Setting the stroke release time to default PtDefault, PTD" on page 331](#))
- PtBlkEnd
(refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE" on page 329](#))
- PtInpos
(refer to [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332](#)) is opened.

Syntax:

Nibble (1) or Nibble	Nibbling on
Nibble (0) or Nibble ()	Nibbling OFF
Short form: NIB (. .)	

Tab. 6-99: Syntax of Nibble (NIB)

Example:

- G90 is active (absolute dimension programming)
- Active plane: X/Y
- Current position: X=0, Y=0, C=0

NC functions with high-level language syntax

:	
N10 LEN=12	Divide traversing blocks into identical partial paths with a maximum of 12 mm. LEN must be programmed before Nibble is switched on!
N20 C10 Nibble(1) G9(ASHAPE)	Nibbling ON. Modal LEN is in effect. Enable interpolator with nibbling logic C axis rotates to 10 degrees. No stroke because the X and Y axes are not programmed.
N30 X0	No traversing motion because the x-axis is already located at 0 position. A stroke is carried out because the x-axis is located in the active plane.
N40 X110	The traversing block is divided into 10 identical 11 mm path segments. Strokes are executed in positions X11, X22, X33 ... X99, X110. Additional stroke at position X0 because no traversing motion occurred in N30.
N50 Y30 NUM=3	Overrides the modally effective LEN (N10) for the current block. Divide the traversing blocks into 3 identical path segments. Strokes at Y10, Y20, Y30.
N60 Y90	LEN from N10 is in effect again. Strokes at Y42, Y54, Y66, Y78, Y90.
N70 X50 Y50 Nibble() G8	Nibbling OFF Path slope ON
:	traversing motion to X=50, Y=50.

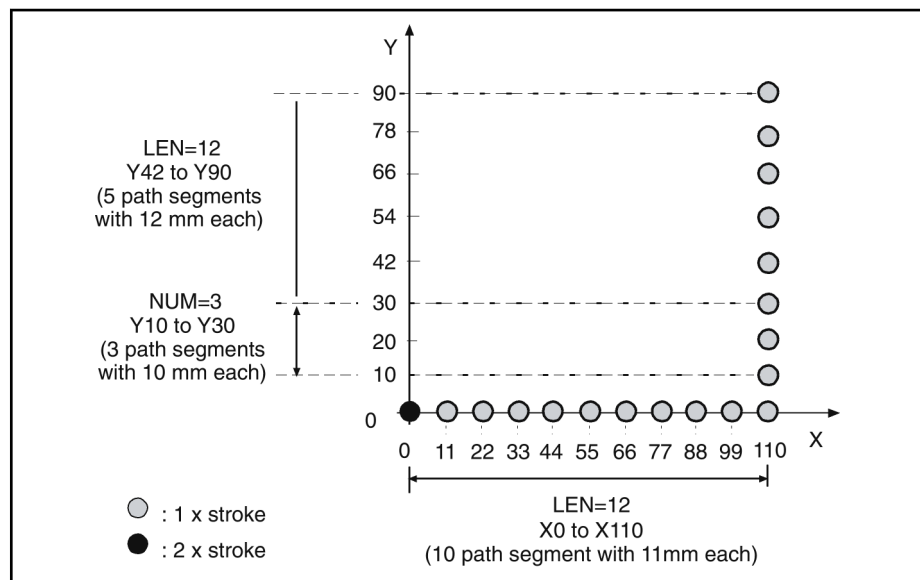


Fig. 6-17: Example Nibble (NIB)



To use the nibbling function, an interpolator supporting the nibbling functionality has to be enabled:

G9(X... , Y... , ...)

G9(ASHAPE)

G9(SHAPE... , PUN)

Special features and restrictions:

- Nibbling must be enabled using the MP 8001 00010.

- Programming of the function "NUM" or "LEN" is obligatory. The control thus automatically generates partial paths from the programmed distance traveled; a stroke is released at the end of each of these path segments.
- The function already has a modal effect on traversing motions that are programmed in the same block.
- If punching is switched on (see [chapter 6.94 "Punching Punch, PUN" on page 335](#)), the function deselects "Nibbling".
- Blocks that do not contain axis coordinates from the active plane also do not release a stroke.
- If the PLC suppresses the stroke release, processing remains at the stroke release position until the PLC releases the stroke release.

6.81 Dividing the traversing block: Number of path segments "NUM"

Description: Divides a programmed traversing block into a defined number of path segments of equal length.



Effective only in conjunction with the functions "punch" (see [page chapter 6.94 "Punching Punch, PUN" on page 335](#)) and "nibble" (see [page chapter 6.80 "Nibbling Nibble, NIB" on page 303](#)).

Syntax:

NUM=<value>	
with	
<Value>	Number of path segments. Input value: Integer, greater than 0. NUM=1 does not produce any division.

Tab. 6-100: Syntax of NUM

Example: See the example in [chapter 6.67 "Dividing the traversing block: Length of path segments LEN" on page 286](#).

- Special features and restrictions:**
- The motion from one stroke to the next is always linear in the case of divided circular blocks.
 - NUM only has an effect in the programmed block and suppresses any active LEN (see [chapter 6.67 "Dividing the traversing block: Length of path segments LEN" on page 286](#)).
 - Programming NUM is possible only when Punching/Nibbling has been enabled (MP 8001 00010) and is active.

6.82 Orientation programming "O()", ROTAX()", "Phi, Theta, Psi"

6.82.1 Overview of the orientation programming using O(...)

Tool orientations can often be programmed at the positions of the phi, theta, psi using the syntax O (<parameter list>). The parameter list can consist of one to four individual parameters. The following table provides an overview:

Syntax	Description
$O(<\beta>)$	Rotation of the tool around a fixed rotary axis with angle β . Before or in the same block, a rotary axis has to be programmed using RotAx. For vector and tensor orientation
$O(<\varphi>,<\vartheta>)$	Tool orientation in the coordinates phi and theta. Optionally, $\text{phi}<\varphi> \text{ theta}<\vartheta>$ can be programmed. Only for vector orientation
$O(<x>,<y>,<z>)$	Tool orientation, programmed as orientation vector with the three Cartesian components x, y and z. Only for vector orientation
$O(<\varphi>,<\vartheta>,<\psi>)$	Tool orientation in the coordinates (Euler angle) phi, theta and psi. Optionally, $\text{phi}<\varphi> \text{ theta}<\vartheta> \text{ psi}<\psi>$ can be programmed. Only for tensor orientation
$O(<w>,<x>,<y>,<z>)$	Tool orientation, programmed as quaternion (w,x,y,z). Only for tensor orientation

Tab. 6-101: Overview O-programming:



The vector orientation becomes active by enabling an axis transformation whose 100th place is equal to 2 regarding the type. Example: Type = 3232201

The tensor orientation becomes active by enabling an axis transformation whose 100th place is equal to 3 regarding the type. Example: Type = 3306301

6.82.2 General description

Orients a tool (milling machine, drill, laser, gripper) anywhere in space based on the current program coordinate system (PCS). Various types of orientation programming are available.



If the TCP is to be moved in addition to the tool orientation motion, also see "space coordinate programming" function in the "MTX Functional Description" documentation.

6.82.3 Linear orientation motion with axis programming

Description:



Only applies to 5-axis transformation supporting a mixed linear coordinate/rotary axis programming, such as axis transformation type 3032001.

To program the tool orientation, the position values of two rotary axes that act on the tool are programmed (e.g. B, C).

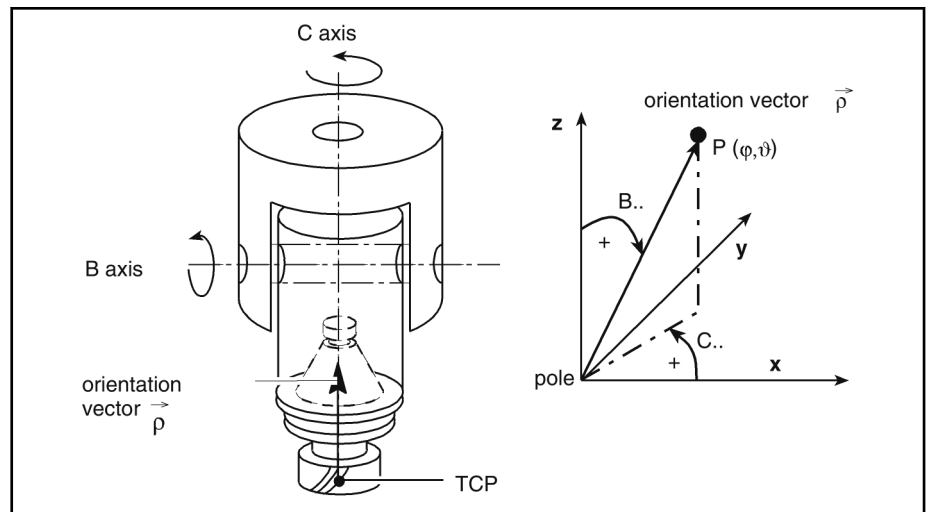


Fig. 6-18: Linear orientation motion with axis programming

The following applies:

- Only the axis kinematics are permitted for which the position values of both rotary axes can be mapped on a one-to-one basis to the orientation coordinates of the orientation vector (e.g. phi and theta).
- The orientation motion is performed as a linear motion in the rotary axes.
- This is suitable for rotation-symmetrical tools.

- Syntax:**
1. Activate axis transformation type **3032001** with "Coord(...)" (see [chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245](#)).
 2. Use the syntax shown below.

{<No>} {<TCP motion>} {<Pos>} {<C><Pos>}	
with	
<No>	Block number Refer to chapter 3.8.2 "Block numbers" on page 27 .
<TCP mot>	Additional TCP motion For the syntax, refer to the documentation "MTX Functional Description" in the section on "Space coordinate programming".
,<C>	Axis addresses of the rotary axes located at the tool.
<Pos>	Absolute axis position in degree

Tab. 6-102: "Linear orientation motion with axis programming" syntax

Example: Example settings:

Parameter TRA/AxTrafo/TrafoType (MP 1030 00110) for axis transformation 2: 3032001

Parameter CHAN/Ch/Coord/Wcs/LinCoord [1] (MP 7080 00010 [1]): x

Parameter CHAN/Ch/Coord/Wcs/LinCoord [2] (MP 7080 00010 [2]): y

Parameter CHAN/Ch/Coord/Wcs/LinCoord [3] (MP 7080 00010 [3]): z

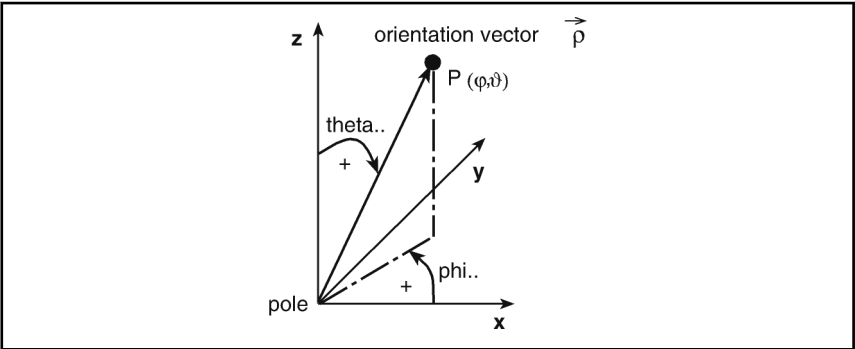
N10 G1 X0 Y0 Z0 B0 C0	Programming logical/physical axis names
N20 Coord(2)	Activates axis transformation 2 Linear coordinates x, y, z and the orientation coordinates B and C can now be programmed.
N30 x100 y200 z300 B20 C60	Linear coordinate interpolation with additional orientation motion
N40 G2 x.. y.. z.. I.. J.. B20 C60	TCP helical motion with additional orientation motion
N50 G1 B20 C10	Pure orientation motion TCP remains constant.
N60 Coord(0)	Axis transformation OFF
:	

- Special features and restrictions:
- The active axis transformation type 3032001 has to be active.
 - Special path search logic avoids rotary axis rotations of more than 180 degrees.

6.82.4 Linear orientation motion with coordinate programming

Description:	 Applies to all 5-axis transformations whose 100th place is equal to 1 regarding the type, such as type = 3232101.

- The tool orientation is programmed using the
- orientation coordinates of the orientation vector
(e.g. phi and theta).



- Fig. 6-19: Orientation coordinates of the orientation vector
- This is suitable for rotation-symmetrical tools.
 - The orientation motion is realized as linear interpolation in φ and ϑ , i.e., as a straight line in an assumed φ/ϑ -plane.
- Syntax:
1. Enable axis transformation (e.g. with type = **3232101**) with "Coord...)" (see [chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245](#)).
 2. Use the syntax shown below.

{<No>} {<TCP mot>} {<phi><φ>} {<theta><θ>}
with

<No>	Block number See chapter 3.8.2 "Block numbers" on page 27
<TCP mot>	Additional TCP motion For the syntax, refer to the documentation " MTX Functional Description" in the section on "Space coordinate programming".
<phi>	Angle name entered in the parameter CHAN/Ch/Coord/Wcs/OriCoord[1] (MP 7080 00010 [4]). Default: phi
<theta>	Angle name entered in the parameter CHAN/Ch/Coord/Wcs/OriCoord [2] (MP 7080 00010 [5]). Default: theta
<φ>,<θ>	Absolute angle values in degrees

Tab. 6-103: "Linear orientation motion with coordinate programming" syntax

Example: Example settings:

Parameter TRA/AxTrafo/TrafoType (MP 1030 00110) for axis transformation 2: 3232101

Parameter CHAN/Ch/Coord/Wcs/LinCoord [1] (MP 7080 00010 [1]): x

Parameter CHAN/Ch/Coord/Wcs/LinCoord [2] (MP 7080 00010 [2]): y

Parameter CHAN/Ch/Coord/Wcs/LinCoord [3] (MP 7080 00010 [3]): z

Parameter CHAN/Ch/Coord/Wcs/OriCoord [1] (MP 7080 00010 [4]): phi

Parameter CHAN/Ch/Coord/Wcs/OriCoord [2] (MP 7080 00010 [5]): theta

N10 G1 X0 Y0 Z0 B0 C0	Programming channel or system axis names
N20 Coord(2)	Activates axis transformation 2 The linear coordinates x, y, z and the orientation coordinates phi, theta can be programmed now.
N30 x1 y2 z3 phi5 theta5	Linear coordinate interpolation with additional orientation motion
N40 Coord(0)	Axis transformation OFF

- Special features and restrictions:**
- Axis transformation type with 100th place = 1 is required.
 - Special path search logic avoids rotary axis rotations of more than 180 degrees.

6.82.5 Vector orientation

Description: *Programming the tool orientation is also possible through*

- Orientation coordinates of the orientation vector
(e.g. phi and theta).

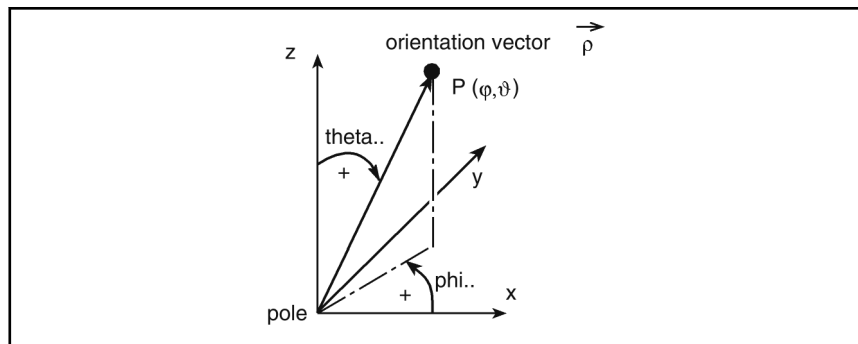
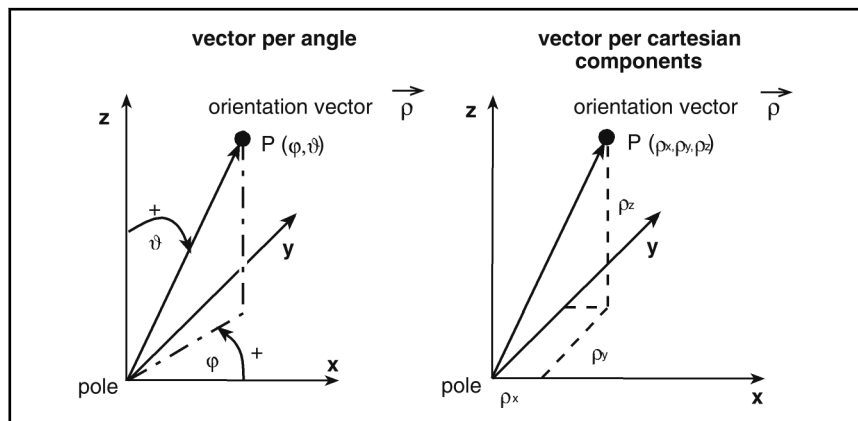


Fig. 6-20: Orientation coordinates of the orientation vector

- Function "O(..)" with the polar angles (φ , ϑ) or Cartesian components (ρ_x , ρ_y , ρ_z) of the orientation vector.

Fig. 6-21: Function O(..) with polar angles (φ , ϑ) or the Cartesian components (ρ_x , ρ_y , ρ_z) of the orientation vector

- Function "ROTAX(..)" for defining a rotary axis oriented anywhere in space so that the orientation vector can be rotated by a specific angle β using the function "O(..)".

The axis of rotation can be programmed both with polar angles (φ_u , ϑ_u) and also via Cartesian components (u_x , u_y , u_z).

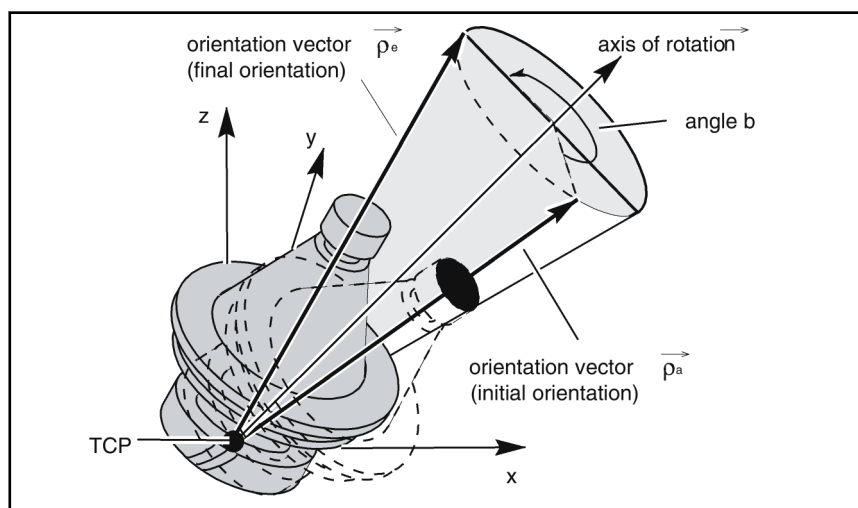


Fig. 6-22: Function ROTAX(..) for the definition of a rotary axis oriented anywhere in space

The following applies:

- Vector orientation can be programmed only together with an axis transformation whose 100th position is of the type = 2.
- The orientation motion is carried out as a rotary pointer motion of the orientation vector from the programmed initial orientation to the final orientation.
- This is suitable for rotation-symmetrical tools.

- Syntax:**
1. Activate the permitted axis transformation type (type with the 100th position = 2) with "Coord(...)" (see [chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245](#)).
 2. Use the syntax shown below.

{<No>} {<TCP motion>} {<phi><φ>} {<theta><θ>} or {<No>} {<TCP motion>} O (<φ>, <θ>) or {<No>} {<TCP motion>} O (<px>, <py>, <pz>) or {<No>} {<TCP motion>} ROTAX (<φu>, <θu>) O (<β>) or {<No>} {<TCP motion>} ROTAX (<ux>, <uy>, <uz>) O (<β>)	
with	
<No>	Block number Refer to chapter 3.8.2 "Block numbers" on page 27 .
<TCP mot>	Additional TCP motion For the syntax, refer to the documentation "MTX Functional Description" in the section on "Space coordinate programming".
<phi>	Angle name that is entered in MP 7080 00010[4]. Default: phi
<theta>	Angle name that is entered in MP 7080 00010[5]. Default: theta
<φ>, <θ>, <φu>, <θu>	Absolute angle values in degrees. Value range: $0^\circ \leq \varphi < 360^\circ$; $0^\circ \leq \vartheta \leq 180^\circ$. If a different value is programmed for ϑ , it is automatically converted into the specified interval.
O (<φ>, <θ>)	Orientation using the function O(..) and the polar angles <φ> and <θ> of the orientation vector.
O (<px>, <py>, <pz>)	Orientation using the function O(..) and the Cartesian components <px>, <py>, <pz> of the orientation vector in the absolute dimension. Automatic standardization to 1. This is why the following examples lead to identical orientation: O(1,2,4), O(2,4,8)
ROTAX (<φu>, <θu>)	Defining the rotary axis by means of the polar angles (φ_u, ϑ_u).

ROTAX (<ux>, <uy>, <uz>)	Defining the rotary axis via the Cartesian components <u _x >, <u _y >, <u _z > in absolute dimensions. Automatic standardization to 1.
O (<β>)	States the incremental angle <β> (in degree) by which the orientation vector is to be rotated about the rotation axis. Values greater than 360 degrees are allowed. The direction of rotation can be selected using the positive/negative sign.

Tab. 6-104: Vector orientation syntax

Special features and restrictions: The initial and final orientation of the orientation vector must not be parallel or anti-parallel, except in the case of programming ROTAX(..).

6.82.6 Tensor orientation

Description: Tool orientation is affected by orienting the entire tool coordinate system TCS, in relation to the current PCS.

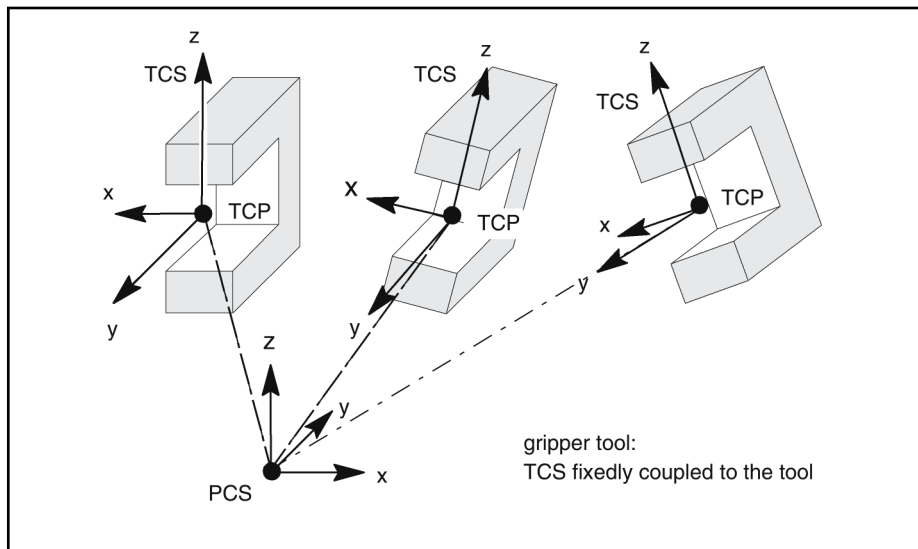


Fig. 6-23: Tensor Orientation

- Programming is also possible using
 - Euler angles phi ϕ , theta ϑ and psi ψ .
As a result of successive rotations with the Euler angles, the TCS is given its desired orientation.
 - O(<φ>,<θ>,<ψ>)
 - 3x3 orientation tensor (rotation matrix that realigns the entire TCS around the TCP). The components of its column vectors (O_x, O_y, O_z) precisely specify the TCS orientation and can be programmed as polar angles or Cartesian components.
 - Quaternion with syntax O(<w>,<x>,<y>,<z>).
 - Function "ROTAX(..)" for defining a rotary axis oriented anywhere in space so that the orientation tensor can be rotated by a specific angle β using the function "O(..)".
The rotary axis can be programmed with polar angles (φ_u, θ_u) as well as with Cartesian components (u_x, u_y, u_z).

The following applies:

- Tensor orientation can be programmed only together with an axis transformation whose 100th position is of the type = 3.

- The orientation motion is carried out as a rotary pointer motion of the orientation tensor from the programmed initial orientation to the final orientation.
- This is suitable only for tools that are **not** rotation-symmetric because the TCS is permanently coupled to the tool in this case.

- Syntax:**
1. Activate the permitted axis transformation type (type with the 100th position = 3) with "Coord(...)" (see [chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245](#)).
 2. Use the syntax shown below.

{<No>} {<TCP>} {<phi><φ>} {<theta><θ>} {<psi><ψ>} or	
{<No>} {<TCP>} {O(<φ>,<θ>,<ψ>)} or	
{<No>} {<TCP>} O _x (<φ _{xxy} (<φ _{yy}	
{<No>} {<TCP>} O _x (<φ _{xxz} (<φ _{zz}	
{<No>} {<TCP>} O _y (<φ _{yyz} (<φ _{zz}	
{<No>} {<TCP>} O _x (<O11>,<O21>,<O31>) O _y (<O12>,<O22>,<O32>) or	
{<No>} {<TCP>} O _x (<O11>,<O21>,<O31>) O _z (<O13>,<O23>,<O33>) or	
{<No>} {<TCP>} O _y (<O12>,<O22>,<O32>) O _z (<O13>,<O23>,<O33>) or	
{<No>} {<TCP>} O(<w>,<x>,<y>,<z>) or	
{<No>} {<TCP>} ROTAX (<φ _{uu}	
{<No>} {<TCP>} ROTAX (<u _{xyz}	
with	
<No>	Block number Refer to chapter 3.8.2 "Block numbers" on page 27 .
<TCP mot>	Additional TCP motion For the syntax, refer to the documentation " MTX Functional Description" in the section on "Space coordinate programming".
<phi>	Angle name entered in the parameter CHAN/Ch/Coord/Wcs/OriCoord[1] (MP 7080 00010 [4]). Default: phi
<theta>	Angle name entered in the parameter CHAN/Ch/Coord/Wcs/OriCoord [2] (MP 7080 00010 [5]). Default: theta
<psi>	Angle name entered in the parameter CHAN/Ch/Coord/Wcs/OriCoord[3] (MP 7080 00010 [6]). Default: psi
<φ>,<θ>,<ψ>	Absolute Euler angles in degrees Value range: 0° ≤ φ < 360°; 0° ≤ θ ≤ 180°; 0° ≤ ψ ≤ 360°. Values outside this range are automatically converted to the respective interval.

NC functions with high-level language syntax

$\langle\varphi_{..}\rangle, \langle\vartheta_{..}\rangle$	absolute angle values of a vector in degrees Value range: $0^\circ \leq \varphi_{..} < 360^\circ$; $0^\circ \leq \vartheta_{..} \leq 180^\circ$. If a $\vartheta_{..}$ value outside the range of values is programmed, it is automatically converted into the given interval.
$O_x(\langle O11\rangle, \langle O21\rangle, \langle O31\rangle)$ $O_y(\langle O12\rangle, \langle O22\rangle, \langle O32\rangle)$ $O_z(\langle O13\rangle, \langle O23\rangle, \langle O33\rangle)$ $O_x(\langle\varphi_x\rangle, \langle\vartheta_x\rangle)$ $O_y(\langle\varphi_y\rangle, \langle\vartheta_y\rangle)$ $O_z(\langle\varphi_z\rangle, \langle\vartheta_z\rangle)$	Orientation using the function $O_x(..)$, $O_y(..)$, $O_z(..)$ $O_x(..)$ defines the direction of the x-coordinate of the TCS in the reference coordinate system (first column of the orientation tensor). The same applies analogously to $O_y(..)$ and $O_z(..)$. The direction can be specified either by the corresponding polar angles $\langle\varphi_{..}\rangle$ and $\langle\vartheta_{..}\rangle$ or by specifying the Cartesian components of the column vectors of the orientation sensor. Only absolute dimensions are allowed. The component values of the column vectors (o..) are automatically standardized to 1.
$\langle w\rangle, \langle x\rangle, \langle y\rangle, \langle z\rangle$	Four components of the orientation quaternion
$ROTAX(\langle\varphi_u\rangle, \langle\vartheta_u\rangle)$	Definition of the rotary axis via polar angles (φ_u, ϑ_u) .
$ROTAX(\langle u_x\rangle, \langle u_y\rangle, \langle u_z\rangle)$	Definition of the rotary axis via the Cartesian components $\langle u_x\rangle, \langle u_y\rangle, \langle u_z\rangle$ in absolute dimensions. Automatic standardization to 1.
$O(\langle\beta\rangle)$	States the incremental angle $\langle\beta\rangle$ (in degree) by which the orientation tensor is to be rotated about the rotation axis. Values greater than 360 degrees are allowed. The direction of rotation can be selected using the positive/negative sign.

Tab. 6-105: Tensor orientation syntax

Example: Example settings:

Parameter TRA/AxTrafo/TrafoType (MP 1030 00110) for axis transformation 2: 3333301

Parameter CHAN/Ch/Coord/Wcs/LinCoord [1] (MP 7080 00010 [1]): x

Parameter CHAN/Ch/Coord/Wcs/LinCoord [2] (MP 7080 00010 [2]): y

Parameter CHAN/Ch/Coord/Wcs/LinCoord [3] (MP 7080 00010 [3]): z

Parameter CHAN/Ch/Coord/Wcs/OriCoord [1] (MP 7080 00010 [4]): phi

Parameter CHAN/Ch/Coord/Wcs/OriCoord [2] (MP 7080 00010 [5]): theta

Parameter CHAN/Ch/Coord/Wcs/OriCoord [3] (MP 7080 00010 [6]): psi

N100 G1 X10 Y20 Z30 $O_x(1,0,0)$ $O_y(0,0.707,-0.707)$

- or -

N100 G1 X10 Y20 Z30 $O_x(1,0,0)$ $O_z(0,0.707,0.707)$

- or -

N100 G1 X10 Y20 Z30 $O_y(0,0.707,-0.707)$ $O_z(0,0.707,0.707)$

- or -

N100 G1 X10 Y20 Z30 O(0.9239,0.3827,0,0)

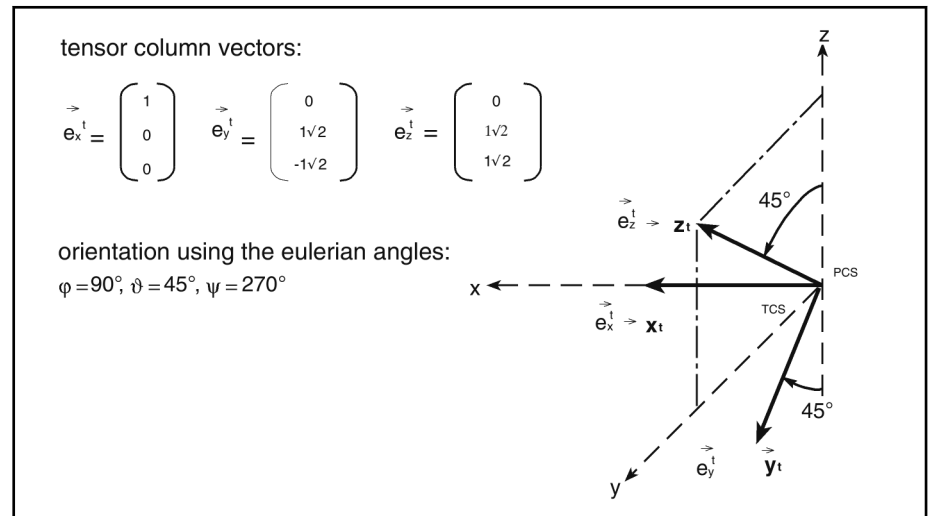


Fig. 6-24: Example of tensor orientation

Special features and restrictions:

- Except when ROTAX(..) is programmed, the orientation motion always occurs on the shortest path to the final orientation.
 For ROTAX(..), the direction of rotation depends on the positive/negative sign of $\langle \beta \rangle$.
- The following limitations are required to uniquely generate the orientation tensor:
 For $\vartheta = 0^\circ$:
 the sum of φ and ψ is required to specify an orientation.
 For $\vartheta = 180^\circ$:
 the difference of φ and ψ is required to specify an orientation.
- If programming of the tensor columns results in two column vectors being parallel or anti-parallel, it is not possible to calculate the orientation tensor.
 A runtime error is displayed.

6.83 Feed 100% "OvrDis, OVD", "OvrEna, OVE"

Description: The program control influences the effect of the channel feed potentiometer (qCh_Override_xy) for feed mode and rapid traverse.
 The functions are active in the operation modes "manual data input" and "processing".

Asynchronous axes are not affected by the function "OVD/OVE".

Syntax:

OvrDis	Feed potentiometer OFF The feed is set to 100% of the programmed value irrespective of the feed potentiometer.
Short form: OVD	

OvrEna	Feed potentiometer ON The feed depends on the position of the feed potentiometer.
Short form: OVE	

Tab. 6-106: Syntax of OvrDis (OVD) and OvrEna (OVE)

The following applies:

- Both functions are modal and deselect each other mutually.
- The switched-on state can be specified via machine parameters.
- Both functions may be written together with other path conditions in the same block.
- The effect of the overriding for asynchronous axes can be influenced in an axis-specific way using the interface signal "qAx_Override_100".

Example:

```

:                               Feed potentiometer set to 100%
N40 OVD G1 X5 Z-2 F200 S100 M4
                               Deactivates feed potentiometer

:                               Switches feed potentiometer to 40%, feed remains at 100%
N80 OVE X100 Activates feed potentiometer
Y50                               Feed changes to 40%
```

6.84 Changing the maximum path acceleration "PathAcc, PAC"

Description: In the part program, reduces the upper limits for

- path acceleration and
- Path deceleration.

Both the acceleration and deceleration values are preset using the machine parameters and can be switched together or separately in the part program.



The programmed or preset path acceleration can be limited by the maximum permitted axis accelerations of the axes participating in the path.

Syntax:

PathAcc (ACC<value>)	Simultaneous setting of the path acceleration and deceleration.
PathAcc ({UP<value1>, } {DOWN<value2>})	Separate setting of the path acceleration and deceleration.
PathAcc () or PathAcc (0)	Activates the acceleration values from MP 7030 00210 and 7030 00220 again.
Short form: PAC (. .)	
with	
<Value>	Acceleration value Depending on the active measuring unit (G71/G70), the control interprets the programmed value as "1000 Inch/s ² " or "m/s ² ".

<value1>	Value for path acceleration In all other respects identical to <value>
<value2>	Value for path deceleration. In all other respects identical to <value>

Tab. 6-107: Syntax of PathAcc (PAC)

Example:

N30 G71	Activates metric programming
N40 PAC (UP1.5)	Sets path acceleration to 1.5 m/s ²
:	
N140 PAC (ACC5)	Sets path acceleration and path deceleration to 5 m/s ²
:	
N200 PAC (UP3.5, DOWN2)	Sets path acceleration to 3.5 m/s ² and path deceleration to 2 m/s ²
:	
N240 PAC ()	Resets acceleration values to machine parameter setting.
:	

Special features and restrictions:

- The maximum programmable acceleration values are limited at all times by the machine parameter values.
- We recommend that the function be programmed in a separate block.

6.85 Activating placement tables "PmTsel, PMS"

Description: Activates placement tables (correction tables of the "inclined plane" placement). Such correction tables are stored as XML files in the file system of the control.

Syntax:

PmTsel ({<path>}<FileName>)	
Short form: PMS (. .)	
with	
<Path>	Optional path specification about the directory the <FileName> is stored in. If there is no specification, it is searched in the "/database" path. If file name cannot be found, the control uses the search path for subroutines to search for <FileName> in other directories.
<FileName>	File name of the placement table, incl. file extensions Tables with standard names (PM<Number>.pmt) can directly be activated using this number, e.g. PmTsel(3) activates the table PM3.pmt.

Tab. 6-108: Syntax of PmTsel (PMS)



Refer to the Control Operating Instructions for information on how to create and edit placement tables.

6.86 Programming polar coordinates

6.86.1 Defining the pole "PolarPol, POP"

- Description:**
- Defines the origin of the polar coordinate system, referring to the active plane of the active program coordinate system.
 - Sets polar angle 1 for polar coordinate programming (standard address: A; definable in MP 8005 00001) to 0 degrees.
 - Sets polar angle 2 for polar coordinate programming (standard address: B; definable in MP 8005 00002) to 90 degrees.

Syntax:

PolarPol (<PCvalue>,<SCvalue>)	
	Specifies the pole
PolarPol (ACT-POS)	Sets pole to current position
PolarPol () or PolarPol (0)	Sets pole to coordinate 0.0
Short form: POP (. .)	
with	
<PCvalue>	Pole value for the primary coordinate
<SCvalue>	Pole value for the secondary coordinate

Tab. 6-109: Syntax of PolarPol (POP)

Example:

```

:
N10 G18                Switches to the ZX plane with Z as the primary and X as
:                      the secondary coordinate.
N30 G0 X10 Z25         Positioning motion in rapid traverse to Cartesian coordi-
:                      nates X10 and Z25.
N40 POP (ACTPOS)       The current position values (PCvalue=25, SCvalue=10)
:                      are applied for the pole definition.

```

- Special features and restrictions:**
- If "PolarPol" is not used, the NC uses the origin of the active program coordinate system as the pole.
 - A programmed pole is active for the current plane only until the next time the plane is switched (see [chapter 5.2.11 "Plane switching G17, G18, G19" on page 126](#)).

6.86.2 Programming polar coordinates

- Description:** The MTX CNC system allows for programming of polar coordinates both in a plane (2D) and in space (3D).

The desired end point is entered using

- radius value (distance of target point P from pole),
- polar angle 1, and
- polar angle 2.

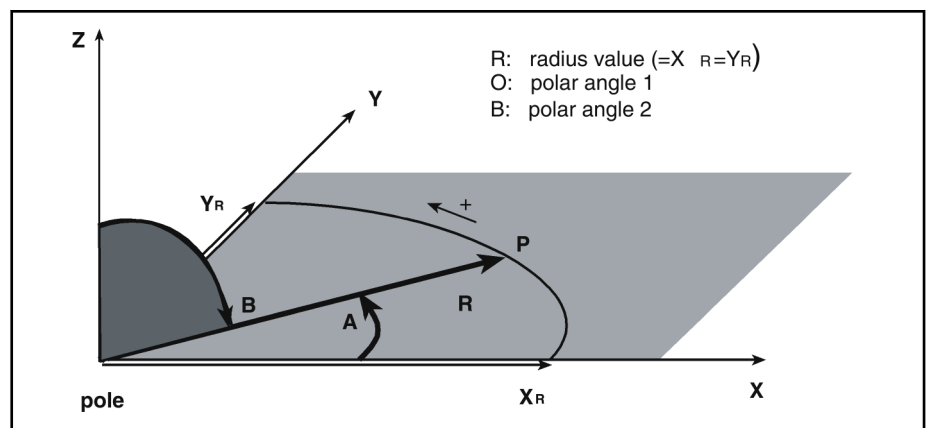


Fig. 6-25: Example: Polar coordinates in the plane with active G17

Radius value:

An axis address is always used to program the current working plane (e.g. "X" or "Y" for an active G17). This programmed coordinate is also called a radius coordinate.

In the case of absolute dimension programming, it indicates the distance of the target point from the pole; in the case of incremental programming, it indicates the distance of the target point from the starting point.

If both possible axis addresses are programmed in a single block, a runtime error occurs.

Polar angle 1:

- By default, this is programmed as address "A" and it is always located in the current working plane.
- Each programmed polar angle 1 refers to **the last-programmed** radius coordinate axis. A positive value refers to a rotation in a mathematically positive sense. The value remains active until a new value is programmed.

If incremental programming is active, the programmed angle is calculated incrementally.

- When polar coordinate programming is activated with POL, polar angle 1 is set to a value of 0 degrees.

Polar angle 2:

- By default, this is programmed as address "B", if absolute dimension programming is active and it describes the angle between the normal axis (of the current working plane) and the vector of the current pole to the target point. It remains in effect until a new value is programmed.

If incremental programming is active, the programmed angle is calculated incrementally.

- When polar coordinate programming is activated with POL, polar angle 2 is set to a value of 90 degrees.

In this case, the target point is located precisely in the active working plane. Now the polar coordinates have an effect only in the plane. All other coordinates outside of the working plane are interpreted according to Cartesian coordinates. This applies especially to programmed coordinate values of the current normal axis!

- "Real" spatial polar coordinates (spherical coordinates) occur if polar angle 2 becomes unequal to 90 degrees. Programming coordinate values of the current normal axis then leads to a runtime error. However, all co-

ordinates outside of the 3D space may be included simultaneously in programming. As in the 2D case, they are also interpreted according to Cartesian coordinates.

- If polar angle 2 is 0 degrees, the value of polar angle 1 is unimportant because the programmed point is in an exactly perpendicular position above the pole.

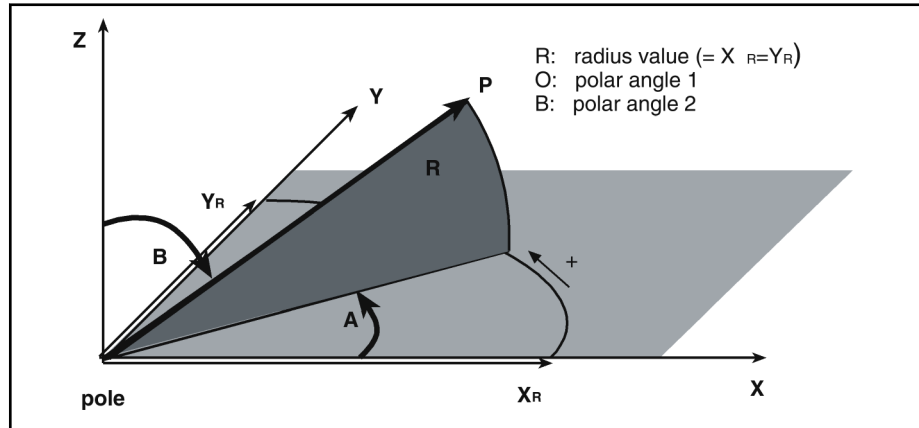


Fig. 6-26: Example: Polar coordinates in space with active G17



The default addresses for polar angles 1 and 2 can be changed using the machine parameters 8005 0001 and 8005 0002. Therefore, they may be different in your system.

Examples: Polar coordinate programming in absolute dimensions

N10 G18	Plane switching
:	Here: Switches to the ZX plane with Z as the primary and X as the secondary coordinate.
N40 POP (25,10)	Pole definition in the current plane (here : with Cartesian coordinates Z=25 and X=10).
:	
N50 G1 (POL)	Path motion in polar coordinates according to P2 (see Fig.).
Z20 A70	P2 is defined by the absolute distance from the pole (here: 20 mm) and polar angle 1 (here: A=70 degrees), based on the programmed radius axis (here: Z-axis).
:	

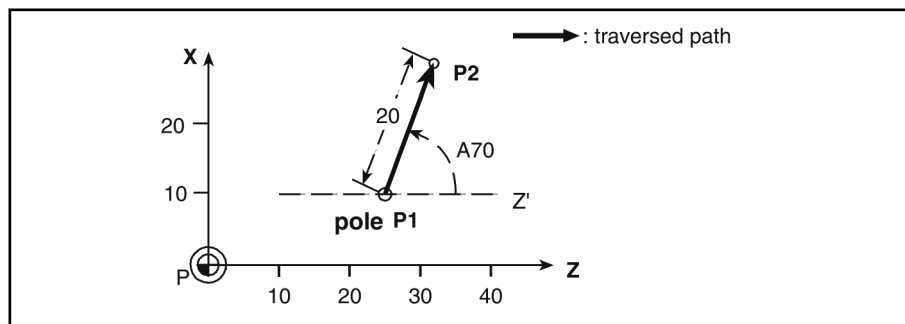


Fig. 6-27: Path motion to P2 in the polar coordinates

```

N60 G1 (POL)      Path motion in polar coordinates according to P3 (see Fig.).
X20 A90           P3 is defined by the absolute distance from the pole (here: 20
:                mm) and polar angle 1 (here: A=90 degrees), based on the
:                programmed radius axis (here: X-axis).
:
:                Alternatively, the following programming is also possible:
:
:                N60 G1 (POL) Z20 A180
:                or
:                N60 Z20 A180

```

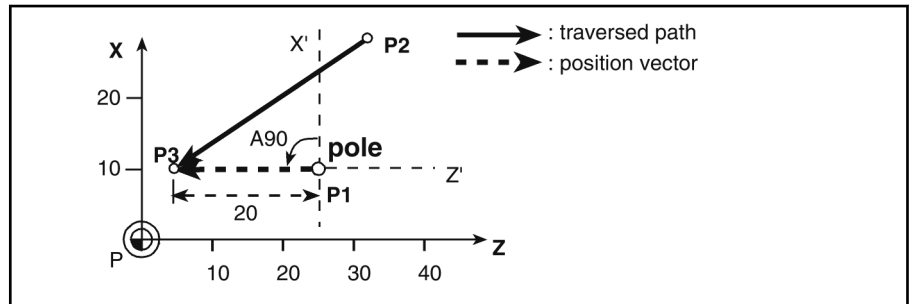


Fig. 6-28: Path motion in the polar coordinates according to P3

Polar coordinate programming in incremental dimensions

```

N10 G17 G90 G0 X0 Y0   Starting position: P0
N20 G1 (POL) X200 A0    P1
N30 G91 Y200 A30       P2
N40 A60                P3
N50 A60                P4
N60 A60                P5
N70 A60                P6
N80 A60                P7
N90 X0                 P0
N100 M30

```

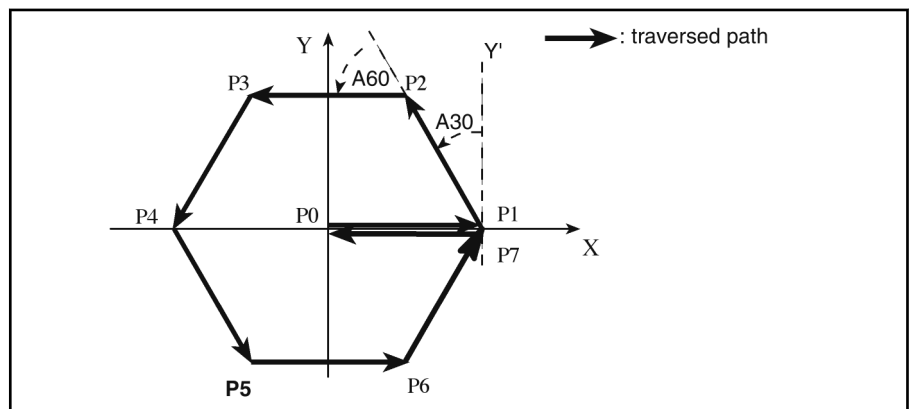


Fig. 6-29: Polar coordinate programming in incremental dimensions

6.87 Defining the reflection/rotation point "PoleSet, PLS"

Description: This function specifies the absolute position of the pole for mirroring, scaling and rotating input tools. The defined pole is then the reference point of these input tools, thus providing the reflection/rotation point.



The function is not required if mirroring or rotation is to take place with reference to the program zero point.

The following applies:

- PoleSet is modally effective. The position of the pole refers to the current program coordinate system and thus remains in effect until it is reset to the program coordinate origin or is redefined.
- The function does not cause any axis traveling.
- The function may be written in the same block as other path conditions and input tools.
- The programmed pole refers to the functions Mirror, Scale and Rotate.

Syntax:

PoleSet (<Coordinates>)	Set pole to the given coordinates. The given coordinates are to be separated by commas (e.g.: PoleSet (X5, Y2)).
PoleSet (0) or PoleSet ()	Resets the pole to the origin of the program coordinate system.
Short form: PLS (. .)	

Tab. 6-110: Syntax of PoleSet (PLS)

Example:

```

:
N30 PLS (X5, Y2)   Sets the mirror/rotation point to a position of X=5 and Y=2.
:
N130 PLS ()        Resets the mirror/rotation point to the program coordinate origin.
:

```

Special features and restrictions: The position of the pole must be entered absolutely, i.e. based on the current program zero point.

6.88 Position-dependent high-speed output "PosDepHSOut, PHS"

Description: Affects the status of a high-speed (HS) output using a programmed traversing motion.

As soon as the **NC command position**, referring to the start or end of the current block, reaches a programmable value, the HS output is set or reset, depending on the programming.

It can be programmed for how long the output is to remain set after the specified event has occurred.



Parameterization of available HS outputs per MP 4075 00102.

Syntax:

PosDepHSOut (<Mode>{, {<distance>}{, <duration>}})
Short form: PHS (. .)

with	
<Mode>	Desired method of operation of the function Input values: 0, 1 or -1 0: Modally saves the data programmed as <distance> and/or <duration>. The current status of the HS output is not changed. 1: Modally saves the data programmed as <distance> and/or <duration> and sets the HS output. The relevant modal data values apply to parameters which have not been programmed. -1: Resets the HS output. If a <distance> is programmed, it only has a local effect restricted to this block. If a <duration> is programmed, it is ignored.
<Distance>	Distance to the start/end of the block after which the signal is to be set (in mm/inches). Default value: 0 0: at end of block greater than 0: distance to the start of the block smaller than 0: distance to the end of the block
<Duration>	Maximum time that the HS output is switched on (in ms) Value range: 0.5 ... 10000.0 Programmed values are internally rounded up to the next larger integer multiple of the NC cycle time.

Tab. 6-111: Syntax PosDepHSOut (PHS)

Example:

```

:
N05 G71
N10 PHS(0,-1.2,40)      Configures the function
:                        In a subsequent call using PHS(1) syntax, the HS
:                        output is set 1.2 mm before reaching the end point
:                        for approx. 40 ms.
N210 G1 G91 F1000
N220 X10 Y23 PHS(1)     Sets HS output according to configuration in N10
:
N330 X10 PHS(1,0.1,900) Sets HS output for max. 900 ms 0.1 mm after the
:                        start position
N350 X20 PHS(-1,-0.3)   Resets HS output 0.3 mm before the end position
:

```

Special features and restrictions:

- One HS output is supported per channel.
- The function has an effect on a traversing motion only if it is programmed in the same block.
- A set HS output is reset only after the ON time has elapsed. A control reset does not affect this time.

- <Distance> always refers to the current command position from NC block perspective.
System-related delays (e.g. caused by drive functions or axis coasting) are not taken into account here.
- If a new PHS job arrives during an elapsing ON time, the NC deletes the still-active job and executes the new job.



If no signal change occurs at the HS output in the case of two subsequent PHS jobs, the external hardware can generally not detect the new job.

6.89 Positioning type for endless axes "PosMode, PMD", Local positioning type for endless axes "DC", "ACP", "ACN"

Description: Specifies in which direction the axes of the "endless" type are to rotate during a positioning procedure.

The following applies:

- Contrary to "PosMode", DC(...), ACP(...) and ACN(...) have a block-to-block effect, overriding the active positioning type of the programmed axis in the current block.
- Position information programmed with DC, ACP and ACN functions are always interpreted as absolute position values.

Syntax:

PosMode (<Addr><Type>)	Activates the positioning type for the axis with the address <Addr> according to the <Type>. Several axes may be programmed in the parentheses. Activates the positioning type for non-programmed rotary/endless axes in the current channel according to MP 1003 00005.
PosMode () or PosMode (0)	Activates the positioning type for all axes in the current channel according to MP 1003 00005.
Short form: PMD (. .)	
<Addr>=DC (<value>)	The axis <Addr> approaches the absolute position <value> along the shortest path.
<Addr>=ACP (<value>)	The axis <Addr> approaches the absolute position <value> in the mathematically positive direction (see the following note).
<Addr>=ACN (<value>)	The axis <Addr> approaches the absolute position <value> in the mathematically negative direction.
with	
<Addr>	Axis address Axes of "endless" type are permitted.

<Value>	Path information for <Addr>
<Art>	Positioning type 0: No positioning logic The direction of travel always results solely from the difference between the old and the new position. 1: Shortest possible path The maximal travel path of the axis is no more than half of the relevant modulo value. 2: According to programmed positive/negative sign: "+": Clockwise rotation; "-": counter-clockwise rotation. 3: The active software limit switches (1020 00001/3 and 1020 00002/4) define the permitted travelling range and the possible direction. The permissible traversing range extends from the negative to the positive limit switch. The limit switch ranges turn active once the axis has been referenced. Coordinate specifications are valid both in the negative and the positive modulo ranges (e.g. 20° corresponds to -340°) The coordinate signs and the local coordinate attribute "DC" are ignored. The attributes "ACP", "ACN" and "IC", on the other hand, are observed. In case of violation of the working range, they cause an error message.

Tab. 6-112: Syntax of PosMode (PMD), DC, ACP and ACN



Mathematically positive direction:

Counterclockwise sense of rotation seen from a coordinate axis in the direction of the coordinate origin.

Example:

N40	Irrespective of the "PosMode", the physical axis with the designation "B" travels in the mathematically positive direction to the position of 258 degrees. The positive/negative sign is ignored.
B=ACP (-258)	
:	
N80 PMD (A1, C2)	Positioning type of
:	Axis A: Shortest possible path
:	Axis B: According to MP 1003 00005
:	Axis C: Sign

Special features and restrictions:

- Except for the positioning type "restricted traversing range", the software limit switches for endless axes have to be always hidden.
- The positioning type for an endless axis can only be switched if switching has been enabled in the MP 1003 00050.
- Several DC, ACP and ACN functions (for different axes) can be programmed in the same block.
- DC, ACP and/or ACN functions do only have an effect on axes of the following types: "rotary" or "endless". If they are programmed in conjunction with other axis types, the control ignores them.
- Negative signs in the position information on DC, ACP and ACN functions are ignored.
- The following applies to the positioning type "limited path":

Negative end angles are always displayed in the positive modulo range (0...360°).

The coordinate attribute DC "shortest path" is ignored. Instead, the control selects the "permissible path" with automatic adjustment of direction.

Direction positioning by means of signs, coordinate attributes or incremental (+/-, ACP/ACN and G91/IC) are not adjusted; these are permissible only within the limit switch range.

Motions are only permitted within a full rotation.

6.90 Precision programming "PrecProg, PRP"

Description: Reduces the feed at contour transitions and on circular path segments so that a preset precision can be maintained.

The required accuracy can also be influenced by specifying the

- maximum permitted **contour or radius error ϵ** on the contour transition or for arcs or the
- Maximum permissible **path lag δ** (distance from corners) that may to be exceeded when traveling through a contour transition.

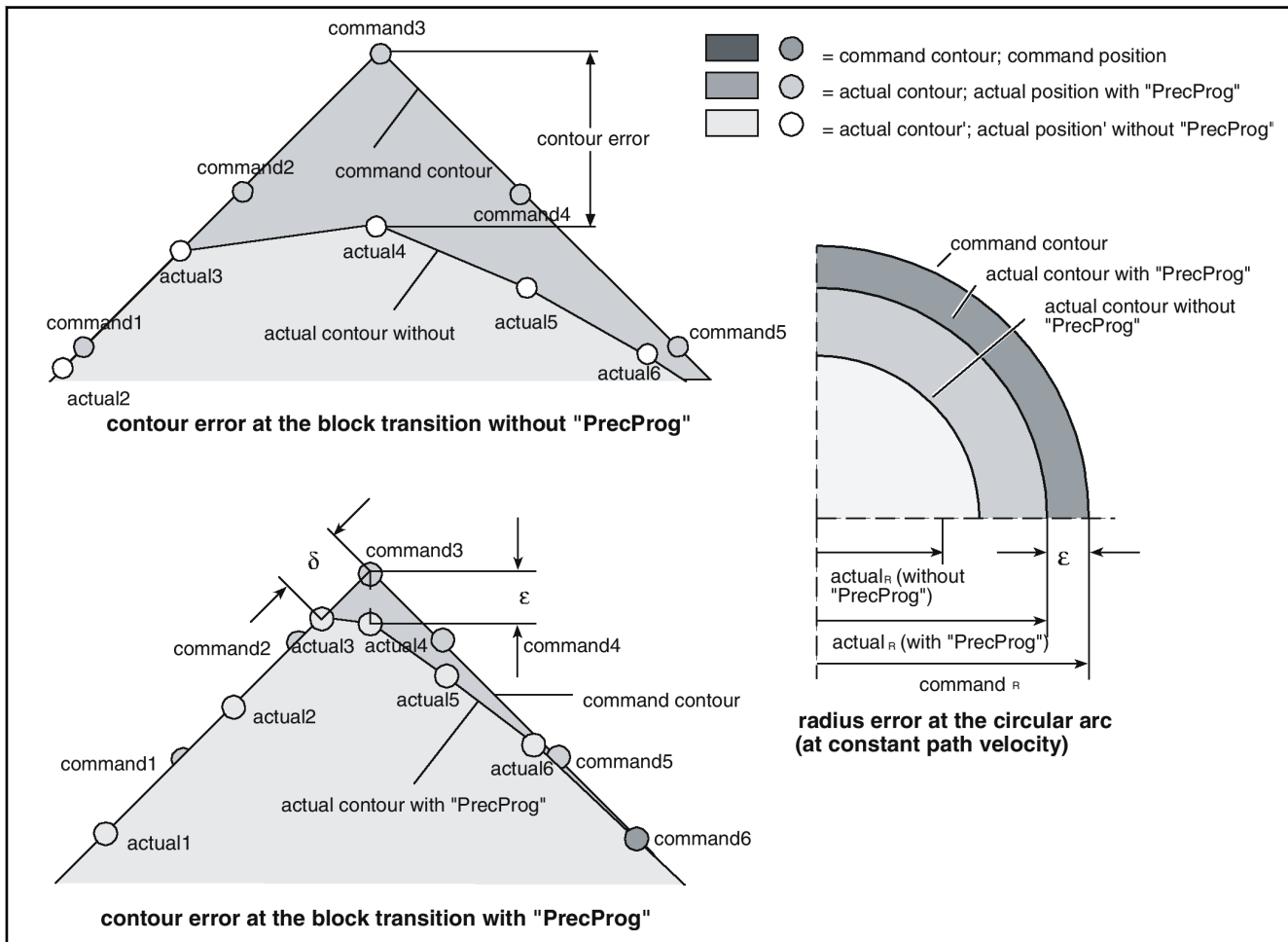


Fig. 6-30: Accuracy programming



In contrast to the "precision" function (G61/G62; see [chapter 5.2.22 "Exact stop ON/OFF G61, G62" on page 145](#)), the velocity does not have to be decelerated to 0 at a block switching. Refer to ["Special features and restrictions:" on page 327](#).

Syntax:

PrecProg or PrecProg(1)	Accuracy programming ON Max. contour/radius error ϵ allowed according to MP 8003 00001.
PrecProg(EPS<e>)	Accuracy programming ON Maximum permitted contour/radius error: < ϵ >
PrecProg(DIST<d>)	Accuracy programming ON Maximum permitted path lagging : < δ > Consider ϵ value of MP 8003 00001 for circular path segments.
PrecProg() or PrecProg(0)	Accuracy programming OFF
Short form: PRP (. .)	
with	
< ϵ >	Maximum permitted contour/radius error in mm or inches (depending on G70/G71).
< δ >	Maximum permitted path lagging in mm or inches (depending on G70/G71).

Tab. 6-113: Syntax of PrecProg (PRP)

Special features and restrictions:

- All affected axes must be set with the identical dynamics.
- If G8 and G62 are not active, decelerating to $v=0$ occurs at every block switching.
- Precision programming provides reasonable results only if the influence of the acceleration on lagging can be ignored.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.91 Program processing depending on the block preparation "PREP"

Description

If there is a critical passage in the part program with (temporarily) very short blocks, it might be necessary to ensure that the buffer of prepared block is nearly completely filled before this passage is processed. Thus, there is not an immediate abort (processing cannot get a new block if none has been completely prepared) if the block preparation takes longer than the block processing.

It might also be not suitable that the MTX starts the program execution as soon as the first block of the part program has been completely prepared. In this case, the preparation buffer is still empty. If the first block is extremely short, there might be an abort.

NC functions with high-level language syntax

Finally, the MTX provides the option to transfer a program to the control and execute it using a buffered NC block specification. The communication has a low priority and might be delayed. This can also cause an abort.

In any of the three application scenarios, execution can be purposefully stopped prior to the critical section, until a sufficient number of blocks has been completely prepared by block preparation. Therefore, the NC function "PREP" is used. The number of blocks to be prepared is programmed with this function.

The function delays the execution of the blocks for a period of 2 seconds max. A runtime error is output if the programmed number of blocks could not have been prepared within this period. This error can be suppressed by programming the keyword "SILENT".



Completely prepared blocks are saved in the buffer of the block look-ahead.

This buffer size is defined with NCO / LookAh / Ch[k] / NofBlkPrep and has to have at least the size for the programmed number of blocks.

The PREP function may **not be programmed in asynchronous subroutines**.

To mark the end of a passage that is to be prepared, a `PREP()` can be programmed. In this case, at the start of a the program section, a `PREP(<N>)` with a capitalized <N> of blocks to be prepared and at the end a `PREP()` can be programmed. The first block after the first `PREP` is only executed if all block until the second `PREP` have been completely prepared.

Syntax:

<code>PREP (<BlkCnt> { , SILENT })</code>	
with	
<code><BlkCnt></code>	Number of blocks to be prepared
<code>SILENT</code>	If the keyword is programmed, no error message is generated in case of a timeout.
<code>PREP ()</code>	
	End of a program section that is to be completely prepared

Tab. 6-114: Syntax of the NC function PREP

Example: Program:

```

N10 PREP(30)
N20 G1 X10 Y10 F12000 ; Block 1
; Pure comment -> it is not counted.
N30 X10.3 Y10.2 ; Block 2
...
N300 X120.4 Y13.1 ; Block 29
N310 X121.2 Y13.2 ; Block 30, when this block is completely prepared, N20 is executed

```

If the buffer of the block look-ahead is too small in the example or if the program is specified via buffered NC block specification and causes a delay there, a runtime error is generated since the requested number of blocks could not be prepared.

To suppress a runtime error, the block `N10 PREP(30, SILENT)` could be programmed instead of line N10.

Special features and restrictions:

- The block preparation buffer must at least be as big as the programmed number of blocks to be prepared.

- The function only waits in the modes Manual Input, Buffered NC Block Specification and Automatic. It does not wait in any of the other operating modes (e.g., single block).
- If WAIT or the program end is found while waiting, the blocks are completely prepared till there. Then, the execution continues.
- It is never waited at reset.
- The PREP command may not be used in an asynchronous subroutine. If it was programmed in an asynchronous subroutine, a runtime error is generated.
- A communication abort (in case of a buffered NC block specification) generates an error message if the SILENT switch was not programmed.
- No channel waiting states are activated.
- The blocks to be prepared only include blocks that go to the interpolator. That means that CPL blocks and empty blocks are not counted for example. For blocks split due to their geometry, each part is counted separately.
- If the buffered NC block specification is used, the first block of a buffered NC block message may either not contain an auto start flag or a PREP.

6.92 Stroke release time

6.92.1 Stroke release time (interpolation end point) "PtBlkEnd, PTE"

Description:



Effective only in conjunction with the functions "punch" (see [chapter 6.94 "Punching Punch, PUN" on page 335](#)) and "nibble" (see [chapter 6.80 "Nibbling Nibble, NIB" on page 303](#)).

The function

- defines time reference event "NC interpolator is reaching the end point of the traversing motion" for the given axes and
- specifies the period between the time reference event and the stroke release.

In this manner, the stroke release time (based on the time reference) can be offset as far as desired.

The following is possible:

- Early stroke release
(e.g. to compensate a constant, application-dependent waiting period that is caused by signal processing) and
- Subsequent stroke release
(e.g. to increase the positioning precision for axes with lower dynamics, or for punching with hold-down devices).



Additional functions that influence the stroke release time:

- PtBlkEnd (refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE" on page 329](#))
- PtInpos (see [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332](#))

Syntax:

PtBlkEnd (<Axis><Time>{, <Axis><Time>}...)	
Short form: PTE (. .)	
with	
<Axis>	Logical axis name
<Time>	Desired period of time (in ms) between the time reference event and stroke release. 0: stroke release at event reference moment Negative value: early stroke release Positive value: delayed stroke release

Tab. 6-115: Syntax of PtBlkEnd (PTE)

- Examples:**
- Axis configuration: X, Y, C
 - Sercos cycle time: 3 ms

Setting the stroke release times:

```

:
N10 PTE (X-10)  Time reference event for axis X:
:               "NC interpolator is reaching the end point of the traversing motion"
:
               Early stroke release by 10 ms

N20             Time reference event for the axes Y and C:
PTI (Y10,C2)    "Inpos window reached"
:
:               (for the function see chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332).
:
:               Y-axis: delayed stroke release by 10 ms
:               C-axis: delayed stroke release by 2 ms

```

Effect during traversing motions:

```

:
N100 G1 Y20     Y determines the stroke release behavior because a longer waiting time has been programmed with an identical time reference (see N20).
C10
:
:               Since the waiting times are rounded up to the Sercos cycle time slots, this results in an actual delayed stroke release of 12 ms.

N110 X20 C20    C determines the stroke release behavior because axes whose time reference = "Inpos window" are "weaker" than axes whose time reference = "Interpolation end point".
:
:               Since the waiting times are rounded up to the Sercos cycle time slots, this results in an actual delayed stroke release of 3 ms.

```

N120 X30 : : :	X determines the stroke release behavior because only X is moved. Actual early stroke release: -9 ms
N130 PTD : : :	Sets a stroke release of all axes in the system according to MP 8001 00020 and MP 8001 00021 (for the function see chapter 6.92.2 "Setting the stroke release time to default PtDefault, PTD" on page 331).

Special features and restrictions:

- Functions PTD, PTE and PTI act modally and cancel each other mutually.
- Punching or nibbling have to be applied to MP 8001 00010.
- All programmed time values are rounded up to match the Sercos cycle time slots.
- In the case of traversing motions in which several axes with differently defined time references and waiting periods are involved, the "weakest" axis determines the actual stroke release behavior.

The following applies:

- Axes whose time reference = "Inpos window" (see [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332](#)) are "weaker" than axes whose time reference = "interpolation end point".
- If the time reference is identical, the "weakest" axis is the one with the longest waiting time.
- The function PtDefault (see [chapter 6.92.2 "Setting the stroke release time to default PtDefault, PTD" on page 331](#)) can be used to reset the stroke release of all axes in the system to the default values.

6.92.2 Setting the stroke release time to default "PtDefault, PTD"

Description:

Effective only in conjunction with the functions "punch" (see [chapter 6.94 "Punching Punch, PUN" on page 335](#)) and "nibble" (see [chapter 6.80 "Nibbling Nibble, NIB" on page 303](#)).

Sets the time of the stroke release of all axes in the system according to the values defined in the MP 8001 00020 and MP 8001 00021.

Syntax:

PtDefault
Short form: PTD

Tab. 6-116: Syntax PtDefault, PTD

Example:

Refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE" on page 329](#).



Additional functions that influence the stroke release time:

- PtBlkEnd
(refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE" on page 329](#))
- PtInpos
(refer to [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332](#)) is opened.

- Special features and restrictions:**
- Functions "PTD", "PTE" and "PTI" act modally and cancel each other mutually.
 - Punching or nibbling have to be applied to MP 8001 00010.

6.92.3 Stroke release time (Inpos window) "PtInpos, PTI"

Description:



Effective only in conjunction with the functions "punch" (see [chapter 6.94 "Punching Punch, PUN" on page 335](#)) and "nibble" (see [chapter 6.80 "Nibbling Nibble, NIB" on page 303](#)).

The function

- defines the time reference event "Inpos window reached" for the given axes and
- specifies the period between the time reference event and the stroke release.

In this manner, the stroke release moment (based on the time reference) can be offset as far as desired (e.g. to increase the positioning precision for axes with lower dynamics, or for punching with hold-down devices).



Additional functions that influence the stroke release time:

- PtBlkEnd
(refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE" on page 329](#))
- PtInpos
(refer to [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332](#)) is opened.

Syntax:

PtInpos (<Axis><Time>{, <Axis><Time>}...)	
Short form: PTI (. .)	
with	
<Axis>	Logical axis name
<Time>	Desired period of time (in ms) between the time reference event and stroke release.

Tab. 6-117: Syntax of PtInpos (PTI)

Example: Refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE" on page 329](#).

- Special features and restrictions:**
- Functions "PTD", "PTE" and "PTI" act modally and cancel each other mutually.
 - Punching or nibbling have to be applied to MP 8001 00010.
 - All programmed time values are rounded up to match the Sercos cycle time slots.
 - In the case of traversing motions in which several axes with differently defined time references and waiting periods are involved, the "weakest" axis determines the actual stroke release behavior.

The following applies:

- Axes whose time reference = "Inpos window" (see [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI" on page 332](#)) are "weaker" than axes whose time reference = "interpolation end point".

- If the time reference is identical, the "weakest" axis is the one with the longest waiting time.
- The function "PtDefault" (see [chapter 6.92.2 "Setting the stroke release time to default PtDefault, PTD" on page 331](#)) can be used to reset the stroke release of all axes in the system to the default values.

6.93 PTP traversing motion "PtpMove, PTP"

Description: When axis transformation is active, a **PointToPoint** traversing motion (PTP) is carried out, i.e. the motion is performed in MCS or ACS without deactivation of the axis transformation. The programmed end position can either be specified in WCS, MCS, or ACS. The respective conversion is performed within the control.



For comprehensive information on the use and parameterization of PTP, refer to the documentation "MTX Functional Description".

Syntax:

PTP (<EndposSys>, <MveSys>, TB<Mode>) and programming of the end position in EndposSys (alternatively: PtpMove (,))	
with	
<EndposSys>	Coordinate system in which the end position has been programmed. Valid: WCS: Programmed end position in the workpiece coordinates MCS: Programmed end position in the machine coordinates AC: Programmed end position in the axis coordinates

NC functions with high-level language syntax

<MveSys>	Coordinate system in which the motion is carried out. Valid: MCS: Motion in machine coordinates ACS: Motion in axis coordinates
TB<Mode>	TrafoBranch To control the transformation branch change in case of dual branch axis transformations. Only possible for the PTP(WCS,MCS) variant. "Mode" can have the following values: 0(Default): The position closer to the starting point is approached out of the two possible MCS positions MCS(+1) and MCS(-1). If required, the transformation branch is changed from +1 to -1 or vice versa. 10: Retain the transformation branch during the motion 11: Change the transformation branch during the motion +1: Move to transformation branch +1, i.e. move to position MCS(+1) -1 : Move to transformation branch -1, i.e. move to position MCS(-1) Further properties: TB is ignored in case of one branch transformations Only the PTP(WCS,MCS) variant supports TB programming. A syntax error is generated for other variants.

Tab. 6-118: Syntax

In each case, "MCS" is used as a default value.

Example:

```

PTP (WCS,MCS) x50 y55 ; WCS coord. x=50, y=55 is approached
                        ; motion in MCS

PTP (WCS,ACS) x=IC(50) ; WCS coord. x incremental traveled by 50
                        ; motion in ACS

PTP (MCS, MCS) X100 Y200 ; MCS coord. X=100, Y=200 is approached
                        ; motion in MCS

PTP (MCS, ACS) X101 Y201 ; MCS coord. X=101, Y=201 is approached
                        ; motion in ACS

PTP (ACS, MCS) XA50 C45 ; ACS position. XA=50, C=45 is approached
                        ; motion in MCS

PTP (ACS, MCS) XA51 C46 ; ACS position. XA=51, C=46 is approached
                        ; motion in ACS

PTP () corresponds to PTP (MCS, MCS)
PTP (,) corresponds to PTP (MCS, MCS)
PTP (WCS, ) corresponds to PTP (WCS, MCS)
PTP (,ACS) corresponds to PTP (MCS, ACS)

```


Examples for TB programming with 5-axis transformation 3232201:

B10 C0	Approaching the initial orientation Phi=0, Theta=30
COORD(1)	The axes move to B30, C0 (transformation branch is +1)
G1 phi0 theta30	
PTP (WCS,,TB0) phi180 theta30	MCS(-1) is approached out of the two possible MCS positions MCS(+1) = (B30,C180) and MCS(-1) = (B-30,C0) (shortest distance with transformation branch change)
or	
PTP (WCS) phi180 the- ta30	
PTP (WCS,,TB10) phi0 theta30	Transformation branch is not changed: MCS(-1) = (B-30,C180) is approached
PTP (WCS,MCS,TB11)	Transformation branch is changed: MCS(+1) = (B30,C0) is approached. WCS position is still Phi=0, Theta=30
PTP (WCS,,TB+1)	Approach transformation branch +1. No motion as transformation branch is already +1
PTP (WCS,MCS,TB-1)	Approach transformation branch -1. MCS(-1) = (B-30,C180) is approached. WCS position is still Phi=0, Theta=30

Special features and restrictions:

- This is a local, non-modal function.
- A PTP traversing motion is always interpreted as absolute programming, i.e. G91 is ignored. Incremental programming can be performed via the programming attribute "IC" (e.g. x=IC(50)).
- During a PTP traversing motion, the 3D radius correction has to be deactivated. Otherwise, a runtime error is generated.
- Only PTP command can be programmed per line.

6.94 Punching "Punch, PUN"

Description: Enabling/disabling the "punching" function on or off.

When punching is enabled, a stroke is triggered at the end in case of

- a stroke is triggered at the end of each programmed traversing motion (single-stroke operation) or
- at the end of every path segment generated by the "NUM" (see [chapter 6.81 "Dividing the traversing block: Number of path segments NUM" on page 305](#)) or "LEN" function (see [chapter 6.67 "Dividing the traversing block: Length of path segments LEN" on page 286](#))

The subsequent traversing motion always starts before the stroke is finished.



Functions to influence the stroke release time:

- **PtDefault**
(refer to [chapter 6.92.2 "Setting the stroke release time to default PtDefault, PTD"](#) on page 331)
- **PtBlkEnd**
(refer to [chapter 6.92.1 "Stroke release time \(interpolation end point\) PtBlkEnd, PTE"](#) on page 329)
- **PtInpos**
(refer to [chapter 6.92.3 "Stroke release time \(Inpos window\) PtInpos, PTI"](#) on page 332) is opened.

Syntax:

Punch (1) or Punch	Punching ON
Punch (0) or Punch ()	Punching OFF
Short form: PUN (. .)	

Tab. 6-119: Syntax of Punch (PUN)

Example:

- G90 is active (absolute dimension programming)
- Active plane: X/Y
- Current position: X=0, Y=0, C=0

:	
N10 C10	Punching ON
Punch (1) G9 (ASHAPE)	Enable interpolator with nibbling logic
	C-axis rotates to 10 degrees
	No stroke because the X and Y axes are not programmed.
N20 C60	C axis rotates to 60 degrees
	No stroke because the X and Y axes are not programmed.
N30 X0	No traversing motion because the x-axis is already located at 0 position.
	A stroke is carried out because the x-axis is located in the active plane.
N40 LEN=12	Divide the following traversing blocks into equal path segments with a maximum length of 12 mm each.
N50 X110	The traversing block is divided into 10 identical 11 mm path segments.
	Strokes are executed in positions X11, X22, X33 ... X99, X110
N60 Y30 NUM=3	Overrides the modally effective LEN (N40) for the current block.
	Divide the traversing blocks into 3 identical path segments.
	Strokes at Y10, Y20, Y30

N70 Y90	LEN from N40 is in effect again
	Strokes at Y42, Y54, Y66, Y78, Y90.
N80 X50 Y50	Punching OFF
Punch(0) G8	Path slope ON
:	traversing motion to X=50, Y=50

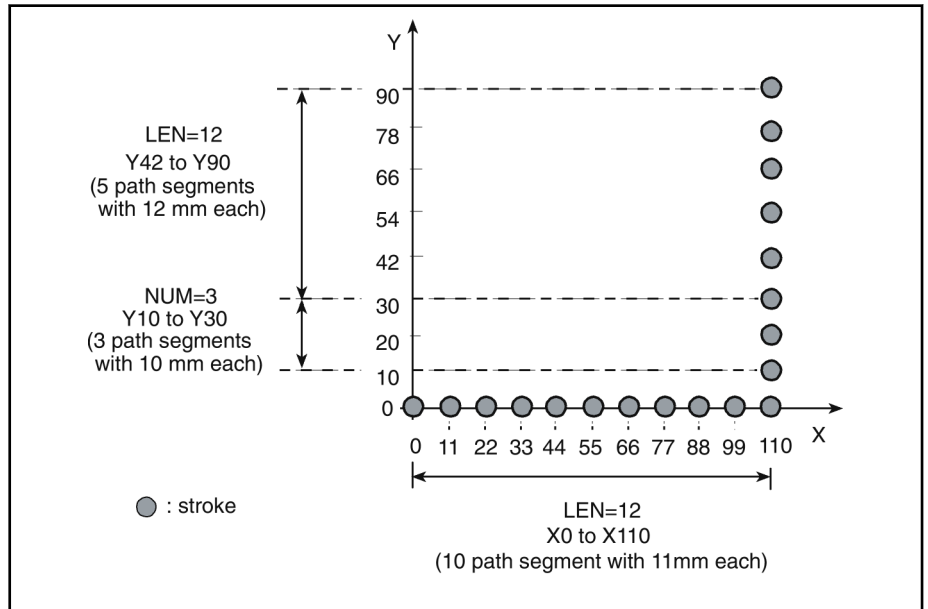


Fig. 6-31: Example Punch (PUN)



To use the nibbling function, an interpolator supporting the nibbling functionality has to be enabled:

G9(X... , Y... , ...)

G9(ASHAPE)

G9(SHAPE..., PUN)

Special features and restrictions:

- Punching must be released by MP 8001 00010.
- The function already has a modal effect on traversing motions that are programmed in the same block.
- The control can use the function "NUM" or "LEN" to automatically generate path segments from the programmed distance traveled; a stroke is released at the end of each of these path segments.
- If nibbling is enabled (see [chapter 6.80 "Nibbling Nibble, NIB" on page 303](#)), the function deselects "Punching".
- Blocks that do not contain axis coordinates from the active plane also do not release a stroke.
- If the PLC suppresses the stroke release, processing remains at the stroke release position until the PLC releases the stroke release.

6.95 Maximum radial acceleration "RadialAcc, RAC"

Description: Reduces the path feed for splines and arcs to such an extent that the specified radial acceleration in the workpiece coordinate system (WCS) is not exceeded.

The radial acceleration limit is disabled after control startup.

NC functions with high-level language syntax

Syntax:

RadialAcc (<value>)	Setting maximum radial acceleration
RadialAcc(0)	Canceling radial acceleration limit
Short form: RAC (. .)	
with	
<Value>	Acceleration value Depending on the active measuring unit (G71/G70), the control interprets the programmed value as "1000 Inch/s ² " or "m/s ² ".

Tab. 6-120: Syntax RadialAcc (RAC)

Example:

```

N20 RAC(0.1) ;Limit radial acceleration up to 0.1 m/s2
N30 G1 G8 G90 G94 F5000 ;Path feed 5000 mm/min max
:
N50 G3 Y20 R10 ;Vmax= 1897 mm/min (Vmax=sqrt(RAC*R10))
N60 G2 Y30 R5 ;Vmax= 1341 mm/min (Vmax=sqrt(RAC*R5))
:
N100 RAC(0) ;Disable radial acceleration limit

```

Special features and restrictions:

- The maximum curvature for splines is randomly checked at the beginning, in the middle and at the end of a block
- The reduction is always carried out for the whole spline path or circular path and taking all coordinates related to the feed into consideration.
- The feed can only be reduced. Other feed limits by means machine parameters or part program work normally.

6.96 Recording of the reverse vector as system data "RecordRevVec, RRV, RRV"

Description: Enables/disables the function "Recording the retract vector as system data".

One prerequisite for the operability of this NC function is that there is a system data with the name <name>RevVec</name> and that it has been defined with <type>RevVec_t</type> and dimension="Channel".

If an NC block in which this NC function is programmed is being executed, zeros are saved for the orientation vector and the binormal vector in the system data. This is only not true if the NC function is programmed in the Init string.

If the recording has been activated, the orientation vector and the binormal vector are scaled to 1 immediately after the initialization and saved in the interpolator cycle in the system data.

The binormal vector is the vector perpendicular to the plane created from direction of motion and orientation vector. With active G41/G42, this is, for example, the vector from the cutter contact point to the cutter center point. For G41 it points to the left as the cutter is on the left side of the contour, for G42 to the right.



In this connection, see the detailed descriptions of the functions G41/G42 and G141/G142 in the "MTX Functional Description" documentation.

Syntax:	RecordRevVec (1) or RecordRevVec ()	Activates the recording of orientation and binormal vector whereas the binormal vector points to the left unless there is another specification by an active radius correction.
	RecordRevVec (2)	Activates the recording of orientation and binormal vector whereas the binormal vector points to the right unless there is another specification by an active radius correction.
	RecordRevVec (0) or RecordRevVec ()	Deactivates the recording of orientation and binormal vector.
Short form: RRV (. .)		

Tab. 6-121: Syntax RecordRevVec (RRV)

Example:	N3 G17 (X,Y,Z) X0 Y0	Selection of level: Processing in the x-y-plane.
	N5 RRV (2)	Activation with binormal vector to the right.
	:	Orientation vector = [0,0,0], binormal vector = [0,0,0]
	N7 G41 X8 ED3	Binormal vector to the left due to G41.
	N8 X20	Orientation vector = [0,0,1], binormal vector = [0,1,0]

Special features and restrictions: If G20 is active, there is no recording.

6.97 Torque reduction "RedTorque, RDT"

Description: "RedTorque" suppresses the value of machine parameter 1003 00010 by means of the programmed value on an axis-to-axis basis.

The value of the machine parameter itself is not changed.

This way, the reduced maximum torque of an axis that can go into effect after a positive edge of the axis interface signal "Torque reduction" (qAx_TrqLim) is no longer permanently limited to the value of the machine parameter, but can instead be set "dynamically".

Syntax:	RedTorque (<Axis1> {,<Axis2>{, ...}})	Sets the values for the reduced maximum torque to the programmed values.
	RedTorque (0) or RedTorque ()	Sets the values for reduced maximum torque to the machine parameters (MP 1003 00010).
	Short form: RDT (. .)	
	with	
	<Axis i>	Physical or logical axis designation, incl. the corresponding maximum torque (in %) of the axis standstill torque. Value range: 0 to 500 %

Tab. 6-122: Syntax of RedTorque (RDT)

Example:	N8 RDT (X5)	With the next positive edge of the relevant axis-based "torque reduction" signal, the maximum torque for the axis (X in this case) is limited to 5% of its standstill torque.
	:	

6.98 Program continuation with block pre-run after program abort "ReentContBlk, RCB"

Description: Marks a position in the program from which the program can be aborted after having been canceled.

Syntax:

ReentContBlk, short form: RCB RCB(1 {,<Cond 1>{, <Cond 2>{, <Cond 3>}}}) RCBD(1)	
	Marks the position as possible continuation point and memorizes the syntax and values of the optional conditions. See examples 1, 3 and 4
Cond [1,2,3]	CPL variable or CPL expression: The variable or expression must be of the Boolean, Int, Real or Double type. If the variable is of the string or array string type, the string or array string variable is evaluated in a way so that the string variable content is interpreted as if it has been directly programmed in the function. See example 4.
RCB(0)	Deletes the last memorized position. See example 2. The first parameter can be replaced by an integer variable. This variable must have the value 0 or 1. See example 5.
RCBD(1)	Deletes the last memorized position. In contrast to RCB(0), it is also effective in an asynchronous subroutine.

Tab. 6-123: Syntax

Examples: **Example 1:**

N130 RCB(1)	Marks block N130 as possible target of the block pre-run. No conditions specified.

Example 2:

N190 RCB(0)	Deletes the last target of the block pre-run, e. g. the target defined in example no.1.

Example 3:

310 FOR RUN%=1 TO 10 N320 X[10 * RUN%] N330 P DrillHole N340 RCB(1, RUN%) 350 NEXT RUN%	The subroutine "DrillHole" is executed 10 times in the loop. If, for example, the loop or the subroutine is canceled in the seventh loop due to an emergency stop, the value "OPERATION% = 6" has been saved with the last successful processing of RCB(). If the block pre-run was executed with this specification, the target block N340 would be found in the sixth loop iteration, i.e. with the NC program start, processing would start in the seventh loop iteration.

Example 4:

500 DIM STRVAR	is the string type.
\$ (30)	
510 STRVAR\$="RUN	The square brackets are required so that the CPL inter-
%<8"	preter deletes the string variable and replaces it by "RUN
	%<8".
...	Thus, programming of RCB (1, [STRVAR\$]) with
N550 RCB (1,	STRVAR\$="OPERATION%<8" corresponds exactly to
[STRVAR\$])	RCB (1, OPERATION%<8).
...	
STRVAR\$	

Example 5:

N800 RCB ([ENABLE%])	Parameterization of the first parameter.

Special features and restrictions:

CPL variables and expressions are only evaluated during the block preparation time. No additional evaluation of the expressions takes place after "Delete distance to go" and in retrace mode if these expressions have already been evaluated.

Conditions can only be specified if the program is linked. When performing a selection in the DIN mode, a runtime error is generated in the case of conditions to be evaluated.

A condition may only be evaluated for the following types:

Integer, Real, Double, Boolean.

Furthermore, structured types and array types must not be specified.

The programs involved in the call activating RCB must not have been modified between a canceled processing and a following "Block pre-run after program abort". The involved programs are all programs which were contained in the subroutine list of the last executed RCB command, i. e. the main program and at least the program in which RCB was programmed.

RCB is ignored without any comment in asynchronous subroutines. If a re-entry mark is deleted in an asynchronous subroutine, use the syntax "RCBD(1)" or "RCBDelete(1)". This is a functional extension of the function RCB(0).

6.99 Backing up the program starting point for re-entry after program cancelation "ReEntryRef, RERF"

Description: Backs up the current absolute coordinates of the channel as program reference. In case of cancellation, the program reference of the call chain is stored.

Saving and restoring the program reference is required for incremental programs as otherwise, the point of interruption in the target block cannot be found during re-entry.

NC functions with high-level language syntax

Syntax:

ReEntryRef, short form: RERF	
RERF() RERF	Backs up all current workpiece coordinates of the channel.
RERF(<coord>=IC(<shift>), ... ,<coord>=IC(<shift>))	Offsets the reference point incrementally. This can be required after leaving and returning to contour to a contour, offset by leaving of contour. See example 2.

Tab. 6-124: Syntax

Examples:**Example 1: Reference at program start**

0010 IF (SD.SysSRun.Active<> 0) THEN	Program restart active?
0020 CNT%=SD(20,2)	
0030 DIM RP_WCS!(8)	Reading in the reference point of the re-entry data
0040 FOR I%=1 TO CNT%	
0050 RP_WCS![I%]=SDR(100,I%,1)	
0060 NEXT	Restore reference position upon re-entry
N070 G1 G90	
N080 X=[RP_WCS![1]]	
N090 Y=[RP_WCS![2]]	Backup current position as reference
N100 Z=[RP_WCS![3]]	
0110 ENDIF	
N120 RERF	

Example 2: Reference offset in re-entry ASUP

```

0010 IF SD.SysReentry.AsupActive=1 THEN
0020 REM STRATEGY%=1 : REM startpoint
0030 STRATEGY%=2 : REM endpoint
0040 REM STRATEGY%=3 : REM breakpoint
0050 END_X! = WCS("X")
0060 END_Y! = WCS("Y")
0070 XIND% = SD(22,1,2)
0080 YIND% = SD(22,2,2)
0090 SPOS_X!=SDR(78,XIND%,1)
0100 SPOS_Y!=SDR(78,YIND%,1)
0110 EPOS_X!=SDR(78,XIND%,2)
0120 EPOS_Y!=SDR(78,YIND%,2)
0130 BPOS_X!=SDR(78,XIND%,3)
0140 BPOS_Y!=SDR(78,YIND%,3)
0150 CASE STRATEGY% OF
0160 LABEL 1
0170 DELTA_X! = END_X! - BPOS_X! - EPOS_X! + SPOS_X!
0180 DELTA_Y! = END_Y! - BPOS_Y! - EPOS_Y! + SPOS_Y!
0190 LABEL 2
0200 DELTA_X! = END_X! - BPOS_X!
0210 DELTA_Y! = END_Y! - BPOS_Y!
0220 LABEL 3
0230 DELTA_X! = END_X! - EPOS_X!
0240 DELTA_Y! = END_Y! - EPOS_Y!
0250 ENDCASE
N260 RERF(X=IC([DELTA_X!]),Y=IC([DELTA_Y!]))
N270 REPOSTP([STRATEGY%])
N280 G77 X2 Y2 ;relative
0290 ENDIF

```

Special features and restrictions: Between setting of program reference and reference offset, no functions transforming the workpiece coordinates (Coord, G15x, G5x, RAX, GAX etc.) must be programmed.

6.100 Removing the axis from the axis group "RemAxis, RAX"

Description: Removes a synchronous axis from the calling channel. This turns a synchronous axis into an asynchronous axis.

The asynchronous axis can then be programmed in any channel using its physical axis name.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

RemAxis (<PAN> <PAI> <LAN>{, <PAN> <PAI> <LAN>}...)	
Short form: RAX (. .)	
with	
<PAN>	Physical axis name Specifies the axis that is to be removed from the current channel.
<PAI>	Physical axis index Same effect as <PAN>
<LAN>	Logical axis name Same effect as <PAN>

Tab. 6-125: Syntax of RemAxis (RAX)

Example:

```

:
N030 RAX (XP,      Physical axis XP, the physical axis assigned index 2 and logi-
2, Z)             cal axis Z are removed from the channel.
:

```

Special features and restrictions:

- Block preparation is not stopped when removing an axis.
- Invalid axis names generate an error message.
- If an axis to be removed is defined in the system but no longer present in the current channel, no error message is displayed.
- Axis positions in the same block must always be programmed according to RemAxis(...).

6.101 Removing logical axis name "RemLogName, RLN"

Description: Removes the logical axis name of a synchronous axis from the calling channel.

The axis remains in the channel, but can only be programmed there through its physical axis name or its physical axis index.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

RemLogName (<PAN> <PAI> <LAN>{, <PAN> <PAI> <LAN>}...)	
Short form: RLN (. .)	
with	
<PAN>	Physical axis name Specifies the axis whose logical axis name is to be deleted in the current channel.
<PAI>	Physical axis index Same effect as <PAN>
<LAN>	Logical axis name Same effect as <PAN>

Tab. 6-126: Syntax of RemLogName (RLN)

Example:

```

:
N030 RLN(YP,3,Z)    The logical names of physical axis YP, the third physical
:                  axis and logical axis Z are removed from the calling chan-
                    nel.

```

Special features and restrictions:

- A given axis must be at a standstill.
If this is not the case, the control generates an error message and aborts the program.
- Axis positions in the same block must always be programmed according to the expression RemLogName(...).

6.102 Removing channel spindles "RemSpindle, RSP"

Description: Removes a logic spindle from the calling channel.

Syntax:

```
RemSpindle (<SysSpNo> | <SysSpName> | <ChanSpName> , {...})
```

Removes a channel spindle from the channel.

with

<SysSpNo>	1..32
<SysSpName>	System spindle name
<ChanSpName>	S1 .. S8

Short form: RSP

Tab. 6-127: Syntax of RemSpindle (RSP)

Example:

```

...
N50 RSP(S2)          submits the second channel spindle from the channel
N60 M203 S2=500      generates a runtime error, as no second channel spindle is
...                  available

```

Special features and restrictions:

- A spindle which is operated at constant cutting velocity (G96) cannot be removed from the channel
- If the channel-specific integer auxiliary functions CSP1 to CSP8 are defined, the PLC is informed about the current channel spindle assignment (e.g., CSP2 0 is no longer assigned to ChannelSpindle2).

6.103 Block repetition "REPEAT, RPT"

Description: NC function REPEAT(<RepetitionFactor>) (short form: RPT) can be used to repeat NC blocks.

The repeat factor specifies how often the block is executed (1 = once). The permissible value range is 1 to 65535.

If containing a subroutine call, the complete block including subroutine is executed repeatedly. The subroutines may contain further blocks with RPTs.

If a traversing motion is programmed and a modal subroutine is active, the complete block including the modal subroutine is executed.

RPT is allowed in automatic and manual data input modes.

CPL parts are run through only when called for the first time.

Syntax: RPT<Repetition factor> or REPEAT(<Repetition factor>)

- <Repetition factor> specifies how often the NC block is to be executed.
Value range: $1 \leq n \leq 65535$
- CPL variables are allowed

Example:

```

...
N10 G91 X10 Y10 RPT5 ; The block is executed 5 times
...
20 MAXCNT%=10
N20 G91 X5 Y5 ; The complete block including subroutine is executed
RPT[ MAXCNT% ] P Up 10 times
...
N10 G81
N20 G91 X20 Y20 ; The block is executed 8 times. The modal subroutine
RPT(8) is called each time.
...

```

Special features and restrictions: RPT is not effective in InitString nor in the action block of the block pre-run.

6.104 Asynchronous subroutines

6.104.1 Defining the repositioning behavior "REPOSDEF"

Description: Defines the repositioning feed and/or the point of return to contour locally (in an asynchronous subroutine).

Syntax:

REPOSDEF({F<Feed> OF<Feed>} , {TP<point>})	
with	
F<Feed>	Feed for automatic repositioning motion at the end of the asynchronous subroutine.
OF<Feed>	Optional feed rate for automatic repositioning motion at the end of the asynchronous subroutine. This feed value is used only if no feed is active on initiating the asynchronous subroutine.
TP<Point>	1: Starting point 2: End point 3: Interruption point

Tab. 6-128: Syntax

- Special features and restrictions:**
- The function is intended for use in an asynchronous subroutine.
 - The repositioning point set previously using ASP RTP (see [chapter 6.8.5 "Asynchronous subroutines: register ASPSET" on page 211](#)) for the running asynchronous subroutine automatically goes back into effect after the end of the subroutine.

6.104.2 Defining point of return to contour in asynchronous subroutine "REPOSTP"

Description: Defines whether the control, after the end of the asynchronous subroutine in which function "REPOSTP" is programmed, is to position to the

- Starting point

- End point
- Point of interruption

of a traversing block that could be interrupted.

If no traversing block was active at the time of interruption, the control always positions on the last active coordinates.

REPOSTP thus suppresses a point of return to contour that was set earlier using ASPRTP for the currently running asynchronous subroutine (see [chapter 6.8.4 "Asynchronous subroutines: Define point of return to contour ASPRTP" on page 210](#)).



For comprehensive information on the use and parameterization of asynchronous subroutines, refer to the documentation "MTX Functional Description".

Syntax:

REPOSTP (<point>)	
with	
<Point>	Desired point of return to contour: 1: Starting point 2: End point 3: Interruption point

Tab. 6-129: Syntax of REPOSTP

Special features and restrictions:

- The function is intended for use in an asynchronous subroutine.
- The repositioning point set previously using ASPRTP (see [chapter 6.8.4 "Asynchronous subroutines: Define point of return to contour ASPRTP" on page 210](#)) for the running asynchronous subroutine automatically goes back into effect after the end of the subroutine.
- Any correction changes made in an asynchronous subroutine are automatically included in the internal calculation of the required point of return to contour.

6.105 Retrace, RETRACE

Description: Program sections within a part program for which the "retrace on the contour" functionality is to be enabled are marked with the help of the "retrace" NC function.

Within this identifying marking, the control generates a forward and a backward program when processing the part program. These programs can be processed in reverse operation.

Line-by-line, the forward and the backward program contains the contour generating specifications of the part program. The programmed coordinates and part of the contour-relevant NC functions are automatically transmitted to the forward and the return program. Asynchronous axis motions are not transmitted. For an application-specific adjustment of the forward and the backward programs, also see ADVA, ADVS, REVA and REVS.

The retrace itself is controlled with the help of interface signals.

NC functions with high-level language syntax

Syntax:

Retrace(1)	Marking the beginning of an area for which retrace on the contour is to be enabled.
Retrace ([<fwd>][,<bwd>])	As Retrace(1) but with data storage of the forward and the backward program. The files are the image of the internal programs for the retrace mode. The program files are generated at runtime. After the end of range identification - Retrace(0)- has been processed, the files can be viewed.
with	
<fwd>	Retrace forward program file name
<bwd>	Retrace backward program file name
Retrace(0)	Marking the end of an area for which retrace on the contour is to be enabled.
Short form: -	

Tab. 6-130: Syntax Retrace, RETRACE

The table below shows the automatically transmitted NC function in forward and return programs:

NC function	Function group
G0, G1, G2, G3, G5, G6	Interpolation functions *) **)
G8, G9	Path slope *)
G17, G18, G19	Plane preselection *)
G40, G41, G42	Milling path correction *)
G43, G44	Transition path segment
G45, G46	Feed at milling cutter contact point/milling cutter center point
G47, G48	Tool correction *)
G53.x,...,G59.x	Zero offsets
G61, G62	Exact stop
G70, G71	Inch/metric programming
G90, G91	Absolute/relative programming:
G93, G94, G95	Feed programming
G140, G141, G142	3D tool radius correction
G153.x,...,G159.x	Placements
CHLL, CHSL	Local chamfer programming
SCO, SCOL	Modal or local corner rounding with splines
RNDL	Local rounding

LockRetrace	Retrace switching lock (on)
NcfLockRetraceOff	Retrace switching lock (off)

- *) No parameter support in the forward and the backward program
- **) For G6, only the spline programmings specified in the documentation "Functional Description - Special Functions, chapter "Channel control", "→ Retrace: Retracing on the contour", → "Description", are supported

Tab. 6-131: Switchable NC functions in the retrace area

Example:

```
...
N100 Retrace(1)
N110 G1 F1000 X27.571 Y3.958
N120 X38.729 Y26.945
N130 M125 ADVA(M125) REVA(M126)
...
N200 Retrace(0)
...
```

Special features and restrictions:

- The retrace area limits the retrace mode. Retracing is only possible within the active area. This also applies if several areas are defined in the part program
- Forward and backward programs contain the resolved NC block sequence without CPL statements and without subroutine calls
- No subroutine calls must be inserted in the forward and backward program
- The backward operation supports linear and circular interpolation
- Modal switchings in the retrace are only supported for the functions listed in [tab. 6-131 "Switchable NC functions in the retrace area" on page 348](#)



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.106 Retrace switching lock LockRetrace, LOCKRETRACE

Description: Use the NC function "LockRetrace" to identify program parts within a parts program in which retrace switching is locked and which result in a warning/message. Areas labeled that way can be passed. However, it is not possible

- to start the retrace mode with subsequent retracing
- to change the direction of machining in retrace mode
- to end the retrace mode by continuing the interrupted part program

Syntax:

LockRetrace(1)	Identification of the program area start in which retrace switchings are locked.
LockRetrace(0)	Identification of the program area end in which retrace switchings are locked.
Short form: -	

Tab. 6-132: Syntax LockRetrace, LOCKRETRACE

Example:

```

...
N100 LockRetrace(1)
N110 SplineDef(2203,X,Y,Z)
N120 G6
...
N200 LockRetrace(0)
...
```

- Special features and restrictions:**
- With a channel reset, a potentially still active retrace switching lock is automatically canceled.
 - If a retrace switching lock has been enabled and has not been disabled again before a retrace area starts, the locked area cannot be exited during retrace operation. The program can only be deselected with a channel reset.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.107 Extending the retrace backward program block "ReverseAppend, REVA"

Description: For reverse operation, a backward program block is generated for each part program block in the retrace area Retrace(1)..Retrace(0). The function ReverseAppend inserts the specified NC functions at the end of the current backward program block.

Syntax:

ReverseAppend(<NCF>...<NCF>)	
Short form: REVA(...)	
with	
<NCF>...<NCF>	List of NC functions

Tab. 6-133: Syntax ReverseAppend (REVA)



Contour changes result in switching errors in reverse operation. CPL functions and subroutine calls are not allowed.

Example:

Part program:	Backward program:	Note:
N100 X100 M10	N1 X=10	Auxiliary functions are not applied
N200 X0 REVA(M125 M126)	N2 X=100 M125 M126	In the backward program, M125 and M126 are to be output to the PLC.



For checking purposes, save the modified backward program in a file (see [chapter 6.105 "Retrace, RETRACE" on page 347](#)).

6.108 Replacing the retrace backward program block "ReverseSubstitute, REVS"

Description: For reverse operation, a backward program block is generated for each part program block in the retrace area Retrace(1)..Retrace(0). The "ReverseSubstitute" function replaces the current backward program block by the specified NC functions.

Syntax:

ReverseSubstitute(<NCF>...<NCF>)	
Short form: REVS(...)	
with	
<NCF>...<NCF>	List of NC functions

Tab. 6-134: Syntax ReverseSubstitute (REVS)



Contour changes result in switching errors in reverse operation.
CPL functions and subroutine calls are not allowed.

Example:

Part program:	Backward program:	Note:
N1 X100	N1 X10	
N2 X0 F10 REVS(X100 F100)	N1 X100 F100	Allow a higher feed in the backward program. Set starting point in the backward program for programmed coordinates!



For checking purposes, save the modified backward program in a file (see [chapter 6.105 "Retrace, RETRACE" on page 347](#)).

6.109 Input tool: Rotating "Rotate(...), ROT(...)"

Description: The "rotate" function is one of the input tools.
The control rotates a programmed contour in the active plane (for planes, refer to G17, G18, G19 or G20).

Rotation always refers to the current point of rotation (see the function "Pole-Set", [chapter 6.87 "Defining the reflection/rotation point PoleSet, PLS" on page 322](#)). If this was not explicitly programmed, the current program zero point is the point of rotation.

Using this function, recurrent programming steps rotated about a specific angle have to be programmed only once.

In addition, dimensions of angular workpieces to the basic workpiece coordinate system do not have to be converted anymore; just take them directly from a production drawing and specify the relevant angle of rotation. The control does the rest.



Rotate is an input tool, which does not change the current program coordinate system. An input tool merely provides another input method for the program coordinates.

Rotate can also be used together with Mirror and Scale.

The following applies:

- The function acts modally. It remains active until it is disabled.
- It may be programmed in the same block as other path conditions and input tools.

Syntax:	Rotate (<angle of rotation>)	Activates rotation about the desired <AngleOfRotation>.
	Rotate (0) or Rotate ()	Deactivates rotation for all the axes in the channel. Sets all angles of rotation to "0". Approached axis positions are retained until they are reprogrammed.
	Short form: ROT (. .)	
	with	
	<Angle of rotation>	> 0: counter-clockwise rotation < 0: clockwise rotation = 0: rotation off All subsequently programmed coordinates of the active plane are rotated around the point of rotation (see PoleSet, chapter 6.87 "Defining the reflection/rotation point PoleSet, PLS" on page 322). Rotation is not applied until the next traversing information.

Tab. 6-135: Syntax of Rotate(...), ROT(...)

Example:	:	Rotation ON
	N30 ROT (45)	All subsequently programmed coordinates of the active plane are rotated counter-clockwise by 45 degrees about a possibly programmed rotation point.
	:	

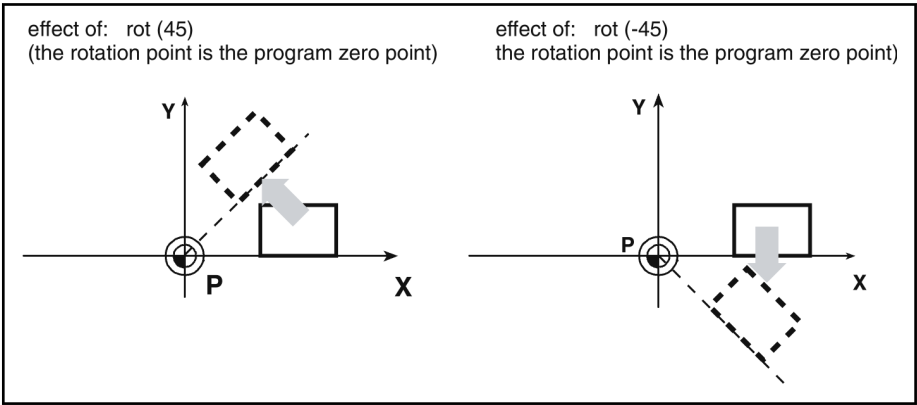


Fig. 6-32: Example Rotate(...), ROT(...)

Rotating an orientation vector: An orientation vector is rotated about the normal of the active plane. Any scaling that may be active does not affect the result, nor does a mirror/rotation point.

Syntax as described under "Programming".

- Special features and restrictions:**
- The function takes into account the interpolation parameters in the case of circular interpolation.
 - It influences the programmable contour shift.
- Refer to [chapter 6.116 "Programmed contour shift Shift, SHT" on page 363](#).

- *It does **not** influence:*
 - Zero offsets (G54-G59.5; see chapter 5.2.21 "Zero offsets (ZO) G53, G53.1-G59.1 to G53.5-G59.5" on page 142),
 - program coordinate offsets (Trans or ATrans; see chapter 6.151 "Program coordinate offset Trans, TRS, Additive Program Coordinate Offset ATrans, ATR" on page 405),
 - setting of the program position ("SetPos"; see chapter 6.115 "Setting program position SetPos, SPS" on page 362),
 - Approaching reference point coordinates (G74 ; see chapter 5.2.26 "Approaching the reference point coordinates G74" on page 149),
 - Approaching the fixed machine axis position (G76 ; see page chapter 5.2.29 "Approaching the fixed machine axis position G76" on page 152),
 - Milling tool radius and tool length correction values.

6.110 Rounding corners

6.110.1 Rounding corners specifying the deviation "RoundEps, RNE"

Description: This function inserts tangential transition arcs between 2 linear blocks in the main plane.

This causes a minor modification of the programmed contour at these corners, but continuous velocity and acceleration characteristics are achieved during interpolation.

Syntax:

RoundEps (<deviation>)	Activate rounding corners between two linear blocks.
RoundEps (DEF)	Rounding of edges ON between 2 linear blocks with default deviation from MP 7050 00110.
RoundEps (0) or RoundEps ()	Rounding of edges OFF between 2 linear blocks.
Short form: RNE (. .)	
with	
<Deviation>	Maximum permitted deviation (in mm) between the modified and the programmed contour. Number of decimals point are permitted. The control calculates a suitable tangential transition arc.

Tab. 6-136: Syntax of RoundEps, RNE

Special features and restrictions:

- The functions "ChLength", "ChSection", "RoundEps" and "Rounding" **act modally** and deselect each other.
- *The control does **not** execute the "Rounding corners" function if*
 - at least one of the two neighboring blocks is not a linear block,
 - at least one of the two neighboring blocks has a path segment outside the selected principal plane, or
 - At least one of the two neighboring blocks has a travel distance which is smaller than the distance set in MP 7050 00120 (2 to 90 mm, default value: 10 mm) or

- if the block transition is considered to be continuous according to the machine parameters, i.e. the angle between the two blocks is smaller than the maximum angle specified in MP 7050 00130 (default = 1 degree).
- If the plane, a zero offset or an axis transformation is switched between two subsequent blocks, no rounding is generated.

6.110.2 Rounding corners with radius specification "Rounding", "RND", "RNDL"

Description: This function inserts tangential transition arcs between two linear, circular or helical blocks in the main plane.

This causes a minor modification of the programmed contour at these corners, but continuous velocity and acceleration characteristics are achieved during interpolation.

The "Rounding corners with radius specification" functionality is provided modally (Rounding, RND) and block by block (RNDL). When programming a rounding block by block, a local feed can be programmed used to machine the rounding.

Syntax:

Rounding (<Radius>)	Enabling the modal rounding of corners between two linear/circular/helical blocks.
Rounding (0) or Rounding ()	Disabling the rounding corners between two linear/circular/helical blocks.
Short form: RND (. .)	
with	
<Radius>	Desired radius of the transition arc. Decimal digits are allowed.

Tab. 6-137: Rounding (RND) syntax

RNDL (<Radius> {, <Loc-Feed>})	Rounding local corners
with	
<Radius>	Desired radius of the transition arc. Decimal digits are allowed.
<LocFeed>	Local feed used to machine the rounding.

Tab. 6-138: RNDL syntax

Special features and restrictions:

- The functions "ChLength", "ChSection", "RoundEps" and "Rounding" **act modally** and deselect each other.
- The "RNDL" function **acts locally** (block by block).
- If either a chamfer or a rounding is modally active and if a local rounding is programmed at the same time, the local function cross-fades the modal function block by block.
- *The control does **not** execute the "Rounding corners" function if*
 - At least one of the two neighboring blocks has a travel distance which is smaller than the distance set in MP 7050 00120 (2 to 90 mm, default value: 10 mm) or
 - the block transition is considered to be continuous according to the machine parameters, i.e. the angle between the two blocks is

smaller than the maximum angle specified in MP 7050 00130 (default = 1 degree).

- If the plane, a zero offset or an axis transformation is switched between two subsequent blocks, no rounding is generated.
- Only the components of the active working plane for rounding are taken into account. In case of spatial straight lines, this causes a change in direction in space. Similar applies to helical path segments.

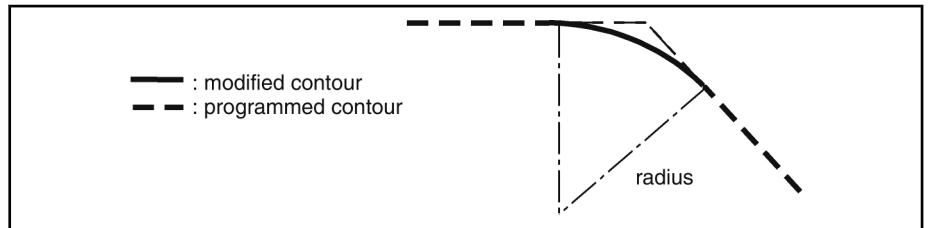


Fig. 6-33: Rounding corners with radius specification

6.111 Synchronizing system axis coupling "SACSYNC"

Description: Synchronizes the NC program with the condition of the system axis coupling (SAC) and transfers the decoupling positions (\$target position) into the work-piece coordinate system without triggering an axis motion.

In case of a programmed motion in the same block, the command reduces the path velocity at the end point to zero and takes effect after the end point has been reached.

As soon as the system axis coupling is in the specified condition, the position is applied and processing of the program is continued.

Syntax:

SACSYNC	Waits until the system axis coupling (SysAxCoupleSta[i].State=0) of all channel axes is disabled and applies the decoupling position to the NC coordinate system.
SACSYNC (TO=0)	Applies the decoupling positions of all decoupled channel axes without waiting.
SACSYNC (<Axis1>=<SAC status>, ..., <AxisN>=<SAC status>)	
	It waits until all specified SAC states for the axes have been reached and applies the deactivation position for decoupled axes.
SACSYNC (TO=<time>, <Axis1>=<SAC status>, ..., <AxisN>=<SAC status>)	
	As before, it is, however, only waited until the time <Time> in ms has elapsed. The control outputs an error without applying the coupling positions if the wait condition does not occur.

NC functions with high-level language syntax

with <Axis i>	Channel axis identifier or System axis identifier or Channel spindle identifier (S1,...,S8) or System spindle identifier
The SAC state can assume the following values from SysAxCoupleSta[i].State: 0: Off 1: Coupling (HUNTING) 2: Recoupling (RECOUPLING) 3: Synchronous (COUPLED) 4: Stopping 5: Stopped 6: Decoupling (DISCOUPLE) 7: Error 1x: Abort (ABORT-x) with x from 1-7 2x: Real-time switching (PENDING-x) with x from 1-7 The following values from SysAxCoupleSta[i].StateAddOn can be overwritten as waiting condition in the SAC state (with x from 1-7 and y from 1-2): 1000, 10yx: Synchronous window 1 reached (INPOS1) 2000, 20yx: Synchronous window 2 reached (INPOS2) 3000, 30yx: Synchronous window 1 and 2 reached (INPOS1 + INPOS2)	Remark: The waiting positions should use static states such as: 0: OFF, 3: Synchronous or 5: stopped since transition states such as: 1: decoupling, 2: recoupling, 4: stopping or 6: decoupling may not be taken or only taken shortly according to the axis position or axis velocity. SACSYNC compares the waiting condition with the current SAC state as follows: - Waiting condition is 0 or contains only possible states from SysAxCoupleSta[i].State: SACSYNC waits for an exact match of the waiting condition with SysAxCoupleSta[i].State. - Waiting condition contains only possible states from SysAxCoupleSta[i].StateAddOn: SACSYNC waits for matching of the states actively set in the waiting condition with SysAxCoupleSta[i].StateAddOn. - Waiting condition contains possible states from SysAxCoupleSta[i].State and from SysAxCoupleSta[i].StateAddOn: SACSYNC waits for identical states set to active with SysAxCoupleSta[i].StateAddOn in the waiting condition in StateAddOn and for an exact match of the waiting condition in the State area with SysAxCoupleSta[i].State.

Tab. 6-139: SACSYNC syntax

Example

;Enable the XB-axis coupling with position correction:

```

1 XB%=10 ;System axis index 10
1 SD.SysAxCoupleCmd[XB%].SyncMode = 0 ;Position-specific coupling
1 SD.SysAxCoupleCmd[XB%].OnOff = 1 ;Enable coupling
1 SD.SysAxCoupleCtr.Validate = 1 ;Activate data
SACSYNC (XB=3) ;Waiting for synchronous run
... ;Machining

;Deactivate xb-axis coupling without compensation motion:
1 SD.SysAxCoupleCmd[XB%].SyncMode = 2 ;Velocity compensation

```

```

1 SD.SysAxCoupleCmd[XB%].OnOff = 0      ;Deactivate coupling
1 SD.SysAxCoupleCtr.Valid = 1           ;Activate data
SACSYNC                                ;Waiting for deactivation and ap-
                                         plying the decoupling position

```

Tab. 6-140: Example of the SACSYNC usage

More examples

SACSYNC (XB=0)	;Wait until the coupling of the xb-axis is deactivated.
SACSYNC (S1=3, SSP11=3)	;wait until channel spindle S1 and system spindle SSP11 are command-value coupled.
SACSYNC (XB=5)	;Wait until the coupling of the xb-axis is stopped.
SACSYNC (XB=1000)	;Wait until the be-axis reached the INPOS1 window. Whether it also reached the INPOS2 window is of no importance.
SACSYNC (XB=3000)	;wait until the xb-axis reached the INPOS1 window and the INPOS2 window.
SACSYNC (XB=2003)	;Wait until the xb-axis is command value-coupled and reached the INPOS2 window. Whether it also reached the INPOS1 window is of no importance.
SACSYNC (XB=3003)	;wait until the xb-axis is command value-coupled and has reached the INPOS1 window and the INPOS2 window.

Tab. 6-141: Examples of further SACSYNC calls

Special features and restrictions

- Asynchronous axes have to have completed the interpolation before programming a deactivation condition with SACSYNC(<AsyncAxis>=0). Other conditions may also be specified with interpolation being active.
- In case of abort of the function by means of channel or system reset, the coupling offset remains active if the axis has been decoupled with SyncMode=2 (without compensation motion).
- In order to also apply the decoupling positions (\$target position) of channel axes with reset, SACSYNC(TO=0) can be stored in the Init string. Active couplings are not be deactivated by a reset.
- The coupling offset of spindles can only be reduced using spindle interpolators (M3/M4, M19, SPCC switching) or drive parameter set switchings (C-axis mode, gear change, etc.).
- It can be waited on the conditioned activation (SwitchCond > 0) of the master axis data with SACSYNC(...<axis>=<state>...), since the current coupling state till switching with PENDING (+20) is superimposed.
- SACSYNC only waits for reaching the synchronous window, but not for exiting the it.

6.112 Input tool: Scale"Scale(...), SCL(...)"

Description: The "scaling" function is one of the input tools.

The control scales a programmed contour up or down in reference to the program zero point.

Scaling always refers to the current reflection point (see the function "Pole-Set", [chapter 6.87 "Defining the reflection/rotation point PoleSet, PLS" on page 322](#)). If this was not explicitly programmed, the current program zero point is the reflection point.

NC functions with high-level language syntax

The "Scale" function can be used in part programs for programming contours using one fixed size (standard size). Then, prior to calling such a "standardized" part program (e.g. as a subroutine), use scaling factors for each axis to determine the scale of the programmed contour.

Thus, it is easy to compensate, for instance, for the contraction of the workpieces in the manufacture of the moulds for cast and forged parts.



Scaling is an input tool; as a result, it does not change the current program coordinate system. An input tool merely provides another input method for the program coordinates.

Scale can also be used together with Mirror and Rotate.

The following applies:

- The function acts modally. It remains active until it is disabled.
- It may be programmed in the same block as other path conditions and input tools.

Syntax:

<code>Scale (<Axis 1><Factor>{,...})</code>	
	Enabling scaling for the indicated axes using the programmed factor.
<code>Scale(0)</code> or <code>Scale()</code>	Deactivates scaling for all the axes in the channel. Sets all scaling factors to a value of "1". Approached axis positions are retained until they are reprogrammed.
Short form: <code>SCL(. . .)</code>	
with	
<code><Axis 1></code>	Axis address (e.g. X) that is to be scaled.
<code><Factor></code>	Programming the axis address with a positive factor activates the function. By doing so, all subsequently programmed path commands of the corresponding axis (e.g. X10) are multiplied internally by this value: Factor > 1: The contour is scaled up Factor < 1: The contour is scaled down Factor = 1: The contour remains unchanged Scaling itself does not release a traversing motion; it is not applied before the next traversing information.

Tab. 6-142: Syntax of Scale(...), SCL(...)

Example:

:

N30 SCL(X3,Y0.5)

:

Scaling ON

All x-coordinates programmed subsequently are multiplied by "3", and the y-coordinates are multiplied by "0.5".

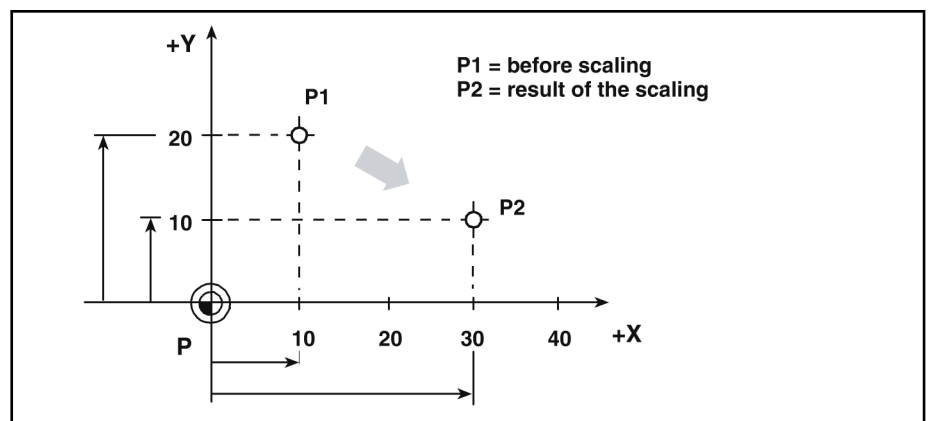


Fig. 6-34: Example Scale(...), SCL(...)

Special features and restrictions:

- Negative scaling factors are not permitted.
- The scaling factors of all participating coordinates of the circle plane must be identical for a circular/helical/helical-N interpolation! Otherwise, an error message is generated.
- The function also affects the I, J, K interpolation parameters as well as the amount of the R address (for radius programming).
- It influences the programmable contour shift.

Refer to [chapter 6.116 "Programmed contour shift Shift, SHT" on page 363](#).

- *It does **not** influence:*
 - feed programming or the active feed,
 - zero offsets (G54-G59.5; see [chapter 5.2.21 "Zero offsets \(ZO\) G53, G53.1-G59.1 to G53.5-G59.5" on page 142](#)),
 - program coordinate offsets (Trans or ATrans; see [chapter 6.151 "Program coordinate offset Trans, TRS, Additive Program Coordinate Offset ATrans, ATR" on page 405](#)),
 - setting of the program position ("SetPos"; see [chapter 6.115 "Setting program position SetPos, SPS" on page 362](#)),
 - Approaching reference point coordinates (G74 ; see [chapter 5.2.26 "Approaching the reference point coordinates G74" on page 149](#)),
 - Approaching the fixed machine axis position (G76 ; see [page chapter 5.2.29 "Approaching the fixed machine axis position G76" on page 152](#)),
 - Milling tool radius and tool length correction values.

6.113 Selected additional coordinate coupling

6.113.1 Selected additional coordinate coupling "SelCrdCouple, SCC"

Description: Couples a workpiece coordinate of the current channel (target) to a workpiece coordinate of another channel (source). The coordinate value of the source thus has an additive effect for the coordinate value of the target.

Thus, motions in the current channel can be superimposed by motions that are programmed in another channel.

NC functions with high-level language syntax

Syntax:

SelCrdCouple (SC<Channel>, CL(<Q1>,<Z1>{{,<Qn>,<Zn>}...}))	
	Selective additive coordinate coupling ON
SelCrdCouple (0) or SelCrdCouple ()	All active coordinate couplings OFF
Short form: SCC (. .)	
with	
<Channel>	Number of the channel where <Qn> is located. Input value: Integer <Channel> must be less than the number of the current channel.
<Qn>	Source Logical name or logical number of the coordinate in <Channel>.
<Zn>	Target Logical name or logical number of the coordinate in the current channel.

Tab. 6-143: Syntax of SelCrdCouple (SCC)

Examples:

SCC (SC1, CL (YA, YB))	Couples coordinate YB in the current channel to coordinate YA from channel 1.
SCC (SC1, CL (1, 1, 2, 2))	Couples the coordinates with the logical numbers 1 and 2 in the current channel to the coordinates with the logical numbers 1 or 2 from channel 1.

6.113.2 Selected additional coordinate coupling with table "SelCrdCoupleTab, SCCT"

Description: Couples a workpiece coordinate of the current channel (target) to one or two workpiece or machine coordinate(s) of one or two other channels (sources). The coordinate values of the sources thus have an additive effect for the coordinate value of the target. Thus, motions in the current channel can be superimposed by motions that are programmed in one or two other channels.

For further information, refer to the "MTX Functional Description", chapter "Couplings", section "Selected additional coordinate coupling with tables".

Syntax:

SelCrdCoupleTab (DE (<Name>,<Fact>) , SO (<Channel>, <Name>,<Fact>,<Tab>,<Offs>, WCS MCS) {, SO2 (<Channel>,<Name>,<Fact>,<Tab>,<Offs>,WCS MCS) })	
Short form: SCCT	
with	
DE (destination)	
<Name>	Name of the target coordinate
<Fact>	Scaling factor of the target coordinate (1.0 unless programmed)
SO (source)	

<Channel>	Number of the source channel
<Name>	Name of the source coordinate
<Fact>	Coupling factor (1.0 unless programmed)
<Tab>	Name of the coupling table
<Offs>	Offset value for table access (0.0 unless programmed)
WCS MCS	Coordinate system of the source: WCS = workpiece coordinate MCS = machine coordinate Default: WCS
SO2 (source)	Analog to SO

Tab. 6-144: Syntax of SelCrdCoupleTab (SCCT)

The function SCCT can be programmed several times for different coordinates.

Examples:

SCCT (DE (YB) , SO (1, YA) , SO2 (1, YC))	The YB results from the sum of the programmed by, the YA and the YC position.
SCCT (DE (YB, 0) , SO (1, YA))	The by is equal to the YA. The programmed by has no effect.
SCCT (DE (ZB) , SO (1, ZA , , SCCTab.fct))	Couples the coordinate ZA of channel 1 to the coordinate ZB of the active channel using the coupling table "SCCTab.fct".

Canceling a coupling:

SCCT ()

SCCT() opens all couplings generated with SCCT(...).

6.114 Suppressing the collision monitoring "SCLN"

Description: The function SCLN locally suppresses the collision monitoring CLN of the tool radius correction. Thus, it is possible to process self-contained geometric contour loops in case of active collision monitoring. If the collision monitoring encounters a block labeled SCLN, the block look-ahead is closed.

Syntax:

SCLN [(<mode>)]	
with	
<mode>	1: Block look-ahead of the collision monitoring is canceled at the marked block. 2: To identify the approaching block to a closed contour: The block look-ahead of the collision monitoring is canceled at the block colliding with the approaching block. Default value: 1

Tab. 6-145: Syntax SCLN

Example:

N100 CLN (DLA10)	Collision monitoring with 10 program blocks, block look-ahead.
N110 G0 X4.4645 Y-5.5355	Position to starting position.
N120 G1 F750 Z5	Infeed tool axis

N130 G1 F1250 G42 ED1	Activate the tool radius correction
X6.4645 Y-5.5355	
N140 G2 R5 X5 Y0 SCLN(2)	Tangential approach circle to contour. Collision with N190. SCLN(2) results in cancelling of the collision monitoring in block N190.
N150 G1 X10 Y0	Start of contour
N160 X10 Y10	
N170 X0 Y10	
N180 X0 Y0	
N190 X6 Y0	End of contour
N200 G2 R5 X8.5355 Y-3.5355	Tangential leaving of contour
N210 G40 G1 X10.85355 Y-5.5355	Deactivating the tool radius correction

Special features and restrictions: The SCLN function is effective on a block-to-block basis.

6.115 Setting program position "SetPos, SPS"

Description: Sets the current program zero point (based on the current program coordinate system and the active zero point) to the corresponding programmed value without triggering axis motions.

Then the new position values are displayed automatically.

Syntax:	
SetPos (<Coordinates>)	Sets the program zero point for the axes programmed under <Coordinates>.
SetPos	Resolves all offsets initiated by SetPos.
Short form: SPS (. .)	
with	
<Coordinates>	Set axis values to the given coordinates. The given coordinates are to be separated by commas (e.g.: (X0, Y0)).

Tab. 6-146: Syntax of SetPos, SPS

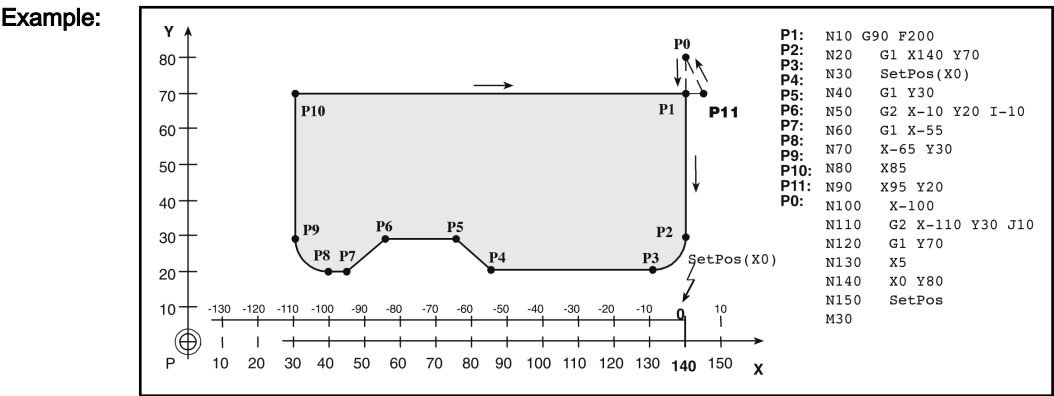


Fig. 6-35: Example: Set program position

Special features and restrictions: Whether SetPos offsets are cleared or maintained after a control reset can be set for each channel using the machine parameters.

6.116 Programmed contour shift "Shift, SHT"

Description: The "shift" function is one of the input tools.

The control shifts a programmed contour parallel to the axes of the program coordinate system.



"Shift" is an input tool; as a result, it does not change the current program coordinate system. An input tool merely provides another input method for the program coordinates.

The following applies:

- Programmed offset coordinates are applied until they are overwritten by a new shift block or disabled.
- If "Shift" is programmed by itself, it does not result in a traversing motion; however, traversing information can be included in the programming for the same block.



Incorrect programming may cause damage to the workpiece and the machine!

"Shift" is influenced by Mirror, Scale and Rotate, i.e. the coordinates of the new contour zero point which were programmed in the Shift block will also be mirrored, scaled or rotated!

Syntax:

Shift (<Coordinates>)	Enable contour shift
Shift (0) or Shift ()	Disable contour shift
Short form: SHT (..)	
with	
<Coordinates>	Program coordinates of the shifted contour zero point. If there are several coordinates, they are to be separated by commas (e.g.: SHT (X5, Y2)).

Tab. 6-147: Syntax of shift (SHT)

Example:

N10 SHT (X10, Y10, Z50)	New contour zero point at X10, Y10, Z50
:	Does not cause any axis traveling
N100 G1 X...Y...Z...	Traveling axes considering the offset.
:	
N110 SHT (X20, Y20)	New contour zero point at X20, Y20, Z50 (Z shift is retained!)
:	Does not cause any axis traveling
N210 SHT ()	Deactivates shift; the axes travel to the programmed position.
X...Y...Z...	
:	

6.117 Speed limitation "SMin, SMN", "SMax, SMX"

Description: Defines the speed range in which the spindle speed may vary in case of direct speed programming (G97) and constant cutting velocity G96 (see [chapter 5.2.42 "Constant cutting velocity G96, Direct speed programming G97"](#))

on page 166) during machining. The defined speed range applies for all gear stages. When speed limitation is active, all speed information is limited to the programmed values.

The system executes modifications to the speed (including those that are caused by the spindle potentiometer) only if they are located within the given speed range.

Syntax:	<code>SMin (S<speed1>)</code>	Activates <Speed1> as the lower limit of the permitted speed range.
	<code>SMin (S0)</code>	Cancels the lower limit of the first spindle or spindle group.
	Short form: <code>SMN (. .)</code>	
	<code>SMax (S<speed2>)</code>	Activates <Speed2> as the upper limit of the permitted speed range.
	<code>SMax (S-1)</code>	Cancels the upper limit of the first spindle or spindle group.
	Short form: <code>SMX (. .)</code>	
	with	
	<Speed1>	Lowest speed admissible Value range: > 0 Must be lower than <Speed2>
	<Speed2>	Highest admissible speed Value range: > 0 Must be higher than <Speed1>

Tab. 6-148: Syntax of *SMin*, *SMN*, *SMax* and *SMX*

The general syntax is as follows:

`SMin(S1=Value1 {,S2=Value2} ... )` or `SMax(S1=Value1 {,S2=Value2} ... )`.

Example:	<code>N50 X.. Y.. SMN (S1500)</code>	The spindle speed must be between 1500 and 2500 rpm.
	<code>N60 X.. Y.. SMX (S2500)</code>	
	:	
	<code>N90 X.. Y.. SMN (S0)</code> <code>SMX (S-1)</code>	Deactivate the lower and upper limits (=speed limitation OFF)

Special features and restrictions: Speed limitation has an effect only if the programmed speed limit values lie within the gear range limits.

6.118 Adjusting spindles in block pre-run init adjustment program "SpAdjSp"

Description:



For more detailed information on the "block pre-run" function, refer to the "MTX Functional Description" documentation.

Function "SPADJSP" generates the logic spindles of the channel when called in the first pre-run adjustment program.

Syntax:

`SpAdjSp`

Tab. 6-149: Syntax *SpAdjSp*

Example: *Excerpt from an InitAdjPrg*

```
; Produce channel spindles and spindle groups
; Also sends aux. functions CSPx to the PLC!
N100 SpAdjSp
;
```

- Special features and restrictions:**
- Function "SpAdjSp" can only be used in an Init adjustment program (see SD.SysSRun. NameOfInitAdjPrg).
 - If the channel-specific integer auxiliary functions CSP1 to CSP8 are defined, the PLC is informed about the current channel spindle assignment - as is the case with GetSpindle, RemSpindle and DefSpindle.

6.119 Determining the spindle syntax for the block pre-run adjustment program "SpAdjStr"

Description:

For more detailed information on the "block pre-run" function, refer to the "MTX Functional Description" documentation.

The function "SPADJSTR" <SpindleNumber> fills the following components of the "SD.SysChSpSRun[]" system data when being called in an adjustment program:

- SpeedStr
- MoveStr
- GearStr

After calling, the SpeedStr, MoveStr and GearStr variables contain the auxiliary function relevant for the programmed spindle at the end of the block pre-run.

Syntax:

SpAdjStr<spindle>

The components of the system data are filled with the valid string of the end of the block pre-run (e.g. S1=, M103, M141).

with

<Spindle>	Number of the (channel) spindle 1...8

Tab. 6-150: Syntax SpAdjStr

Example:

Excerpt from an InitAdjPrg

```
;
; Specify speed or cutting velocity
;
0000 DIM SADR$(8) :REM For spindle syntax (e.g., S1=)
;
0100 FOR SP% = 1 TO SD.SysChSpSRun.MaxSpNumber
0102 IF SD.SysChSpSRun.Sp[SP%].SpeedNotAdj <> 0 THEN
;
N104 SpAdjStr[SP%] ;Load the syntax elements of the current spindle
0106 SADR$ = SD.SysChSpSRun.SpeedStr
;
N108 [SADR$] [SD.SysChSpSRun.Sp[SP%].ProgSpeed]
0110 ENDIF
0112 NEXT SP%
;
```

Special features and restrictions:

- The SpAdjStr function can only be used in the adjustment program (see SD.SysSRun. NameOfInitAdjPrg und SD.SysSRun.NameOfAsup).
- When calling "SpAdjStr", the spindle configuration must have been adjusted (see "SpAdjSp").

6.120 Enabling/transferring reserved spindle "SpAdmin, SPA"

Description: *Permits*

- enabling of a spindle that is currently reserved in the current channel without stopping a turning spindle,
- followed by its transfer to any channel.



The function "Spindle Stop" is also used to enable a spindle (see [chapter 5.3.6 "Spindle stop M5, M105, M205" on page 184](#)).

Syntax:

SpAdmin (S<Num>=<Mode>{ , S<Num>=<Mode>}...)

Short form: SPA (...)

with

<Num>	Number of the spindle (spindle index) Input range: 1...8 Integer
<Mode>	0: enable spindle 1: reserve and transfer an enabled spindle

Tab. 6-151: Syntax of SpAdmin (SPA)

Example:

N60 SPA (S1=0, S2=0) Enable 1. and 2. spindle
:

Special features and restrictions: A control reset or M30 enables spindles reserved by the current channel only if the command for "Spindle Stop" is entered at the correct position in MP7060 00020.

6.121 Defining (activating) the coupling group, removing (deactivating) the coupling group"SpCoupleConfig, SPCC"



For more information, refer to the "MTX Functional Description - Basics" in the chapter "Position-synchronous spindle operation".

Description:

- Defines a coupling group and activates its spindle coupling.
The control automatically switches the spindle drives involved to the positioning interface.
- Adds slave spindles to an existing coupling group or removes them from an existing coupling group.
Added spindle drives are automatically switched to the positioning interface; removed spindle drives are switched to the speed interface (if the speed interface was active on the affected spindles before coupling).
- Specifies the direction of rotation for the slave axes to the coupled to the master axis.
- Deactivates the spindle coupling of a couple group and thus deactivates the complete coupling group.
All spindle drives involved are switched to the speed interface if the speed interface had been active on the particular spindles before coupling.

The coupling law for slave spindle positions is:

$$p_s = p_m * k + o_{dist} + o_{offs}$$

with

p_s	of the slave spindle command position
p_m	of the master axis command position
k	of the direction of rotation from slave spindle to master spindle (1,-1)
O_{dist}	of the preset coupling distance between master axis and slave axis (SPCD)
O_{offs}	of the additive dynamic offset to the coupling distance (SPCP)

Syntax:

Defines coupling group <code>SpCoupleConfig (CP=<Group>, MA=<Master>, S<Sequence>=1 -1 {{, S<Sequence>=1 -1}...})</code>	
Adds/removes slave spindles to and from the coupling group <code>SpCoupleConfig (CP=<Group>, S<Sequence>=<Mode> {{, S<Sequence>=<Mode>}...})</code>	
Resolves coupling group <code>SpCoupleConfig (CP=<Group>, MA=0)</code>	
Short form: SPCC (...)	
with	
<Coupling>	Number of the group of coupled spindles Input range: 1...4 Integer
<Master>	Number of the main spindle (spindle index) Input range: 1...8 Integer
<Consequence>	Number of the slave spindle (spindle index) Input range: 1...8 Integer
<Mode>	0: Remove slave spindle <link> 1: Add slave spindle in the direction of rotation of the master spindle to the <Coupling> -1: Add slave spindle in opposite direction of rotation of the master spindle to the <Coupling>

Tab. 6-152: Syntax of *SpCoupleConfig* (SPCC)

Example:

CoupleAndDecouple

```

N010 SPCD(S3=0)           ;coupling distance 0°
...
N210 SPCC(CP=1, MA=1, S3=1) ;activate coupling 1
N220 SPC_WAIT(CP=1)       ;wait for synchronous mode
...
N310 M103 S1=2000
...
```

```
...
N760 SPCC(CP=1, MA=0)           ;deactivate coupling 1
N770 SPCC_WAIT(CP=1)           ;wait for end of decoupling
...
```

6.122 Coupling distance of slave spindle "SpCoupleDist, SPCD"

Description: Configures the desired position lag between the master and slave spindles at the time of coupling (when the group of coupled spindles is created) for the "spindle coupling" function.

If the function is not used for a desired slave spindle, its position lag to the master spindle is 0 degrees (at the time of coupling).



To retract additional angle displacement between the master and the slave spindle during an active spindle coupling, see the function "SpCouplePosOffs, [chapter 6.124 "Angular offset with active coupling SpCouplePosOffs, SPCP"](#) on page 369.

Syntax:

SpCoupleDist (S<Num>=<Distance>{, S<Num>=<Distance>}...)	
Short form: SPCD (...)	
with	
<Num>	Number of the slave spindle (spindle index) Input range: 1...8 Integer
<Distance>	<i>Coupling distance of the slave spindle:</i> - Positions the lag between the master and the slave spindle (in degrees). Value range: $0^\circ \leq \text{Position difference} < 360^\circ$ If another value is programmed, it is automatically converted to the given interval. ACT for a coupling at the current position. This can avoid an unwanted compensation motion at stand-still of master and slave axis.

Tab. 6-153: Syntax of SpCoupleDist (SPCD)

Example Refer to "Defining coupling group (activate), resolving coupling group (deactivate)".

Special features and restrictions: The function may only be programmed with spindles that currently do not belong to a coupling group.

6.123 Synchronous mode error window "SpCoupleErrWin, SPCE"

Description: Configures the maximum permitted position deviation between the target and actual values of a slave spindle for the "spindle coupling" function.

If the position deviation lies within the defined interval during an active spindle coupling, the spindle-based output signal "**synchronous run 2**" is sent.

If the function is not used for a desired slave spindle, its synchronous mode error window is +/-10 degrees.



See also the function "SpCoupleSyncWin", [chapter 6.126 "Synchronous window SpCoupleSyncWin, SPCS"](#) on page 370.

Syntax:

SpCoupleErrWin(S<Num>=<Window>{, S<Num>=<Window>}...)	
Short form: SPCE (...)	
with	
<Num>	Number of the slave spindle (spindle index) Input range: 1...8 Integer
<Window>	Maximum permitted position deviation from the value (in degrees). Input range: 0 to 359.9999

Tab. 6-154: Syntax of SpCoupleErrWin (SPCE)

Example

Refer to the "Synchronization window "SpCoupleSyncWin, SPCS".

Special features and restrictions:

The function may only be programmed with spindles that currently do not belong to a coupling group.

6.124 Angular offset with active coupling "SpCouplePosOffs, SPCP"

Description:

Retracts an angular offset between master and slave spindle during an active spindle coupling. The relative speed between the master and the slave spindle at which the angular offset is to be retracted can be optionally programmed.



The control resets the spindle-specific output signal "synchronous run 1" for the duration of rotation.

Syntax:

SpCouplePosOffs(S<num>=<offset>{{, S<num>=<offset>}...} {, POSVEL<Speed>})	
Short form: SPCP (...)	
with	
<Num>	Number of the slave spindle (spindle index) Input range: 1...8 Integer
<Offset>	Absolute angle of rotation between the master and the slave spindle (in degrees). Value range: -3600°...+3600°
<Speed>	The relative speed between the master and the slave spindle at which the angular offset is to be retracted. The unit and the default value depend on drive parameter S-0-0222. Once entered, <Speed> remains stored internally until it is modified by calling the function again.

Tab. 6-155: Syntax of SpCouplePosOffs (SPCP)

Example:*AngularOffset*

```
N010 SPCD(S3=0)           ;coupling distance 0°
...
```

```

N210 SPCC(CP=1, MA=1, S3=1) ;activate coupling 1
N220 SPC_WAIT(CP=1)          ;wait for synchronous mode
...
...
N310 M103 S1=2000
...
...
N410 SPCP(POSVEL20, S3=195) ;rotate S3 in relation to S1
N420 SPCP_WAIT(CP=1)        ;wait for end of rotation
...

```

Special features and restrictions: The angular offset is added to the possibly configured coupling lag (see [chapter 6.122 "Coupling distance of slave spindle SpCoupleDist, SPCD" on page 368](#)).

6.125 Waiting for angular offset "SpCouplePosOffs_Wait, SPCP_WAIT"

Description: Pauses the part program until a programmed angular offset of a group of coupled spindles (SPCP; see [chapter 6.124 "Angular offset with active coupling SpCouplePosOffs, SPCP" on page 369](#)) has been actually retracted on the machine..

Syntax:

SpCouplePosOffs_Wait (CP=<Coupling>)	
Short form: SPCP_WAIT (...)	
with	
<Coupling>	Number of the group of coupled spindles Input range: 1...4 Integer

Tab. 6-156: Syntax of SpCouplePosOffs_Wait (SPCP_WAIT)

Example Refer to "Phase offset in case of active coupling "SpCouplePosOffs, SPCP"".

6.126 Synchronous window "SpCoupleSyncWin, SPCS"

Description: Configures the maximum permitted position deviation between the target and actual values of a slave spindle for the "spindle coupling" function:

- Effect when a coupling is made/changed (at the start of the synchronization phase):
The part program waits until the position deviation lies within the defined interval.
- Effect while the coupling is being made:
If the position deviation lies within the defined interval, the spindle-based output signal "**Synchronous run 1**" is sent.

If the function is not used for a desired slave spindle, its synchronization window is +/-1 degrees.



For additional monitoring while coupling is active, see the function "SpCoupleErrWin, "[chapter 6.123 "Synchronous mode error window SpCoupleErrWin, SPCE" on page 368](#)".

Syntax:

SpCoupleSyncWin (S<Num>=<Window>{, S<Num>=<Window>}...)	
Short form: SPCS (...)	
with	

<Num>	Number of the slave spindle (spindle index) Input range: 1...8 Integer
<Window>	Maximum permitted position deviation from the value (in degrees). Input range: 0 to 20

Tab. 6-157: Syntax of *SpCoupleSyncWin* (SPCS)

Example: *ConfigureAndCouple*

```
;Configure slave spindle
N010 SPCD(S3=0)           ;coupling distance 0°
N020 SPCS(S3=0.5)         ;synchronous window 0.5°
N030 SPCE(S3=3.0)         ;error window 3°
...
N210 SPCC(CP=1, MA=1, S3=1) ;activate coupling
N220 SPC_WAIT(CP=1)       ;wait for synchronous mode
...
...
```

Special features and restrictions: The function may only be programmed with spindles that currently do not belong to a coupling group.

6.127 Waiting for synchronous mode "SpCouple_Wait, SPC_WAIT"

Description: The part program waits until a coupling group is successfully created, reconfigured or removed (SPCC; see [chapter 6.121 "Defining \(activating\) the coupling group, removing \(deactivating\) the coupling groupSpCoupleConfig, SPCC" on page 366](#)).

Syntax:

SpCouple_Wait (CP=<Coupling>)	
Short form: SPC_WAIT (...)	
with	
<Coupling>	Number of the group of coupled spindles Input range: 1...4 Integer

Tab. 6-158: Syntax of *SpCouple_Wait* (SPC_WAIT)

Example Refer to "Defining coupling group (activate), resolving coupling group (deactivate)".

6.128 Defining/removing spindle groups "SPG.., SPGALL"

Description: If the "channel spindles" function is activated (MP 1040 00999 =1), the spindle groups are automatically preset during control startup.

The first channel spindle is a member of the first spindle group.

Spindle groups (also called parallel spindles) permit simplified programming of several spindles combined with the functions for "gear ranges", "clockwise/counterclockwise rotation", "spindle stop" and "spindle orientation". Thus, the individual spindles does not have to be programmed separately.



The function "spindle group" **must not** be confused with a "coupling group" consisting of two or more position-controlled spindles.

Whereas all the spindles in a spindle group may be running at different speeds, all the spindles in a group of coupled spindles are always running synchronously.

Syntax:

SPG <Group>(<Numbers>)	Specifies which channel spindles are to be combined as spindle group in the current channel.
SPG <Group>(0)	Resets the group in the current channel to the default setting.
SPG <Group>(-1)	Cancels the corresponding spindle group in the current channel.
SPGALL (0)	Resets all groups to default setting in the current channel.
with	
<Group>	Number of the spindle group Input range: 1 to 4 Integer
<Number>	Numbers (separated by commas) of all the channel spindles that are to be assigned to the corresponding spindle group. Input range: 1...8
SPG(<numbers>) is a short form of SPG1 (<numbers>)	

Tab. 6-159: Syntax of SPG.. (SPGALL)

Example:

GSP (SSP01, S1, SSP07, S2)	The channel spindles 1 and 2 are assigned to the system spindles 1 and 7.
SPG1 (1, 2)	Spindle group 1 consists of S1 and S2.
M3	Starts S1 and S2 (SSP01 and SSP07) with clockwise rotation.
GSP (SSP14, S2)	System spindle 14 is assigned to spindle 2.
M4	Start S1 and S2 (SSP01 and SSP14) with counter-clockwise rotation.
RSP (S1)	No system spindle is assigned to channel spindle 1(S1).
M3	Start S2 (SSP14) with clockwise rotation.

Special features and restrictions:

- At a spindle transfer (GSP, RSP, DSP), the assignment of channel spindle to system spindle changes. This assignment change directly affects the spindle group.
- Competing jobs between single spindles and spindle groups that are programmed in the same block lead to a runtime error (example: N10 M3 M104).

6.129 Writing into a system data queue "SSDQ"

Description: The "SSDQ" function writes a programmed message (an integer) at the **execution time** in a queue in the system data (an "Int_t" system data array). The queue is used as ring buffer. If writing is done to the last element of the sys-

tem data array, the function enters the programmed message in the first element of the queue when called the next time. In all other cases, it is written to the subsequent element.

In order for the user to learn which queue elements have been written, the "SSDQ" function simultaneously modifies a control structure. This control structure is also located in the system data and consists of the two "Int_t" elements **LastFilled** and **LastGet**.

If the function wrote a programmed message into the queue, the **LastFilled** element is set to the index of the modified entry in the queue. This index is one-based.

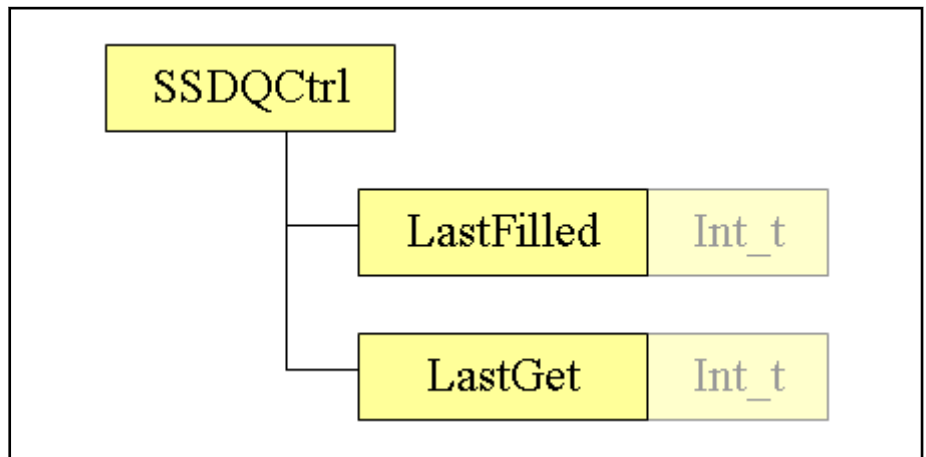


Fig. 6-36: Structure of the SSDQ control structure

The programmer can specify both, the system data queue and the control structure used. For this purpose, they specify the corresponding structures as XPath. Both system data structures can be both, root elements and part of a larger structure.



It is also permitted to create the structures in a channel-dependent form. If in such constellation, the channel index is omitted, the function automatically uses the substructure of the current channel (in which the part program is being executed).

If the system data structures are not specified explicitly, the function tries to use "/SSDQueue" as queue and "/SSDQCtrl" as control structure.

In case of a writing process, a modification even is only triggered for the control structure. The queue is modified, but a change event is, however, not triggered.

After the "SSDQ" function has been executed, the user is responsible for reading the message from the queue. The **LastGet** element in the control structure has to be set to the index of the latest queue element read.

If the user reads the control structure and the values of **LastFilled** and **LastGet** are not identical, new messages have arrived in the queue. If **LastFilled** is larger than **LastGet**, the new messages are positioned in the entries of the queue from **LastGet+1** to **LastFilled** (in the order programmed). If **LastFilled** is smaller than **LastGet**, the ring buffer behavior of the queue is applied. In this case, the new messages are located in **LastGet+1** up to the end of the queue as well as from the beginning of the queue to **LastFilled** (in the order programmed).

If the user does not succeed in setting **LastGet** in due time, there is the risk of the queue being full and messages being lost. For this reason, the "SSDQ"

NC functions with high-level language syntax

function checks whether there are still free elements in the queue (next writing position unequal **LastGet**). If this is not the case, a runtime error is created.

With the "SSDQInit" function, it is possible to initialize a control structure. When programming "SSDQInit", **LastGet** and **LastFilled** are set to 1 at the execution time.

Syntax:

SSDQ (M <Message> {, Q <XPath>} {, C <XPath>})	
with	
M <message>	Integer, that is to be written in the queue.
Q <XPath>	XPath to the system data or the substructure that is used as queue. Default: "/SSDQueue"
C <XPath>	XPath to the system data or the substructure that is used as control structure. Default: "/SSDQCtrl"
SSDQInit ({C <XPath> })	
C <XPath>	XPath to the system data or the substructure that is to be initialized as control structure Default: "/SSDQCtrl"

Tab. 6-160: Syntax SSDQ and SSDQInit

Example:

for SSDQ and SSDQInit

N10 SSDQ (M32)	Writing the message "32" into the queue "/SSDQueue" with the control structure "/SSDQCtrl"
N20 SSDQ (M=17, Q / myQueue)	Writing the message "17" into the queue "/myQueue" with the control structure "/SSDQCtrl"
N30 SSDQ (M 42, C=/ myCtrl)	Writing the message "42" into the queue "/SSDQueue" with the control structure "/myCtrl"
40 ST\$="/Msg[1,,3]/ Queue"	Writing the message in I% into the queue "/Msg[1,CHANNELNO,3]/Queue" with the control structure "/SSDQCtrl"
N50 SSDQ (M[I%], Q=[ST\$])	If this program is, for example, executed in chan- nel 5, the function writes into the queue "/Msg[1,5,3]/Queue"
N60 SSDQInit	Initializing the control structure "/SSDQCtrl" with LastFilled=1 and LastGet=1
N70 SSDQInit (C=/ Msg[1,,2]/Ctrl	Initializing the control structure "/Msg[1,CHANNELNO,2]/Ctrl" with LastFilled=1 and LastGet=1 If this block is, for example, executed in channel 4, the function writes into the control structure "/Msg[1,4,3]/Ctrl"

Special features and restrictions:

- The queue has to comprise at least three elements.

- The size of the message queue depends on the free memory space of the target.
- If the execution of the programmed motion of a block with SSDQ requires less than one IPO cycle, there is a "jerk".
- As the communication via NCS to Windows is only low priority, there may be delays when reading messages.

6.130 System spindle programming "SSp....."

6.130.1 General information

The following functions allow for the activation of system spindles. These functions supplement the traditional spindle activation through auxiliary functions.

The abbreviation **SSp** stands for **SystemSpindle**.

6.130.2 Enabling/applying the reserved spindle "SSpAdm"

Description: Releases one or several system spindles from the current channel that have been reserved by motion or orientation settings. Enabled spindles may be transferred to the current channel.

This function only exists for reasons of symmetry. We recommend to use the "SpAdmin" channel function.

Syntax:

SSpAdm, SSPADM	
SSpAdm (<SSpNo.> <SSpName>, <Mode>{ , <SSpNo.> <SSpName>, <Mode>} ...)	
with	
<SSpNo>	1..32
<SSpName>	System spindle name
<Mode>	1 transfers a released spindle 0 conditionally enables a spindle

Tab. 6-161: Syntax of SSpAdm

Example:

SSpAdm(SSP05, 0, SSP08, 1) Enables system spindle 5, and transfers the enabled system spindle 8

6.130.3 Disabling management of the spindle motion for specific channels "SSpAdmOff"

Description: Disable the management of spindle motions of a system spindle for specific channels. This function has the same effect as machine parameter 1040 00070 "Spindle management".

Syntax:

SSpAdmOff, SSPADMOff	
SSpAdmOff (<SSpNo.> <SSpName>, <ChannelNo> {, <ChannelNo> } ...)	
SSpAdmOff (<SSpNo.> <SSpName>)	
with	
<SSpNo>	1..32

<SSpName>	System spindle name
<ChannelNo>	1 .. 60: channel numbers 0: Enabled for tasks of the bit interface

Tab. 6-162: Syntax of SSpAdmOff

Example:	<code>SSpAdmOff (SSP05, 1, 2, 4)</code>	Disables the management of motions of system spindle 5 for the channels 1, 2 and 4
	<code>SSpAdmOff (5)</code>	Resets the management of motions of system spindle 5 to the state according to MP 1040 00070

- Special features and restrictions:**
- In deviation from the schema of the other "SSp" functions, this function can only affect one system spindle at a time.
 - A specification by channel 0 disables the motion management of this system spindle for specifications by the bit interface.

6.130.4 Changing the gear range "SSpGear"

Description: Changes the gear range of one or several system spindles.

Syntax:

<code>SSpGear, SSPGEAR</code>	
<code>SSpGear (<SSpNo.> <SSpName>, <Level>{ , <SSpNo.> <SSpName>, <Level>} ...)</code>	
with	
<SSpNo>	1..32
<SSpName>	System spindle name
<Stage>	1..4 for the gear stages 1 to 4 8 for idling

Tab. 6-163: Syntax of SSpGear

Example:	<code>SSpGear (SSP05, 4)</code>	Loads the fourth gear stage for system spindle 5

- Special features and restrictions:**
- The NC waits for execution of the gear range change.
 - Automatic gear selection is a channel property. For this reason, "SSpGear" cannot be used to switch between automatic and manual mode.
 - Automatic gear selection, if active in a channel, is not affected by "SSpGear". This is to say that the next programmed S-word may execute a gear shift if "M40" is active.
 - Gear switching is only supported for Sercos spindles.

6.130.5 Limiting the maximum speed "SSpMax"

Description: Limits the upper maximum speed of one or several system spindles.

Syntax:

<code>SSpMax, SSPMAX</code>	
<code>SSpMax (<SSpNo.> <SSpName>, <Speed>{ , <SSpNo.> <SSpName>, <Speed>} ...)</code>	
with	
<SSpNo>	1..32

<SSpName>	System spindle name
<Speed>	> 0 upper speed limit in rpm -1 disables speed limitation

Tab. 6-164: Syntax of SSpMax

Example:	<code>SSpMax (SSp05, 2000, SSp08, 3000)</code>	Sets the upper speed limit of system spindle 5 to 2000, and of system spindle 8 to 3000 rpm.
	<code>SSpMax (SSp05, -1, SSp08, -1)</code>	Cancels the upper speed limit of the system spindles 5 and 8.

6.130.6 Limiting the minimum speed "SSpMin"

Description: Limits the lower speed limit of one or several system spindles.

Syntax:

<code>SSpMin, SSPMIN</code>	
<code>SSpMin (<SSpNo.> <SSpName>, <speed>{ , <SSpNo.> <SSpName>, <speed>} ...)</code>	
with	
<SSpNo>	1..32
<SSpName>	System spindle name

Tab. 6-165: Syntax of SSpMin (SSPMIN)

Example:	<code>SSpMin (5, 100, 8, 500)</code>	Sets the minimum speed of system spindle 5 to 100, and of system spindle 8 to 500 rpm.
	<code>SSpMin (5, 0, 8, 0)</code>	Cancels the lower speed limit of the system spindles 5 and 8.

6.130.7 Switching between position/speed interface "SSpMode"

Description: Switches the drive of the programmed system spindles between speed and position mode.

Syntax:

<code>SSpMode, SSPMODE</code>	
<code>SSpMode (<SSpNo.> <SSpName>, <Mode> { , <SSpNo.> <SSpName>, <Mode>} ...)</code>	
with	
<SSpNo>	1..32
<SSpName>	System spindle name
<Mode>	1: activates positioning interface 0: deactivates positioning interface

Tab. 6-166: Syntax of SSpMode

Example:	<code>SSpMode (5, 1, 8, 0)</code>	Enables the position interface for system spindle 5 and disables it for system spindle 8

Special features and restrictions: The positioning interface is always activated when the spindle is at standstill.

6.130.8 Motion programming "SSpMove (SSPM)"

Description: Starts or stops one or several system spindles.

Syntax:

SSpMove, SSPMOVE, SSPM	
SSpMove (<SSpNo.> <SSpName>, <Job>{ , <SSpNo.> <SSpName>, <Job>} ...)	
with	
<SSpNo>	1..32
<SSpName>	System spindle name
<job>	3 - clockwise spindle rotation 4 - counter-clockwise spindle rotation 5 - spindle stop

Tab. 6-167: Syntax of SSpMove (SSPM)

Example:

SSpMove (SSP05, 3, SSP08, 5) Starts system spindle 5 in clockwise rotation, and stops system spindle 8.

6.130.9 Spindle orientation "SSpOri (SSPO)"

Description: The programmed system spindles move to the programmed angular position.

Syntax:

SSpOri, SSPORI, SSPO	
SSpOri (<SSpNo.> <SSpName>, <angle>{ , <SSpNo.> <SSpName>, <angle>} ...)	
with	
<SSpNo>	1..32
<SSpName>	System spindle name
<Angle>	0.0 .. 359.9999

Tab. 6-168: Syntax of SSpOri (SSPO)

Example:

SSpOri (5, 20, 8, 0) Orients system spindle 5 to 20 degrees and system spindle 8 to 0 degrees.

Special features and restrictions:

- The NC waits for execution of the orientation motion.
- Contrary to the orientation motion via M19, M119, ... the angle always has to be programmed.
- If the input of a traversing motion has been programmed via SSpMove in the same NC block, SSpMove is ignored (no error message).
- Set the interface signal "qSp_SValueSD" and the position is assigned via the system data "SysSpCmdData[n]/OriPos". The programmed specification is ignored.

6.130.10 Speed programming "SSpSpeed (SSPS)"

Description: Speed specification for one or more system spindles.

Syntax:

SSpSpeed, SSPSPEED, SSPS	
SSpSpeed (<SSpNo.> <SSpName>, <Speed>{ , <SSpNo.> <SSpName>, <Speed>} ...)	
with	

<SSpNo>	1..32
<SSpName>	System spindle name

Tab. 6-169: Syntax of SSpSpeed (SSPS)

Example: `SSpSpeed(SSp05, 1000, SSp08, 2000)` Sets the speed of system spindle 5 to 1000, and of system spindle 8 to 2000 rpm.

- Special features and restrictions:**
- By default, the programmed speed values are interpreted as rpm.
 - When G96 is active, the programmed speed is interpreted as cutting velocity in m/min.
 - The control always limits the output speed command values in such a manner that the limit values of the active gear range and the programmed speed limit (see SMin, SMax, SSpMin and SSpMax) are observed.
 - The set speed value is applied until it is overwritten by a new speed (acting modally).
 - By setting the interface signal "qSp_SValueSD", the speed is specified via the system data "SysSpCmdData[n]/Speed". The programmed specification is ignored.

6.130.11 Tool-specific spindle limitations"SSpToolLim"

Description: Enables the entered tool-specific spindle limitations in one or several system spindles in the `/SysSpCmdData[<SysSp>]/Tool` system data.



For more information, refer to the "MTX Functional Description Basics" in the chapter "Spindle functions" under "Tool-specific spindle limitations".

Syntax:

<code>SSpToolLim, SSPTOOLLIM</code>	
<code>SSpToolLim(<SSpNo.> <SSpName>, <0 1>{ , <SSpNo.> <SSpName>, <0 1> } ...)</code>	
with	
<SSpNo>	System spindle number
<SSpName>	System spindle name
<0 1>	1 enables the tool-specific limitations of <code>/SysSpCmdData[]/Tool</code> 0 disables the tool-specific limitations

Tab. 6-170: Syntax SSpToolLim

Example: `SSpToolLim(3,1)` enables the tool-specific limitations of the third system spindle.

- Special features and restrictions:**
- SSpToolLim stops the programmed spindles and subsequently restarts them.
 - SSpToolLim reports a gear shifting to the "SpindleWait, SPWAIT" and "SpindleWaitMode, SWM" functions.
 - SSpToolLim comprises an NC-internal WAIT functionality.

6.131 Additional function to orient/position spindles "SpindleHome, SPHOME"

Description: SPHOME can be programmed in one line together with the M19/M119/SSpOr.

SPHOME forces to redetermine the reference point for all programmed spindles.

Syntax:

SpindleHome	
Short form: SPHOME	
	Forces to redetermine the reference point for all programmed spindles.

Tab. 6-171: SpindleHome syntax (SPHOME)

Example: Switching the spindle/turret axis to the turret mode

:	
N020 M219 SPHOME	Orienting milling spindle by redetermining the reference point
N030 STAW (REV)	Switch on NC-side from spindle to turret mode and wait till the C-axis becomes active.
N040 Mxxx	Instruct the PLC to perform the mechanical switch to turret mode.
N050 WAX (REV)	Integrate the turret axis REV into the active channel.
:	

- Special features and restrictions:**
- SPHOME is only applied to the currently speed-controlled spindles.
 - SPHOME is only applied if the initial motion state of the spindle is not "Oriented".

6.132 Enabling C-axis mode for spindles "SpindleToAxis, STA"

Description: *Switches a spindle entered in the following machine parameters to C-axis mode:*

- as a spindle/C-axis in MP 1001 00001 (drive function type)
- as a Sercos spindle in MP 1040 00001 (selection of spindle type)

In terms of machining, this turns a spindle into an asynchronous axis.

It is displayed in the display as an asynchronous axis that is initially located on any position between 0 and 359.9999 degrees.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

SpindleToAxis (<PAN> <PAI>{ , <PAN> <PAI>}...)
Short form: STA (. .)
with

<PAN>	Physical axis name Specifies the spindle that is to be switched to C-axis mode.
<PAI>	Physical axis index Same effect as <PAN>

Tab. 6-172: Syntax of SpindleToAxis (STA)

Example:

```

:
N030      The physical axis "CH" (i.e. the spindle with the name "CH" during axis
STA (CH)  operation) is switched to an asynchronous axis.
:

```

Special features and restrictions: Axis positions in the same block must always be programmed according to the expression "SpindleToAxis(...)".

6.133 Enabling C-axis mode for spindles and wait "SpindleToAxisWait, STAW"

Description: Switches a spindle entered in the following machine parameters to C-axis mode and waits until the C-axis is active:

- as a spindle/C-axis in MP 1001 00001 (drive function type)
- as a Sercos spindle in MP 1040 00001 (selection of spindle type)

In terms of machining, this turns a spindle into an asynchronous axis.

It is displayed in the display as an asynchronous axis that is initially located on any position between 0 and 359.9999 degrees.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

SpindleToAxisWait (<PAN> <PAI>{ , <PAN> <PAI>}...)	
Short form: STAW (. .)	
with	
<PAN>	Physical axis name Specifies the spindle that is to be switched to C-axis mode.
<PAI>	Physical axis index Same effect as <PAN>

Tab. 6-173: Syntax SpindleToAxisWait (STAW)

Example:

```

:
N030 STAW (CH)  The physical axis "CH" (i.e. the spindle with the name "CH"
                  during axis operation) is switched to an asynchronous axis.
N040 CH90      The asynchronous axis CH is positioned to 90 degrees.
:

```

Special features and restrictions: Axis positions in the same block must always be programmed according to the expression "SpindleToAxisWait(...)".

6.134 Waiting for spindle programming at end of block "SpindleWait, SPWAIT"

Description: It waits at the end of each block till the respective spindle reached the desired state.



For more information, refer to the documentation "MTX Functional Description" in the chapter "Spindle functions" under "Path motion and spindle functions".

Syntax:

<code>SpindleWait (<Spindle> { , <Condition> })</code>	
Short form: <code>SPWAIT (. .)</code>	
with	
<Spindle>	It specifies the channel spindle(s) that should be waited for. -2: All spindles except the main spindle -1 or MSP: Main spindle 0 or ALL: all spindles 1 .. 8 or S1... S8: Number or name of a spindle
<Condition>	Optional. List of the waiting conditions with a maximum content of 3, 5, 19, 40 3: Wait for "Speed reached" at M3 or M4 5: Wait for spindle stop at M5 19: Wait for spindle oriented at M19 40: Wait for the end of a gear switching Default: For non-programmed conditions, it is waited for all conditions.
<code>SpindleWait</code>	
	<code>SpindleWait</code> without parameter list waits for the spindle functions programmed in this block to reach.

Tab. 6-174: SpindleWait (SPWAIT) syntax

Example:

```

:
N30 M3 S1000 SPWAIT           It waits at block end till notification "Speed
                                reached".
:
N260 M203 S2=500 G0 X1000     Spindle 2 is started.
N270 G0 Y10 SPWAIT(S2,3)     It waits at block end till spindle 2 notifies
                                "Speed reached".
N280 G1 ...
:

```

6.135 Selecting a waiting strategy "SpindleWaitMode, SWM"

Description: It selects and activates a waiting strategy for the function "Path Motion and Spindle Functions".



For more detailed information, refer to the documentation "MTX Functional Description" in chapter "Spindle functions" under "Path motion and spindle functions" and to the documentation "IndraMotion MTX Machine Parameters" in chapter "Spindles".

Syntax:

<code>SpindleWaitMode (<Spindle> , <Switch on> {, <Strategy>)</code>	
Short form: <code>SWM (..)</code>	
with	
<Spindle>	It specifies the channel spindle(s) for which the strategy should be changed. -2: All spindles except the main spindle -1 or MSP: Main spindle 0 or ALL: All spindles 1 .. 8 or S1... S8: Number or name of a spindle
<Switch on>	0: Disable waiting strategy 1: Enable waiting strategy
<Strategy>	Optional. 1 .. 3: Strategy number 1: Predefined strategy 2 and 3: SP/SpAdmin/BlkSwiCtrl

Tab. 6-175: *SpindleWaitMode (SWM) syntax*

Example:

```

:
N010 SpindleWaitMode (MSP,      Strategy 2 is selected and activated for the
1, 2)                      main spindle.
:
N150 SpindleWaitMode (MSP, 0)  The current waiting strategy (2) of the main
:                               spindle is disabled.
N190 SpindleWaitMode (MSP, 1)  The current waiting strategy (2) of the main
:                               spindle is enabled.

```

6.136 Rounding corners with splines

6.136.1 Rounding corners with splines (modal) "SplineCornering, SCO"

Description: The modal NC function "round corners with splines" rounds all contour transitions (with a spline) whose knee angles are equal or bigger than a configured minimum knee angle.

Syntax:

<code>SplineCornering (E<lag>)</code>	
	"Corner rounding on" with specification of a maximum permissible tolerance E.
<code>SplineCornering (DEF)</code>	"Corner rounding on" with tolerance E of MP 7050 00110.
<code>SplineCornering (L1=<L1>, L2=<L2>)</code>	

NC functions with high-level language syntax

	"Corner rounding on" with specification of a pre- and a post-corner distance L1 or L2.
	SplineCornering(L<L>) "Corner rounding ON" with symmetrical pre-corner and post-corner distance $L_1 = L_2 = L$ specification SplineCornering(L<L>) "Corner rounding ON" with symmetrical pre-corner and post-corner distance $L_1 = L_2 = L$ specification SplineCornering(L<L>,E<distance>) "Corner rounding on" with specification of tolerance E and simultaneous pre- and a post-corner distance on L
SplineCornering(0)	"Corner rounding off"
Short form: SCO	

Tab. 6-176: Syntax of SplineCornering (SCO)

- Special features and restrictions:**
- The corners of any contour type are rounded in all coordinates (max. 8) of the channel.
 - The modal NC function "corner rounding with splines" can be overridden locally, that means, for the transition to the next block, by the local (non-modal) NC function "Cornering with splines".
 - The same machine parameters apply as for circular rounding of corners.

6.136.2 Rounding corners with splines (local) "SplineCorneringLocal, SCOL"

Description: The local (non-modal) NC function "Rounding corners with splines" rounds the transition to the following NC block with a spline if it is an non-tangential contour transitions .

Syntax:

SplineCorneringLocal (E<Distance>)	
	Corner rounding with specification of a maximum permissible tolerance E.
SplineCornering(DEF)	Corner rounding with tolerance E of MP 7050 00110.
SplineCornering (L1=<L1>, L2=<L2>)	
	Corner rounding with specification of a pre- and a post-corner distance L1 or L2.
SplineCornering(0)	No corner rounding for the transition to the following NC block.
Short form: SCOL	

Tab. 6-177: Syntax of SplineCornering (SCOL)

- Special features and restrictions:**
- The corners of any contour type are rounded in all coordinates (max. 8) of the channel.
 - The modal NC function "corner rounding with splines" can be overridden locally (that means for the transition to the next block) by the local (non-modal) NC function "cornering with splines".
 - The same machine parameters apply as for circular rounding of corners.

6.137 Defining the spline type "SplineDef, SPD"

Description: *The MTX CNC system supports the following spline types:*

- **Spline type 0:**
Spline with Coefficient Programming
(polynomial coefficients of the CAD/CAM system)
- **Spline type 1:**
C1-constant cubic splines with programming of support points and tangent specification if required (Hermite splines)
(tangential transitions at the data points)
- **Spline type 2:**
C2-continuous cubic splines with programming of support points
(constant curvature transitions at the data points)
- **Spline type 3:**
B-spline with checkpoint programming
(curve characteristics near the data points).
- **Spline type 4:**
B-spline approximation
(Specified tolerance margin around the data points).

The desired spline type is selected and initialized using the function "SplineDef" (SPD). Then, spline programming can be activated using the function "G6".

Syntax:

SplineDef (<Id>, {<Members>})	
	Initialize spline type
Short form: SPD (. .)	
with	
<Id>	Integer number with a maximum of 4 digits for the spline variant

Tab. 6-178: Syntax SplineDef (SPD)

Spline type	Position			
	1000s	100s	10s	1s
	Spline type 0...4	Parameterization 0: None 1: Equidistant 2: Chordal 3: Centripetal	Tangent calculation or specification 0: None 1: Bessel 2: Akima 3: False position 4 = Hermite	Spline degree 1...5
Spline type 0	0	0	0	1...5
Spline type 1	1	1...3	0...4	1...5
Spline type 2	2	1...3	0	3

Spline type	Position			
	1000s	100s	10s	1s
	Spline type 0...4	Parameterization 0: None 1: Equidistant 2: Chordal 3: Centripetal	Tangent calculation or specification 0: None 1: Bessel 2: Akima 3: False position 4 = Hermite	Spline degree 1...5
Spline type 3	3	1...3	0	1...5
Spline type 4	4	2	3	3

Tab. 6-179: Overview of the spline types

<Members>	Not applicable to spline type 0 and Hermite spline! List of names of the coordinates/axes to be involved in the spline motion. Non-programmed coordinates/axes are linearly taken along in <Members>. Orientation coordinates can also be programmed as splines. To achieve this, specify "O" for orientation, or the polar coordinates "phi" and "theta", under <Members>. If no coordinates are programmed in <Members>, the following activation values apply: In case of an active 5-axis transformation: x,y,z,O Others: All channel members

Tab. 6-180: Syntax of SplineDef (SPD) continued

Example:

N10 SplineDef(5)	Initializes spline type 0 with spline degree 5
N15 SplineDef(1143)	Initializes spline type 1 Hermite spline with spline degree 3
N20 SplineDef(2203,X,Y,Z)	Initializes spline type 2 with spline degree 3 and chordal parameters. Coordinates X, Y and Z are involved.
N30 Spline-Def(3103,x,y,z,O)	Initializes spline type 3 with spline degree 3 and equidistant parameters. Coordinates x, y and z, as well as the orientation coordinates, are involved.
N30 SplineDef(4233)	Initializes spline type 4 with spline degree 3. All channel coordinates are involved.

6.138 Programmable path split "Split, SPT"

Description: Splits the programmed motion blocks in several path segments if they exceed a certain length.

Syntax:	Split ({<Mode>}{, <part length>})
	Short form: SPT (...)
	with

<Mode>	0: Function has no effect 1: Function acts modally 2: Function acts only in the programmed block Not programmed: Effect as for 0
<Partial length>	For linear blocks: Length of path segments For circular blocks: Arc length The same programming unit as for the axis coordinates The following applies if <part length> has not been programmed: The active <part length> roughly corresponds to the maximum path that can be traveled in 2 subsequent interpolation cycles, depending on the current feed.

Tab. 6-181: Split syntax

Example:

```

:
N20 Split(2,10) X100    Effective path split block by block for N20 ON
:                      The traversing motion in N20 is split into partial paths, each with a length of 10 mm.
N120 Split(1)          Modally effective path split starting at N120 ON
:                      Traversing motions from N120 are split into path segments that can be traveled in approx. 2
:                      interpolation cycles.
N220 Split(1,3.0)      Modally effective path split starting at N220 ON
:                      Traversing motions starting at N220 are split into path segments, each with a length of 3
:                      mm.
N320 Split()           Programmable path split OFF
:

```

- Special features and restrictions:**
- The programmable path segmentation divides a traversing block only if its length exceeds <part length>.
 - If <part length> is not a whole divisor of the travel path, the distance-to-go is also traveled separately.
 - If the programmable split is active on a block-to-block basis (<Mode>=2), a traversing motion must be programmed in the same block.

6.139 Spindle: Switching between position and speed interface "SpMode, SPM"

Description: Switches the spindle drive between speed and position mode.

Syntax:

SpMode (S<Num>=<Mode>{, S<Num>=<Mode>}...)
Short form: SPM (...)
with

<Num>	Number of the spindle (spindle index) Input range: 1...8 Integer
<Mode>	0: Speed mode 1: Position mode

Tab. 6-182: Syntax of SpMode (SPM)

Example: N60 SPM(S1=1,S2=1) Switch 1st and 2nd spindle to position mode
:

- Special features and restrictions:**
- Spindle alignment is stopped.
 - A rotating spindle is briefly stopped when a switch to position mode is made.

6.140 Parameterizing the static tool orientation "StatToolOri, STO"

Description: Using G47(...) or G78, tool length corrections L_1 , L_2 and L_3 are assigned to the L_1 , L_2 , and L_3 coordinate directions. These coordinate directions form a coordinate system in which tool length correction vector L has components L_1 , L_2 and L_3 .

Command "StatToolOri" is used to rotate L about the Euler angles Φ (φ), Θ (ϑ) and Ψ (ψ) in this coordinate system:

- first about the L_3 coordinate direction by the angle **Phi**,
- then around the angle **Theta** around the L_2 coordinate direction **rotated during the previous rotation**
- then around the angle **Psi** around the coordinate direction **rotated during the previous rotation** L_3 .

The following rotations of L have the same result:

- first about the L_3 coordinate direction by the angle **Psi**,
- then around the angle **Theta** around the **original, not rotated** L_2 coordinate direction
- then around the angle **Phi** around the **original, not rotated** L_3 coordinate direction.

As both approaches can lead to the same result, use the approach that appeals more to you.

The static tool orientation always refers to the reversed tool correction vector. Consecutive, static tool orientations thus are not additive.

Syntax:

StatToolOri ({< φ >} {,< ϑ >} {,< ψ >})
Short form: STO (. .)

with	
<φ>,<θ>,<ψ>	Euler angle in degree Value range: 0° ≤ φ < 360°; 0° ≤ θ ≤ 180°; 0° ≤ ψ ≤ 360°. Values outside this range are automatically converted to the respective interval.

Tab. 6-183: Syntax of StatToolOri (STO)

Rotation of the tool correction vector around the angle Alpha around the L₁ coordinate direction:

L₁ coordinate direction:

```
StatToolOri(-90,<Alpha>,90)
```

Example:

```
1 DCT(1,1,0) = 100
1 DCT(2,1,0) = 0
1 DCT(3,1,0) = 0
; Components of the tool correction vector
G0 X0 Y0 Z0 ; (L1,L2,L3)
G47(X,Y,Z) ED1 ; (100,0,0)
STO(90,0,0) ; (0,100,0)
STO(0,0,0) ; (100,0,0)
STO(0,90,0) ; (0,0,-100)
G48 ED0
M30
```

Example:

```
1 DCT(1,1,0) = 0
1 DCT(2,1,0) = 0
1 DCT(3,1,0) = 100
; Components of the tool correction vector
G0 X0 Y0 Z0 ; (L1,L2,L3)
G47(X,Y,Z) ED1 ; (0,0,100)
STO(0,90,0) ; (100,0,0)
STO(-90,45,90) ; ( 0 , -50*sqrt(2) , 50*sqrt(2) )
STO(0,30,0) ; ( 50 , 0 , 50*sqrt(3) )
STO(45,30,0) ; (25*sqrt(2),25*sqrt(2),50*sqrt(3))
G48 ED0
M30
```

Special features and restrictions:

- The parameters of static tool orientation are set using "STO" and activated by "G47" (see [chapter 5.2.19 "Tool length correction G47, G48" on page 139](#)).
- If "G47" is already active when STO is executed, the change to the tool orientation is effective immediately.

- If coordinate designations were not assigned to all three tool length corrections L_1 , L_2 and L_3 using G47(...) or G78, the control takes only the assigned coordinates during tool orientation and generates a warning.



Also see "STP" in the "MTX Standard NC Cycles" documentation.

6.141 Suppressing single step modes (on/off)"StepModeDisable, SMD"/"StepModeEnable, SME"

Description: The Single Step suppression is used to completely process parts of an NC program in one of the operation modes "Single Step", "Single Block", "Program Block" and "Debug Step" and in one step. That means that the blocks are processed without confirming the NC start.

It is possible to completely encapsulate (asynchronous) subroutines and cycles in a way that the complete (asynchronous) subroutine or cycle is processed with one NC start even though it was not planned in the current operation mode.

The Single Step mode suppression is not a change of the operation mode. That means that the geometry calculation and the velocity calculation are carried out in the operation mode selected (e.g. Single Block). However, waiting on the NC start signal is suppressed.

It is possible to control the effect of the function separately for the operation modes Single Block/Single Step, Program Block and Debug Block. The NC start suppression can for example be enabled for Single Block/Single Step and Program Block while the NC start remains unsuppressed in the "Debug Step" mode.

Programming:

The suppression of the Single Block is programmed as modally effective NC function. There is a function for enabling and disabling.

Syntax:

StepModeDisable(({Variable={}{Value}}{},{Variable={}{Value}}))	
Short form: SMD { (. .) }	
with the optionally programmable variables	
SBLK	The following applies to the operation modes "Single Block" and "Single Step": 0 - NC start is not suppressed 1 - NC start is suppressed Default setting: 1
PBLK	It applies to the operation mode "Program Block" 0 - NC start is not suppressed 1 - NC start is suppressed Default setting: 1
DBGS	It applies to the operation mode "Debug Step" 0 - NC start is not suppressed 1 - NC start is suppressed Default setting: 1

NOW	It indicates whether the suppression applies from the block onwards in which the function was programmed. 0 - NC start is suppressed starting from next block 1 - NC start is also suppressed for this block Default setting: 0
If a variable is not programmed, the specified presettings are applied. If the value is not programmed, a value of 1 is assigned to the variable. If no variables are programmed, the brackets are optional. Empty brackets are not like a disabling function, but like an enabling function with preset values.	

Tab. 6-184: Syntax *StepModeDisable (SMD)*

StepModeEnable
Short form: SME
No parameters can be programmed. The block - in which the SME is programmed - is one of the blocks the NC start is suppressed in.

Tab. 6-185: *StepModeEnable (SME) syntax***Examples:**

Example 1: ... N50 SMD N60 X100 N70 SME ...	The blocks N60 and N70 are not confirmed with NC start in all Single Step modes
Example 2: ... N50 SMD X100 N60 Y100 N70 SME ...	Analog example 1
Example 3: ... N50 SMD(NOW1) X100 N60 Y100 N70 SME ...	In contrast to the examples 1 and 2, the block N50 is not confirmed with the NC start even though there is a traversing motion. The blocks N60 and N70 are also not confirmed with NC start.

Example 4: ... N50 SMD(DBGS0) X100 N60 Y100 SME ...	 The block N60 can be confirmed with NC start in the "Debug Step" mode. The NC start is suppressed in the operation modes "Program Block" and "Single Step/Single Block".
Example 5: Main program ... N50 P Up (SR) ... Subroutine SR N10 SMD(NOW) N999 SME	 The subroutine is completely processed in all Single Step modes in the N50 block of the main program by the NC start. Thus, no NC start is required with the subroutine. Note that the single block suppression has to be activated in the first block using the variable "NOW" - and that it has to be deactivated in the last block.

Tab. 6-186: *Programming examples of the Single Step mode suppression*

Special features and restrictions: The Single Step mode suppression does not impede a change of the operation mode in a locked operation mode. The operation mode is changed, but the locked blocks are still processed without NC start.

A new SMD programming overwrites the variables of a previously effective SMD. Only one SME is required.

SME is active after control startup and is additionally implicitly activated at control reset and program end. That means the operation mode suppression does not act modally via a program restart or a program change. However, SME is not generally activated in case of subroutine returns. This also applies to asynchronous subroutines. Ensure that the "SME" function is programmed to disable a suppression, since the block of an NC program in a single block mode is not confirmed with NC start jobs by accident.

Each program is first started with the "SME" function. This behavior cannot be set due to safety reasons.

6.142 Tool-specific spindle limits"SToolLim"

Description: Enables the entered tool-specific spindle limitations in one or several channel spindles in the /SysSpCmdData[<SysSp>]/Tool system date.



For more information, refer to the "MTX Functional Description Basics" in the chapter "Spindle functions" under "Tool-specific spindle limitations".

Syntax:

SToolLim, STOOLLIM
SToolLim(<SpName> = <0 1>{ , <SpName> = <0 1>} ...)
with

<SpName>	Spindle name (S, S1..S8)
<0 1>	1 enables the tool-specific limitations of /SysSpCmdData[]/Tool 0 disables the tool-specific limitations

Tab. 6-187: Syntax *SToolLim*

Example: `SToolLim(S2=1)` enables the tool-specific limitations of the system spindle currently assigned to the second channel spindle.

- Special features and restrictions:**
- SToolLim stops the programmed spindles and subsequently restarts them.
 - SToolLim reports a gear shifting to the "SpindleWait, SPWAIT" and "SpindleWaitMode, SWM" functions.
 - SToolLim comprises an NC-internal WAIT functionality.

6.143 Traveling in TCS "TCM(, ,)"

Description: Linear traversing motions in the tool coordinate system TCS, also in case of active axis transformation 2.

It is a local relative dimension programming (that means that the programmed position specified is interpreted incrementally irrespective of G90/G91).

For comprehensive information on the use and parameterization of "TcsMove", refer to the documentation "MTX Functional Description".

Syntax:

TCM({<xVal>},{<yVal>},{<zVal>}) (alternatively: TcsMove(, ,))	
with	
<xVal>, <yVal>, <zVal>	Values to be traveled incrementally in respect to the tool coordinate system TCS.

Tab. 6-188: Syntax of TCM

Example:

TCM(,,-100)	;plunging 100 [mm] in tool direction
TCM(, ,200)	;retract 200 [mm] in tool direction

6.144 Tangential tool control "TangTool, TTL"

Description: Takes a rotary or coordinate along the programmed path in the active plane with an adjustable angle. In this way, a tool can move at any time during application with a required positioning angle to the programmed path.

In order to rotate the tool axis to the corresponding required positioning angle, the function takes any existing tool symmetry into account in the case of tools with multiple tool edges. Furthermore, the control can automatically insert an intermediate block that rotates the tool axis by the required angle if there is a knee in the contour at the block transition.

The following applies:

- At the 0 degree position of the tool rotary axis, the default tool edge direction of the tool is parallel to the positive traveling direction of the main axis in the current plane.

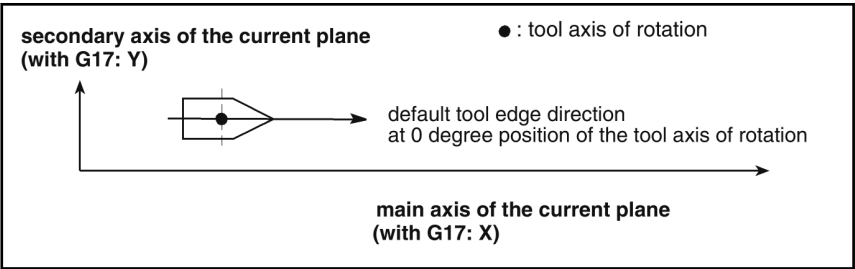


Fig. 6-37: 0 degree position of the tool rotary axis

- The positioning angle indicates the angular difference between the path and the default tool edge direction. It is 0 degrees if the default tool edge direction runs tangentially to the programmed path.

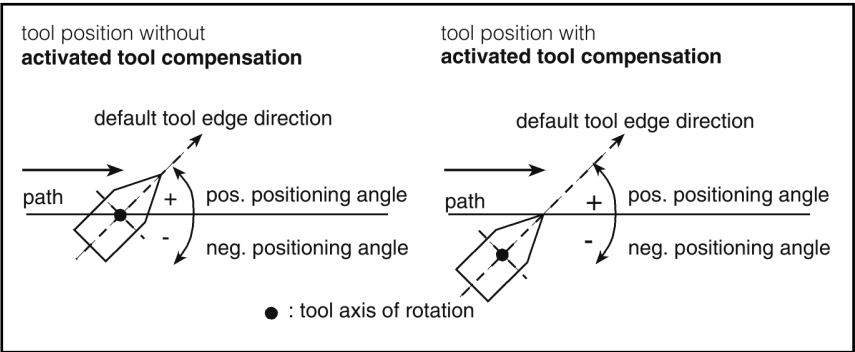


Fig. 6-38: Setting angle

- "TangTool" does not cause any traversing motion at activation. A programmed tool axis is not executed before the next traversing motion. Depending on the parameterized adaptation angle,
 - an intermediate block for tool rotation is traveled previously, or
 - the tool jumps to its new position at the beginning of the block.
- To calculate the required rotation of the tool axis in case of a specified setting angle during a circular path is done in an interpolator cycle. Consequently, the tool axis keeps turning by the respective setting angle calculated with each interpolator clock pulse in the case of such contour sections.



The maximum permissible intermediate block angle (ZSW) must not be larger than the angle the axis traverses at rapid traverse velocity within a Sercos cycle.
$$ZSW [^\circ] < AXDr[i]/Vel/MaxVel * SCS/ScsIf[1]/ScsCycTime * 1/60.000.000$$

Syntax:	<pre>TangTool (({TAX={}<Coordinate> } {,SYM={}<s> } {,ANG={}<a> } {,IA={}<zsw> } {,PLC={}<p> }))</pre>
	Tangential tool guidance ON
TangTool () or TangTool (0)	Tangential tool guidance OFF
Short form: TTL (. .)	

with	
<Coordinate>	Designation of the coordinate/axis to be subject to the function Tangential Tool Guidance. Admissible are: Name or number of the channel coordinate or axis. If this is not programmed, MP 7050 00210 is active. Only the system axis number can be entered.
<s>	Tool symmetry (normally the number of edges). Input value: Integer, not equal to 0. A tool with symmetry value <s> obtains a technologically identical position after a rotation of 360 degrees / <s>. Examples: Rectangular tools: <s>= 2, Quadrangular tool: <s>= 4. 1: Asymmetric tool or only one tool edge exists. >0: Symmetric tool having several, equally spaced tool edges. If there is a knee in the contour, the tool is rotated just enough for the nearest tool edge to be positioned at the positioning angle with the contour. <0: In the case of a change of direction (180 degree knee) the tool is not rotated, regardless of the positioning angle. Otherwise, the same applies as for ">0". If this is not programmed, MP 7050 00220 is active.
<a>	Setting angle Value range: -180 degrees to +180 degrees This indicates the angular difference between the path and the default tool edge direction. If this is not programmed, MP 7050 00250 is active. This is to specify whether the current angle of the tool rotary axis is to be used as setting angle or if the value of MP 7050 002540 should be used instead.

<zsw>	Intermediate block angle Value range: 0 to 180 degree Defines the knee angle between two blocks from which on an intermediate block is inserted to rotate the tool axis. If the angle at the contour knee is less than <zsw>, no separate block is inserted to rotate the tool axis. Instead, at the start of the next block, the tool jumps to its new position. The maximum permissible intermediate block angle (ZSW) must not be larger than the angle the axis traverses at rapid traverse velocity within a Sercos cycle. ZSW [°] < AXDr[i]/Vel/MaxVel * SCS/ScsIf[1]/ScsCycTime * 1/60.000.000 If this is not programmed, MP 7050 00230 is active.
<p>	Activates or deactivates an NC-PLC communication during execution of an intermediate block. 0: NC-PLC communication OFF. The NC executes the intermediate block unconditionally. 1: NC-PLC communication ON. The execution of a rotation block is controlled by the NC-PLC communication.

Tab. 6-189: Syntax of TangTool (TTL)

Example:	TTL	Tangential tool guidance is activated with the corresponding machine parameter values for the axis entered in MP 7050 00210.
	TTL (TAX=C, SYM1, ANG90, IA20, PLC0)	Programming with channel axis name
	TTL (TAX=theta, ANG45)	Programming with coordinate name
	TTL (TAX3, SYM1, ANG90, IA20, PLC0)	Programming with coordinate number
	TTL (TAX [NAME\$], SYM1, ANG90, IA20)	Programming with CPL variable

- Special features and restrictions:
- The function "tangential tool guidance" must not be active at the same time as "tangential tool orientation" (see [chapter 6.145 "Tangential tool orientation TangToolOri, TTO" on page 396](#)).
 - "Tangential tool guidance" must never be programmed in combination with a traversing motion.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.145 Tangential tool orientation "TangToolOri, TTO"

Description:	 Effective only in conjunction with the functions "punch" (see page chapter 6.94 "Punching Punch, PUN" on page 335) and "nibble" (see page chapter 6.80 "Nibbling Nibble, NIB" on page 303).

When Punch/Nibble is enabled, this ensures that a punching/nibbling tool is located at a definable angle to the programmed path during every stroke procedure.

For a stroke at the start of the block, a block for rotating the tool is automatically inserted, if required.

The control automatically determines the optimum direction of rotation (shortest path).

The rotary axis is specified by the MP 7050 00210.

Syntax:

TangToolOri({SYM<s>},{ANG<a>}) or TangToolOri(1) TangToolOri	
	Tangential tool orientation ON
TangToolOri() or TangToolOri(0)	Tangential tool orientation OFF
Short form: TTO...	
with	
<s>	Tool symmetry (normally the number of edges) Input value: Integer, greater than 0 A tool with symmetry value <s> obtains a technologically identical position after a rotation of 360 degrees / <s>. Examples: Rectangular tools: <s>= 2, Quadrangular tool: <s>= 4. 1: Asymmetric tool or only one tool edge exists. >1: Symmetric tool having several, equally spaced tool edges. If SYM is not programmed: effect as for SYM1.
<a>	Positioning angle to programmed path Value range: -180 degrees to +180 degrees 0: Tool axis is positioned at an angle of 0 degrees to the programmed path. If ANG is not programmed: effect as for ANG0.

Tab. 6-190: Syntax of TangToolOri, TTO

- Example:**
- G90 is active (absolute dimension programming)
 - Active plane: X/Y
 - Tool axis: C (modulo 360)
 - Current position: X=0, Y=0, C=0
 - Punching/nibbling is disabled.

NC functions with high-level language syntax

:	
N10 TangToolOri(1)	Tangential tool orientation ON; effect as for SYN1 and ANG0.
N20 G1 G91 X10 Y10	No orientation of the C-axis yet, since punching/nibbling is disabled.
N30 X10 Y10 Punch(1)	Punching ON C-axis rotates to 45°
N40 Y-10	C axis rotates to -90° After modulo calculation: C=270°
N50 Punch(0)	Punching OFF
N60 LEN=30 Nibble(1)	Nibbling ON Desired block segment lengths: 30 mm
N70 G2 X114.6 I57.3 J0	Semicircle with 180 mm length of arc The block is divided into 6 block segments, each with an arc length of 30 mm.
N80 TTO()	Tangential tool orientation OFF
:	

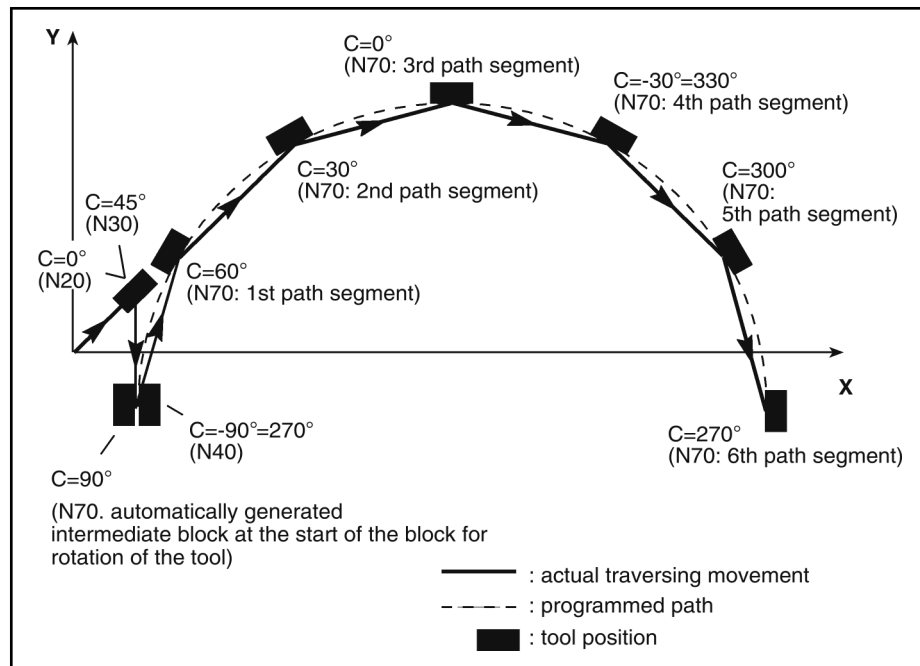


Fig. 6-39: Example TangToolOri (TTO)

Special features and restrictions:

- The function must not be programmed together with the function "Tang-Tool" (tangential tool guidance, TTL).
- When the function is active, "tangential tool guidance" (TTL, see [chapter 6.144 "Tangential tool control TangTool, TTL" on page 393](#)) is not possible.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.146 Retracting from tapping "TappRet1, TappRet2"

Description: If the "tapping without compensation chuck" function (G63) was canceled (by control reset, voltage drop) while the screw tap was still in operation, "TappRet1" and "TappRet2" can be used to retract the tap from the tapped hole. Programming can be done by manual input or within a part program / a subroutine (cycle).

Syntax:

TappRet1	Switches the spindle(s) - running again in spindle speed mode after control reset (control startup) - to position mode.
TappRet2 F...	Initiates the actual retract motion. with the programmed F-value.

Tab. 6-191: Syntax of TappRet1, TappRet2



"TappRet1" must be programmed before retracting.

Two situations must be distinguished when applying the function "Retract from Tapping":

- **Retract after control reset** (automatic retract):
The data saved when starting the tapping process is retained. With "TappRet1" the spindles are switched to the position mode. If "TappRet2" is programmed next, the tap moves out of the thread and on to the stored starting position. Only "TappRet2" must be programmed together with the desired feed (F-value).
- **Retract after power failure** (manual retract):
The retract information saved in the beginning of the tapping is lost. In this case, the relevant parameters have to be programmed explicitly together with "TappRet2":

```
G91 TappRet2 S<speed> F<feed> M3/M4 <boring axis>
<incremental distance>
```

Prerequisite to Using TappRet1/TappRet2:

- The c-axes involved have to be defined as endless rotary axes:
 - MP 1001 00004 (/AXSP/Dr[1]/AxFun/ModCalc/ENA_ModCalc)
 - Sercos parameter S-0-0076 = 0b1xxxxxxx.
- In the Sercos secondary mode 1 (S-0-0033), bit 8 for drive-controlled change of operation mode must not be set:
S-0-0033 = 0b000001011 or 0b000001100.

Application of subroutines for thread retracting:

To facilitate the application of the function "Retract from tapping", write one subroutine (cycle) for the automatic and one subroutine for the manual retract. These subroutines can be assigned to any unassigned G-code with the following machine parameters.

- 3090 00001 (/NCP/SubProg/GFun/Fun[1]/GCode)
- 3090 00002 (/NCP/SubProg/GFun/Fun[1]/SubProg)

6.147 Spindle selection for tapping without compensation chuck "TappSp, TSP"

Description: • Determines to which spindle G63 is to refer.

"TappSp" is in effect until it is reprogrammed. It is still in effect even after a "control reset"!

- If "TappSp" is not programmed, G63 always refers to the first spindle.

Syntax:

TappSp (CAX<i>)	Activate tapping for individual spindles using spindle number(s).
TappSp (GRP<j>)	Activate tapping for all spindles using a spindle group.
TappSp (GRP<j> { , CAX<i> })	
	Activate tapping for all spindles using a spindle group as well as for additional spindles using spindle number(s).
Short form: TSP (. .)	
with	
<i>	Number of the Spindle to which G63 refers. Value range: 1...n, where n stands for the highest spindle number in the system (max. 8). Whole number. Several spindles are programmed by several CAX<i>, separated by commas.
<j>	Number of the Spindle Group to which G63 refers. Value range: 1...n, where n stands for the highest spindle group number (max. 4). Whole number.

Tab. 6-192: Syntax of TappSp (TSP)

Example:

N20 TappSp (CAX2)	Select the spindle with a spindle number of 2 for G63.
:	
N120 TappSp (CAX2, CAX4, CAX7)	Selects the spindles with spindle numbers of 2, 4 and 7 for G63.
:	
N220 TappSp (GRP2)	Selects the spindle group with a spindle number of 2 for G63.
:	
N320 TappSp (GRP3, CAX4)	Selects the spindle group with a number of 3 plus the spindle with a spindle number of 4 for G63.
:	

Special features and restrictions:

- Only a single spindle group can be activated for G63.
- Spindle numbers, plus 1 spindle group if required, can be combined in any way.

6.148 Adjusting the axis configuration "TargetAxisConf, TAX"

Description: During block pre-run, the "TargetAxisConf" (short form: TAX) NC function can be used to conveniently adjust the axis and axis name configuration in the adjustment programs and in the action block.

The control uses the TAX command to activate the axis configuration prior to the target block of the block pre-run. Superfluous axes are removed. Required axes are included into the channel. Axis names are also applied.

Axes assigned otherwise are not taken into account:

- An axis that is assigned to another channel cannot be included.
- A C-axis that is in spindle mode cannot be included.

- An axis that is part of a transformation cannot be removed.

Axes that are not taken into account can subsequently be subjected to single commands. The necessary information is filed in the system data (Sys_ChSRun).

TAX can be called repeatedly, e.g., after a transformation has been deactivated.

Syntax: TAX or TargetAxisConf

Example: During the computation, the second system axis was removed from the channel, "Y" was the name assigned to the third system axis, and the fifth and eleventh system axes were included into the channel under the names of "Z" and "U".

Before computation		During computation	After computation	
SysNo	Name	...	SysNo	Name
1	X	N50 RAX(Y)	1	X
2	Y	N60 ALN(Z, Y)	3	Y
3	Z	N70 WAX(5, Z, 11, U)	5	Z
7	C	...	7	C
			11	U

Init adjustment program:

...

N100 X10 Y10 Z10 C10 **First, second, third and seventh** system axes move to position 10

N110 TAX

N120 X20 Y20 Z20 C20 **first, third, fifth, seventh and eleventh** system axes move to position 20
U20

...

Special features and restrictions: TAX is only allowed in the adjustment phase of the pre-run, i.e., in the adjustment programs and in the action block.

6.149 TCS definition in program coordinates "TcsDef, TCS"

Description: When 6-axis transformation is active, this generates a tool coordinate system TCS_p that can be offset and/or turned in reference to the last current TCS.

The coordinate values given for TCS_p are converted within the NC and stored in the tool correction memory.

The generated TCS_p can be cleared by disabling the function. In this process, the latest current TCS is activated again.

- Syntax:**
1. If not already executed, activate axis transformation type **3333301** (see function "Coord(...)", [chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245](#)).
 2. Use the syntax shown below.

NC functions with high-level language syntax

TcsDef ({<Position>},{<orientation>})	
	Define and activate the position and orientation of TCS _p . For detailed syntax, see below.
TcsDef () or TcsDef (0)	Deletes TCS _p and activates the latest current TCS again.
Short form: TCS (...)	

Tab. 6-193: Syntax of TcsDef (TCS)

Detailed Syntax:

TcsDef ({<x><px>},{<y><py>},{<z><pz>},{<phi><φ>},{<theta><θ>},{<psi><ψ>} or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{O (<φ> , <θ> , <ψ>) } or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{Ox (<φ _{xxy} (<φ _{yy}	
or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{Ox (<φ _{xxzz}	
or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{O _y (<φ _{yyzz}	
or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{Ox (<o11> , <o21> , <o31>) O _y (<o12> , <o22> , <o32>) } or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{Ox (<o11> , <o21> , <o31>) Oz (<o13> , <o23> , <o33>) } or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{O _y (<o12> , <o22> , <o32>) Oz (<o13> , <o23> , <o33>) } or	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{ROTAX (<φ _{uu}	
TcsDef ({<x><px>},{<y><py>},{<z><pz>},{ROTAX (<u _{xyz}	
with	
<x>	Coordinate name that is entered in MP 7080 00010[1]. Default: x
<y>	Coordinate name that is entered in MP 7080 00010[2]. Default: y
<z>	Coordinate name that is entered in MP 7080 00010[3]. Default: z
<px> , <py> , <pz>	Absolute Cartesian coordinates of the axis addresses <x> , <y> und <z> , referring to the current PCS. The values define the origin of the new TCS _p . AC/IC programming is permitted.
<phi>	Angle name that is entered in MP 7080 00010[4]. Default: phi
<theta>	Angle name that is entered in MP 7080 00010[5]. Default: theta
<psi>	Angle name that is entered in MP 7080 00010[6]. Default: psi

$\langle\varphi\rangle, \langle\vartheta\rangle, \langle\psi\rangle$	Absolute Euler angles in degrees, in relation to the current PCS. The values define the orientation of the new TCS_p. AC/IC programming is permitted. Value range: $0^\circ \leq \varphi < 360^\circ$ $0^\circ \leq \vartheta \leq 180^\circ$ $0^\circ \leq \psi \leq 360^\circ$ Values outside this range are automatically converted into the corresponding definition interval.
$\langle\varphi_{..}\rangle, \langle\vartheta_{..}\rangle$	Absolute angle values in degrees. Value range: $0^\circ \leq \varphi_{..} < 360^\circ$ $0^\circ \leq \vartheta_{..} \leq 180^\circ$ If a $\vartheta_{..}$ value outside the range of values is programmed, it is automatically converted into the given interval.
$Ox(\langle o11\rangle, \langle o21\rangle, \langle o31\rangle)$ $Oy(\langle o12\rangle, \langle o22\rangle, \langle o32\rangle)$ $Oz(\langle o13\rangle, \langle o23\rangle, \langle o33\rangle)$ $Ox(\langle\varphi_x\rangle, \langle\vartheta_x\rangle)$ $Oy(\langle\varphi_y\rangle, \langle\vartheta_y\rangle)$ $Oz(\langle\varphi_z\rangle, \langle\vartheta_z\rangle)$	Orientation using the function $Ox(..)$, $Oy(..)$, $Oz(..)$. For example, $Ox(..)$ defines the direction of the x-coordinate of the TCS_i in the reference coordinate system PCS. The same applies analogously to $Oy(..)$ and $Oz(..)$. The direction can be specified either by the corresponding polar angles $\langle\varphi_{..}\rangle$ and $\langle\vartheta_{..}\rangle$ or by specifying the Cartesian components of the column vectors of the TCS_p orientation sensor. Only absolute dimensions are allowed. The component values of the column vectors (o..) are automatically standardized to 1.
$ROTAX(\langle\varphi_u\rangle, \langle\vartheta_u\rangle)$	Defining the rotary axis by means of the polar angles (φ_u, ϑ_u) .
$ROTAX(\langle u_x\rangle, \langle u_y\rangle, \langle u_z\rangle)$	Definition of the rotary axis via the Cartesian components $\langle u_x\rangle, \langle u_y\rangle, \langle u_z\rangle$ in absolute dimensions. Automatic standardization to 1.
$O(\langle\beta\rangle)$	Specifies the incremental angle $\langle\beta\rangle$ in degrees by which the TCS_p orientation tensor is to be rotated around the rotary axis, originating at the orientation tensor of the last current TCS. Values greater than 360 degrees are allowed. The direction of rotation can be selected using the positive/negative sign.

Tab. 6-194: Detailed syntax of *TcsDef* (TCS)**Special features and restrictions:**

- The active axis transformation type 3333301 has to be active.
- If the function "Coord(...)" (see [chapter 6.28 "Selecting the axis transformation Coord, CRD" on page 245](#)) is reprogrammed, a previously generated TCS_p is automatically cleared and the last current TCS is activated again.
- If programming of the tensor columns results in two column vectors being parallel or anti-parallel, it is not possible to calculate the orientation tensor.
A runtime error is displayed.



In case of bracket expressions "(0)" or "(1)" do not represent parameters of the NC function. Thus, the numerical value within the brackets cannot be transmitted via CPL variables.

6.150 Special functions for thread cutting "ThreadSet, TST"

6.150.1 General information

Description: "ThreadSet" allows to temporarily adjust partial areas of G33.

In this case, the control superimposes the static values stored in the machine parameters.

"ThreadSet" allows for:

- Adjusting the dynamics and retract motion,
- Switching the spindle mode (speed control, position control)
- setting a signal at the channel interface (configurable within the "active function" signals using machine parameters).

Control reset or M30

- cancels the settings superimposed by "ThreadSet",
- clears an interface signal that had been set by "ThreadSet",
- switches the main spindle back to speed-controlled operation if it had previously changed to position-controlled spindle operation using "ThreadSet(SPC1)".



All partial functions described below can be jointly programmed (separated by commas) in one "ThreadSet" block.

6.150.2 Configuring the retract data ThreadSet(...)

Syntax:

ThreadSet (RD(<HA-value>,<NA-value>))	
Short form: TST (. .)	
with	
<HA value>	Retract path (incremental in mm) towards the main axis of the currently selected plane (G17, G18, G19, G20). The value must always be programmed; however, it is only relevant for longitudinal and tapered threads.
<NA value>	Retract path (incremental in mm) towards the secondary axis of the currently selected plane. The value must always be programmed; however, it is only relevant for transversal and tapered threads.
or:	
ThreadSet (RD(, -1))	
-1	The retract data from MP 7050 00645 and MP 7050 00650 are applied.

Tab. 6-195: Syntax of ThreadSet (TST), configure retract data

6.150.3 Configuring the dynamics ThreadSet(DYN)

Syntax:

ThreadSet (DYN({<jump>},{<Accel>},{<Decel>})) or TST (. .)	
with	
<Jump>	Max. permitted jump velocity in mm/min Entering "-1" enables MP 7050 00610 again.

<Acc>	Acceleration in m/s ² Entering "-1" enables MP 7050 00615 again.
<Brake accel>	Deceleration in m/s ² Entering "-1" enables MP 7050 00620 again.

Tab. 6-196: Syntax of ThreadSet (TST), configure dynamics

6.150.4 Enabling the retraction ThreadSet(ron)

Syntax:

ThreadSet (ron<Status>) or TST (..)	
with	
<Status>	0: Deactivates fast retraction 1: Activates fast retract

Tab. 6-197: Syntax of ThreadSet (TST), enable retraction

6.150.5 Switching the spindle mode ThreadSet(spc)

Syntax:

ThreadSet (spc<Status>) or TST (..)	
with	
<Status>	0: Switches the main spindle to speed-controlled mode. 1: Switches the main spindle to position-controlled mode, depending on the setting of 7050 00600 [3].

Tab. 6-198: Syntax of ThreadSet (TST), switch spindle mode

For the main spindle see [chapter 6.73 "Main spindle switching MainSp, MSP" on page 294](#).

6.150.6 Influencing the channel interface signal ThreadSet(tci)

Syntax:

ThreadSet (tci<status>) or TST (..)	
with	
<Status>	>0: clears the channel interface signal 1: sets the channel interface signal

Tab. 6-199: Syntax of ThreadSet (TST), influence channel interface signal



Which signal is influenced at the channel interface can be configured within the "Active function" signals using machine parameters.

6.151 Program coordinate offset "Trans, TRS", Additive Program Coordinate Offset "ATrans, ATR"

Description: "Trans" and "ATrans" displace the active program coordinate system relative to the current workpiece coordinate system.

NC functions with high-level language syntax

Both offsets can be activated and deactivated irrespective from each other. If both are active, they act additively.

In this way, a part program can be executed anywhere within the workpiece coordinate system without modifications in the programmed contour. Input tools based on the active program coordinate system (e.g. programmed contour offset, scaling, mirroring, rotation) are independent of a program coordinate offset and therefore do not need to be adapted.

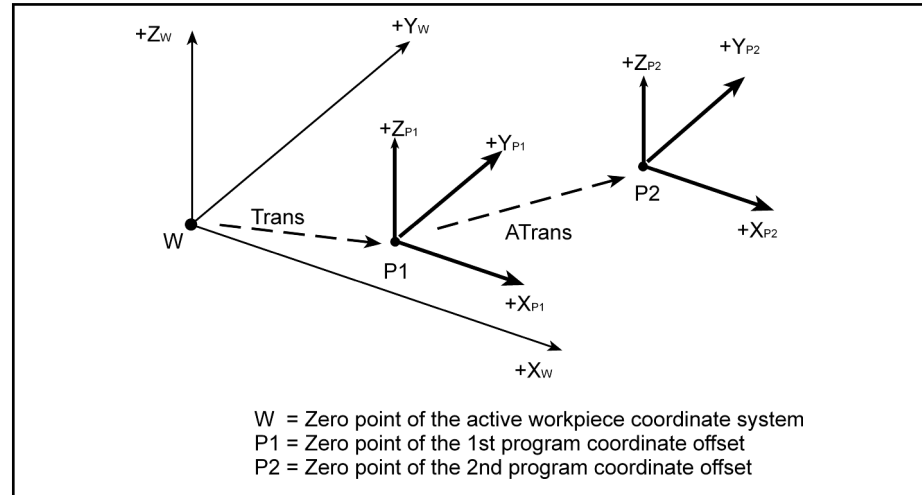


Fig. 6-40: Program coordinate offset, additive program coordinate offset

The following applies:

Programming a program coordinate offset does not trigger a traversing motion.

Syntax:

Trans (<Coordinates>)	Program coordinate offset ON The zero point of the program coordinate system is displaced to the workpiece coordinate system by the programmed <coordinates>. The offset acts additively to "ATrans". Several coordinate axes, separated by commas, may be programmed in the bracket. (Example: TRS (X100, Y50, Z50)).
Trans () or Trans (0)	All program coordinate offsets OFF
Short form: TRS (..)	
ATrans (<coordinates>)	Additive program coordinate offset ON The zero point of the program coordinate system is displaced to the workpiece coordinate system by the programmed <coordinates>. The offset acts additively to "Trans". Several coordinate axes, separated by commas, may be programmed in the bracket. (Example: ATR (X10, Z70)).

ATrans () or ATrans (0)	Additive program coordinate offsets OFF
Short form: ATR (..)	

Tab. 6-200: Syntax of Trans (TRS) and ATrans (ATR)

Example:	N10 ATR () TRS ()	Both program coordinate offsets OFF.
	:	The zero point of the program coordinate system is set to the position X0 Y0 Z0 in the current workpiece coordinate system.
	N30 TRS (X10, Y10, Z50)	The zero point of the program coordinate system is set to the position X10 Y10 Z50 in the current workpiece coordinate system.
	:	
	N50 ATR (X20, Y10)	The zero point of the program coordinate system is set to the position X30 Y20 Z50 in the current workpiece coordinate system.
	:	
	N70 ATR (X50, Z30)	The zero point of the program coordinate system is set to the position X60 Y20 Z80 in the current workpiece coordinate system.
	:	
	N90 TRS ()	Program coordinate offset OFF
	:	The zero point of the program coordinate system is set to the position X50 Y10 Z30 in the current workpiece coordinate system.

6.152 Volumetric compensation

6.152.1 Enabling and disabling "VCPEna"

Description: The NC function enables/disables the volumetric compensation in the channel. Before the compensation can be enabled, a table file is loaded with the NC function VCPLoad ([chapter 6.152.2 "Loading the table VCPLoad" on page 408](#)).

There are two memory slots for table files for each channel.

Syntax:

VCPEna (<TabIndex>)	Enabling compensation
VCPEna	Enable compensation in table 1
VCPEna (0) VCPEna ()	Disable compensation
with	
<TabIndex>	Number of the memory slot whose table should be used for the compensation Value range: 1 or 2

Tab. 6-201: VCPEna syntax

Example 1:

Program:

```

N10 VCPLoad (/Machine.ofm) ; Load table file in memory 1
N20 VCPEna                ; Enable compensation, table from 1
N30 ...                   ; Machining
N40 VCPEna ()              ; Disable compensation

```

Example 2: *Program:*

```

N10 VCPLoad(/mnt/ToolHead1.ofm)      ; Load file in memory 1
N20 VCPLoad(/mnt/ToolHead2.ofm, 2)   ; Load file in memory 2
N30 VCPEna                          ; Compensation on, table 1
N40 ...                             ; Machining
N50 VCPEna ()                        ; Disable compensation
N60 ToolHeadChange (2)               ; Change tool head 2
N70 VCPEna (2)                       ; Compensation on, table 2
N80 ...                             ; Machining
N90 VCPEna ()                        ; Disable compensation

```

Special features and restrictions:

- When enabling and disabling the compensation, the correction value is traveled in within 3s. Thus, the machine moves.
- The period of the motion depends on the correction value and the permitted (safe) velocity.
- If the required axes are not in the channel or the specified axis transformation AT1 is not active, a warning is created at activation and no compensation takes place.
- If the axis transformation AT1 is switched or disabled, the compensation deactivates itself and a warning is created.
- If an axis, which is compensated, is removed from the channel, a warning is output and the compensation is disabled.
- To adjust "VCPEna" and "VCPLoad" after a block pre-run, the NC function "ADJ(VCP)" can be used.

6.152.2 Loading the table "VCPLoad"

Description: The NC function can be used to load a table file for the volumetric compensation. The NC function VCPEna ([chapter 6.152.1 "Enabling and disabling VCPEna" on page 407](#)) enables and disables the compensation.

There are two memory slots for table files for each channel.

Syntax:

VCPLoad (<TabName> {, <TabIndex> })	Load table file
with	
<TabName>	File name including path of the table file to be loaded
<TabIndex>	Number of the memory slot whose table should be used for the compensation Value range: 1 or 2, specification: 1

Tab. 6-202: VCPLoad syntax

Example 1: see VCPLoad ([chapter 6.152.1 "Enabling and disabling VCPEna" on page 407](#))

Special features and restrictions:

- If the table is currently active in the specified memory location, a warning is generated and the programmed table is not loaded.
- To load a table to the same memory location directly after switch-off, "WAIT" and a short waiting time to retrace the correction has to be programmed in between for synchronization purposes.
- The table file may be located anywhere.
- The memory available for a table file is limited. If the data do not fit into the memory, a warning is generated.

6.153 Virtual drives "VirtAxisPos, VAP, VAPR"

Description: Sets the axis position of virtual synchronous axes in the current channel. When using the VAPR syntax, the interface signal "Reference point known" is set.



For further information on the virtual drives, refer to the "MTX Functional Description" documentation.

Syntax:

VirtAxisPos (<Axis 1><value>,<Axis n><value>, ...)

Short form: VAP (...)

with

<Axis x>	Address of the virtual axis

<Value>	Position information for <Axis x>

Tab. 6-203: Syntax of VirtAxisPos (VAP)

Example:

VAP (VX150) Position axis VX to 150 mm

Special features and restrictions: Local incremental programming (IC...) is not permitted.

6.154 Diagnostics of velocity control "VREC_START", "VREC_STOP"

Description: For editing of freeform surface programs with G8 velocity control, many factors affect the resulting velocity profile directly or indirectly. The diagnostics of the velocity control allows the user to find out which variable has the dominating influence on the reached path velocity at a specified position in the part program.

For block preparation, the **VREC_START** function records data which are calculated block by block by the velocity control and which affect the velocity curve. The name and storage location (directory) of the log file as well as the physical units in which data should be logged are specified during activation.

The created diagnostic file can be evaluated offline. With the help of the logged data, the parameters can be identified which must be optimized to implement a more favorable velocity curve.

During the VREC activation, the valid axis limit values for all axes located in the channel under the header "Limits of channel axes at VREC activation:" are output:

Column index:	Column title:	Meaning:
1	SysAxNo	System axis number
2	MaxAxVel	Maximum axis velocity
3	MaxAxAccUp	Maximum axis acceleration in the path acceleration phases as it can be programmed with the NcfAxAccUpSlope NC function.
4	MaxAxAcc	Maximum axis acceleration in the path deceleration phases
5	MaxAxJerk	Maximum axis jerk

Tab. 6-204: Structure of the log file

Subsequently, the following data is recorded block by block under the header "Resulting dynamic limits:"

NC functions with high-level language syntax

Column index:	Column title:	Meaning:
1	Way	Interpolation distance (path length) in the NC block
2	F100	Programmed velocity including the orientation motion (in the selected unit for G93 and G94, related to spindle revolutions for G95).
3	Fmax	Maximum path velocity including the orientation motion (limited)
4	F100(TCP)	Programmed velocity of the tip of the tool (for G93 and G94 in the selected unit, for G95 related to spindle revolutions)
5	Fmax(TCP)	Maximum path velocity of the tool tip (limited).
6	Fmax limiter	Characteristics and number of the system axis that limits the maximum path velocity to Fmax. The characteristics can be the maximum axis velocity, the maximum axis acceleration (Acc) and the maximum axis jerk (Jerk).
7	AccUp	Maximum path acceleration of the tool tip (limited). This value can only deviate from AccDown if one or several maximum axis accelerations were modified by the NC function NcfAxAccUpSlope.
8	AccDown	Maximum path deceleration of the tool tip (limited).
9	Acc limiter	Characteristics and number of the system axis that limits the maximum path deceleration to AccDown. The characteristics can be the maximum axis acceleration (Acc) and the maximum axis jerk (Jerk) in question.
10	Vend	Maximum path velocity at the end of the block or the current deceleration ramp including the orientation motion
11	Vend(TCP)	Maximum path velocity of the tip of the tool at the end of the block or the current deceleration ramp including the orientation motion.
12	MaxStepVel	Maximum path velocity that would be theoretically possible at the transition to the next block.
13	0Vel	yes: For the NC block, that attribute has been set that the subsequent block transition should be carried out with $V = 0$. This attribute is set for example of block which are to travel with exact stop.
14	Look ahead state	Look-ahead state including the number of currently looked-ahead blocks: 0: Dummy block: No Look-Ahead 1: Traversing block without Look-ahead, e.g. with precise stop, subsequent block is dummy block, etc. 2: Set Look_ahead is sufficient for brake path examination. 3: Set Look_ahead is sufficient for brake path examination. However, a $v = 0$ block transition has been found within the look-ahead range.

Column index:	Column title:	Meaning:
15	Insert state	Insert block state: -: No insert block 1: Common insert block 2: First split block (no insert block) 3: Split block (insert block) 4: Last split block (insert block)
16	Addr	Memory address of the NC block within the control
17	NC-Source	Source text of the NC block (here: CPL portions are initiated)

Tab. 6-205: Structure of the log file

Syntax:

VREC_START({file path/}<file name>{, Units})	
	Start diagnostic recording for velocity control
With:	
{File path/}<File name>	Directory and file name in which the diagnostic recording should be saved
Units	It is stored in the following units: Units = 0 or not programmed: Distance in increments Velocities in increments per lpo cycle Acceleration in increments per lpo cycle ² Jerks in increments per lpo cycle ³ Units = 1 and G70 active: Distance in inch Velocities in inch/min Accelerations in in/s ² Jerks in in/s ³ Units = 1 and G71 active: Distance in mm Velocities in mm/min Accelerations in m/s ² Jerks in m/s ³
VREC_STOP	Exits diagnostic recording

Tab. 6-206: Syntax of VREC_START; VRECSTOP

Example:

```

...
N30 VREC_START (/mnt/VDIAG1.txt)
...
N110 VREC_STOP
...
N300 VREC_START (/mnt/VDIAG2.txt, 1)
...
N500 VREC_STOP

```

...

M30

Special features and restrictions: When using the 'diagnostics of velocity control' function, ensure that there is sufficient memory for the diagnostic file. The required memory capacity is usually approx. 3 - 4 times greater than the memory required for the respective section of the part program to be recorded.

6.155 Applying the axis, wait, if required "WaitAxis, WAX"

Description: Applies an asynchronous axis to the calling channel. This turns an asynchronous axis to a synchronous axis.

The axis can then be programmed in the current channel using its physical or logical axis name.



For detailed information on the "axis transfer" functionality, refer to the documentation "MTX Functional Description".

Syntax:

`WaitAxis (<PAN>|<PAI>, {<LAN>}{, <PAN>|<PAI>, {<LAN>}}...)`

Short form: `WAX ( . . )`

with

<PAN>	Physical axis name Specifies the axis that is to be transferred to the current channel.
<PAI>	Physical axis index Same effect as <PAN>.
<LAN>	Logical axis name If this is programmed, the axis to be integrated into the current channel will be given the logical name <LAN>. <LAN> has to be defined either in the parameter CHAN/Ch/Coord/Acs/ChAx/ChAxName (MP 7010 00010) (channel axis name) or in the parameter CHAN/Ch/Coord/Acs/OptAxName/OptChAxName (MP 7010 00020) (optional axis name).

Tab. 6-207: Syntax of WaitAxis (WAX)

Example:

N030 WAX (YP, , ZP, Z) If necessary, block preparation waits for physical axes YP and ZP to come to a standstill. Then the axes are transferred to the calling channel.
:
While YP is also addressed as YP in the calling channel, ZP receives the address Z in the calling channel.

- Special features and restrictions:**
- If an axis to be transferred is not yet at a standstill, the block preparation waits for it to come to a standstill. The axis is subsequently transferred. As opposed to the function "GetAxis" (see [chapter 6.52 "Applying the axis GetAxis, GAX" on page 271](#)), no error message is displayed and the program is not terminated.
 - Axis positions in the same block must always be programmed after WaitAxis(...) and may be programmed only if axis transformation is not active.
 - Axes to be transferred must not be participating in any active monitoring area (see [chapter 6.7 "Area monitoring area, ARA" on page 203](#)).

6.156 Writing Sercos parameters with extended Sercos ID number "WritelD3, WID3"

6.156.1 General information

Description: It writes the programmed Sercos drive parameters (S-x-xxxx-xx.xx, P-x-xxxx.xx.xx) to one or more drives.

Apart from single values, complete value lists with a maximum of 16 values or ID lists with a maximum of 8 ident numbers can also be written.

Prerequisites:

- The parameter must be valid and editable in Sercos Phase 4.
- The parameter value must be within the permitted range.
- The drive must be connected to the control using the Sercos interface; cyclic communication between the control and the drive is running.



The CPL command "SCS3" is available to help read Sercos drive parameters with extended Sercos ident number.



Using the Sercos multiplex channel accelerates the data transfer.



Improper or random modifications of Sercos drive parameters can lead to damage on the workpiece and/or machine and to dangerous and unpredictable machine reactions.

Information on the Sercos drive parameters available in the drive is required for proper use. For this, consult your drive documentation.

6.156.2 Writing parameters to one or several drives (for spindles and asynchronous axes)

Syntax:

```
WriteID3(<Par>, DRIVE(<AA1>, <W1>{<AAn>, <Wn>}...))
```

Short form: WID3 (...)

with

<Par>	Default parameter (S-x-xxxx.xx.xx) - or - Product-specific parameter (P-x-xxxx.xx.xx)
<AA1>...<AAn>	Physical axis index or physical axis name of the Sercos drive. A maximum of 8 drives may be defined using DRIVE(...) syntax.
<W1>...<Wn>	Value to be written in <Par>

Tab. 6-208: Syntax 2 WritelD3 (WID3)

Example:

```
N100 WID3 (S-0-0104.0.0, DRIVE (1, 2.22) )
```

Writes the value 2.22 to S-0-0104.0.0 in drive 1

6.156.3 Writing parameters to one or several drives (for synchronous axes)

Syntax:

WriteId3 (<Par>, <SA1><W1>{, <SA<An><Wn>}...)	
Short form: WID3 (...)	
with	
<Par>	Default parameter (S-x-xxxx.xx.xx) - or - Product-specific parameter (P-x-xxxx.xx.xx)
<SA1>...<SA<An>	Logical axis address of the Sercos drive
<W1>...<Wn>	Value to be written in <Par>.

Tab. 6-209: Syntax 1 Writeld3 (WID3)

Example:

```
N100 WID3 (S-0-0159.0.0, X10, Y20, Z30)
```

S-0-0159.0.0 write:
in the drive of the x-axis: 10
in the drive of the y-axis: 20
in the drive of the z-axis: 30

6.156.4 Writing extended ident numbers of an ID list to a drive

Syntax:

WriteID3 (<Par>, EID_LIST (<A1>, <ID1>{, <Idn>}...))	
Short form: WID3 (...)	
with	
<Par>	Default parameter (S-x-xxxx.xx.xx) - or - Product-specific parameter (P-x-xxxx.xx.xx) containing an extended ID list.
<A1>...<A<An>	Physical axis index - or - physical axis name of the Sercos drive.
<Id1>...<Idn>	A maximum of 8 ident numbers to be written in <Par>.

Tab. 6-210: Syntax 4 Writeld3 (WID3)

Example:

```
N100 WID3 (S-0-1050.0.20, EID_LIST (2, P-0-556, S-0-1050.0.6))
```

Write S-0-1050.0.20 in the drive of axis 2 with two ident numbers.

Special features and restrictions:

- The function must not be programmed during contour machining.
- Programmed parameters are written only after the corresponding drive comes to a standstill.

6.156.5 Writing values of a parameter list to a drive

Syntax:

WriteID3 (<Par>, LIST (<A1>, W1{, <Wn>}...))	
Short form: WID3 (...)	
with	

7 CPL commands

Some basic properties of the CPL commands are displayed in a property field in front of the description of each command. The fields relevant for the respective command are highlighted in grey.

<type>	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Key:

- **<type>**: CPL data type of the return value of the CPL command (if the command is provided with a return value): **DOUBLE**, **REAL**, **INT**, **BOOL**, **CHAR**
For commands with a varying return value of the data type, the property field shows **MULTYP**
For commands providing a system data structure or substructure as return value, the property field shows **STRUCT**
- **READ**: CPL command to a read control-internal data, e.g. position, correction register or table element
- **WRITE**: CPL command to write a control-internal data, e.g. correction register or table element
- **FUNCT(ion)**: CPL command triggering a calculation or an action and providing a return value
- **PROC(edure)**: CPL command triggering an action and not providing any return value
- **WAIT**: Using the CPL command requires a preceding programmed WAIT for synchronization, as data of the active point in time is accessed for example
- **ERRNO**: The CPL command supports the programming of the ERRNO parameter
- **!**: Feature to be considered when using the CPL command. This feature is explained in the following description

7.1 Axes and coordinates

7.1.1 Overview

CPL provides commands to access axis and coordinate data.

The following commands are distinguished:

- Names and properties
(AXINF, AXADR\$, AXP)
- to read coordinate and axis positions
(PCS, WCS, MCS, ACS, SPOS, APOS)
- to read touch probes
(PCSPROBE, PROBE)

In the part program, coordinates interpolated while processing the program are always programmed. The axis transformations calculate the corresponding command values for the respective axes (spatial and machine coordinates) from the current coordinate values.



For detailed information on "coordinates, axes and transformations", refer to the documentation "MTX Functional Description".

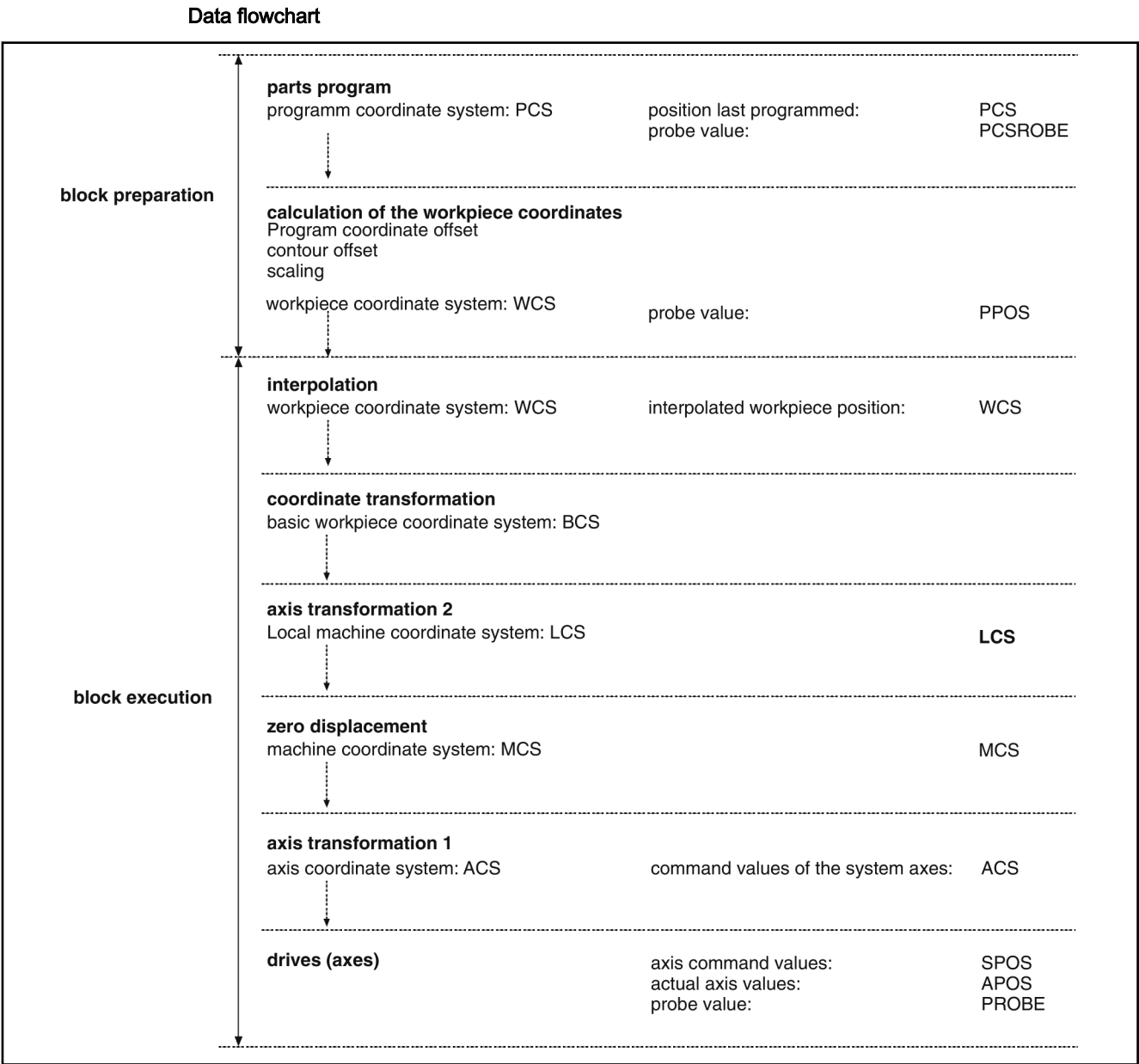


Fig. 7-1: Data flowchart

Measuring units for provided axis and coordinate positions

Synchronous linear axes and Translatory spatial coordinates:	"mm" or "inch"; Depends on the setting (G71, G70) in the calling channel.
Synchronous rotary axes and Rotary spatial coordinates:	"Degree"
Asynchronous linear axes:	"mm"
Asynchronous rotary axes:	"Degree"

Example: Configuration and assignment of axis names to channels

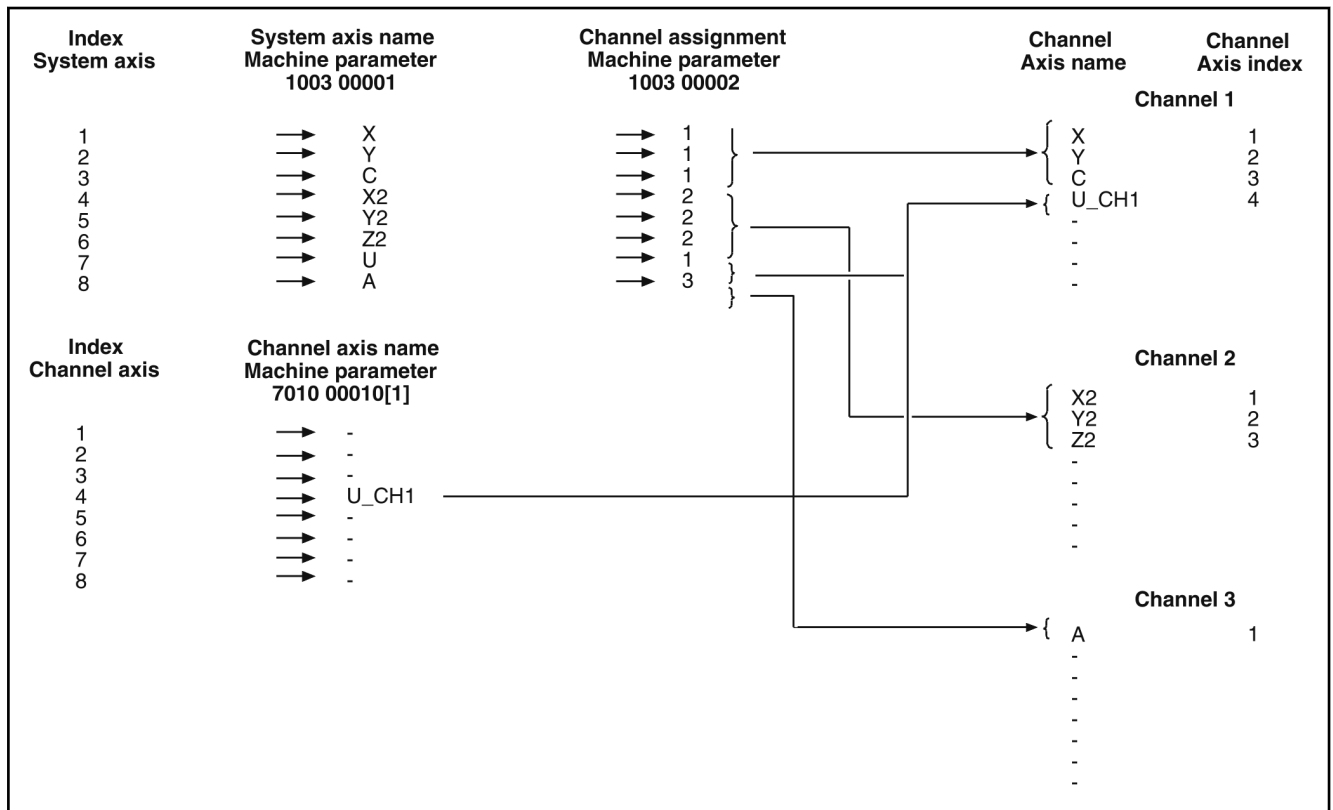


Fig. 7-2: Configuration and assignment of axis names to channels

7.1.2 Commands to read names and properties

AXADR\$

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Axis Address** - Axis and coordinate address

The CPL command **AXADR\$** returns the name of program and machine coordinates as well as channel and system axis names as return value.

The input parameter is used for the:

- System axis index
- SACI index
- program coordinate index (channel coordinate index) for the program coordinate names
- machine coordinate index for machine coordinate names
- Channel axis index for channel and system axis names

The AXADR\$ command supports the ERRNO parameter.

Syntax:

AXADR\$(<Index> [, [Selection type] [Channel number]])	
<Index>	Coordinate, channel, system or SACI index. The name is interpreted according to the specified <Selection type>.

CPL commands

<Selection type>	optional: Specifies how to interpret the transferred index: 0: System axis index The command returns the relevant system axis or SACI name 1: Channel coordinate index (default) The command returns the respective program coordinate name 2: Machine coordinate index The command returns the respective machine coordinate names 3: Channel axis index The command returns the respective channel axis name 4: Channel axis index The command returns the system axis name of the linked system axis
<Channel number>	optional: Channel number to which the specified channel coordinate or channel axis index refers. -1: The command returns the valid names in the freeze block (only in connection with block pre-run and asynchronous sub-routines). 0: Current channel (default) The channel number is valid only in connection with <Selection type> = 1, 2, 3, 4. In connection with the <Selection type> = 0, the channel number can only be specified if a value < 0 for <Index> is specified, i.e. the workpiece or machine coordinate names configured in the channel are to be accessed. If the channel number of a third-party channel is specified, the AXADR\$ command returns names that are valid at the active point in time of the specified channel.
ERRNO	CPL variable If the CPL variable ERRNO is programmed at any position in the parameter list, the command does not generate any internal runtime error. The error is returned by a corresponding return value of the variable. <i>The following return values are possible:</i> 0: Access OK. <0: Error: For a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69 If the CPL variable ERRNO is not specified, a runtime error is generated automatically if an internal access error occurs.

Tab. 7-1: Syntax AXADR\$

Syntax examples:

AXADR\$(2,0)	returns the system axis name of the second system axis
AXADR\$(3)	returns the program coordinate name of the third program coordinate in the current channel

AXADR\$(1,2,2)	returns the name of the first machine coordinate in channel 2
AXADR\$(4,3,4)	returns the channel axis name of the fourth channel axis in 4
AXADR\$(5,4)	returns the system axis name of the fifth channel axis in the current channel

AXINF

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description

Axis Information

The CPL command **AXINF** returns multiple axis and coordinate-specific properties as return value.

Possible input parameters:

- System axis name
- System axis number
- Channel axis name and (optionally) the channel index - or -
- channel axis index and (optionally) the channel index.

The following axis and coordinate-specific information can be determined via the AXINF command:

- System axis index for one channel or system axis.
- Motion type for one axis (linear, rotary, endlessly linear modulo).
- Virtual axis.
- Axis and coordinate meaning (X, Y, Z)
- Diameter programming

The AXINF command supports the ERRNO parameter.

Syntax:

AXINF(<Axis> [, Coordinate type [, Channel number, <Selection>]])	
<Axis>	Channel axis number or axis name: System axis name or channel axis name can be entered. The name is interpreted according to the specified <Selection type>.
<Coordinates>	optional: Defines whether a programmed name is interpreted as system axis name or channel axis name: 0: System 1: Channel (default)
<Channel number>	optional: Channel number referring to the specified channel axis index or channel axis name. 0: Current channel (default) The channel number is permitted if <Selection type> = 1. If the <Selection type> is 1 and no channel number is programmed, the data of the current channel is defined.

<Selection>	optional: Selection of the axis-specific information to be provided by AX-INF: 0: Axis and coordinate index (default) 1: Motion type of the axis <i>Return values:</i> • 1: Linear axis • 2: Endless Axis • 3: Rotary axis • 4: Linear modulo axis 2: Virtual axis <i>Return values:</i> • 0: Real axis • 1: Virtual axis 11: Axis meaning <i>Return values:</i> • 0: No axis meaning • 1: Meaning X • 2: Meaning Y • 3: Meaning Z 12: Diameter programming <i>Return values:</i> • 0: No diameter programming • 1: Diameter programming
ERRNO	CPL variable If the CPL variable ERRNO is programmed at any position in the parameter list, the command does not generate any internal runtime error. The error is returned by a corresponding return value of the variable. <i>The following return values are possible:</i> • 0: Access OK. • <0: Error: For a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69 If the CPL variable ERRNO is not specified, a runtime error is generated automatically if an internal access error occurs.

Tab. 7-2: AXINF syntax

Syntax examples:

AXINF("VA",0)	returns the system axis index of the system axis "VA"
AXINF("Z")	returns the system axis index of the channel axis "Z" in the current channel

AXINF("Y",1,5)	returns the system axis index of the channel axis "ZY" in channel 5
AXINF(4,1,0,ERRNO)	returns the system axis index of the fourth channel axis in the current channel
The error code is returned in the variable ERRNO.	

AXP

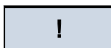
Description: **Axis Programming** - Variable axis address

The **AXP** command facilitates plane-independent writing of part and measurement programs.

Syntax

AXP (<Axis number>,<Path information>[,<Axis type>])	
The command is applied to an NC block. It has to be in square brackets "[]" and is programmed instead of the address values.	
<Axis number>	Index of workpiece coordinate or system axis
<Path information>	Path information variable or value
<Axis type>	optional: Determines how an index programmed under <Axis type> is interpreted: 0: System axis index 1: Index of the workpiece coordinate of the channel in which the program is currently processed. If there is no explicit programming, <Axis type> is set to 1.

Tab. 7-3: Syntax AXP



The AXP command can only be programmed within NC blocks using square brackets []. Calling under the operation mode "Manual Data Input" is thus not possible!

Example: **Subroutine:**

10 A%=P1% : B%=P2%	Transfers axis number from P1% and P2% to A% and B%
20 C=P3:D=P4:RA=P5	Transfers command values for G2
30 E=0	Constant for pole at G20
N40 G20 [AXP(A%,E)] [AXP(B%,E)]	Plane switching with G20; Pole on 0,0
N50 G2 [AXP(A%,C)] [AXP(B%,D)]	Radius programming with G2
R[RA]] F1000	

Plane definition with $A\%$ and $B\%$. Subsequent plane switching with $G20$. Finally, the axes move with $F1000$ through an arc defined by the variables C , D (end point) and RA (radius).

7.1.3 Commands to read coordinate and axis positions

General information

Note:

- The coordinate index in the channel is always specified for spatial and machine coordinates
- However, coordinates can be accepted or transmitted to the channel using the axis transfer NC functions. Thus, the coordinate index of the other coordinates in the channel are subject to change
Due to the option of presetting the system axis index, it is also possible to use fixed indices
- A runtime error is displayed if a non-configured system axis has been preset

The described commands run with similar parameters each. Therefore, they are explained in the following.

<Coordinate>	Index or name of a coordinate: A name is interpreted as coordinate name. Only if no corresponding coordinate name exists, the name is interpreted as channel or system axis name. An index is interpreted according to the specified <SelectionType>. Programming a non-configured coordinate/axis leads to a runtime error if the optional variable ERRNO is not been programmed.
<Axis>	Index or name of an axis: A name is interpreted as channel axis name. If no name exists, the system axis name is used. An index is interpreted according to the specified <SelectionType>. Programming a non-configured axis results in a runtime error if the optional variable ERRNO is not programmed.
<Selection type>	optional: Determines how an index programmed under <coordinate> or <axis> is interpreted: 0: System axis index 1: Coordinate index or channel axis index (default) If there is no explicit programming, <Selection type> =1 is set.

<Channel>	optional: Channel number; permitted only in connection with <Selection type> = 1. If coordinates/axes are to be read by third-party channels, and if they can be addressed using their index or name, the number of the channel to which the coordinate/axis is currently assigned to is entered in <Channel>. If no channel is specified, the coordinates/axes of the current channel are accessed. If a system axis is addressed (with name or index) and a channel is specified at the same time, an error message is generated, even if the variable ERRNO is programmed.
ERRNO	CPL variable that can be programmed at any position. <i>When ERRNO is programmed, no runtime error is output. The return values are:</i> • 0: Access OK • -1: Parameter error • -2: Coordinate/axis does not exist • -3: Invalid coordinate/axis in channel • -4: Axis is not a pseudo-coordinate • -5: Channel does not exist • -6: Command may only be called in its own channel • -7: Data could not be read

ACS

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Axis Coordinate System** - Reading axis command values

Provides the current command position of an axis.

Axes can be addressed with their channel axis name or their channel axis index.

Optionally, axes can also be addressed with their system axis name or their system axis index.

The following applies:

- The ACS result corresponds to the "SPOS" command if the system coupling is inactive for this axis.
- If the returned value is to be determined at the time of block execution, "WAIT" has to be programmed first in an individual block (see also [chapter 3.19.2 "Synchronization functions of block preparation" on page 79](#)).

Without WAIT, no uniquely predictable values are provided, since it is not known how far block execution "lags behind" the block preparation.

- When accessing the axis values of a third-party channel, it can be required to take synchronization measures to measure a defined position.

Syntax:

ACS (<Axis>[,<SelectionType>[,<Channel>]])	
	(For the parameters, refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424, For programming ERRNO, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69)

*Tab. 7-4: Syntax ACS***Syntax examples:**

```
ACS ("X", ERRNO)
```

```
ACS ("X", 1)
```

```
ACS ("X", 1, 1)
```

Channel 2 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels"](#) on page 419):

```
N10 G0 G90 X2=150 Y2=100
```

```
20 WAIT
```

```
30 X2VALUE=ACS ("X2", ERRNO)
```

The currently interpolated axis position of channel axis X2 is assigned to X2VALUE.

```
31 IF ERRNO <>0 THEN
```

```
32 PRN# (0, "error:", ERRNO)
```

```
33 ENDIF
```

```
N40 G91 X2=10 Y2=10
```

```
50 WAIT
```

```
60 X2VALUE=ACS (1, 1)
```

X2VALUE is assigned the currently interpolated axis position of the first channel axis of the current channel (X2 axis).

```
70 Y2WERT=ACS ("Y2", 1, 2)
```

The currently interpolated axis position of channel axis Y2 of the second channel is assigned to Y2VALUE.

```
80 XWERT=ACS (1, 0)
```

The currently interpolated axis position of the first system axis is assigned to XWERT.

APOS

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Actual Position - Reading actual axis values

Returns the current **actual position** of an axis based on the axis coordinate system.

All system axes can be addressed with their system axis name or their system axis index.

The following applies:

- If the returned value is to be determined at the time of block execution, WAIT has to be programmed first in an individual block (see also [chapter "WAIT \(without parameters\)"](#) on page 79).

Without WAIT, no uniquely predictable values are provided, since it is not known how far block execution "lags behind" the block preparation.

- When accessing the axis values of a third-party channel, it can be required to take synchronization measures to measure a defined position.

Syntax:

APOS (<axis>)	
<Axis>	Index or name of a system axis Programming a non-configured axis results in a runtime error.

Tab. 7-5: Syntax APOS

ERRNO cannot be programmed.

Example:

Channels according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)):

30 AKT4=APOS (4)	The current actual axis value of the fourth system axis (X2-axis in channel 2) is assigned to the variable AKT4.
:	
50 AKT8=APOS ("A")	The current actual axis value of the eighth system axis (a-axis in channel 3) is assigned to the variable AKT8.
:	

MCS

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Machine Coordinate System - Reading machine coordinates

Provides the current machine position (MCS) for a machine coordinate without straightness and angle error correction.

Third-party channel coordinates can also be accessed.

The following applies:

- If the returned value is to be determined at the time of block execution, WAIT has to be programmed first in an individual block (see also [chapter "WAIT \(without parameters\)" on page 79](#)).
Without WAIT, no uniquely predictable values are provided, since it is not known how far block execution "lags behind" the block preparation.
- When accessing the coordinate values of a third-party channel, it can be required to take synchronization measures to measure a defined position.

Syntax:

MCS (<Coordinate> [,<Selection type>[,<Channel>]])	
	(For the parameters, refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424 , For programming ERRNO, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69)

Tab. 7-6: Syntax MCS

Syntax examples:

MCS ("X", ERRNO)

MCS ("X", 1)

MCS ("X", 1, 1)

Channel 2 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)):

N10 G0 G90 X2=150 Y2=100

20 WAIT

CPL commands

30 X2WERT=MCS ("X2",ERRNO)	The currently interpolated machine position of the X2 coordinate is assigned to X2WERT.
31 IF ERRNO <>0 THEN	
32 PRN# (0,"error:",ERRNO)	
33 ENDIF	
N40 G91 X2=10 Y2=10	
50 WAIT	
60 X2VALUE=MCS (1,1)	The currently interpolated machine position of the first machine coordinate of the current channel (X2-coordinate) is assigned to X2VALUE.
70 Y2VALUE=MCS ("Y2",1,2)	The currently interpolated position of the Y2-coordinate of the second channel is assigned to Y2VALUE.
80 XVALUE=MCS (1,0)	The currently interpolated axis position of the first system axis is assigned to XWERT. This access is only permitted if the system axis is identical to the machine coordinate.

PCS

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Program Coordinate System** - Reading programmed coordinates

Returns the last programmed value for a coordinate.

The following applies:

- If the workpiece coordinate system is switched after programming the coordinate and before querying the position, the provided value already takes the new workpiece coordinate system into account.
- Only coordinates of the own channel can be queried.

Syntax:

PCS (<Coordinate> [,<SelectionType>])	
	(For the parameter, refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424; for programming ERRNO, refer chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69)

Tab. 7-7: Syntax PCS

Syntax examples:

PCS ("X",ERRNO)

PCS ("X",1)

Channel 2 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels"](#) on page 419):

N1 G0 G90 X2=150 Y2=100	
02 X2VALUE=PCS (1,ERRNO)	The programmed absolute position of the first coordinate in the current channel (X2VALUE = 150) is assigned to X2VALUE.
03 IF ERRNO <>0 THEN	
04 PRN# (0,"error:",ERRNO)	
05 ENDIF	

N10 G91 X2=10	
011 X2VALUE=PCS (1,1)	The programmed absolute position of the first coordinate in the current channel (X2VALUE = 160) is assigned to X2VALUE.
N12 X2=5 Y2=10	
13 Y2VALUE=PCS ("Y2",1)	The programmed absolute position of system axis Y2 (Y2WERT = 110) is assigned to Y2WERT.
14 X2VALUE=PCS ("X2")	The programmed absolute position of the X2-coordinate in the current channel (X2WERT = 165) is assigned to X2WERT.
15 XVALUE=PCS (1,0)	Runtime error: Access to the first system axis in channel 2 is not permitted. (Axis is assigned to channel 1)

SPOS

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

- Description:** **System axis position** - Reading command position of a system axis
- Returns the current **command position** of an axis based on the axis coordinate system.
- An active system axis coupling is considered in the command position. That means that the position corresponds to the slave axis value resulting from the total of the active master axis parts and the command position of the axis coordinate system.
- All system axes can be addressed with their system axis name or their system axis index.
- The following applies:*
- SPOS and ACS always provide identical values if the system axis coupling is deactivated.
 - SPOS provides the slave axis value SysAxCoupleSta[i].ActPos in the \$Commandposition display if the system axis coupling is active.
 - If the returned value is to be determined at the time of block execution, WAIT has to be programmed first in an individual block (see also [chapter 3.19.2 "Synchronization functions of block preparation" on page 79](#)).
Without WAIT, no uniquely predictable values are provided, since it is not known how far the block execution "lags behind" the block preparation.
 - When accessing the axis values of a third-party channel, it can be required to take synchronization measures to measure a defined position.

Syntax:

SPOS (<Axis>)	
<Axis>	Index or name of a system axis Programming a non-configured axis results in a runtime error.

Tab. 7-8: Syntax SPOS

ERRNO cannot be programmed.

Example: Channels according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)).

30 POS1=SPOS (1)	The current command axis value of the first system axis (x-axis in channel 1) is assigned to the variable POS1.
:	
50 POS5=SPOS ("Y2")	The current command axis value of the fifth system axis (Y2-axis in channel 2) is assigned to the variable POS5.
:	

WCS

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Workpiece Coordinate System** - Reading workpiece position

Provides the current workpiece position without online correction values for a coordinate.

The following applies:

- If the returned value is to be determined at the time of block execution, "WAIT" has to be programmed first in an individual block (see also [chapter 3.19.2 "Synchronization functions of block preparation" on page 79](#)).
Without WAIT, no uniquely predictable values are provided, since it is not known how far block execution "lags behind" the block preparation.
- When accessing the coordinate values of a third-party channel, it can be required to take synchronization measures to measure a defined position.

Syntax:

WCS (<Coordinate> [,<SelectionType>[,<Channel>]])	
	(For the parameters, refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424 , For programming ERRNO, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69)

Tab. 7-9: WCS syntax

Syntax examples:

```
WCS ("X", ERRNO)
WCS ("X", 1)
WCS ("X", 1, 1)
```

Channel 2 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)):

10 WAIT	
20 Z2POS=WCS (3,1,2,ERRNO)	The currently interpolated workpiece position of the third coordinate of the second channel (Z2-axis) is assigned to Z2POS.
21 IF ERRNO <>0 THEN	
22 PRN# (0,"Error:",ERRNO)	
23 ENDIF	
30 WAIT	


```
110 YPOS = WCS("Y")
```

The currently interpolated workpiece position of the y-coordinate of the current channel is assigned to YPOS.

```
120 XPOS = WCS(1)
```

The currently interpolated workpiece position of the first coordinate of the current channel is assigned to XPOS.

7.1.4 Commands to read measured values

PCSPROBE

Description: **Program Coordinate System PROBE** - Measured value in program coordinates

PCSPROBE() is used after the NC functions:

- G75.1 or G75 (travel to first measuring probe)
- G75.2 (travel to second measuring probe)
- FME (flying measurement)
- FSP (measuring at fixed stop)
- MOB (measuring on a block)

PCSPROBE() generates a runtime error when for example:

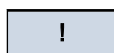
- an asynchronous axis is queried
- a synchronous axis of a third-party channel is queried
- Syntax error ...

ERRNO can be programmed. Refer to [chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO"](#) on page 69.

Syntax:

PCSPROBE (<Coordinate> [,<SelectionType>])	
<Coordinate>	Refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424
<Selection type>	

Tab. 7-10: Syntax PCSPROBE



Before PCSPROBE, it has not to be synchronized further with the WAIT command after the NC functions "Touch probe" (G75.x) and "Measuring at fixed stop" (FSP). The CPL command SD(9) is used to query whether measuring was successful and whether the measuring result is present!

Example:

Channel 1 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels"](#) on page 419):

```
N10 G75.1 X100 Y100 Z50
20 IF SD(9)=1 THEN
N30 (MSG, touch probe 1 has not triggered!)
40 GOTO .ERROR
50 ELSE
```

```

60 ZMEAS=PCSPROBE("Z",ERR-
NO);
61 IF ERRNO <>0 THEN
62 GOTO .Error
63 ENDIF
70 ENDIF

```

The value of the z-coordinate of the measured position is assigned to the variable "ZMEAS"

Special features: PCSPROBE() checks if all measured values of the axis required to calculate the measured value are available, taking the following NC functions into consideration.

- Axis transformation (CRD)
- Coordinate transformations (BCR, G152...G159)
- Rotation (ROT)

A missing measured axis value is reported via ERRNO -45 if programmed.

PPOS



From MTX 14VRS, the measured position can always be determined using PCSPROBE().

In the new application, please only use the **PCSPROBE()** and **PROBE()** commands.

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Probe POSition**

PROS() is used after the NC functions:

- G75.1 or G75 (travel to first measuring probe)
- G75.2 (travel to second measuring probe)
- FME (flying measurement)
- FSP (measuring at fixed stop)

PPOS() generates a runtime error when e.g.:

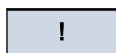
- an asynchronous axis is queried
- a synchronous axis of a third-party channel is queried
- Syntax error ...

ERRNO cannot be programmed.

Syntax:

PPOS (<Axis> [,<SelectionType>])	
<Axis> <Selection type>	Refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424

Tab. 7-11: PPOS syntax



Before PPOS, it has not to be synchronized further with the WAIT command after the NC functions "Touch probe" (G75.x) and "Measuring at fixed stop" (FSP). The CPL command SD(9) is used to query whether measuring was successful and whether the measuring result is present!

Example: Channel 3 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)):

```

N10 G75.2 A250 F500
20 IF SD(9) = 1 THEN
N30 (MSG, touch probe 2 has not triggered!)
40 GOTO .ERROR
50 ELSE
60 AMEAS = PPOS(1,1);
70 ENDIF

```

The measured value of the first channel axis of the channel is assigned to the variable "AMEAS".

Special features: PPOS considers the following corrections:

- Axis zero offsets (G54...G59)
- Tool corrections (G48, ED)
- Program coordinate offset (Trans, ATrans)
- Lead screw error compensation and cross-compensation

The following are not considered:

- Axis transformation (Coord)
- Coordinate transformations (BcsCorr, G152 - G159)
- Scaling (PoleSet, Mirror, Scale, Rotate)

Without any axis or coordinate transformation, the provided value refers to the latest programmed workpiece coordinate system (WCS).

PROBE

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **PROBE** - Measured value of the drive

PROBE() is used after the NC functions:

- G75.1 or G75 (travel to first measuring probe)
- G75.2 (travel to second measuring probe)
- FME (flying measurement)
- FSP (measuring at fixed stop)
- MOB (measuring on a block)

PROBE() generates a runtime error when for example:

- an asynchronous axis is queried
- a synchronous axis of a third-party channel is queried
- Syntax error ...

ERRNO cannot be programmed.

Syntax:

PROBE (<Axis> [,<SelectionType>])	
<Axis>	Refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424
<Selection type>	

Tab. 7-12: PROBE syntax

!

Before PROBE, it has not to be synchronized further with the WAIT command after the NC functions "Touch probe" (G75.x) and "Measuring at fixed stop" (FSP). The CPL command SD(9) is used to query whether measuring was successful and whether the measuring result is present!

Example: Channel 2 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)):

```

N70 G75.2 Y2 250

80 IF SD(9) = 1 THEN

N90 (MSG, touch probe 2 has not triggered!)

100 GOTO .ERROR

110 ELSE

120 Y2MEAS=PROBE(2);           The measured value of the second
                                channel axis is assigned to the
                                "Y2MEAS" variable (here: Y2 axis of
                                the channel).

130 ENDIF

```

Special features: The command PROBE()

- provides measured values that refer to the ASC axis coordinate (ASC) system
- considers only lead screw error compensation and cross compensation

7.2 Spindles

7.2.1 Overview

CPL provides commands to access spindle data.

7.2.2 Commands to read names

SPADR\$

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: Spindles Address

The CPL command **SPADR\$** returns the name of channel and system axis names.

The input parameter is used for the:

- System spindle index
- Channel spindle index for channel and system spindle names

The SPADR\$ command supports the ERRNO parameter.

Syntax:

SPADR\$(<Index> [, [<Selection type>] [, <Channel number>]])	
<Index>	Channel spindle or system spindle index. The name is interpreted according to the specified <Selection type>.

<Selection type>	optional: Specifies how to interpret the transferred index: 0: System spindle index The command returns the respective system spindle name 1: Channel spindle index (default) The command returns the respective channel spindle name 2: Channel spindle index The command returns the system spindle names of the linked system spindle
<Channel number>	optional: Channel number to which the specified channel spindle index refers to. -1: The command returns the valid names in the freeze block (only in connection with block pre-run and asynchronous sub-routines). 0: Current channel (default) The channel number is permitted only if <SelectionType> = 1 and 2. If the channel number of a third-party channel is specified, the SPADR\$ command returns names that are valid at the active point in time of the specified channel.
ERRNO	CPL variable If the CPL variable ERRNO is programmed at any position in the parameter list, the command does not generate any internal runtime error. The error is returned by a corresponding return value of the variable. <i>The following return values are possible:</i> 0: Access OK. <0: Error: For a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69 If the CPL variable ERRNO is not specified, a runtime error is generated automatically if an internal access error occurs.

Tab. 7-13: Syntax SPADR\$

Syntax examples:

SPADR\$(2,0)	returns the system axis name of the second system spindle
SPADR\$(3)	returns the channel spindle name of the third channel axis in the current channel
SPADR\$(1,2,4)	returns the system axis name of the first channel axis in channel 4
SPADR\$(5,2)	returns the system axis name of the fifth channel axis in the current channel

7.3 Corrections and input helps

7.3.1 Zero point offsets

General information

The following CPL commands permit ZO tables to be read, created and modified.

ZOTCR

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Zero Offset Table CReate** - Creating ZO table

Creates an XML zero offset table with no correction sets within the MTX file system.

Syntax:

ZOTCR (<Table>,<Channel/Template>)	
<Table>	Name of the XML table with file extension ".zot", if applicable including path specification If no path is specified, the table is created in the root directory.
<Channel/template>	Channel number or name of the XML table template, if applicable including path specification Of no path is specified, the table in the root directory is used.
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> 0: Access OK. <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-14: Syntax ZOTCR

Syntax example:

```
ZOTCR("/usr/user/tab.zot",1)
```

Creating a new "/usr/user/tab.zot" table with columns for each axis of the first channel.

ZOCINS

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Zero Offset Column INSert** - Inserting axis in ZO table

Inserts a new axis in front of an existing column in an XML zero offset table. The new axis comprises the same number of correction sets as the first axis of the table. The values are preset to 0.0.

Syntax:

ZOCINS (<Table>,<Position>,<AxisName> [,<AxisType>][,<AxisChannel>][,<PosChannel>]]])	
<Table>	Name of the XML table, if applicable including path specification Of no path is specified, the table in the root directory is used.
<Position>	Coordinate name or column index of the insert position
<Axis name>	Axis name of the new table column
<Axis type>	Optional axis type of the new axis 0: Linear axis (default value) 1: Rotary axis
<Axis channel>	Channel number of the new axis Default: 0
<PosChannel>	Channel number of the coordinate at the insert position if a coordinate name was specified. Otherwise, no meaning.
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> 0: Access OK. <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-15: Syntax ZOCINS

Syntax examples:

```
ZOCINS ("/usr/user/tab.zot", "X", "Y")
```

Inserting a new axis in the "/usr/user/tab.zot" table with the column name "Y" in front of the table column with the name "X".

Variant with axis name

```
ZOCINS ("/ZOT_Trafo.zot", "X", "U", , 1, 1)
```

Inserting a new linear axis (for channel 1) in the table "/ZOT_Trafo.zot" with the column name "U" in front of the table column with the name "X" of channel 1;

After the input of this command, the axis with the name "X" and channel 1 is searched in the table. The new linear axis "U" is inserted in its position for channel 1.

Variant with column index of the insert position

```
ZOCINS ("/ZOT_Trafo.zot", 1, "U", 1, , 2)
```

Inserting a new rotary axis (for channel 2) in the table "/ZOT_Trafo.zot" with the column name "U" in table column 1.

ZOCDEL

	---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!																																													
Description:	Zero Offset Column DElete - Deleting axis in ZO table Deletes an axis from an xml zero offset table.																																																				
Syntax:	<table><tr><td colspan="9">ZOCDEL (<Table>,<Position>[,<Channel>])</td></tr><tr><td><Table></td><td colspan="8">Name of the XML table, if applicable including path specification Of no path is specified, the table in the root directory is used.</td></tr><tr><td><Position></td><td colspan="8">Axis name or column index of the axis to be deleted</td></tr><tr><td><Channel></td><td colspan="8">Channel number of the axis to be deleted Default: 0</td></tr><tr><td>ERRNO</td><td colspan="8">CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> • 0: Access OK. • <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.</td></tr></table>								ZOCDEL (<Table>,<Position>[,<Channel>])									<Table>	Name of the XML table, if applicable including path specification Of no path is specified, the table in the root directory is used.								<Position>	Axis name or column index of the axis to be deleted								<Channel>	Channel number of the axis to be deleted Default: 0								ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> • 0: Access OK. • <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.							
ZOCDEL (<Table>,<Position>[,<Channel>])																																																					
<Table>	Name of the XML table, if applicable including path specification Of no path is specified, the table in the root directory is used.																																																				
<Position>	Axis name or column index of the axis to be deleted																																																				
<Channel>	Channel number of the axis to be deleted Default: 0																																																				
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> • 0: Access OK. • <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.																																																				

Tab. 7-16: Syntax ZOCDEL

Syntax examples:

```
ZOCDEL("/usr/user/tab.zot","X")
```

Deletes the axis with the column name "X" in the table "/usr/user/tab.zot".

Variant with axis name of the axis to be deleted

```
ZOCDEL("/ZOT_Trafo.zot","X",1)
```

Deletes the axis with the column name "X" in the table "/ZOT_Trafo.zot".

After entering this command, the axis with the name "X" (in channel 1) is searched in the table and then deleted.

Variant with column index of the axis to be deleted

```
ZOCDEL("/ZOT_Trafo.zot",1)
```

Deletes the axis with column index 1 in the table "/ZOT_Trafo.zot".

After entering this command, the first axis in the table is deleted.

ZOT

	DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!
Description:	Zero Offset Table - Read and write access to ZO table							

Read and write access to any XML zero offset table within the file system of the MTX CNC system. Only individual elements can be accessed. While writing, incremental modifications can also be specified.



The ZOT command directly writes in the specified table. If this is an active table, the written correction values are applied upon the next NC block.

Be careful when using the functionality "Asynchronous subroutine" (ASUP), "Leaving and returning to contour", "Delete distance-to-go" and "Overwrite":

All NC block already prepared are prepared again from the point of interruption in these functions. If correction values of the already prepared NC blocks are changed with ZOT in this range, the changed values now apply to all NC blocks to be prepared.

Also, the changed correction values are already active within a called asynchronous subroutine or within the approaching motion of leaving an returning to contour.

If this behavior is not desired, program the command "WAIT" before each ZOT!

Optionally, the refresh mechanism of tables and correction registers can be suppressed globally using the **/SysCorrRefrDis** system date. Changed correction values are applied upon the next ZO programming or the next table activation in this case.

Syntax:

ZOT (<ColumnSelection>,<ZO Code>[,<ZO Bank>][,<Table>][,<Unit>][,<Channel>]]])	
<Column selection>	Name of the machine coordinate (axis); programmed in inverted commas or column index in the table. <i>While writing, it is possible</i> - to overwrite the table value or - enter the value additively when the <Column selection> starts with a negative sign.
<ZO code>	54: 1. Zero point offset 55: 2. Zero point offset 56: 3. Zero point offset 57: 4. Zero point offset 58: 5. Zero point offset 59: 6. Zero point offset
<ZO bank>	Index of the ZO bank (1...5) Default value: 1
<Table>	Name of the ZO table, if required with absolute or incremental path information (table extension ".zot" does not have to be programmed). If there is no path information, it is searched in the configured search path. For tables with default names such as ZO1 and ZO2, only the numerical value can be specified. If there is no table name, the latest activated table is accessed.

<Unit>	0 or "MM": mm 1 or "INCH": inch During write access, the assigned value is interpreted in the specified unit. During read access, the value is converted into the specified unit. Default value: mm
<Channel>	Channel assignment (0..60) The parameter <Channel> may only be specified if <Column selection> has been specified as name of a machine coordinate. If the parameter <ColumnSelection> contains a column index and a <Channel> is specified, an error message is displayed. Channel-specific coordinates of the ZO table with identical names can be differentiated by means of the channel assignment. Default: The first column is accessed according to <ColumnSelection>. The channel entry is ignored. The first column found with <ColumnSelection> and <Channel> is used. If not available: Error message
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> • 0: Access OK. • <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-17: ZOT syntax

Syntax examples:

ZOT("X", 54)	Access to the G54 offset of axis / machine coordinate "X" in the last activated ZO table of the channel.
ZOT(2, 55, 2)	Access to the G55.2 offset of the second entered axis / machine coordinate in the last activated ZO table of the channel.
ZOT(3, 57, , "V1")	Access to the G57 offset of the third entered axis / machine coordinate in the ZO table "V1.zot".
ZOT("Z", 58, 4, 21, 1)	Inch access to the G58.4 offset of axis / machine coordinate "Z" in the ZO table "ZO21.zot".
ZOT("Y", 59, 5, "/mnt/esmuser/de/HHGENIUS", "MM")	Metric access to the G59.5 offset of axis / machine coordinate "Y" in the table "HHGENIUS" in the directory "/mnt/esmuser/de".

```
ZOT ("Y", 56, 1, "/mnt/ZOF", "MM", 2)
```

Metric access to the G56.1 offset of axis / machine coordinate "Y" assigned to channel 2 in the table "ZOF" in the directory "/mnt".

ZOV

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Zero Offset Value** - Active ZO values

Provides the total of the latest programmed and thus effective ZO values of a machine coordinate (axis) or, optionally, the effective value of a single ZO bank.

Syntax:

ZOV (<Axis selection>[,<ZO bank>])	
<Axis selection>	Name of machine coordinate (axis), programmed in inverted commas - or - index of the machine coordinate (axis); values 1..8
<ZO bank>	Index of the ZO bank, values 1..5

Tab. 7-18: ZOV syntax

Syntax examples:

ZOV ("X")	returns the total of all effective (latest programmed) ZOs for the machine coordinate (axis) X
ZOV (2, 3)	returns the effective ZO of the third ZO bank for the second machine coordinate (axis)

7.3.2 Position and contour offset AXO

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Axis Offset** - Position offset

AXO provides the current SETPOS offset for a coordinate at the time of block preparation, i.e. the offset that was last activated when the program interpretation is provided.

Syntax:

AXO (<Coordinate>[,<SelectionType>])	
	(For parameters, refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424 ; ERRNO cannot be programmed)

Tab. 7-19: Syntax AXO



AXO permits access only to the offset values of its own channel. Since asynchronous axes do not have any SETPOS offset, AXO is not permitted for asynchronous axes.

Example:

Channel 1 according to the example configuration (see [fig. 7-2 "Configuration and assignment of axis names to channels" on page 419](#)):

```
N10 G1 G90 X100 F1000
N20 SETPOS (X75, Y125)
```

30 XD = AXO ("X")	The latest activated SETPOS offset of the x-coordinate of the current channel (XD=100-75=25) is assigned to XD.
40 YD = AXO (2,0)	The latest activated SETPOS offset of the second system axis (YD=200-125=75) was assigned to YD.
50 X2D = AXO (4,0)	Runtime error, since the fourth system axis is assigned to channel 2.

COF

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Contour offset** - Contour offset

Provides the last programmed contour offset (shift) of a coordinate.

Since the programmed contour offset only affects the coordinates in the current channel, selecting a coordinate that does not exist in the current channel results in an error message.

Corrections values are provided in the active measuring unit of the current channel, i.e. for G70 in "inch" and for G71 in "mm". For rotary axes/rotary spatial coordinates, the unit is always "degree".

Syntax: COF (<Coordinate> [,<SelectionType>])

(For parameters, refer to [chapter 7.1.3 "Commands to read coordinate and axis positions"](#) on page 424; ERRNO cannot be programmed)

Tab. 7-20: Syntax COF

Example:	10 A=COF (3) :	returns the last programmed contour offset of the coordinate with the third coordinate index in the active channel.
	20 B=COF ("X") :	returns the latest programmed contour offset of the x-coordinate in the active channel.
	30 C=COF (2,0) :	returns the last programmed contour offset of the second system axis in the active channel. This access is only permitted if the system axis is identical to the WCS coordinate.
	100 C=COF (0) :	Runtime error, since 0 is an invalid coordinate index.

7.3.3 Workpiece position correction and placements (inclined plane)

DPC

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Datum Point Correction** - Workpiece position correction

Returns the latest programmed parameters of the workpiece position correction (BCR) of a coordinate (offset values and rotating angles).

Since the workpiece position correction only applies to the coordinates in the current channel, selecting a nonexistent coordinate in the current channel results in an error message.

Correction values are provided in the active measurement unit of the current channel. That means for G70 in "inch" and for G71 in "mm". For rotary axes/ rotary spatial coordinates, the unit is always "degree".

Syntax:

DPC (<Coordinate>[,<Selection type>])	
(For parameters, refer to chapter 7.1.3 "Commands to read coordinate and axis positions" on page 424 ; ERRNO cannot be programmed)	
<Coordinate>	Additional effect of the specifications: "1"... "n" or "Name": returns offset value "0" returns angle of rotation Phi

Tab. 7-21: Syntax DPC

Example:

10 A=DPC (1)	returns the last programmed workpiece position correction of the coordinate with the first coordinate index in the channel.
:	
15 B=DPC ("X")	Returns the last programmed workpiece position correction of the x-axis/coordinate in the channel.
:	
20 B=DPC (2)	returns the last programmed workpiece position correction of the coordinate with the second coordinate index in the channel.
:	
25 B=DPC (2, 0)	returns the last programmed workpiece position correction of the second system axis in the active channel.
:	
30 ANGLE=DPC (0)	Returns the last programmed angle of rotation Phi.
:	
100 C=DPC (9)	Runtime error, since 9 is not a valid coordinate index if there are eight axes in the system.
:	

PMV

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

PlaceMent Value - Active placement values

Returns the total of the latest programmed, and thus effective placement values for a coordinate. Optionally, the effective value of an individual placement bank can be read.

Syntax:

PMV (<Coord.Selection>[,<Pl.Bank>])	
<Coord.Selection>	Index of the coordinate (1...6) or fixed identifiers "X", "Y", "Z", "PHI", "THE", "PSI" for the WCS coordinates.
<Pl. bank>	Index of the placement bank (1...5)

Tab. 7-22: Syntax PMV

Syntax examples:

PMV ("Y")	Returns the total number of all active (last programmed) placements for the "Y" coordinate. Due to possible rotations, not only the total number of the individual banks is involved, but also the resulting offset in "y"-direction of the basic workpiece coordinate system.
PMV (4, 4)	Returns the Euler angle PHI of the active placement of the fourth bank.

PMT

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **PlaceMent Table** - Read and write access to placement table

Read and write access to any XML placement table of the MTX CNC system. Only single elements can be accessed. While writing, incremental modifications can also be specified.

Syntax:

PMT (<Coord.Selection>,<Pl.Code>[,<Pl.Bank>][,<Table>][,<Unit>]]])	
<Coord.Selection>	Index of the coordinate (1...6) or fixed names "X", "Y", "Z", "PHI", "THE", "PSI" for the WCS coordinates The value can be read or written. <i>During write access, the following is possible:</i> • To overwrite the table value or • Add the value to the table value if <Coord.Selection> starts with a negative sign.
<Pl. code>	154: Placement 1 155: Placement 2 156: Placement 3 157: Placement 4 158: Placement 5 159: Placement 6
<Pl. bank>	Index of the placement bank (1...5) Default value: 1
<Table>	Name of the placement table; if required, with absolute or incremental path information (table extension ".pmt" does not have to be programmed). If there is no path information, it is searched in the configured search path. For tables with default names such as PM1 and PM2, only the numerical value has to be provided. If there is no table name, the latest activated table is accessed.
<Unit>	0 or "MM": mm 1 or "INCH": inch For the three translatory coordinates, the assigned value is interpreted in the unit specified for write access. In case of read access, the value is converted into the specified unit. Default value: mm

Tab. 7-23: Syntax PMT

Syntax examples:

PMT ("Z", 155)	Access to the G155 offset of the coordinate "Z" in the latest activated placement table of the channel.
PMT (5, 157, 3)	Access to the Euler angle ϑ of G157.3 in the last activated placement table of the channel.
PMT ("X", 154, , "P1")	Access to the G154.1 offset of coordinate "X" in the placement table P1.

```
PMT(3,158,4,,1)
```

Inch access to the G158.4 offset of the coordinate "Z" in the last activated placement table of the channel.

```
PMT("Y",159,5,"/mnt/esmuser/de/PL5","MM")
```

Metric access to the G159.5 offset of coordinate "Y" in the table "PL5" in the directory "/mnt/esmuser/de".

ROT2EUL

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Rotation to Euler angles** - Computing Euler angles from rotations along axes

ROT2EUL is a CPL command to compute Euler angles. Subsequent rotations along the axes x, y and/or z can be specified. These are then converted into Euler angles for placements.

In contrast to ABSROT2EUL ([chapter " ABSROT2EUL" on page 445](#)), it is not rotated along spatially fixed axes, but always along the new (rotated) coordinate system.

Syntax:

ROT2EUL(<EulerAngle>, <RotaryAxis1>, <Angle1> [,<RotaryAxis2>, <Angle2> [,<RotaryAxis3>, <Angle3>]])	
<Euler angle>	Array of three DOUBLE values Input: Euler angle along which the starting coordinate system is rotated Output: Euler angle along which the result coordinate system is rotated
<Rotary axis1..3>	Axis around which it is to be rotated ("x", "y" or "z")
<Angle1..3>	Angle around which it is to be rotated (in degrees)

Tab. 7-24: Syntax ROT2EUL

Example:

Program:

```
1 REM Compute Euler angle for rotation along X (20°) and then along Z (60°)
2 DIM EULER!(3)
3 EULER!(1) = 0 : EULER!(2) = 0 : EULER!(3) = 0
4 ROT2EUL(EULER!, "X", 20, "Z", 60)
5 REM Result in field EULER! (Phi, Theta, Psi)
```

ABSROT2EUL

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Absolute rotation to Euler angles** - Compute Euler angles from rotations along spatially fixed axes

ABSROT2EUL is a CPL command to compute Euler angles. Up to three rotations along the axes x, y and/or z of the BCS (spatially fixed) can be specified. These are then converted into Euler angles for placements.

In contrast to ROT2EUL ([chapter " ROT2EUL" on page 445](#)), it is not rotated along spatially fixed axes, but always along the axes of the basic coordinate system.

Syntax:

ABSROT2EUL (<EulerAngle>, <RotaryAxis1>, <Angle1> [,<RotaryAxis2>, <Angle12> [,<RotaryAxis3>, <Angle3>]])	
<Euler angle>	Array of three DOUBLE values Input: Euler angle along which the starting coordinate system is rotated Output: Euler angle along which the result coordinate system is rotated
<Rotary axis1..3>	Axis around which it is to be rotated ("x", "y" or "z")
<Angle1..3>	Angle around which it is to be rotated (in degrees)

Tab. 7-25: ABSROT2EUL syntax

Example: Program:

```

1 REM Compute Euler angle for rotation along Y (45°) of an already rotated KOS
2 DIM EULER!(3)
3 EULER!(1) = 90 : EULER!(2) = -30 : EULER!(3) = 270
4 ABSROT2EUL(EULER!, "y", 45)
5 REM Result in field EULER! (Phi, Theta, Psi)

```

PT2EUL

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Points to Euler angles - Compute Euler angles for and inclined plane from which three points are specified

PT2EUL is a CPL command to compute Euler angles. Three points (as x, y and z values) to be located in one plane are transferred to the command. The Euler angles are computed from the points in a way that this plane spans the new xy-plane with one placement. Thus, an area with three points can be sampled and the Euler angles can be determined for a corresponding placement via PT2EUL.

All three points have to be different and they not be in a straight line. Even though the parameters are sufficient to determine a completely new plane, only the computed Euler angles are returned. Since Pt1 is equal to the zero point, the plane can be directly set with the angles and Pt1.

Syntax:

PT2EUL (<EulerAngle>, <Pt1X>, <Pt1Y>, <Pt1Z>, <Pt2X>, <Pt2Y>, <Pt2Z>, <Pt3X>, <Pt3Y>, <Pt3Z>)	
<Euler angle>	Array of three DOUBLE values Input: Euler angle along which the starting coordinate system is rotated Output: Euler angle along which the result coordinate system is rotated
<Pt1XYZ>	Zero point of the new plane related to the starting coordinate system
<Pt2XYZ>	New x-axis through Pt2 related to the starting coordinate system
<Pt3XYZ>	Pt3 (related to the starting coordinate system) is located in the new xy-plane

Tab. 7-26: Syntax PT2EUL

Example: Program:

```

1 REM The probed points are located from PT1! to PT3!
2 DIM EULER!(3)
3 EULER!(1) = 0 : EULER!(2) = 0 : EULER!(3) = 0
4 PT2EUL(EULER!, PT1!(1), PT1!(2), PT1!(3), PT2!(1), PT2!(2), PT2!(3), PT3!(1), PT3!(2), PT3!(3))
5 REM Result in field EULER! (Phi, Theta, Psi)
; The placement can be directly set with G152
G152.1([PT1!(1)], [PT1!(2)], [PT1!(3)], [EULER!(1)], [EULER!(2)], [EULER!(3)])

```

VEC2EUL

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Vectors to Euler angles** - Compute Euler angle for inclined plane from which the normal vector and the direction of the new x-axis are specified

VEC2EUL is a CPL command to compute Euler angles. A plane is specified using a normal vector (located vertically on the plane). Furthermore, a direction can be optionally programmed for the x-axis. The command computes the Euler angles for this plane.

If the second vector is not vertical to the normal vector, its vertical projection is used to determine the x-axis. The two vectors may not be in parallel. If the vector of the x-axis is not programmed, the y-axis is specified in parallel to the old xy-plane. The enclosed angle corresponds to the second Euler angle (as G151).

Syntax:

VEC2EUL (<EulerAngle>, <NormX>, <NormY> ,<NormZ> [, <XAxisX>, <XAxisY> ,<XAxisZ>])	
<Euler angle>	Array of three DOUBLE values Input: Euler angle along which the starting coordinate system is rotated Output: Euler angle along which the result coordinate system is rotated
<NormXYZ>	Normal vector of the new plane related to the starting coordinate system
<XAxisXYZ>	Vector of direction of the new x-axis related to the starting coordinate system

Tab. 7-27: Syntax VEC2EUL

Example: Program:

```

1 REM Compute Euler angle with normal vector
2 DIM EULER!(3)
3 EULER!(1) = 0 : EULER!(2) = 0 : EULER!(3) = 0
4 VEC2EUL(EULER!, 1, 0.5, 0)
5 REM Result in field EULER! (Phi, Theta, Psi)

```

WCS2BCS

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **WCS to BCS** - Converts a point from the workpiece coordinate system (WCS) to the basic coordinate system (BCS)

WCS2BCS is a CPL command to convert a point. A transferred point in the workpiece coordinate system (WCS) is converted into the basic coordinate system (BCS). Both points differ in their placements connected in between.

Placement data can optionally be transferred to the command. If this data is not programmed, the currently active placements (without BCR) are used for the computation.

Syntax:

WCS2BCS (<Point> [, <Placement>])	
<Point>	Array of three DOUBLE values Input: Point in WCS Output: Point in BCS
<Placement>	Array of six DOUBLE values Placement data [X, Y, Z, Phi, Theta, Psi]

Tab. 7-28: WCS2BCS syntax

Example: Program:

```
; Set a placement
G151.1(10,20,30, 45,30,0)
WAIT
; The zero point of a new placement is to be located on the current placement related to (10,0,50)
2 DIM PT!(3)
3 PT!(1) = 10 : PT!(2) = 0 : PT!(3) = 50
4 WCS2BCS(PT!)
; Result in the field PT!
; Set the new placement and overwritten the old one
G151.1([PT!(1)], [PT!(2)], [PT!(3)], -45, 30, 0)
```

BCS2WCS

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

BCS to WCS - Converts a point from the basic coordinate system (BCS) into the workpiece coordinate system (WCS)

BCS2WCS is a CPL command to convert a point. A transferred point in the basic coordinate system (BCS) is converted into the workpiece coordinate system (WCS). Both points differ in their placements connected in between.

Placement data can optionally be transferred to the command. If this data is not programmed, the currently active placements (without BCR) are used for the computation.

Syntax:

BCS2WCS (<Point> [, <Placement>])	
<Point>	Array of three DOUBLE values Input: Point in BCS Output: Point in WCS
<Placement>	Array of six DOUBLE values Placement data [X, Y, Z, Phi, Theta, Psi]

Tab. 7-29: Syntax BCS2WCS

Example: Program:

```
; Set a placement
G151.1(10,20,30, 45,30,0)
WAIT
; previously measured point in the field PT! is present in the BCS and is now to be approached
2 BCS2WCS(PT!)
; Result in the field PT!
; Approach point in new coordinate system
G1 F500 X[PT!(1)] Y[PT!(2)] Z[PT!(3)]
```

7.3.4 Input helps**SCL**

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

SCaLing - Active values for scaling, mirroring, rotating and pole

Returns for the current channel (here: channel in which the program with the **SCL** command is running) the latest programmed parameters of the NC functions PLS, SCL, MIR and ROT (pole coordinates, scaling factors and rotation angles).

Since PLS, SCL, MIR and ROT(...) apply only to the coordinates in the current channel, selecting a nonexistent coordinate in the current channel results in an error message.

Position values are provided in the active measuring unit of the current channel, i.e. for G70 in "inch" and for G71 in "mm". For rotary axes or rotary spatial coordinates, the unit is always "degree".

Syntax:

SCL (<Selection>[,<Coordinate>[,<SelectionType>]])	
<Selection>	0: last programmed angle of rotation of the main plane 1: last programmed pole of a channel axis 2: Last programmed scaling factor of a channel axis
<Coordinate>	Index or name of a coordinate: A name is interpreted as coordinate name. Only if no corresponding coordinate name exists is this name, it is interpreted as channel or system axis name. An index is interpreted according to the specified <Selection-Type>. Programming a non-configured coordinate/axis leads to a run-time error if the optional variable ERRNO is not been programmed.
<Selection type>	optional: Determines how an index programmed under <Coordinate> is interpreted: 0: System axis index 1: Coordinate index (default) If there is no <Column selection>, the index is interpreted as coordinate index!

Tab. 7-30: Syntax SCL

Example:

10 W=SCL(0)	Writes the latest programmed angle of rotation to the variable W.
:	
20 P=SCL(1,2)	Writes the pole of the coordinate with the second coordinate index in the channel into the variable P.
:	
30 F=SCL(2,2,1)	Writes the scaling factor of the coordinate with the second coordinate index in the channel into the variable F.
:	
40 D=SCL(2,"X")	Writes the scaling factor of the x-coordinate in the active channel into the variable D.

7.3.5 Tool corrections

General information

The following CPL commands permit access to internal tool correction data (D-corrections) and external tool correction data (ED-corrections) which the PLC can access.

TCV

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **ToolCorrValues** - The latest tool correction values applied
Provides the latest programmed tool correction values, either as a total (D-correction + external correction memory) or as a single value.

Syntax:

TCV (<ValueSelection>[,<CorrectionSelection>])	
<Value selection>	0: Latest activated D-/ED-correction 1 or "L1": L1 correction length 2 or "L2": L2 correction length 3 or "L3": L3 correction length 4 or "RAD": Tool / tool edge radius 5 or "ORI": Edge position 6 or "PHI": Euler angle ϕ (ED-correction only) 7 or "THE": Euler angle ϑ (ED-correction only) 8 or "PSI": Euler angle ψ (ED-correction only)
<Correction selection>	1 or "D": D-correction (internal) 2 or "E": ED-correction (external) Default: Total of D- and ED-correction

Tab. 7-31: Syntax TCV

Syntax examples:

TCV (0, "D")	Provides the latest programmed D-correction.
TCV ("L1")	Reads the total of L1 values of the latest programmed D- and ED-correction.
TCV (4, 1)	Reads the tool radius of the latest programmed D-correction
TCV (2, "E")	Reads the L2 tool length of the latest programmed ED-correction.

DCT

DOUBLE	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **D-Correction Table** - Accessing D-correction tables and ED-correction registers
Read and write access to tool correction values in any D-correction tables as well as to external correction values (ED-correction). While writing, incremental modifications can also be specified.



The DCT command directly writes to the specified table or correction register. If this is an active table or an active correction register, the written correction values are applied upon the next NC block.

Be careful when using the functionality "Asynchronous subroutine" (ASUP), "Leaving and returning to contour", "Delete distance-to-go" and "Overwrite":

All NC block already prepared are prepared again from the point of interruption in these functions. If DCT was used to change correction values within the range of the already prepared NC blocks, the changed value now applies to NC blocks to be prepared.

Also, the changed correction values are already active within a called asynchronous subroutine or within the approaching motion of leaving an returning to contour.

If this behavior is not desired, program the "WAIT" command before each DCT!

Optionally, the refresh mechanism of tables and correction registers can be suppressed globally using the `/SysCorrRefrDis` system data. In this case, correction value changes are applied upon the next correction or the next table activation or edge selection.

Syntax:

DCT (<ValueSelection>, <DataBlock>[, [<Table>] [, <Unit>]])	
<Value selection>	1 or "L1": L1 correction length 2 or "L2": L2 correction length 3 or "L3": L3 correction length 4 or "RAD": Tool / tool edge radius 5 or "ORI": Edge position 6 or "PHI": Euler angle ϕ (ED-correction only) 7 or "THE": Euler angle ϑ (ED-correction only) 8 or "PSI": Euler angle ψ (ED-correction only) The value can be read or written. During write access, it is possible: - to overwrite the table value or - to add the value to the table value if <Value selection> starts with a negative sign. Incremental writing is not possible for the edge position (5 or "ORI").
<Data set>	1...99: For a D-correction 1...16: For an ED-correction

<Table>	Name of the D-correction table, If required with absolute or incremental path information (without table extension ".dct"). If there is no path information, it is searched in the configured search path. For tables with default names such as DC1, DC2, etc., only the numerical value has to be provided. Without the latest accessed D-correction table is used. 0: Access to the ED-correction of the current channel. -1: Access to the ED-correction of channel 1 -2: Access to the ED-correction of channel 2. -3: Access to the ED-correction of channel 3. ... -60: Access to the ED-correction of channel 60.
<Unit>	Only relevant for D-correction tables: 0 or "MM": mm (default) 1 or "INCH": inch During write access, the assigned value is interpreted in the specified unit. During read access, the value is converted into the specified unit. When accessing ED corrections, the value is always interpreted according to the unit (the last unit selected in the calling channel G70/G71) or are provided in this unit.

Tab. 7-32: Syntax DCT

Syntax examples:

DCT (1, 10, "K4")	Access to the "L1"-correction value of data block 10 in D-correction table "K4".
DCT ("RAD", 7)	Incremental writing to the tool radius of data block 7 of the latest activated table.
DCT (3, 1, 0)	Access to the "L3" correction of the external tool correction of the channel.
DCT ("L2", 16, 1, "INCH")	Inch access to the "L2" correction of data block 16 of the table DC1.dct.
DCT (4, 7, -3)	Access to the tool radius of correction set 7 of the third channel.



When reading/writing tool correction data from third-party channels, note that there is no competing access among different channels.



When accessing ED corrections, the value is always interpreted according to the unit (the last unit selected in the calling channel G70/G71) or are provided in this unit.

7.4 Control data and processes

7.4.1 Control version

VERSINF\$

	CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!
Description:	VERSion INFormation - Control version VERSINF\$ allows to query management data of the MTX CNC system in the CPL program. The command provides a value of type "String".							
Syntax:	VERSINF\$(<Index1>[,<Index2>])							
	<Index1>		Constant of type integer Command mode: 1: Read software version 2: Read hardware version					
	<Index2>		Type integer. Additional command mode depending on the parameter value <Index1>. <Index1> = 1: No additional mode available <Index1> = 2 : additional mode:					
			1: Hardware type (default) 2: Vector group number 3: Index of vector group number 4: Serial number 5: Type code					

Tab. 7-33: VERSINF\$ syntax

Example:

```

10 DIM SYS_INFO$(50)
20 SYS_INFO$ = VERSINF$(1)
30 PRN#(0,"Software version: ", SYS_INFO$)
M30

```

7.4.2 Time recording

CLOCK

	INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!
Description	CLOCK - System time Reads the control-internal system time or the internal interpolation cycle counter. The difference of two measurement results can be used to determine the runtime of a program section.							

Syntax:

CLOCK(<Mode>)	
<Mode>	optional (Default: 0) 0: Output of the system time in milliseconds. 1: Output of the system time in microseconds. 2: Indicates the current state of the interpolation cycle counter with the set interpolation cycle in milliseconds.

Tab. 7-34: Syntax CLOCK

Example:

```

:
20 WAIT
30 STARTTIME% = CLOCK(2)
N4 G1 X50 Y70
40 WAIT
50 ENDTIME%=CLOCK(2) : DIFF%=ENDTIME%-STARTTIME%
:

```

The current state of the interpolation cycle counter is assigned to the variables "STARTZEIT%" or "ENDZEIT%" before and after block N4 is executed. The block execution time of N4 (in milliseconds) can be determined from the difference between the contents of the two variables.

Note that the "WAIT" command is absolutely mandatory!



The runtime measurement of a program section with control-internal system time does not provide useful results on an MTX control emulation!

Measuring the number of Ipo cycles causes for the MTX control emulation and real controls nearly the same runtimes as long as there are no delays by unprepared NC blocks, e.g. by block cycle time problems or long CPL passages.

DATE

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

DATE - Date

Provides data regarding the current date.

Syntax:

DATE[(<i><Mode></i>)]	
<i><Mode></i>	optional (Default: 0) 0: Output in format "DD.MM". 1: Output in format "DD.MM.YY". 2: Output of the week day: 1=Monday, 2=Tuesday, etc. 3: Current day of the year, e.g. "40" for February 9th. 4: Calendar week of the year.

Tab. 7-35: Syntax DATE



If DATE is used several times in a program, the corresponding result variables have to be dimensioned. Otherwise, the latest value read of the DATE command is assigned to all non-dimensioned variables that contain the result of a DATE assignment.

Example:

```

:
10 DIM A$ (5)      Dimensioning of the string variable A$.
20 DIM B$ (8)      Dimensioning of the string variable B$.
30 DIM C$ (1)      Dimensioning of the string variable C$.
40 DIM D$ (3)      Dimensioning of the string variable D$.
50 DIM E$ (2)      Dimensioning of the string variable E$.
60 A$ = DATE       The date as "DD.MM" is assigned to the string variable A$.
70 B$ =           The date as "DD.MM.YY" is assigned to the string variable B$.
DATE (1)
80 C$ =           The weekday as numerical value "1" - "7" is assigned to the
DATE (2)          string variable C$.
90 D$ =           The current day as numerical value "1" - "366" is assigned to
DATE (3)          the string variable D$.
100 E$ =          The calendar week of the year as numerical value "1" - "53" is
DATE (4)          assigned to the string variable E$
:

```

TIME

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:**TIME** - Time

Provides the current value for the time.

Example:

```

:
40 B$ = TIME       The time as "HH.MM.SS" is assigned to the string variable B$.
:

```



If TIME is used several times in a program, the corresponding result variables have to be dimensioned. Otherwise, the latest value read of the TIME command is assigned to all non-dimensioned variables that contain the result of a TIME assignment.

7.4.3 Machine parameters

MCA

MULTYP	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Machine COnfiguration Data** - Machine parameters

Transfers the contents of a single MACODA parameter. Depending on the data type, this value can be of the following types "INT", "REAL", "DOUBLE" or "CHAR". The variable in which the transferred value is to be stored, has to be of the same type!

Type conflicts between the transferred value and the target variables are detected at the program's runtime and are acknowledged with an error message.

Syntax:

MCA (<BlockId>,<ParameterIndex>[,<ClassIndex>])	
<BlockId>	Number of one MACODA parameter One MACODA may contain multiple single MACODA parameters (parameter list). If a non-existent parameter number is programmed, a runtime error is output.
<Parameter index>	Index of the single MACODA parameter starting with "0". If a non-existent index number is programmed, a runtime error is output.
<Class index>	Optional: Only for channel, spindle and transformation-specific machine parameters. This parameter specifies channel, spindle or transformation index for which the machine parameter value has to be read. If not programmed and in case of a channel parameter, the command provides the MACODA single parameter of the channel in which the CPL program is currently executed. If an invalid class index is programmed, a runtime error is output.

Tab. 7-36: Syntax MCA

Example: **MCA command**

```
10 BLOCKNO%=100300004
```

```
20 ERG%=MCA(BLOCKNO%,0)
```

The content of the first single parameter of the machine parameter 100300004 is assigned to the integer variable ERG%.

7.4.4 Simple system data

SD (general description)

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **System Data** - Simple system-internal data in integer format

Reads from a collection of system-internal data of the MTX CNC system in integer format.

Syntax:

SD (<Group>[,<Index1>[,<Index2>[,<Index3>]]])	
	The SD command provides INTEGER values.

Tab. 7-37: Syntax SD

SD(2,...) - Current override position

Syntax	Return Value
SD(2,1)	Provides the current position of the feed override in the current channel.
SD(2,2)	Provides the current position of the rapid traverse override in the current channel.
SD(2,3) ($\hat{=}$ SD(202,1))	Returns the current position of the override for spindle 1. 0 if no spindle is applied.
SD(2,4) ($\hat{=}$ SD(202,1))	Provides the current position of the override for spindle 2. 0 if no spindle is applied.

Tab. 7-38: SD(2,...)

SD(5,...) - Velocity and speed

Syntax	Return value
	Active values:
SD(5,1,1)	Returns the active feed velocity (100% value) rounded to an integer value in mm/min or inch/min.
SD(5,2,1)	Returns the active rapid traverse velocity (100% value) rounded to an integer value in mm/min or inch/min.
SD(5,3,1) ($\hat{=}$ SD(205,1,2))	returns the last programmed workpiece position correction of the second system axis in the active channel. 0 if no spindle is applied.
SD(5,4,1) ($\hat{=}$ SD(205,1,2))	returns the current spindle speed (command value) of the second spindle in rpm (incl. override evaluation). 0 if no spindle is applied.
	Programmed values:
SD(5,1,2)	Returns the last programmed feed in input unit per minute.
SD(5,3,2) ($\hat{=}$ SD(205,2,1))	returns the last spindle speed (command value) programmed in the channel of the first spindle in rpm. If no spindle is applied, 0 is returned.
SD(5,4,2) ($\hat{=}$ SD(205,2,2))	returns the last spindle speed (command value) programmed in the channel of the second spindle in rpm. If no spindle is applied, 0 is returned.
	Actual values:

Syntax	Return value
SD(5,3,3) ($\triangleq$ SD(205,3,1))	Returns the actual spindle speed of the first spindle.
SD(5,4,3) ($\triangleq$ SD(205,3,2))	Returns the actual spindle speed of the second spindle.

Tab. 7-39: SD(5,...)

SD(8) - Channel query

Syntax	Return value
SD(8)	Returns the channel number of the calling channel.

Tab. 7-40: SD(8)

SD(9) - Touch probe

Syntax	Return value
	returns information on whether a touch probe was triggered after G75, G75.2 or MOB or whether "measuring at fixed stop" FSP or FsProbe is active:
SD(9)	• 0: – to G75, G75.2 or MOB: Touch probe switched. – Acc. to FSP (FsProbe): Measuring at fixed stop executed. • 1: – to G75, G75.2 or MOB: Touch probe not switched. – Acc. to FSP (FsProbe): Measuring at fixed stop not yet executed.

Tab. 7-41: SD(9)

SD(10,...) - Coordinate index on which the L1, L2 or L3 correction is applied

Syntax	Return Value
	If the correction is settled in the workpiece coordinate system, SD(10,...) provides the channel axis index of the axis on which the L1, L2 or L3 correction is applied: If the correction is settled in the workpiece coordinate system TCS, SD(10,...). SD(10,...) provides the following return values: 101: Settlement in x-direction of the TCS $\triangleq$ XTR. 102: Settlement in y-direction of the TCS $\triangleq$ YTR. 103: Settlement in z-direction of the TCS $\triangleq$ ZTR.
SD(10,1,<Corr>) SD(10,1) ($\triangleq$ SD(10,1,3))	at the point in time of block preparation.

Syntax	Return Value	
SD(10,2,<Corr>) SD(10,2) ($\triangleq$ SD(10,2,3))	at the active point in time.	
	<Corr>	1: Returns the index for the coordinate on which the L1 correction applies. 2: Returns the index for the coordinate on which the L2 correction applies. 3: Returns the index for the coordinate on which the L3 correction applies.

Tab. 7-42: SD(10,...)

SD(11,...) - Channel axis index of main, secondary and infeed axis

Syntax	Return value	
	Provides the channel axis index of main, secondary and infeed axis depending on the active plane:	
SD(11,<Axis>,1)	at the point in time of block preparation.	
SD(11,<Axis>,2)	At the active point in time (block execution).	
	<Axis>	1: Main axis 2: Secondary axis 3: Infeed axis

Tab. 7-43: SD(11,...)

SD(12,...) - Direction of spindle rotation

Syntax	Return value
SD(12,1) ($\triangleq$ SD(212,1,1))	Active direction of spindle rotation: 0: Spindle stop 3: Clockwise spindle rotation 4: Counter-clockwise spindle rotation 19: Orient spindle -1: Spindle not applied
SD(12,2) ($\triangleq$ SD(212,2,1))	<i>Last programmed direction of spindle rotation:</i> - Commands such as "Active direction of spindle rotation" (SD(12,1)). - An active change of direction of rotation for each interface signal is not considered!

Syntax	Return value
SD(12,3) ($\triangleq$ SD(212,1,2))	<i>Active direction of spindle rotation (second spindle):</i> Commands such as "Active direction of spindle rotation" (SD(12,1)).
SD(12,4) ($\triangleq$ SD(212,2,2))	<i>Last programmed direction of spindle rotation (second spindle):</i> - Functions as in "Active direction of spindle rotation" (SD(12,1)). - An active change of direction of rotation for each interface signal is not considered!

Tab. 7-44: SD(12,...)

SD(13) - Processing method at interpretation time

Syntax	Return Value
SD(13)	Processing methods when processing at interpretation time: 0: Single block, single step 1: Continuous block 2: Program block 11: Block pre-run with continuous block (selected block not yet interpreted).

Tab. 7-45: SD(13)

SD(14) - National language

Syntax	Return Value
SD(14)	Returns the number (country code) of the active national language (machine parameter 6010 00010).

Tab. 7-46: SD(14)

SD(15) - Test without motion

Syntax	Return Value
SD(15)	<i>Returns information that "test without motion" is active:</i> 0 = No 1 = Yes

Tab. 7-47: SD(15)

SD(17,...) - Channel index of a coordinate with specified meaning

Syntax	Return Value	
SD(17,<Class>,1)	Returns the channel-related index (1..8) of the coordinate with the meaning <Class> at the point of block preparation.	
SD(17,<Class>,2)	Returns the channel-related index of the coordinate with the meaning <Class> at the active point of time (block execution).	
	<Class>	Coordinate description: 1 = "X" 2 = "Y" 3 = "Z"

Tab. 7-48: SD(17,...)

SD(20,...) - Number of axes in calling channel

Syntax	Return Value
	Returns the number of synchronous axes in the calling channel:
SD(20) or SD(20,1)	at the point in time of block preparation.
SD(20,2)	At the active point in time (block execution).

Tab. 7-49: SD(20,...)

SD(21,...) - Number of axes in a channel

Syntax	Return Value	
	Returns the number of synchronous axes of a channel:	
SD(21,<Chan>) or SD(21,<Chan>,1)	at the point in time of block preparation.	
SD(21,<Chan>,2)	At the active point in time (block execution).	
	<Chan>	1..60 = Channel index

Tab. 7-50: SD(21,...)



If the specified channel is not active, axes of this channel can already be assigned (they are currently active in another channel). Nevertheless, the axes belong to the specified channel.

Example: Axis X2 belongs to channel 2 (not active) and X2 is currently traversed synchronously in channel 1. In the instructions SD(21,2,...), the axis X2 is seen as part of channel 2.

SD(22,...) - Channel axis number of a system axis of the calling channel

Syntax	Return Value	
	It provides the channel axis number of a system axis of the calling channel or -1:	
SD(22,<SysAxis>) or SD(22,<SysAxis>, 1)	at the point in time of block preparation.	
SD(22,<SysAxis>, 2)	At the active point in time (block execution).	
	<SysAxis>	1..250 = System axis index or system axis name in inverted commas.

Tab. 7-51: SD(22,...)

SD(23,...) - System axis number of a channel axis of the calling channel

Syntax	Return Value	
	Returns the system axis number of a channel axis of the calling channel or -1:	
SD(23,<ChanAxis>) or SD(23,<ChanAxis>, 1)	at the point in time of block preparation.	
SD(23,<ChanAxis>, 2)	At the active point in time (block execution).	
	<ChanAxis>	1..8 = Channel axis index or channel axis name in inverted commas.

Tab. 7-52: SD(23,...)

SD(24,...) - System axis number of a channel axis

Syntax	Return Value	
	Returns the system axis number of a channel axis or -1:	
SD(24,<ChanAxis>,<Chan>) or SD(24,<ChanAxis>,<Chan>,1)	at the point in time of block preparation.	
SD(24,<ChanAxis>,<Chan>,2)	At the active point in time (block execution).	

Syntax	Return Value	
	<ChanAxis>	1..8 = Channel axis index or channel axis name in inverted commas.
	<Chan>	1..60: Channel index

Tab. 7-53: SD(24,...)



If the specified channel is not active, axes of this channel can already be assigned (they are currently active in another channel). Nevertheless, the axes belong to the specified channel.

Example: Axis X2 belongs to channel 2 (not active) and X2 is currently traversed synchronously in channel 1. In the instructions SD(24,2,...), the axis X2 is seen as belonging to channel 2.

SD(25,...) - Channel of system axis

Syntax	Return Value	
SD(25,<SysAxis>)	Returns the channel of a system axis at the active point in time. For asynchronous axes, "-1" is returned.	
	<SysAxis>	1..250 = System axis index or system axis name in inverted commas.

Tab. 7-54: SD(25,...)

SD(30) - Triggering event for init program call

Syntax	Return value
SD(30)	Returns the triggering event to call the channel-specific init program in the respective channel: • 1: after program end (M30) of a main program • 2: after channel reset • 3: after system reset • 0: not called as init program but as NC program SD(30) is intended for use in init programs.

Tab. 7-55: SD(30)

SD(60,...) - Spline data

Syntax	Return Value
SD(60,1,<Chan>,<BlkType>)	Spline ID programmed with SplineDef(<Spline ID>).
SD(60,11,<Chan>,<BlkType>)	Spline ID used by LCTS
	Also refer to SDR(60) Values >= 0: No error Wert = -2: NC block could not be determined Value = -3: incorrect BlkType

Syntax	Return Value	
	<Chan>	Optional: Channel index or -1 for own channel Default: -1
	<BlkType>	Optional: NC block selection: 1: Block preparation 2: Block execution -1: Re-entry block -2: Freeze block Default: 1 for Chan = Own channel 2 for Chan = Third-party channel

Tab. 7-56: SD(60,...)



BlkType = 1 may not be used for the third-party channel. Return value is -3.

BlkType = -1, -2 return only reasonable return values for "Cancel distance to go", "Block pre-run" and "Asup". Otherwise, -2 results.

SD(68,...) - Total of the program coordinate offset

Syntax	Return value	
	Provides the total (Trans + ATrans) of the program coordinate offset for one channel coordinate	
SD(68,1,<CoordIndex>)	At the point in time of the block preparation (latest programmed value).	
SD(68,2,<CoordIndex>)	At the active point in time (block execution).	
	<CoordIndex>	1..8 = Coordinate index

Tab. 7-57: SD(68,...)

SD(77) - Point of return to contour for asynchronous subroutine

Syntax	Return Value
SD(77)	<i>Returns the currently set approaching point of an asynchronous subroutine:</i> • 1 = Starting point • 2 = End point • 3 = Interruption point

Tab. 7-58: SD(77)

SD(131,...) - Tangential tool guidance

Syntax	Return Value
SD(131,1)	Returns the channel axis number of the rotary axis
SD(131,2)	Returns the tool symmetry.

Tab. 7-59: SD(131,...)

SD(168,...) - Program coordinate offset

Syntax	Return Value
	Returns the value of the program coordinate offset (Trans) for one channel coordinate
SD(168,1,<CoordIndex>)	At the point in time of the block preparation (latest programmed value).
SD(168,2,<CoordIndex>)	At the active point in time (block execution).
	<CoordIndex> 1..8 = Coordinate index

Tab. 7-60: SD(168,...)

SD(200,...) - Working areas and dead areas

Syntax	Return Value
SD(200,1)	Returns the number of active areas in the channel.
SD(200,1,<Area>)	<i>State of area in the channel</i> 1: area is active in the channel. 0: area is inactive in the channel.
SD(200,3,<Area>)	<i>Type of range:</i> 0: not defined 1: dead area 2: Workspace
SD(200,11,<Area>)	Area center point in programming units (first area axis)
SD(200,12,<Area>)	Area center point in programming units (second area axis)
SD(200,21,<Area>)	Area extension in programming units (first area axis)
SD(200,22,<Area>)	Area extension in programming units (second area axis)
SD(200,31,<Area>)	System axis number (first area axis)
SD(200,32,<Area>)	System axis number (second area axis)
	<Area> 1..32 = area index

Tab. 7-61: SD(200,...)

SD(202,...) - Spindle override

Syntax	Return Value
SD(202,<ChanSpindle>)	Returns the active override value for a channel spindle.
	<ChanSpindle> 1..8 = Channel spindle number

Tab. 7-62: SD(202,...)

SD(205,...) - Speed of channel spindle

Syntax	Return Value	
SD(205,1,<ChanSpindle>)	Returns the active command value speed including override.	
SD(205,2,<ChanSpindle>)	Returns the last programmed speed.	
SD(205,3,<ChanSpindle>)	Returns the current actual speed.	
	<ChanSpindle>	1..8 = Channel spindle number

Tab. 7-63: SD(205,...)

SD(211,...) - Channel spindle numbers of a spindle group

Syntax	Return Value	
	Returns the current configuration of a spindle group at the time of block preparation as zipped list.	
SD(211,<Group>,0)	Returns number of members of the group (0 .. 8).	
SD(211,<Group>,<N>)	Returns the channel spindle number of the N. member of the group.	
	<Group>	1..4 = group number
	<N>	1..8 = member number

Tab. 7-64: SD(211,...)

SD(212,...) - Motion function of a channel spindle

Syntax	Return Value	
	Returns the motion function of a channel spindle	
SD(212,1,<ChanSpindle>)	at the active point in time.	
SD(212,2,<ChanSpindle>)	at the point in time of block preparation.	
	<ChanSpindle>	1..8 = Channel spindle number

Tab. 7-65: SD(212,...)

SD(222,...)- Channel spindle number of system spindle

Syntax	Return Value	
	Returns the channel spindle number of a system spindle of the active channel or -1.	
SD(222,<SysSpindle>) or SD(222,<SysSpindle>,1)	Value at the time of block preparation.	
SD(222,<SysSpindle>,2)	Value at the active point in time.	
	<SysSpindle>	System spindle name or 1..60 = System spindle number

Tab. 7-66: SD(222,...)

SD(223,...) - System spindle number of a channel spindle

Syntax	Return Value	
	Returns the system spindle number of a channel spindle of the active channel or -1.	
SD(223,<ChanSpindle>) or SD(223,<ChanSpindle>,1)	Value at the time of block preparation.	
SD(223,<ChanSpindle>,2)	Value at the active point in time.	
	<ChanSpindle>	"S1"..."S8" or 1..8 = Channel spindle number

Tab. 7-67: SD(223,...)

SD(230,...) - System axis number of system spindle

Syntax	Return Value	
SD(230,<SysSpindle>)	Returns the system axis number of a system spindle or -1.	
	<SysSpindle>	System spindle name or 1..60 = System spindle number

Tab. 7-68: SD(230,...)

SD(231,...) - System axis number of a channel spindle

Syntax	Return Value	
SD(231,<ChanSpindle>)	Returns the system axis number of a channel spindle or -1.	
	<ChanSpindle>	"S1"..."S8" or 1..8 = Channel spindle number

Tab. 7-69: SD(231,...)

SD(232,...) - System spindle number of a system axis

Syntax	Return Value	
SD(232,<SysAxis>)	Returns the system spindle number of a system axis or -1.	
	<SysAxis>	1..250 = System axis index or system axis name in inverted commas.

Tab. 7-70: SD(232,...)

SD(268,...) - Additive program coordinate offset

Syntax	Return Value	
	Returns the value of the additive coordinate offset (ATrans) for a coordinate of the channel	
SD(268,1,<CoordIndex>)	At the point in time of the block preparation (latest programmed value).	

Syntax	Return Value	
SD(268,2,<CoordIndex>)	At the active point in time (block execution).	
	<CoordIndex>	1..8 = Coordinate index

Tab. 7-71: SD(268,...)

SD(328,...) - Accuracy programming

Syntax	Return Value
SD(328,1) or SD(328)	Last programmed accuracy limit of the NC function "PrecProg".
SD(328,2)	Last programmed corner distance of the NC function "PrecProg".

Tab. 7-72: SD(328,...)

SD(581,...) - Channel axis coupling

Syntax	Return Value	
SD(581,0,<Master>)	- Channel axis number (<Master>), if <Master> is a master axis. 0: If <Master> is not a master axis.	
SD(581,<Slave>,0)	- Channel axis number of the master axis for which the <Slave> axis is a slave axis. 0: If <Slave> is not a slave axis.	
SD(581,<Slave>,1)	- Programmed slave axis offset in programming units for the <Slave> axis. 0: If <Slave> is not a slave axis.	
SD(581,<Slave>,2)	- Programmed coupling factor for the <Slave> axis. 0: If <Slave> is not a slave axis.	
SD(581,<Slave>,3)	- Programmed master axis offset in programming units for the master axis of the <Slave> axis . 0: If <Slave> is not a slave axis.	
	<Master>	Channel axis index of the master index (1..8).
	<Slave>	Channel axis index of the slave index (1..8).

Tab. 7-73: SD(581,...)

SD(594,...) - Feed group

Syntax	Return Value	
	Provides information on the feed groups (FeedGrp) and feed adaptation (FeedAd).	
SD(594,<Coord>,0)	FeedGrp and FeedAd: 2: Coordinate is part of the current feed group. 1: Coordinate is moved synchronously 0: Coordinate is a slave (e.g., of AXC) -1: Coordinate does not exist in the channel. E.g. SD(594,"Y",0)	
SD(594,<Coord>,1)	FeedGrp: 1: FeedGrp is active and the coordinate is a programmed member. 0: No member or FeedGrp is active -1: Coordinate does not exist in the channel. e.g. SD(594,"Y",1)	
SD(594,<Coord>,2)	FeedAd: 2: FeedAd is active and the coordinate is a programmed member. 1: FeedAd is active and the coordinate is a member via MP 0: FeedAd is not active or not a member -1: Coordinate does not exist in the channel. e.g. SD(594,"Y",2)	
	<Coord>	Channel axis index (1..8) or Coordinate name or 9 (= Orientation coordinate O)

Tab. 7-74: SD(594,...)

SD - Programming example

Examples:

```

30 A% = SD(2,1)    A% contains the active position of the feed potentiometer
:                  in percent.
40 B% = SD(5,1,1)  B% contains the active feed rate.
:
```

Program example:

SD (9)

```

N4 G75.2 X120
60 IF SD(9) = 1 THEN
N7 (MSG, touch probe 2 has not triggered)
80 GOTO .ERROR
```

```

90 ELSE
100 XMESS = PPOS(1)
110 ENDIF

```

In the programming example "SD (9)", the x-axis is traversed in the direction of the specified position. If the position is reached and the touch probe has not triggered, a message (line N7) is output and it is jumped to label .ERROR. If the touch probe was triggered, the current position, based on the program coordinate system in XMESS, can be saved.

SDR (general description)

REAL	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **System Data Real** - System-internal data in Real_Format

Reads from a collection of system data of the MTX CNC system in **REAL format**. The format and application of the command correspond to the SD command.

Syntax:

SDR (<Group>[,<Index1>[,<Index2>]])
The SDR command outputs REAL values.

Tab. 7-75: Syntax SDR

SD(1,...) - Target position after block pre-run

Syntax	Return Value
SDR(1,<CoordIndex>)	Workpiece processing of all coordinates calculated in the block pre-run/re-entry. If there is no block pre-run, "0" is returned. If an unapplied axis, coordinate or an auxiliary axis is addressed, a runtime error is caused.
	<CoordIndex> 1..8 = Coordinate index

Tab. 7-76: SDR(1,...)

SDR(2,...) - Current override position

Syntax	Return Value
SDR(2,1)	Returns the current position of the feed override in the current channel in percent.
SDR(2,3)	Returns the current position of the override for spindle 1 in percent. 0 if no spindle is applied.
SDR(2,4)	Returns the current position of the override for spindle 2 in percent. 0 if no spindle is applied.

Tab. 7-77: SDR(2,...)

SDR(5,...) - Velocity

Syntax	Return Value
SDR(5,1,1)	Returns the active feed velocity (100% value) rounded to an integer value in mm/min or inch/min.
SDR(5,2,1)	Returns the active rapid traverse velocity (100% value) rounded to an integer value in mm/min or inch/min.
SDR(5,1,2)	Returns the last programmed feed in input unit per minute.

Tab. 7-78: SDR(5,...)

SDR(60,...) - Spline data

Syntax	Return Value
SDR(60,2,<Chan>)	Tolerance value of the B-spline approximation programmed with BAP(E..)
SDR(60,3,<Chan>)	Orientation tolerance value of the B-spline approximation programmed with BAP(OE..) or -1.0, if BAP(OE...) has not been programmed
	The return value refers to the preparation block in the own channel and to the active block in the third-party channel Also refer to SD(60,1) Values >= 0: Tolerance value of B-spline approximation Value = -1.0: BAP(OE...) was not programmed Wert = -2.0: NC block could not be determined
	<Chan> Optional: Channel index or -1 for own channel Default: -1

Tab. 7-79: SDR(60,...)

SDR(68,...) - Total of program coordinate offset

Syntax	Return Value
	Provides the total (Trans + ATrans) of the program coordinate offset for one channel coordinate
SDR(68,1,<CoordIndex>)	At the point in time of the block preparation (latest programmed value).
SDR(68,2,<CoordIndex>)	At the active point in time (block execution).
	<CoordIndex> 1..8 = Coordinate index

Tab. 7-80: SDR(68,...)

SDR(77,...) - Returning to contour after Asup

Syntax	Return Value
	Returns the coordinates of the possible approaching points after an Asup: The returned position permits repositioning of the coordinates in the vicinity of the desired approaching point in one asynchronous subroutine using "G76".
SDR(77,<CoordIndex>,0) or SDR(77)	Returns the current approaching point according to ASPRTP/REPOSTP of a machine coordinate in the freeze block.
SDR(77,<CoordIndex>,1)	Returns the starting point of a machine coordinate in the freeze block.
SDR(77,<CoordIndex>,2)	Returns the end point of a machine coordinate in the freeze block.
SDR(77,<CoordIndex>,3)	Returns the interruption point of a machine coordinate in the freeze block.
	<CoordIndex> Machine coordinate index

Tab. 7-81: *SDR(77,...)***SDR(78,...) - Returning to contour after Asup**

Syntax	Function value
	Returns the coordinates of the possible approaching points in a WCS after an Asup: The returned positions facilitates the calculation of the contour offset in the WCS between approaching position and current position.
SDR(78,<CoordIndex>,0) or SDR(78)	Returns the current approaching point according to ASPRTP/REPOSTP of a workpiece coordinate in the freeze block.
SDR(78,<CoordIndex>,1)	Returns the starting point of a workpiece coordinate in the freeze block.
SDR(78,<CoordIndex>,2)	Returns the end point of a workpiece coordinate in the freeze block.
SDR(78,<CoordIndex>,3)	Returns the point of interruption of a workpiece coordinate in the freeze block.
	<CoordIndex> Workpiece coordinate index

Tab. 7-82: *SDR(78,...)*

SDR(100) - Re-entry points for block pre-run

Syntax	Function value
SDR(100,<CoordIndex>,0)	Returns the point of interruption (WCS) of the call chain during re-entry.
SDR(100,<CoordIndex>,1)	Returns the reference point (WCS) of the call chain during re-entry.
SDR(100,<CoordIndex>,2)	Returns the point of interruption (ACS) of the call chain during re-entry.

Tab. 7-83: SDR(100,...)

SD(131,...) - Tangential tool guidance

Syntax	Return Value
SDR(131,3)	Returns the positioning angle in degrees
SDR(131,4)	Returns the intermediate block angle in degrees.

Tab. 7-84: SDR(131,...)

SD(168,...) - Program coordinate offset

Syntax	Return Value
	Returns the value of the program coordinate offset (Trans) for one channel coordinate
SDR(168,1,<CoordIndex>)	At the point in time of the block preparation (latest programmed value).
SDR(168,2,<CoordIndex>)	At the active point in time (block execution).
	<CoordIndex> 1..8 = Coordinate index

Tab. 7-85: SDR(168,...)

SDR(202,...) - Spindle override

Syntax	Return Value
SDR(202,<ChanSpindle>)	Returns the active override value for a channel spindle.
	<ChanSpindle> 1..8 = Channel spindle number

Tab. 7-86: SDR(202,...)

SDR(205,...) - Speed of channel spindle

Syntax	Return Value
SDR(205,1,<ChanSpindle>)	Returns the active command value speed including override.
SDR(205,2,<ChanSpindle>)	Returns the last programmed speed.
SDR(205,3,<ChanSpindle>)	Returns the current actual speed.
	<ChanSpindle> 1..8 = Channel spindle number

Tab. 7-87: SDR(205,...)

SDR(219,...) - Orienting position of a channel spindle

Syntax	Return Value	
SDR(219,<ChanSpindle>)	Last programmed spindle position at M19 (if M19 is not programmed, 0 is returned).	
	<ChanSpindle>	1..8 = Channel spindle number

Tab. 7-88: SDR(219,...)

SDR(268,...) - Additive program coordinate offset

Syntax	Return Value	
	Returns the value of the additive program coordinate offset (ATrans) for a coordinate of the channel	
SDR(268,1,<CoordIndex>)	At the point in time of the block preparation (latest programmed value).	
SDR(268,2,<CoordIndex>)	At the active point in time (block execution).	
	<CoordIndex>	1..8 = Coordinate index

Tab. 7-89: SDR(268,...)

SDR(300) - Epsilon environment around the north pole

Syntax	Return Value
SDR(300)	Size of the active epsilon environment around the north pole of an active 5-axis transformation in degrees. The active value can be greater than the value set in EpsAx-Trafo (MP 1030 00160).

Tab. 7-90: SDR(300)

SDR(328,...) - Accuracy programming

Syntax	Return Value
SDR(328,1) or SDR(328)	Last programmed accuracy limit of the NC function "PrecProg".
SDR(328,2)	Last programmed corner distance of the NC function "PrecProg".

Tab. 7-91: SDR(328,...)

SDR(581,...) - Channel axis coupling

Syntax	Return Value
SDR(581,0,<Master>)	- Channel axis number (<Master>), if <Master> is a master axis. 0: If <Master> is not a master axis.
SDR(581,<Slave>,0)	- Channel axis number of the master axis for which the <Slave> axis is a slave axis. 0: If <Slave> is not a slave axis.

Syntax	Return Value	
SDR(581,<Slave>,1)	- Programmed slave axis offset in programming units for the <Slave> axis. 0: If <Slave> is not a slave axis.	
SDR(581,<Slave>,2)	- Programmed coupling factor for the <Slave> axis. 0: If <Slave> is not a slave axis.	
SDR(581,<Slave>,3)	- Programmed master axis offset in programming units for the master axis of the <Slave> axis. 0: If <Slave> is not a slave axis.	
	<Master>	Channel axis index of the master index (1..8).
	<Slave>	Channel axis index of the slave index (1..8).

Tab. 7-92: SDR(581,...)

7.4.5 System data of structured types

General information

Structured system data (abbreviated SD) forms a class of data in the overall system with the following properties:

- Individual SD can either be located in the volatile or nonvolatile memory (permanent SD).
- The number, size and structure can be freely defined within the existing memory.

Permanent system data is initialized again once while volatile system data is initialized again at each system startup. The values can be specified in an initialization file. If nothing is specified, the values are preset to "0".

Structured variables "SV" can be replaced by SDs. However, they are retained due to reasons of compatibility.

For comprehensive information on the use and definition of system data, refer to the documentation "MTX Functional Description".

Access to system data

Description: The keyword SD is used to access in CPL. As opposed to XPath addressing (standardized access to structured data), the point operator "." is used in place of the separator "/". Arrays are addressed with [], e.g.:

Xpath: /MyArrVar[2,3,4]/subcomponent

CPL: SD.MyArrVar[2,3,4].subcomponent

Example: 10 SD.MyChanVar=DBSEA("/dbt1/Rec",-1,-1,"Key1=1",I%)

For channel-specific SDs, the channel index can be omitted, e.g.:

SD.MyArrVar[,3,4] or

SD.MyChanVar without [] for a one-dimensional channel SD.

Any CPL expression providing an integer value can be programmed within the square brackets.

SDs with the same structure types can be assigned to each other.

The following assignment rules apply to basic types:

CPL commands

Name	Length in bytes	Description	C-type	IndraLogic type	CPL
String	2 * maxLength + 1	String (UTF-8 format)	char []	String()	CHAR
IsoLatin1String	maxLength + 1	String (Latin1 format)	char []	String()	CHAR
Byte_t	1	Signed 8 bit integer	char	SINT	INT
Short_t	2	Signed 16-bit integer	short	INT	INT
Int_t	4	Signed 32-bit integer	int	DINT	INT
UnsignedByte_t	1	Unsigned 8 bit integer	unsigned char	USINT	INT
UnsignedShort_t	2	Unsigned 16-bit integer	unsigned short	UINT	INT
UnsignedInt_t	4	Unsigned 32-bit integer	unsigned int	UDINT	INT
Float_t	4	32 bit real	float	REAL	REAL
Double_t	8	64-bit real	double	LREAL	DOUBLE
Boolean_t	1	true, false, 1, 0	char	BOOL	BOOL

Tab. 7-93: Assignment rules for basic types

VARINF

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **VARiable INFormation** - Variable information

Using the **VARINF** command, a query can be submitted to the part program whether it contains a structured system date or a permanent variable. The command provides 1 as result if the entered variable exists. If the variable does not exist, the command provides 0.

Syntax:

VARINF(<VariableName>)	
with	
<Variable name>	Name of the structured system date or permanent variable.

Tab. 7-94: VARINF syntax

Example:

```

10 DIM A$(20)
10 I% = VARINF("@_RES_DOUBLE")
...
40 A$ = "SD.SysSRun[1].Active"
50 J% = VARINF(A$)
...
```

VARVAL

MULTYP	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **VARiable VALue** - Variable value

The CPL command returns the content of a system data variable or a permanent variable. The variable name is transferred as character string expression. Thus, the variable can be indirectly addressed. Additionally, the variable does not have to exist to link the program.

If the specified variable does not exist or has an invalid type, a runtime error is generated. To check in advance whether a system data variable or a per-

manent variable exists, use the `VARINF` ([chapter "VARINF" on page 476](#)) command.

The `VARVA` command supports the `ERRNO` parameter.

Syntax:

<code>VARVAL(<VariableName>)</code>	
with	
<code><Variable name></code>	Name of the structured system data or permanent variable.
<code>ERRNO</code>	CPL variable If the CPL variable <code>ERRNO</code> is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. <i>The following return values are possible:</i> • 0: Access OK. • <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69) If the CPL variable <code>ERRNO</code> is not specified, a runtime error is generated in case of an access error.

Tab. 7-95: `VARVAL` syntax

Example:**Program:**

```
1 DIM A$(30) : DIM B$(30) : IDX%=4
1 A$="SD.MySDVar[" : B$="].Active"
1 RES% = VARVAL(A$+STR$(IDX%)+B$)
; in RES% is the content of SD.MySDVar[4].Active
```

Restrictions:

- Only system data variables and permanent variables are supported (neither global nor local CPL variables).
- The transferred character string is the variable name and not the expression to be calculated.
- If an array variable is accessed, all indices have to be available as constants (digits) and not as expressions.
- Only simple types can be read (INTEGER, REAL, DOUBLE, BOOLEAN, STRING) (neither arrays nor structures).

SDLOAD

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:**System Data LOAD** - Load structured system data

The CPL command `SDLOAD` can be used to read system data from a file. Several root elements can be specified. The specified subcomponents are ignored. All data starting from the root element are always read.

Root array elements can be read completely (without array index) or individually (with array index).

Syntax

<code>SDLOAD (<FileName>,<Root-Element_1>[,...,<Root-Element_n>])</code>	
<code><FileName></code>	Directory path and file name of the target file If no path is specified, the file is read out of the current directory.

<Root element>	Root element of the data group to be read
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error -7: Data could not be read -9: Access to source file not possible -10: Target file including path is too long -20: Incorrect root tag -21: File access not possible -35: Internal error at system data access If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-96: *SDLOAD syntax***Syntax example:**

```
SDLOAD("/usrfep/SDDatUsr.xml", "DgnUsr[1]", ERRNO)
```

The user-specific system data "DgnUsr[1]" is read from the file "SDDatUsr.xml" and loaded to the "/usrfep" (user FEPRM).

SDSAVE

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **System Data SAVE** - Save structured system data

Using the CPL command SDSAVE, system data is saved in a **newly created** file. Files in an already existing file with the same name are overwritten. Several root elements can be specified. The specified subcomponents are ignored. All data starting from the root element are always saved.

Root array elements can be saved (without array index) or individually (with array index).

Syntax

```
SDSAVE (<FileName>, <Root-Element_1>[, ..., <Root element_n>])
```

<FileName>	Directory path and file name of the target file Without path specification, it is searched for the file in the current directory.

<Root element>	Root element of the data group to be saved
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error -10: Target file including path is too long -13: Data could not be written -20: Incorrect root tag -21: File access not possible -35: Internal error at system data access If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-97: Syntax of SDSAVE

Syntax example:

```
10 SDSAVE("cplsave.xml", "SysAPrg", ERRNO)
```

The complete root element "SysAPrg" is saved in the file "cplsave.xml" in the current directory.

```
20 SDSAVE("SDDatUsr.xml", "DgnUsr[1]", ERRNO)
```

The individual root array element "DgnUsr[1]" is saved in the "SDDatUsr.xml" file in the current directory.

```
30 SDSAVE("/SDDatUsr.xml", "DgnUsr", ERRNO)
```

All root array elements "DgnUsr" are saved in the "SDDatUsr.xml" file in the root directory.

7.4.6 Sercos Interface

SCS3

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: SerCoS 3 - Sercos - Read parameters

Allows read-only access to Sercos drive parameters with extended ident number.

Syntax:

```
SCS3 (<AxisSelection>, <EID-no>)
```

<Axis selection>	System axis index or system axis name

<EID no>	Extended Sercos ident number (S-x-xxxx.xx.xx, P-x-xxxx.xx.xx)
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error -7: Data could not be read -10: Target file including path is too long -21: File access not possible -27: Command with invalid parameter type -30: Insufficient memory -36: Service channel blocked -37: Unknown ident number -38: Invalid ident number -39: Invalid axis number If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-98: SCS3 syntax

The parameter content is provided without unit or scaling.



The command is only available for Sercos drives with an extended ident number.



Parameters containing a list (multiple values separated by commas) cannot be read. In such cases, the control issues an error message.



Using the Sercos multiplex channel accelerates the data transfer.

If the drive data is in the Sercos drive telegram, this data is read from the telegram (refer to the parameter manual of the Sercos drive). Otherwise, the drive data is read directly in the drive.

If **other** applications access drive data, such drive data cannot be accessed at the **same** time. The parameter ERRNO can be used to respond to this error in the part program. Repeating the access can provide the desired drive data.



Continuous access to drive data can prevent access for other applications!

A response to the error in the part program can be provided by evaluating the error variable ERRNO.

Example:

```
10 LAGE%=SCS3 (1, "S-0-51.0.0", ERRNO)
```

The actual position value of the first axis is assigned to the POSITION% integer variable.

```
12 IF ERRNO = 0 THEN
```

Error evaluation

```
13 REM***Actual position value was read correctly***
```

```
14 ELSE
```

```
15 REM***Actual position value could not be read***
```

```
16 ENDIF
```

SCS3L

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **SerCoS 3 List** - Sercos - Read parameter lists

Certain Sercos parameters are stored as lists. These lists can be read using the command "SCS3L". Since the length (required memory) for a list is unknown, the read list elements are stored in ASCII files. Subsequently, the read data can be processed using CPL file commands.

By the SCS3L command, the file specified in the command is created if it does not yet exist. The content of an existing file is overwritten.

Syntax:

SCS3L (<AxisIndex>,<EID no>,<FileName>)]	
<Axis index>	System axis index or system axis name
<EID no>	Extended Sercos ident number (S-x-xxxx.xx.xx, P-x-xxxx.xx.xx)
<FileName>	Name of the ASCII file in which the read list is to be stored.
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error -7: Data could not be read -10: Target file including path is too long -21: File access not possible -27: Command with invalid parameter type -30: Insufficient memory -36: Service channel blocked -37: Unknown ident number -38: Invalid ident number -39: Invalid axis number If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-99: SCS3L syntax

It can sometimes be impossible to access drive data if other applications access drive data. The parameter <ERRNO> can be used to respond to this

error in the part program. Repeating the access can provide the desired drive date.



The command is only available for Sercos drives with an extended ident number



Continuous access to drive data can impede access for other applications.

Example:

```
10 SCS3L(1,"P-0-0021","OUTFILE.txt")
```

Write list of ident numbers for the axis oscilloscope of drive 1 to file "OUTFILE.txt".

WSCS

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Facilitates the writing access to Sercos drive parameters.

Apart from individual values, value lists with a maximum of 16 values can be written.

Prerequisites:

- The parameter must be valid and editable in Sercos Phase 4.
- The parameter value must be within the permitted range.
- The drive must be connected to the control using the Sercos interface; cyclic communication between the control and the drive is running.



When calling WSCS, no NC blocks can be processed. This can be achieved by using a WAIT command.



WSCS can be used optionally to the NC function "Write-ID3(WID3)".



To read the Sercos drive parameters, the CPL command "SCS3" is available.



Improper or random modifications of Sercos drive parameters can lead to damage on the workpiece and/or machine and to dangerous and unpredictable machine reactions.

Information on the Sercos drive parameters available in the drive is required for proper use. For this, consult your drive documentation.

Syntax:

WSCS (<Axisselection>,<ID-No>,<Arraylength>,<Value(s)>,{<Unit>})	
<Axis selection>	System axis index or system axis name
<ID no>	Sercos ident number (S-x-xxxx, P-x-xxxx, S-x-xxxx.xx.xx, P-x-xxxx.xx.xx)

<Field length>	0 Write individual parameters 1..16 Write list parameters
<Value (s)>	For array length = 0: A CPL expression For array length = 1..16: Name of an array variable
{<Unit>}	Optional. 1: (Default) Writing by taking into consideration the unit and the scaling 0: Writing without unit and scaling
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error -13: Data could not be written -26: Command with invalid parameter value -27: Command with invalid parameter type -36: Service channel blocked -37: Unknown ident number -38: Invalid ident number -39: Invalid axis number -41: Command not permitted -42: Limit value violated -43: Invalid element number If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-100: Syntax WSCS

Write individual parameters: With <ArrayLength> **0**, individual parameters are written.
 <Value (s)> can be an arbitrary CPL expression.

A parameter of the 5th system axis is written with a value. In case of an assigned service channel, it is waited for 100 ms and the writing operation is repeated:

```

...
N10 WAIT
000 .WRTORQUE
001 WSCS (5,"S-0-92",0,400.0,ERRNO)
002 IF ERRNO <> 0 THEN
003   IF ERRNO = -36 THEN
004     WAIT(,100)
005     GOTO .WRTORQUE
006   ELSE

```

```

007 REM error handling
009 ENDIF
010 ENDIF
...

```

Fig. 7-3: Write individual parameters (example 1)

A parameter of the channel axis Z is written:

```

...
N10 WAIT
000 SAX% = SD(23,"Z")
001 WSCS(SAX%,"S-0-92",0,400.0,ERRNO)
002 IF ERRNO <> 0 THEN
007 REM error handling
010 ENDIF
...

```

Fig. 7-4: Write individual parameters (example 2)

Write parameter list

Parameter lists are written by transmitting the name and the number of the used elements of a field variable.

The <Field length> (1 to 16) specifies the number of the list element to be written.

<Value (s)> has to be name of a one-dimensional array variable.

In the following program section, the field variable `VALUE!` is created and written with values.

The name of the field variables and the number of elements to be used is transmitted to `WSCS`.

Program:

```

...
N10 WAIT
000 DIM VALUE!(5)
001 VALUE!(1) = 3.5
002 VALUE!(2) = 0.2
003 WSCS("X", "P-0-4006", 2, VALUE!, ERRNO)
004 IF ERRNO <> 0 THEN
005 REM error handling
006 ENDIF
...

```

Fig. 7-5: Write parameter list

Special features: With {<unit>} = 0, parameter values read by "SCS3" can be written to the drive again.

7.4.7 PLC interface

BITIF

Description: **BIT InterFace** - Bit interface

This command allows to access the digital interface between the NC and PLC:

- All inputs and outputs can be read.
- CPL customer outputs Ch_Cpl01 - 16 in the channel interface can also be written.

Syntax:

BOOL	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

```

BITIF (<BitSignal>[,<Index>[,<IF Unit>]])

```

```

with

```

<Bit signal>	Bit offset or symbolic address Bit offset: 0...31 for global interface 0...111 for channel interface 0...95 for axis interface 0...95 for spindle interface 0...7 for high-speed interface For the symbolic addresses, refer to the "MTX PLC Interface" documentation.
<Index>	Index of the axis, the spindle or the channel <i>If no index is programmed, the following applies for the query:</i> • Active channel for channel interface • Axis 1 for axis interface • Spindle 1 for spindle interface.
<IF unit>	0 or QCH: PLC output signals, channel-specific 1 or QAX: PLC output signals, axis-specific 2 or QSP: PLC output signals, spindle-specific 3 or ICH: PLC input signals, channel-specific (incl. CPL customer outputs of the NC) 4 or IAX: PLC input signals, axis-specific 5 or ISP: PLC input signals, spindle-specific 6 or QGEN: PLC output signals, global 7 or IGEN: PLC input signals, global 8 or QHS: PLC input signals, high-speed 9 or IHS: PLC axis signals, high-speed

Tab. 7-101: Syntax of BITIF

Syntax examples:

<code>I?=BITIF("QGEN_RESET")</code>	Reads the global output signal "system reset"
<code>I?=BITIF("ICH_RESET",1)</code>	Reads the channel input signal "channel reset" of channel 1
<code>I?=BITIF(0,1,"IAX")</code>	Reads the axis input signal "reference point known" for axis 1
<code>I?=BITIF(88,2,5)</code>	Reads the spindle input signal "Error of state class-1" for spindle 2
<code>BITIF("ICH_CPL05")=TRUE</code>	Setting CPL customer output 5 to TRUE
<code>BITIF(50,1,3)=TRUE</code>	
<code>BITIF("ICH_CPL02")=FALSE</code>	



In case of high-speed signals, "QHS" (8) is specified for input signals as IF unit and "IHS" (7) is specified for output signals.

PLC

MULTYP	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Programmable Logic Controller** - Access to PLC operands

This command is used to access PLC operands.

Syntax:

PLC (<Type>,<EmptyParam.>,<Address>,<Size>)	
with	
<Type>	1: Input (E) 2: Output (A) 3: Flag (M) Read-only access is permitted for all types. Write-access is only permitted to flags.
<empty parameter>	(not assigned)
<Address>	Relevant byte address starting at the beginning of the range The control checks the parameter according to the active PLC.
<Size>	Size of the data type: 1: byte 2: Word 4: double-length word
PLC (<Type>, <EmptyParam.>, <Address>, <Size> [, <DataType>])	
<Data type>	optional: Type to access the operands on the PLC 0: INTEGER (default) 1: REAL
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible 0: Access OK. <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69) If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-102: PLC syntax



When using the <Data type> parameter, the same type has to be used when accessing (read and write) the same operand!

Example:

```
:
30 REM Read 2 byte ab from input 10
40 I% = PLC(1,,10,2)
:
```


7.4.8 Active functions

NCF

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **NC Functions** - Active NC functions and modal auxiliary functions

Returns the syntax of the latest programmed NC function within the modal group or the auxiliary function group <NC function>. All modal groups of the control as well as the auxiliary functions defined in the auxiliary function groups can be queried.

Optionally, the modal states of the interruption block can also be retrieved within an asynchronous subroutine controlled by a parameter.

A freeze block is

- - with regard to a block pre-run - a previous block of the target block selected
- - with regard to regular asynchronous subroutines - an NC block from where it is jumped to the Asup.

The variable, in which the result is to be stored, has to be of type "dimensioned character array".

Type conflicts are detected at program runtime and acknowledged with an error message.

Syntax:

NCF (<NC Function>)	
NCF (<NC function> [, <DataSelection>])	
<NC function>	Syntax of any NC function or auxiliary function If a non-existent syntax is programmed, a runtime error is displayed.
<DataSelection>	0: (Default) The command provides the latest programmed function. 1: The command provides the active function in the freeze block (only together with block pre-run and asynchronous subroutines).

Tab. 7-103: NCF syntax

Example:

```

10 DIM A$(4)           Dimension character array for a string with up to four characters.
:
20 A$=NCF("G1")        The syntax of the latest programmed NC function of the
:                       group containing "G1" as the syntax is assigned to string
                       variable A$.
N80 [A$]               The previously queried NC function is programmed again.
N10 M3 S1234 T2345
123 DIM A$(4)
20 A$=NCF("M3")
N30 [A$]               The previously queried NC function is programmed again.
50 PRN#(0,"In M3 group ",A$," is active)
N30 M5

```

```

33 A$=NCF("M3")
34 PRN#(0,"In M3 group ",A$," is active")
N70 M30
    
```



The spindle syntax of the MTX CNC system is freely configurable and set in the machine parameters 1040 00101 et seq.

7.4.9 NC interface (NCS)

General information

NCS coupling commands can be used to access process and data services of the internal NCS interface using CPL.

MCODS (general description)

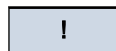
	INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!
Description	Motion Control Data Service - Motion Control Data Services Calls Motion Control data services of the NCS via CPL. Thus, data and states to be read from the CNC.							



All read values refer to the time at which the CPL block is processed by the block preparation.

If the calling program also contains NC blocks, the block preparation is generally performed before processing at the machine. If program executions are to be affected by functions determining current machine data or states, eliminate the "time offset" between the block preparation and the current machine state. However, this applies only to functions that access exactly the channel in which they are executed.

In this case, use the "WAIT" command in the line preceding the call. This stops block preparation until all the blocks before "WAIT" have actually been processed.



The "MCODS" command cannot be used together with block pre-run! The reason is that MCODES accesses data of the active point in time (interpolation). The interpolation is suppressed during block pre-run. Therefore, the active state is still identical with the state at the point of time before the block pre-run when the adjustment starts.

Common CPL functions exist for a high number of "MCODS" commands. These special CPL commands provide identical and similar data. It is also referred to CPL commands in the tabular overview under the column "also refer to"

The commands provide a return value (see [chapter "Error return values" on page 541](#)).

Syntax:

MCODS (<Type>, <Channel>, <Version>, <Buffer>, <Size>, <AxisNumber>, <IdentNumber> [, <P1>])	
<Type>	Integer expression Indicates the function to be executed. The following table lists all the available functions.
<Channel>	Integer expression Indicates the channel to be subject to the function.
<Version>	Initialized integer or real variable (no constant!) If the variable content is 0 when the command is called, the function specified by <Type> immediately writes the requested data to <Buffer>. In addition, the command returns a version ID of the provided data in <Version>. If this version ID is still in the variable the next time the command is called, the command does not write the requested data immediately, but only after the next time the data in <Buffer> is modified. Thus, a program loop can be run until a channel reached a certain state for example. However, include a timeout condition (e.g. counter or elapsed period) in the loop to prevent endless loops!
<Buffer>	The command returns the requested data values in <Buffer>. <i>Depending on the type of data, <Buffer> has to be</i> • a simple variable of type "Integer", "Real", "Double" • an array variable of type "Integer", "Real", "Double" • string variable (one-dimensional character array).

Tab. 7-104: MCODES syntax



For array or string variables, only the variable name without dimension/index may be entered!

<Size>	Integer expression Specifies the array size of <Buffer>. If <Buffer> is not an array variable but rather a simple variable of type "Integer", "Real" or "Double", enter 1 for <Size>.
<Axis number>	Integer expression Specifies the axis number of a system axis.
<Identification number>	Integer expression Provides the value of an <ident number> of the cyclic axis telegram for all axes.

Tab. 7-105: MCODES parameter



The size of an array variable has to be defined first via the DIM command. It must not be exceeded in parameter <Size>!

Dimension these variables before the MCODS command and within the same file!

<P1>	Optional parameter depending of the <Type>.

Tab. 7-106: Syntax 3 MCODS

Command overview
MCODS(...)Function overview
MCODS(...)

Command provides:	Syntax:	See also:
Positions		
Axis command position	MCODS(1..)	SPOS
Axis command position	MCODS(2..)	SPOS
Actual axis position of the drives	MCODS(35..)	APOS
Actual axis position of the CNC	MCODS(147..)	-
Actual axis values (workpiece coordinate system)	MCODS(38..)	-
Lag	MCODS(4..)	-
Axis program value (program coordinate system)	MCODS(37..)	PCS
Programmed end positions including offsets	MCODS(16..)	WCS
Programmed end positions without offsets	MCODS(23..)	PCS
Axis coordinates in ACS1	MCODS(73..)	-
Axis coordinates in ACS0	MCODS(109..)	ACS
Distance to go in the workpiece coordinate system (WCS)	MCODS(83..)	-
Distance to go in the axis coordinate system ACS1	MCODS(85..)	-
Velocity and speed		
Path feed	MCODS(3..)	SDR(5,1,2)
Programmed path feed	MCODS(24..)	SDR(5,1,2)
Programmed path feeds	MCODS(142..)	
Active programmed path feed	MCODS(143..)	
Jog velocities	MCODS(27..)	-
Spindle command speed, spindle cutting velocity	MCODS(5..)	SD(205,2,...)
Actual spindle speed	MCODS(36..)	SDR(205,3,...)
Maximum spindle speed	MCODS(19..)	-
Minimum spindle speed	MCODS(20..)	-
Programmed spindle speeds	MCODS(25..)	SDR(205,2,...)
Current axis velocity	MCODS(119..)	-
Current axis speed	MCODS(120..)	-
Programmed axis speed	MCODS(121..)	-

Command provides:	Syntax:	See also:
Maximum axis velocity	MCODS(131..)	-
Peak value and maximum value of axis velocity	MCODS(145..)	-
Peak value and maximum value of axis acceleration	MCODS(146..)	-
States		
Status "InPos"	MCODS(6..)	BITIF("IAX_INPOS",...)
Status "Approach reference point"	MCODS(26..)	BITIF("IAX_REFKNOWN",...)
Status "test mode" (dry run)	MCODS(29..)	BITIF("ICH_DRYRUN",...)
SAV and IPO state	MCODS(32..)	BITIF("ICH_STATE",...)
Status "dwell time active"	MCODS(39..)	-
Status "auxiliary function active subject to acknowledgement"	MCODS(40..)	-
Status "Block transfer enabled"	MCODS(41..)	BITIF("QCH_TRANSFERLOCK",...)
Status "travel command"	MCODS(47..)	BITIF("IAX_TRVCMD",...)
Status "feed inhibit"	MCODS(49..)	BITIF("QAX_DRVLOCK",...)
Channel waiting states	MCODS(87..)	-
Channel reset; waiting for	MCODS(129,..)	-
Potentiometer		
Values of feed potentiometers	MCODS(7..)	SDR(2,1)
Values of spindle potentiometers	MCODS(8..)	SDR(202,...)
Values of axis potentiometers	MCODS(50..)	-
Corrections		
Active tool radius correction number	MCODS(11..)	TCV(0)
Active tool radius correction	MCODS(12..)	TCV(4)
Name of active tool correction table	MCODS(13..)	-
Name of the active axis/ZO table	MCODS(14..)	-
Active axis ZO values	MCODS(15..)	ZOV
Active tool correction	MCODS(54..)	-
Active data block placement	MCODS(112..)	-
Path and name of active placement table	MCODS(113..)	-
Tool position correction	MCODS(114..)	DPC
Total placements	MCODS(115..)	PMV
Active program coordinate offset	MCODS(116..)	SDR(168)
Active additive program coordinate offset	MCODS(117..)	SDR(268)
Total of the active program coordinate offset	MCODS(118..)	-
Online correction values	MCODS(89..)	-
Online correction status	MCODS(90..)	-
Current temperature compensation	MCODS(132..)	-

CPL commands

Command provides:	Syntax:	See also:
Current screw lead error compensation	MCODS(133..)	-
Current cross compensation	MCODS(134..)	-
Current compensation offset	MCODS(135..)	-
Operation modes		
Channel mode	MCODS(31..)	BITIF("ICH_OPMODE",...)
Axis mode	MCODS(48..)	-
System structure		
Number of feed axes, auxiliary axes, spindles; Motion types, drive types	MCODS(34..)	-
Number of channels	MCODS(44..)	-
Number of axes	MCODS(45..)	-
Axis names	MCODS(33..)	AXINF
Active channel axis names	MCODS(59..)	AXADR\$
Channel axis names (default setting)	MCODS(60..)	-
Axis-channel assignment	MCODS(43..)	SD(25,...)
Axis-channel default assignment	MCODS(58..)	-
Ident number from cyclic axis telegram	MCODS(62..)	-
Coordinate names	MCODS(78..)	AXINF\$
Number of coordinates	MCODS(82..)	-
Current data block of zero offset	MCODS(99..)	ZOV
Current total of zero offset	MCODS(100..)	ZOV
Machine coordinate names	MCODS(106..)	-
Number of machine coordinates	MCODS(107..)	-
Measuring units		
Axis units (default setting)	MCODS(61..)	-
Axis units	MCODS(53..)	-
Programming type (inch/metric)	MCODS(18..)	NCF("G70")
Diameter programming	MCODS(91..)	NCF("DIA")
Auxiliary functions		
Group numbers of auxiliary functions	MCODS(94..)	-
Active syntax of auxiliary function groups	MCODS(65..)	-
Spindles		
Motion functions of spindles	MCODS(63..)	SD(212,2)
Gear ranges of spindles	MCODS(64..)	BITIF("ISP_GEAR1..4SELECT",...)
Group assignment of channel spindles	MCODS(65..)	-
Information on whether gear switching is active	MCODS(66..)	-

Command provides:	Syntax:	See also:
Automatic/manual gear range selection	MCODS(67..)	-
Spindle names	MCODS(102..)	-
Number of spindles	MCODS(103..)	-
Spindle system number	MCODS(104..)	-
Spindle axis number	MCODS(105..)	-
Actual spindle position	MCODS(108..)	-
Spindle torque	MCODS(110..)	-
Spindle performance	MCODS(111..)	-
Drive		
Manufacturer version	MCODS(55..)	-
Control device type	MCODS(56..)	-
Motor type	MCODS(57..)	-
Other		
Messages in part program	MCODS(28..)	-
Path and name of main program	MCODS(30..)	-
Return-to-contour strategy and recording of jog motions	MCODS(46..)	-
Customer-specific data	MCODS(42..)	-
Optional stop (activated)	MCODS(68..)	BITIF("ICH_BLKSLASH",...)
Skip block (activate)	MCODS(69..)	BITIF("ICH_OPTSTOP",...)
Automatic program reselection active	MCODS(70..)	BITIF("QCH_RESELOFF",...)
Angle of rotation	MCODS(122..)	SCL(0)
Position of pole for mirroring and rotation	MCODS(123..)	SCL(1,...)
Offset coordinates	MCODS(124..)	COF
Active scaling factors	MCODS(125..)	SCL(2,...)
Coordinate mirroring, active	MCODS(126..)	SCL(2,...)
Block pre-run		
Adjustment status	MCODS(128,..)	-
Status of block pre-run re-entry mark	MCODS(127,..)	-

Tab. 7-107: Command overview MCODES



In the following table, the parameters partially specified in the syntax are integer constants. Instead of these constants, integer variables can also be programmed. However, they the specified value when the command is called has to be assigned to then.

MCODS(1,...) - Axis command position

Description: Provides the command positions of all **feed and auxiliary axes in the system** in ascending and channel-independent order in <Buffer>:

- for linear axes in mm
 - for rotary axes in degree
- "Set program position" (e.g. SetPos) is included in the values.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(1,-1,<Version>,<Buffer>,<Size>)**

Tab. 7-108: Syntax MCODES(1,...)

MCODS(2,...)- Axis command position (integer)

Description: Provides the command positions of all **feed and auxiliary axes in the system** in ascending and channel-independent order in <Buffer>:

in 0.0001 mm for linear axes

in 0.0001 degrees for rotary axes

"Set program position" (e.g. SetPos) is included in the values.

Update: Cyclic

Data type of <Buffer>: Integer, array

Syntax: **MCODS(2,-1,<Version>,<Buffer>,<Size>)**

Tab. 7-109: Syntax MCODES(2,...)

MCODS(3,...)- Path feed

Description: *Returns three values with current path feeds of <Channel> (including feed potentiometer) in the mm/min in ascending order in <Buffer>:*

1. The command velocity assigned externally to the interpolator.
2. The actual interpolator velocity (= current path velocity).
3. The interpolator-internal command velocity. It can have changed compared to the externally assigned one due to an application (e.g. Feed Adapt function).

During feed programming in mm/rev (G95), it provides the path feed in mm/min.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(3,<Channel>,<Version>,<Buffer>,3)**

Tab. 7-110: Syntax MCODES(3,...)

MCODS(4,...)- Lag

Description: *Returns the lagging of all axes in ascending order in <Buffer>:*

- In mm for linear axes
- in degrees for rotary axes

If lagging transfer (via Sercos parameter) is not supported by the drives, 0.0 is returned.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(4,-1,<Version>,<Buffer>,<Size>)**

Tab. 7-111: Syntax MCODES(4,...)

MCODS(5,...)- Spindle command speed, spindle cutting velocity

Description: Returns the spindle command speeds or cutting velocities of all spindles in <Channel> in ascending order (S or S1, S2, S3, etc.) in <Buffer>. If -1 or 0 is provided for <Channel>, <Buffer> provides the data of the system spindles. Otherwise, the channel spindle data is provided. With an active G96, the cutting velocities are provided in m/min. Otherwise the current spindle command speeds are provided in rpm. The potentiometer, the speed limits (SMin, SMax) and the limits due to the gear range are included in the calculation. If no spindle exists, 0.0 is returned in <Buffer> at the corresponding position.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(5,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-112: Syntax MCODES(5,...)

MCODS(6,...) - InPos

Description: Provides 0 or 1 as InPos signal for each **feed and auxiliary axis** in ascending and channel-independent order in <Buffer>:

Axis is in position: 1

Axis is not in position: 0

An axis is in position if it is in the parameterized InPos window (MP 1015 00100) and if no travel command is pending (see also MCODES(47...)).

Update: Cyclic

Data type of <Buffer>: Integer, array

Syntax: **MCODS(6,-1,<Version>,<Buffer>,<Size>)**

Tab. 7-113: Syntax MCODES(6,...)

MCODS(7,...) - Value of feed potentiometer

Description: Provides the current value of the feed potentiometer of <Channel> in 1/100 percent in <Buffer>.

Update: Cyclic

Data type of <Buffer>: Real

Syntax: **MCODS(7,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-114: Syntax MCODES(7,...)

MCODS(8,...) - Values of spindle potentiometers

Description: Returns the current values of the feed potentiometer in 1/100 percent in ascending order (S or S1, S2, S3, etc.) in <Buffer>.

If -1 or 0 is provided for <Channel>, <Buffer> provides the data of the system spindles. Otherwise, the channel spindle data is provided.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(8,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-115: Syntax MCODES((8,...)

MCODS(9,...)

Description: Does not apply to MTX

MCODS(10,...)

Description: Does not apply to MTX

MCODS(11,...)- Active tool radius correction number

Description: Provides the tool radius correction number active in <Channel> in <Buffer>. If no radius correction is active, -1 is returned.

Update: After each block switching

Data type of <Buffer>: Integer

Syntax: **MCODS(11,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-116: Syntax MCODES(11,...)

MCODS(12,...)- Active tool radius correction

Description: Provides the tool radius correction active in <Channel> in mm in <Buffer>. If no tool radius correction is active, 0.0 is returned.

Update: After each block switching

Data type of <Buffer>: Real

Syntax: **MCODS(12,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-117: Syntax MCODES(12,...)

MCODS(13,...)- Name of active tool correction table

Description: Provides the name of the tool correction table active in <Channel> in <Buffer>.

If not active, three blanks are returned as string.

Update: With each block switching

Data type of <Buffer>: Character, array

Syntax: **MCODS(13,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-118: Syntax MCODES(13,...)

MCODS(14,...)- Name of active axis ZO table

Description: Provides the name of the axis ZO table active in <Channel> in <Buffer>. If not active, three blanks are returned as string.

Update: With each block switching

Data type of <Buffer>: Character, array

Syntax: **MCODS(14,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-119: Syntax MCODES(14,...)

MCODS(15,...)- Active axis ZO values

Description: Returns the ZO values active in <Channel> of the first three banks for all machine coordinates in mm or degrees in <Buffer>.

If no offset is active, 0.0 is returned.

The following order applies:

- Offset of the first machine coordinate in bank 1
- Offset of the second machine coordinate in bank 1
- Offset of the nth Machine coordinate in bank 1

- Offset of the first machine coordinate in bank 2
- :
- Offset of the nth Machine coordinate in bank 2
- :
- Offset of the nth Machine coordinate in bank 3

Values for all five offset banks are provided by the command MCODS(99, ...)

Update: After each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(15,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-120: Syntax MCODS(15,...)

MCODS(16,...)- End position of current block including input helps

Description: Provides the end positions of the active blocks of all **feed and auxiliary axes** referring to the workpiece coordinates in ascending and channel-independent order in <Buffer>:

In mm for linear axes

In degree for rotary axes

All offset values are included in the calculation.

"Set program position" (e. g. SetPos) is **not** included in the values.

Update: With each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(16,0<Version>,<Buffer>,<Size>)**

Tab. 7-121: Syntax MCODS(16,...)

MCODS(17,...)

Description: Does not apply to MTX

MCODS(18,...) - Inch/metric programming

Description: Returns the programming type of the axes in the <Channel> in <Buffer>:

0: inch

1: Metric

2: Degree

3: Axis is not present

Update: With each block switching

Data type of <Buffer>: Integer

Syntax: **MCODS(18,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-122: Syntax MCODS(18,...)

MCODS(19,...) - Maximum spindle speed

Description: Returns the maximum admissible spindle speed in rpm in ascending order (S or S1, S2, S3, etc.) in <Buffer>.

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

The speed limitations are included in the calculation.

Update: With each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(19,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-123: Syntax MCODES(19,...)

MCODS(20,...) - Minimum spindle speed

Description: Returns the minimum admissible spindle speeds in rpm in ascending order (S or S1, S2, S3, etc.) in <Buffer>.

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

The speed limitations are included in the calculation.

Update: With each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(20,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-124: Syntax MCODES(20,...)

MCODS(21,...)

Description: Does not apply to MTX

MCODS(22,...)

Description: Does not apply to MTX

MCODS(23,...)- Programmed end positions without offsets

Description: As MCODES(16, ...) but without offsets.

Update: With each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(23.0,<Version>,<Buffer>,<Size>)**

Tab. 7-125: Syntax MCODES(23,...)

MCODS(24,...)- Programmed path feed

Description: Returns the programmed path feed of <Channel> in mm/min in <Buffer>.

Update: After each block switching

Data type of <Buffer>: Real

Syntax: **MCODS(24,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-126: Syntax MCODES(24,...)

MCODS(25,...) - Programmed spindle speed

Description: Returns the programmed spindle speeds in rpm in ascending order (S or S1, S2, S3, etc.) in <Buffer>.

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After each block switching

Data type of <Buffer>: Real Array

Syntax: **MCODS(25,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-127: Syntax MCODES(25,...)

MCODS(26,...) - Reference point approached

Description: Returns 0 or 1 as the signal "Approached reference point" for each **feed and auxiliary axis** in ascending and channel-independent order in <Buffer>:

Reference point approached: 1

Reference point not approached: 0

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(26,-1,<Version>,<Puffer>,<Size>)**

Tab. 7-128: Syntax MCODES(26,...)

MCODS(27,...) - Jog velocities

Description: Returns the current jog velocities of all **feed and auxiliary axes in the system** in ascending and channel-independent order in <Buffer>:

In mm/min for linear axes

In rpm for rotary axes

Update: After change

Data type of <Buffer>: Real, array

Syntax: **MCODS(27,-1<Version>,<Buffer>,<Size>)**

Tab. 7-129: Syntax MCODES(27,...)

MCODS(28,...) - Notes (messages)

Description: Returns the latest programmed note via MZA command, MSG command or PRN#(0,...) in <Buffer>.

The <Channel> parameter selects either a channel-comprehensive or a channel-specific note:

-1 Returns the latest channel-comprehensive note (MSG).

0..60 Returns the latest note of the MSG and PRN# command in the specified channel.

Update: After change

Data type of <Buffer>: Character, array

Syntax: **MCODS(28,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-130: Syntax MCODES(28,...)

MCODS(29,...) - Test mode

Description: Returns 1 in <Buffer> if test mode is not active. Otherwise 0.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(29,0,<Version>,<Buffer>,<Size>)**

Tab. 7-131: Syntax MCODES(29,...)

MCODS(30,...) - Path and name of main program

Description: Returns the path of the main program addressed in <Channel> including its name as string in <Buffer>.

A value in <Version> is ignored when the command is called.

The length of the file name plus the length of the path are specified in <Size>. In the internal file system, path including file name may not exceed 100 characters.

For files stored in mounted file systems, the value in <Size> depends on the maximum permitted number of characters supported by the external file system for the path and the name of a file.

Update: After change

Data type of <Buffer>: Character, array

Syntax:

MCODS(30,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-132: Syntax MCODES(30,...)

MCODS(31,...) - Channel mode

Description: *Returns in <Buffer> the active in the <Channel>*

- Main mode
- Secondary mode

The following values are defined for the main mode:

- 0: No operation mode. Thus, no process is active
- 1: Jog mode
Axes can be jogged (+/-).
- 2: Approach reference point
Axes can be started using the signals Manual+ / Manual-.
- 3: reserved
- 4: Manual data input
Individual NC blocks can be specified for machining.
- 5: Automatic (continuous block)
Part programs are processed completely.
- 6: Automatic (program block)
Individual blocks of a part program are processed sequentially. Each individual block is prepared and started with NC start.
- 7: Automatic (single step)
Under certain conditions, the NC generates and prepares several blocks from an individual NC block in the part program. In this operation mode, NC start is always used to pass an individual block on to the interpolator for processing.
- 8: Teaching
Axes can be jogged (+/-) or traversed with the handwheel. Individual NC blocks can be specified for machining.
- 9: reserved
- 10: Automatic (single block)
With NC start, all blocks that were generated and prepared due to an individual NC block in the part program are passed on to the interpolator for processing.
- 11: Return to contour
Axes can be moved manually away from the contour and restarted either automatically or manually.

- 12: CPL debugger (program block mode)
Single blocks are processed as in the part program.
- 13: CPL debugger (continuous block mode)
All blocks up to the next point of interruption are processed.

The following values are defined for the secondary mode:

- 0: No secondary mode active
- 1: Overwriting active. Individual NC blocks can be specified for machining.

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(31,<Channel>,<Version>,<Buffer>,2)

Tab. 7-133: Syntax MCODES(31,...)

MCODS(32,...) - SAV and IPO state

Description: *Returns in<Buffer> of <Channel>*

- the SAV state (block preparation) and
- the interpolator state.

The following values are defined for the SAV state:

- 1: Operation mode is not active. A process can be selected
- 2: Operation mode is ready. A process can be started
- 3: Operation mode is active. A program or NC block is being processed
- 4: Reserved
- 5: Reserved
- 6: An error occurred in the operation mode. This error can be cleared only by "Control reset" or "Program deselection"
- 7: Reserved
- 8: Reset is currently executed
- 9: A program is selected and prepared (e.g. linked)
- 10: "Cancel distance to go" was triggered and is not yet completed
- 11: Operation mode is active and processes existing buffers again
- 12: Operation mode is ready. The process is at the beginning of the program and can be started
- 13: All blocks at the buffered NC block specification are processed. Waiting for the next specification

The following values are defined for the IPO state:

- 1: Interpolator running
- 2: Interpolator shuts down due to a feed hold
- 3: Interpolator stopped axes

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(32,<Channel>,<Version>,<Buffer>,2)

Tab. 7-134: Syntax MCODES(32,...)

MCODS(33,...) - Axis names

Description: For <Channel> = -1, <Buffer> returns the names of all configured system axes 1 - 100, separated by "0" (0-byte) in ascending order of nine bytes each.
 For <Channel> = -2, <Buffer> returns the names of all configured system axes 101 - 200, separated by "0" (0-byte) in ascending order of nine bytes each.
 For <Channel> = -3, <Buffer> returns the names of all configured system axes 201 - 250, separated by "0" (0-byte) in ascending order of nine bytes each.
 If <Channel> = existing channel number, <Buffer> returns the names of all **axes in the specified channel** separated by "0" (0-byte) in ascending order.
 Names shorter than eight characters are filled up with spaces to complete the eight characters.
 The size of <Buffer> is specified in <Size>. For 100 axes, it can be a maximum of (9*100) 900 bytes.
 For an example, refer to [chapter " Examples of programming" on page 543](#).

Update: After control startup

Data type of <Buffer>: Character, array

Syntax: **MCODS(33,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-135: Syntax MCODES(33,...)

MCODS(34,...)

Description: Does not apply to MTX

MCODS(35,...) - Actual axis position of the drives

Description: Provides the actual positions of all **feed and auxiliary axes** transmitted to the CNC by the drives via the Sercos interface in ascending and channel-independent order in <Buffer>:
 - for linear axes in mm
 - for rotary axes in degree
 "Set program position" (e. g. SetPos) is **not** included in the values.



The values provided by MCODES(35..) differ from MCODES(147..) by the axis error corrections LSEC, CCOMP and the temperature compensation.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(35,-1<Version>,<Buffer>,<Size>)**

Tab. 7-136: Syntax MCODES(35,...)

MCODS(36,...) - Spindle list speed

Description: Returns the actual spindle speeds in rpm in ascending order (S or S1, S2, S3, etc.) in <Buffer>. If -1 or 0 is transmitted for <Channel>, <Buffer> returns the data of the system spindles, but if not, it returns the data of the channel spindles. The potentiometer, the speed limitations (SMin, SMax) and the limitations due to the gear range are included in the calculation. If no spindle exists, 0.0 is returned in <Buffer> at the corresponding position.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(36,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-137: Syntax MCODES(36,...)

MCODS(37,...) - Programmed coordinate values

Description: Returns the target positions of the interpolator referring to the workpiece for all **feed axes in the system** in ascending and channel-independent order in <Buffer>:

- for linear axes in mm

- for rotary axes in degree

"Set program position" (e.g. SetPos) and the axis ZO (G54.x ...G59.x) is **not** included in the values.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(37,-1<Version>,<Buffer>,<Size>)**

Tab. 7-138: Syntax MCODES(37,...)

MCODS(38,...) - Actual axis values in the workpiece coordinate system

Description: Returns "actual values" in <Buffer>. These values are related to the corresponding (Cartesian) workpiece coordinate system of <Channel>.

The values are calculated from the actual axis positions using the channel-by-channel machine-specific kinematic forward transformation (axis transformation). If no kinematics axis transformation is active, MCODES(38) returns values identical to MCODES(147).

Applying MCODES(38) requires a corresponding setting of machine parameter 9030 00002. This machine parameter is used to configure whether and how often actual values are calculated in the workpiece coordinate system.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(38,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-139: Syntax MCODES(38,...)

MCODS(39,...) - Dwell time active

Description: Returns 1 in the <Buffer> if a dwell time is active in <Channel>. Otherwise 0.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(39,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-140: Syntax MCODES(39,...)

MCODS(40,...) - Auxiliary function requiring an acknowledgment active

Description: Returns 1 in <Buffer> if an auxiliary function in <Channel> is waiting to be acknowledged. Otherwise 0.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(40,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-141: Syntax MCODES(40,...)

MCODS(41,...) - Block transfer enabled

Description: Returns 1 in <Buffer> if the NC input signal "Block transfer inhibit" is set for <Channel>. Otherwise 0.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(41,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-142: Syntax MCODES(41,...)

MCODS(42,...) - Customer-specific data

Description: Returns customer-specific data from <Channel> in <Buffer>.

In <P1>, an integer value ranging between 0 and 65535 can be transferred to the customer server to select certain data upon the command call.

The command is intended for customer-specific internal developments in the area of the "NC kernel".

Update: After change

Data type of <Buffer>: Depends on the type of the customer-specific data

Syntax: **MCODS(42,<Channel>,<Version>,<Buffer>,<Size>,<P1>)**

Tab. 7-143: Syntax MCODES(42,...)

MCODS(43,...) - Axis assignment: Channel

Description: Returns the following default assignment for each system axis in <Buffer>:

- ≥0:** Channel number of the synchronous axis
- 1 :** Axis is asynchronous
- 2:** Axis is a spindle
- 3:** Axis is not defined

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(43,-1<Version>,<Buffer>,<Size>)**

Tab. 7-144: Syntax MCODES(43,...)

MCODS(44,...) - Number of channels

Description: In <Buffer>, the following is returned in ascending order

- Number of usable user channels
- Number of channels at the interface
- Number of internal and external channels

Update: After control startup

Data type of <Buffer>: Integer, array

Syntax: **MCODS(44,-1,<Version>,<Buffer>,3)**

Tab. 7-145: Syntax MCODES(44,...)

MCODS(45,...) - Number of axes

Description: *In <Buffer>, the following is returned in ascending order*

- Maximum index of all drives (axes and spindles) in the system. This is required for grinding over all drives. If no gaps are present, this is the number of drives in the system.
- Maximum index of the axes in the system. This is required for the size of the axis interface for example. If no gaps are present, this is the number of axes in the system.
- Maximum number of spindles in the system. This is required for the size of the spindle interface for example

Update: After control startup

Data type of <Buffer>: Integer, array

Syntax:

MCODS(45,-1,<Version>,<Buffer>,3)

Tab. 7-146: Syntax MCODES(45,...)

MCODS(46,...) - Return-to-contour strategy and recording of jog motions

Description: *Returns the following in <Buffer> in ascending order for the specified <Channel>:*

- Return to contour mode
- Point of return to contour
- Recording status of jog motions.

The following values are valid for the return to contour mode:

- 1: Automatic return to contour
- 2: Return to contour with single block
- 3: Manual return to contour

For the point of return to contour:

- 1: Return to starting point
- 2: Return to end point
- 3: Return to point of interruption

For the recording status:

- 0: Recording not active
- 1: Recording active

Update: After change

Data type of <Buffer>: Integer, change

Syntax:

MCODS(46,<Channel>,<Version>,<Buffer>,3)

Tab. 7-147: Syntax MCODES(46,...)

MCODS(47,...) - Travel command

Description: For <Channel>= -1, <Buffer> returns the travel command signals of all system axes in ascending order.

For <Channel> = existing channel number, <Buffer> returns the travel command signals of all **axes in the specified channel followed by the travel command signals of the asynchronous axes** in ascending order.

Travel command present: 1

Travel command not present: 0

A travel command is always set as soon as an axis is to execute a traversing motion via manual entry or entry in the part program.

Update: Cyclic

Data type of <Buffer>: Integer, array

Syntax: **MCODS(47,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-148: Syntax MCODES(47,...)

MCODS(48,...) - Axis mode

Description: For <Channel>= -1, <Buffer> returns the operation modes of all system axes in ascending order.

For <Channel> = existing channel number, <Buffer> returns the operation modes of all **axes in the specified channel followed by the operation modes of the asynchronous axes** in ascending order.

Possible return values for the operation modes:

- 0:** No operation mode. Thus, no process is active
- 1:** Jog mode
Axes can be jogged (+/-).
- 2:** Approach reference point
Axes can be started using the signals Manual+ / Manual-.
- 3:** Reserved
- 4:** Manual data input
Individual NC blocks can be specified for machining.
- 5:** Automatic (continuous block)
Part programs are processed completely.
- 6:** Automatic (program block)
Individual blocks of a part program are processed sequentially. Each individual block is prepared and started with NC start.
- 7:** Automatic (single step)
Under certain conditions, the NC generates and prepares several blocks from one individual NC block in the part program. In this operation mode, NC start is always used to pass an individual block on to the interpolator for processing.
- 8:** Reserved
- 9:** Reserved
- 10:** Automatic (single block)
With NC start, all blocks that were generated and prepared due to an individual NC block in the part program are passed on to the interpolator for processing.
- 11:** Return to contour
Axes can be moved manually away from the contour and restarted either automatically or manually.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(48,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-149: Syntax MCODES(48,...)

MCODS(49,...) - Feed inhibit

Description: For <Channel>= -1, <Buffer> returns the feed inhibit signals of all system axes in ascending order.

For <Channel> = existing channel number, <Buffer> returns the feed inhibit signals of all **axes in the specified channel followed by the feed inhibit signals of the asynchronous axes** in ascending order.

1: Feed inhibit present

0: Feed inhibit not present

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(49,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-150: Syntax MCODES(49,...)

MCODS(50,...)- Axis potentiometer

Description: For <Channel>= -1, <Buffer> returns the values for the respective axis-relevant overrides (axis potentiometer or channel potentiometer) for all system axes in ascending order (in 0.01 percent).

For <Channel> = existing channel number, <Buffer> returns the value of the channel potentiometer for every **axis in the specified channel followed by the values of the axis potentiometers of all asynchronous axes** in ascending order (in 0.01 percent).

Update: After change

Data type of <Buffer>: Real, array

Syntax:

MCODS(50,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-151: Syntax MCODES(50,...)

MCODS(51,...)

Description: Does not apply to MTX

MCODS(52,...)

Description: Does not apply to MTX

MCODS(53,...) - Measuring unit of axes

Description: For <Channel>= -1, <Buffer> returns the measuring units (metric, inch, degree) of all system axes in ascending order.

For <Channel> = existing channel number, <Buffer> returns the measuring unit for every **axis in the specified channel followed by the measurement unit of all asynchronous axes** in ascending order.

For asynchronous linear axes in the "Jog" and "Approach reference point" axis modes, the axis interface determines the unit. If no axis mode is entered, "metric" is returned.

For synchronous linear axes in the "Jog" and "Approach reference point", channel modes, the axis interface determines the unit. In the other operation modes, it depends on the metric/inch channel unit (G70/G71).

For rotary axes and spindles, the unit is degrees. For Hirth axes with location programming, a corresponding measuring unit is provided.

Possible return values for the units:

0: inch

- 1: Metric
- 2: Degree
- 3: Axis is not present
- 4: Location-programmable Hirth axis

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(53,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-152: Syntax MCODES(53,...)

MCODS(54,...) - Active tool correction

Description: Returns the tool correction factors active in <Channel> in <Buffer>.

Sequence:

- Radius correction
- L3 length correction
- L1 length correction
- L2 length correction
- Edge position
- Correction type

The following correction types are defined:

- 0: No correction
- 1: Drilling tool
- 2: Milling cutter
- 3: Turning tool
- 4: Angle head tool

Update: After block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(54,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-153: Syntax MCODES(54,...)

MCODS(55,...) - Manufacturer version

Description: Returns the manufacturer version of the drive.

The axes are selected in the parameter <axis number> by entering the system axis. (No. 0 also provides the first axis). The manufacturer version corresponds to Sercos parameter S-0-0030. An array of up to 40 characters is provided in the parameter <Buffer>.

Update: After control startup

Data type of <Buffer>: Character, array

Syntax: **MCODS(55,-1,<Version>,<Buffer>,<Size>,<AxisNumber>)**

Tab. 7-154: Syntax MCODES(55,...)

MCODS(56,...) - Control device type

Description: Outputs the control device type of the drive.

The axis is selected in the parameter <Axis number> by specifying the system axis (no. 0 also provides the first axis). The controller device type corre-

sponds to Sercos parameter S-0-0140. An array of up to 40 characters is provided in the parameter <Buffer>.

Update: After control startup

Data type of <Buffer>: Character, array

Syntax:

MCODS(56,-1,<Version>,<Buffer>,<Size>,<AxisNumber>)

Tab. 7-155: MCODES(56,...) syntax

MCODS(57,...)- Motor type

Description: Returns the motor type of the drive.

The axes are selected in the parameter <axis number> by entering the system axis. (No. 0 also returns the first axis). The motor type corresponds to the Sercos parameter S-0-0141. An array of up to 40 characters is provided in the parameter <Puffer>.

Update: After control startup

Data type of <Buffer>: Character, array

Syntax:

MCODS(57,-1,<Version>,<Buffer>,<Size>,<AxisNumber>)

Tab. 7-156: Syntax MCODES(57,...)

MCODS(58,...) - Default axis assignment: Channel

Description: Returns the following default assignment for each system axis in <Buffer>:

≥0: Channel number of the synchronous axis

-1 : Axis is asynchronous

-2: Axis is a spindle

-3: Axis is not defined

In case of 250 axes, <Buffer> has to have the <Size> 250 (integer).

Update: After startup

Data type of <Buffer>: Integer

Syntax:

MCODS(58,-1,<Version>,<Puffer>,<Size>)

Tab. 7-157: Syntax MCODES(58,...)

MCODS(59,...) - Active channel axis names

Description: For <Channel> = -1, <Buffer> returns the names of all **active channel axes** with the system axis indexes 1 - 100, separated by "0" (0 byte) in ascending order of nine bytes each.

For <Channel> = -2, <Buffer> returns the names of all **active channel axes** with the system axis indexes 101 - 200, separated by "0" (0 byte) in ascending order of nine bytes each.

For <Channel> = -1, <Buffer> returns the names of all **active channel axes** with the system axis indexes 201 - 250, separated by "0" (0 byte) in ascending order of nine bytes each.

If <Channel> = existing channel number, <Buffer> returns the names of all **axes in the specified channel** separated by "0" (0-byte) in ascending order.

Names shorter than 8 characters are filled with spaces up to the 8th character.

The size of <Buffer> is specified in <Size>. For 100 axes, it can be a maximum of (9*100) 900 bytes.

For an example, refer to [chapter " Examples of programming" on page 543](#).

Update: After change
Data type of <Buffer>: Character, array

Syntax: **MCODS(59,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-158: Syntax MCODES(59,...)

MCODS(60,...) - Default setting of channel axis names

Description: For <Channel> = -1, <Buffer> returns the names of all active channel axes with the system axis indexes 1 - 100, separated by "0" (0 byte) in ascending order of nine bytes each.

For <Channel> = -2, <Buffer> returns the names of all active channel axes with the system axis indexes 101 - 200, separated by "0" (0 byte) in ascending order of nine bytes each.

For <Channel> = -3, <Buffer> returns the names of all active channel axes with the system axis indexes 201 - 250, separated by "0" (0 byte) in ascending order of nine bytes each.

If <Channel> = existing channel number, <Buffer> returns the names of all **axes in the specified channel** separated by "0" (0-byte) in ascending order.

Names shorter than eight characters are filled up with spaces to complete the eight characters.

The size of <Buffer> is specified in <Size>. For 100 axes, it can be a maximum of (9*100) 900 bytes.

For an example, refer to [chapter " Examples of programming" on page 543](#).

Update: After control startup
Data type of <Buffer>: Character, array

Syntax: **MCODS(60,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-159: Syntax MCODES(60,...)

MCODS(61,...) - Default setting of measuring units of axes

Description: If <Channel> = -1, <Buffer> returns the measuring units (metric, inch, degree) of all system axes in the default setting.

If <Channel> = existing channel number, <Buffer> returns the measuring units (metric, inch, degree) of all **axes specified in the channel** in the default setting.

- For **asynchronous** linear axes, "metric" units are used.
- For **synchronous** linear axes, the measuring unit depends on the switched-on state after startup (parameter CHAN/Ch/Ini/NcResetState (MP 7060 00010)): "Metric/inch" (G70/G71)
- For rotary axes, the unit is degree. For Hirth axes with location programming, a corresponding unit is provided.

Possible return values for the units:

- 0: Inch
 - 1: Metric
 - 2: Degree
 - 3: Axis is not present
 - 4: Location-programmable Hirth axis
- In case of 250 axes, <Buffer> has to have the <Size> 250 (integer).

Update: After control startup

Data type of <Buffer>: Integer, array

Syntax: **MCODS(61,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-160: Syntax MCODES(61,...)

MCODS(62,...) - Ident number from cyclic axis telegram

Description: Provides the value of an <ident number> of the cyclic axis telegram for all axes.

If the <ident number> is not in the cyclic telegram, the value NCS_MCO_NOT_IN_CYCL_AT_C(-2147483648) is output.

The value is "Integer" in Sercos scaling.

In case of 250 axes, <Buffer> has to have the <Size> 250 (integer).

Update: Cyclic

Data type of <Buffer>: Integer, array

Syntax: **MCODS(62,-1,<Version>,<Buffer>,<Size>,<ident number>)**

Tab. 7-161: Syntax MCODES(62,...)

MCODS(63,...) - Motion functions of spindles

Description: Returns the motion functions of the spindles.

Coding the motion functions:

- 0: Spindle not defined
- 1: Clockwise rotation without cooling agent
- 2: Clockwise rotation with cooling agent
- 3: Counterclockwise rotation without cooling agent
- 4: Counterclockwise rotation with cooling agent
- 5: Spindle stop
- 6: Orient spindle

For e.g. 32 spindles, <Buffer> has to have the <Size> 32 (integer).

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(63,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-162: Syntax MCODES(63,...)

MCODS(64,...) - Gear ranges

Description: Returns the gear stages of the spindles.

Coding the gear stages:

- 40: automatic gear stage selection
- 41: Gear stage 1
- 42: Gear stage 2
- 43: Gear stage 3
- 44: Gear stage 4
- 48: Disengage gear

For e.g. 32 spindles, <Buffer> has to have the <Size> 32 (integer).

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: `MCODS(64,<Channel>,<Version>,<Buffer>,<Size>)`

Tab. 7-163: Syntax MCODES(64,...)

MCODS(65,...) - Group assignment of channel spindles

Description: Returns the group assignment of the channel spindles:

Group assignment:

- 0: No group active
- 1.. 4. Group number

For e.g. With eight channel spindles, <Buffer> has to have the <Size> 8 (integer).

If -1 is returned for <Channel>, <Buffer> returns the data of the active channel. Otherwise, the data of the selected channel is returned.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: `MCODS(65,<Channel>,<Version>,<Buffer>,<Size>)`

Tab. 7-164: Syntax MCODES(65,...)

MCODS(66,...) - Gear switching

Description: *Returns information on whether gear switching is active:*

- 0: Gear switching is not active
- 1: Gear switching is active

For e.g. 32 spindles, <Buffer> has to have the <Size> 32 (integer).

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: `MCODS(66,<Channel>,<Version>,<Buffer>,<Size>)`

Tab. 7-165: Syntax MCODES(66,...)

MCODS(67,...) - Automatic/manual gear switching

Description: *Provides the automatic or manual gear selection:*

- 0: Manual
- 1: automatic

For e.g. 32 spindles, <Buffer> has to have the <Size> 32 (integer).

If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: `MCODS(67,<Channel>,<Version>,<Buffer>,<Size>)`

Tab. 7-166: Syntax MCODES(67,...)

MCODS(68,...) - Skipping the block

Description: Returns the state of the NC output signal **activate skip block** and of the input signal **skip block** of <Channel> in <Buffer>.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(68,<Channel>,<Version>,<Buffer>,2)**

Tab. 7-167: Syntax MCODES(68,...)

MCODS(69,...) - Optional stop

Description: Returns the state of the NC output signal **optional stop activated** and of the input signal **optional stop** of <Channel> by <Buffer>.

Update: After change

Data type of <Buffer>: Integer, array

Syntax: **MCODS(69,<Channel>,<Version>,<Buffer>,2)**

Tab. 7-168: Syntax MCODES(69,...)

MCODS(70,...) - Automatic program reselection active

Description: Returns in <Buffer> whether automatic program reselection is applied in the specified channel:

0: Function is not applied

1: Function is applied

Update: Immediately after request

Data type of <Buffer>: Integer

Syntax: **MCODS(70,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-169: Syntax MCODES(70,...)

MCODS(71,...) - Workpiece coordinates

Description: Returns the values of the **workpiece coordinates** (WCS) of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(71,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-170: Syntax MCODES(71,...)

MCODS(72,...) - Command values of basic coordinates

Description: Returns the values of the **command values of basic coordinates** (BCS) of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(72,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-171: Syntax MCODES(72,...)

MCODS(73,...) - Axis coordinates in ACS1

Description: Provides the **values of the axis coordinates** (ACS1) of the specified channel in <Buffer>.

Channel = -1: all axis coordinates

0 < channel ≤ max. channel: Data of the specified channel

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(73,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-172: Syntax MCODES(73,...)

MCODS(74,...) - Machine coordinates

Description: Provides the values of the **machine coordinates** (MCS) of the specified channel in <Buffer>.

Channel = -1: all axis coordinates

0 < channel ≤ max. channel: Data of the specified channel

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(74,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-173: Syntax MCODES(74,...)

MCODS(75,...) - Actual values of basic coordinates

Description: Returns the **actual values of basic coordinates** (BCS) of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(75,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-174: Syntax MCODES(75,...)

MCODS(76,...) - Programmed coordinate end points

Description: Returns the **programmed coordinate end points** of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Update: After each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(76,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-175: Syntax MCODES(76,...)

MCODS(77,...) - Coordinate end points

Description: Returns the **end points of the coordinates** of the specified channel including offsets in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Update: After each block switching

Data type of <Buffer>: Real, array

Syntax:

MCODS(77,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-176: Syntax MCODES(77,...)

MCODS(78,...) - Coordinate names

Description: Returns the **names of the active coordinates** of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.
The size of <Buffer> is specified in <Size>. For 8 axes, it can be a maximum of (8*9) 72 bytes.

Update: After change

Data type of <Buffer>: Character, array

Syntax:

MCODS(78,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-177: Syntax MCODES(78,...)

MCODS(79,...) - INPOS status for coordinates

Description: Returns the **INPOS status of the coordinates** of the specified channel in <Buffer>: first the spatial coordinates, then the pseudo coordinates of the channel.
The status from the AND operation of the axis signals is generated for a spatial coordinate.

Update: Cyclic

Data type of <Buffer>: Integer, array

Syntax:

MCODS(79,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-178: Syntax MCODES(79,...)

MCODS(80,...) - Reference for coordinates

Description: Returns the **reference point status of the coordinates** of the specified channel in <Buffer>:
The status from the AND operation of the axis signals is generated for a spatial coordinate.

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(80,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-179: Syntax MCODES(80,...)

MCODS(81,...) - Measuring unit for coordinates

Description: Returns the **measuring units of the coordinates** of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Possible return values for the units:

- 0: inch
- 1: Metric
- 2: Degree
- 3: Coordinate not present

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(81,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-180: Syntax MCODES(81,...)

MCODS(82,...) - Number of coordinates

Description: Provides the **number of the coordinates/axes** in three elements of the specified channel in <Buffer>:

- 1. value: Total number of axes in the channel
- 2nd value: Number of spatial coordinates + number of pseudo-coordinates of the channel.
- 3. value: Number of pseudo-coordinates of the channel.

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(82,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-181: Syntax MCODES(82,...)

MCODS(83,...) - Distance to go of coordinates in the workpiece coordinate system (WCS)

Description: Returns the **distance-to-go of the workpiece coordinates** (WCS) of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax:

MCODS(83,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-182: Syntax MCODES(83,...)

MCODS(84,...)

Description: Does not apply to MTX

MCODS(85,...) - Distance to go of axes in axis coordinate system

Description: Provides the **distances to go by the axes in the axis coordinate system** (ACS) of the specified channel in <Buffer>.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax:

MCODS(85,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-183: Syntax MCODES(85,...)

MCODS(86,...)

Description: Does not apply to MTX

MCODS(87,...) - Channel waiting states

Description: Provides the **waiting states** of a **channel** in <Buffer>.

This command indicates the reasons a channel is entering a waiting state.

The active waiting states are bit-coded.

The following constants define the corresponding bits of the first integer value starting with the lowest value:

- 0: Dwell time
- 1: Auxiliary function subject to acknowledgement

2:	Block transfer inhibit
3:	Feed in the channel is equal to 0
4:	Program stop with M0/M1
5:	Feed halt in the channel
6:	Feed inhibit in the channel or in a channel axis
7:	Block transfer inhibit specified by the customer
8:	Synchronized motion stop between channels (ASTOP, ...)
9:	Waiting for axis at axis exchange (WAX)
10:	Waiting for permanent variable (WPV)
11:	Waiting for interface signal at active point of time (WAITA, ...)
12:	Waiting for interface signal (WAIT(BITIF...)) or fixed interval (WAIT(,ZEIT%)) in block preparation
13:	Motion Control Data Service (MCODS(...))
14:	Waiting for inpos message of channel axes
15:	G95: Waiting for spindle and rotary axis speed
16:	G33: Waiting for spindle speed
17:	No return to contour velocity, e.g. jog velocity
18:	Waiting for spindle motion
19:	Programmable event (SEV/WEV)
20:	Waiting for share of internal resources (memory)
21:	G63: Waiting for C-axis switching
Refer to the example in chapter " Examples of programming" on page 543.	

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(87,<Channel>,<Version>,<Buffer>,1)**

Tab. 7-184: Syntax MCODES(87,...)

MCODS(88,...)

Description: Does not apply to MTX

MCODS(89,...) - Online correction values (WCS)

Description: Returns the values of the online correction (WCS) of the specified channel in <Buffer>. First all spatial coordinates, then the pseudo-coordinates of the channel.

Update: After change

Data type of <Buffer>: Real, array

Syntax: **MCODS(89,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-185: Syntax MCODES(89,...)

MCODS(90,...) - Status of online correction (WCS)

Description: Returns the current state of the online correction (WCS) of the specified channel in <Buffer>. First all spatial coordinates, then the pseudo-coordinates of the channel.

0:	Inactive
1:	Active

Update: After change
Data type of <Buffer>: Integer, array

Syntax: **MCODS(90,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-186: Syntax MCODES(90,...)

MCODS(91,...) - Diameter programming

Description: Returns 1 <Buffer> for axes with diameter programming in .
 If the value Ncs_MCoNoChannel_C (-1) is specified for the channel, the values of all system axes are returned.
 If a channel number is specified, the values for the channel axes are returned.

Update: After each block switching
Data type of <Buffer>: Integer, array

Syntax: **MCODS(91,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-187: Syntax MCODES(91,...)

MCODS(92,...) - Basic coordinates of tool tip (BCS-Tcp)

Description: In case of active 3D radius correction in <Buffer>, it returns the command values of the tool tip in the basic coordinate System (BCS Tcp). First the spatial coordinates, then the pseudo-coordinates of the specified channel.
 If 3D radius correction is switched off, the values are identical with MCODES(72,...).

Update: Cyclic
Data type of <Buffer>: Real, array

Syntax: **MCODS(92,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-188: Syntax MCODES(92,...)

MCODS(93,...)

Description: Does not apply to MTX

MCODS(94,...) - Group numbers of auxiliary functions

Description: Provides the auxiliary function group numbers for the auxiliary functions specified in <Syntax> of the specified channel in <GroupId>.
 <Syntax> contains the auxiliary functions separated by a space and completed by CHR\$(0); refer to ["Example:" on page 529](#).

Update: Cyclic
Data type of <Buffer>: Character, array

Syntax: **MCODS(94,<Channel>,<Version>,<GroupId>,<Size>,<Syntax>)**

Tab. 7-189: Syntax MCODES(94,...)

MCODS(95,...) - Active syntax of auxiliary function groups

Description: Provides the active syntax for the auxiliary function group numbers specified in <GroupId> of the specified channel in <Syntax>.
 An array (completed with CHR\$(0)) of the desired auxiliary function group numbers is transferred in <GroupId>; refer to the example below.

Update: Cyclic
Data type of <Buffer>: Character, array

Syntax:

MCODS(95,<Channel>,<Version>,<Syntax>,<Size>,<GroupId>)

Tab. 7-190: Syntax MCODES(95,...)

MCODS(96,...)

Description: Does not apply to MTX

MCODS(97,...)

Description: Does not apply to MTX

MCODS(98,...)

Description: Does not apply to MTX

MCODS(99,...) - Active Zero Point Offset Data

Description: Returns the active data block of the zero offset of the machine coordinates in <Buffer>.

For each coordinate, up to five "Double" values are provided for the active offset values of the ZO banks 1..5. If no offset is active, 0.0 is returned.

The data is stored in <Buffer> as follows:

- Offset of 1st coordinate, bank 1
- Offset of 1st coordinate, bank 2
- ...
- Offset of first coordinate, bank 5
- Offset n. Coordinate, bank 1
- ...
- Offset n. Coordinate, bank 5

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(99,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-191: Syntax MCODES(99,...)

MCODS(100,...) - Total of zero point offsets

Description: Returns the total of active zero offsets of the machine coordinates in <Buffer>. For every coordinate, a "Double" value is provided.

If no offset is active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(100,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-192: Syntax MCODES(100,...)

MCODS(101,...) - Axis coordinate end pointsDescription: Provides the **end points of the axis coordinates** (ACS) of the specified channel in <Buffer>:

0 < channel ≤ max. channel: Data of the specified channel

Update: After each block switching

Data type of <Buffer>: Real, array

Syntax:

MCODS(101,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-193: Syntax MCODES(101,...)

MCODS(102,...) - Spindle names

Description: Returns the names of the active spindles of the specified channel in <Buffer>. If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Char, array

Syntax:

MCODS(102,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-194: Syntax MCODES(102,...)

MCODS(103,...) - Number of spindles

Description: Returns the number of spindles of the specified channel in <Buffer>. If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer

Syntax:

MCODS(103,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-195: Syntax MCODES(103,...)

MCODS(104,...) - Spindle system number

Description: Returns the system numbers of the spindles (system spindle number) in ascending order (S or S1, S2, S3, etc.) in <Buffer>. If -1 or 0 is provided for <Channel>, <Buffer> provides the data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(104,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-196: Syntax MCODES(104,...)

MCODS(105,...) - Spindle axis number

Description: Returns the axis numbers of the spindles in ascending order (S or S1, S2, S3, etc.) in <Buffer>. If -1 or 0 is provided for <Channel>, <Buffer> provides data of the system spindles. Otherwise, the channel spindle data is provided.

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(105,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-197: Syntax MCODES(105,...)

MCODS(106,...) -Machine coordinate names

Description: Returns the names of the active machine of the specified channel in <Buffer>: first all spatial coordinates, then the pseudo coordinates of the channel in the machine coordinate system.

The size of <Buffer> is specified in <Size>. For 8 axes, it can be a maximum of (8*9) 72 bytes.

Update: After change

Data type of <Buffer>: Character, array

Syntax:

MCODS(106,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-198: Syntax MCODES(106,...)

MCODS(107,...) - Number of machine coordinates

Description: Returns the number of the coordinates/axes in three elements of the specified channel in <Buffer>:

1. value: Total number of axes in the channel

2nd value: Number of the spatial coordinates plus the number of the pseudo-coordinates of the channel in the machine coordinate system

3. value: Number of the pseudo-coordinates of the channel in the machine coordinate system

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(107,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-199: Syntax MCODES(107,...)

MCODS(108,...) -Actual spindle position

Description: Returns the actual position of the active spindles of the specified channel in <Buffer>.

If -1 or 0 is provided for <Channel>, <Buffer> provides the actual position of the system spindles. Otherwise, the actual position of the channel spindles is provided.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax:

MCODS(108,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-200: Syntax MCODES(108,...)

MCODS(109,...) - Axis coordinates in ACS0

Description: Provides the **values of the axis coordinates** (ACS0) of the specified channel in <Buffer>.

Channel = -1: all axis coordinates

0 < channel ≤ max. channel: Data of the specified channel

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax:

MCODS(109,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-201: Syntax MCODES(109,...)

MCODS(110,...) -Actual spindle torque

Description: Returns the actual and maximum torque (percentage [%]) of the active spindles of the specified channel in <Buffer>.

If -1 or 0 is provided for <Channel>, <Buffer> provides the torque of the system spindles. Otherwise, the torque of the channel spindles is provided.

Two values are transferred for each spindle.

See also the documentation "MTX Standard NC Operation", chapter "Technology display".

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax:

MCODS(110,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-202: Syntax MCODES(110,...)

MCODS(111,...) - Spindle performance

Description: Returns the actual and maximum performance (unit in watts [W]) of the active spindles of the specified channel in <Buffer>.

If -1 or 0 is provided for <Channel>, provides the performance of the system spindles. Otherwise, the performance of the channel spindles is provided.

Two values are transferred for each spindle.

See also the documentation "MTX Standard NC Operation", chapter "Technology display".

Update: Cyclic

Data type of <Buffer>: Integer, array

Syntax:

MCODS(111,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-203: MCODES(111,...) syntax

MCODS(112,...) - Active placement data

Description: Returns the active data block of the placements in <Buffer>. Six "Double" values are provided for each level or bank. If no placement is active, 0.0 is returned. In total, 30 "Double" values are provided.

The data is stored in <Buffer> as follows:

- Bank 1 [X, Y, Z, Phi, Theta, Psi]
- Bank 2 [X, Y, Z, Phi, Theta, Psi]
- ...
- Bank 5 [X, Y, Z, Phi, Theta, Psi]

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(112,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-204: Syntax MCODES(112,...)

MCODS(113,...) - Path and name of active placement table

Description: Provides the name of the placement table active in <Channel> in <Buffer>.

If not active, three blanks are returned as string.

Update: After each block switching

Data type of <Buffer>: Character, array

Syntax:

MCODS(113,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-205: Syntax MCODES(113,...)

MCODS(114,...) - Workpiece position correction

Description: Returns the workpiece position correction active in <Channel> in <Buffer>.

Six "Double" values are provided:

[X_w offset, Y_w offset, Z_w offset, Phi, Theta, Psi].

If no workpiece position correction is active, 0.0 is always returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(114,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-206: Syntax MCODES(114,...)

MCODS(115,...) - Total of placements

Description: Returns the total number of placements active in <Channel> and the workpiece position correction in <Buffer>.

Six "Double" values are provided:

[X_w offset, Y_w offset, Z_w offset, Phi, Theta, Psi].

If no workpiece position correction and no placement is active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(115,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-207: Syntax MCODES(115,...)

MCODS(116,...) - Active program coordinate offset

Description: Returns the program coordinate offset active in <Channel> in <Buffer>.

A "Double" value is provided for each coordinate.

If no program coordinate offset is active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(116,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-208: Syntax MCODES(116,...)

MCODS(117,...) - Active additive program coordinate offset

Description: Returns the additive program coordinate offset active in <Channel> in <Buffer>.

A "Double" value is provided for each coordinate.

If no program coordinate offset is active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(117,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-209: Syntax MCODES(117,...)

MCODS(118,...) - Total of active program coordinate offsets

Description: Returns the total of the program coordinate offset active in <Channel> in <Buffer>.

A "Double" value is provided for each coordinate.

If no program coordinate offset is active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax: **MCODS(118,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-210: Syntax MCODES(118,...)

MCODS(119,...) - Current axis velocity

Description: Provides the **current axis velocities** of the specified channel in <Buffer>.

If -1 is specified for <channel>, the axis velocities of all system axes are returned.

Otherwise, the command provides the current axis velocities of the specified channel.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(119,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-211: Syntax MCODES(119,...)

MCODS(120,...) - Current axis speed

Description: Provides the **current axis speeds** of all system axes in <Buffer>.

The axis speeds are only available for asynchronous, endlessly rotating rotary axes.

Update: After each block switching

Data type of <Buffer>: Double, array

Syntax: **MCODS(120,-1,<Version>,<Puffer>,<Size>)**

Tab. 7-212: Syntax MCODES(120,...)

MCODS(121,...) - Programmed axis speed

Description: Provides the **programmed axis speeds** of all system axes in <Buffer>.

The axis speeds are only available for asynchronous, endlessly rotating rotary axes.

Update: After each block switching

Data type of <Buffer>: Double, array

Syntax: **MCODS(121,-1,<Version>,<Buffer>,<Size>)**

Tab. 7-213: Syntax MCODES(121,...)

MCODS(122,...) - Angle of rotation

Description: Provides the **angle of rotation** for the active plane in <Channel> in <Buffer>.

A "Double" value is provided.

If no angle of rotation is active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double

Syntax:

MCODS(122,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-214: Syntax MCODES(122,...)

MCODS(123,...) - Pole position for mirroring and rotation

Description: Provides the **pole data programmed** in <Channel> in <Buffer>.
A "Double" value is provided for each specified coordinate.
If no pole data is programmed, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(123,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-215: Syntax MCODES(123,...)

MCODS(124,...) - Active offset coordinates

Description: Provides the **active offset coordinates** in <Channel> in <Buffer>.
A "Double" value is provided for each coordinate.
If no offset coordinates are active, 0.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(124,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-216: Syntax MCODES(124,...)

MCODS(125,...) - Active scaling factors

Description: Provides the **active scaling factors** in <Channel> in <Buffer>.
A "Double" value is provided for each factor.
If no scaling factors are active, 1.0 is returned.

Update: After change

Data type of <Buffer>: Double, array

Syntax:

MCODS(125,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-217: Syntax MCODES(125,...)

MCODS(126,...) - Active coordinate mirroring

Description: Provides **active coordinate mirroring** in <Channel> in "Buffer".
An integer value is provided for each coordinate.
If no mirroring is active, 0 is returned. Otherwise, 1 is returned.

Update: After change

Data type of <Buffer>: Integer, array

Syntax:

MCODS(126,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-218: Syntax MCODES(126,...)

MCODS(127,...) - Status of block pre-run re-entry mark

Description: Provides the status of the axes existing in the **block pre-run re-entry mark** in <Buffer>.
If a block pre-run re-entry mark was generated or changed, 1 is returned. If it is deleted, 0 is returned.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(127,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-219: Syntax MCODES(127,...)

MCODS(128,...) - Block pre-run adjustment status

Description: Provides the status of the currently performed **block pre-run adjustment** in the <Buffer>.

The following adjustment state is returned:

- 0: Block pre-run not active
- 5: Computation active
- 10: Target block found; waiting for NC start
- 20: Init adjustment program active
- 30: Manual data input active (overwriting after block pre-run)
- 40: Action block active
- 60: Target block adjustment program active
- 80: Approaching motion active

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(128,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-220: Syntax MCODES(128,...)

MCODS(129,...) - Channel reset

Description: Waiting for a **channel reset** performed in the <Channel>. Initial data cannot be read. That means, the command only returns at next reset of the specified channel.

The channel number of the reset channel is returned in <Buffer>.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(129,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-221: Syntax MCODES(129,...)

MCODS(131,...) - Maximum axis velocity

Description: Provides the **maximum axis velocities** of the specified channel in <Buffer>.

If -1 is specified for <channel>, the axis velocities of all system axes are returned.

Otherwise, the command provides the axis velocities of the specified channel.

The maximum axis velocity is only available for axes in jog mode and asynchronous axes

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(131,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-222: Syntax MCODES(131,...)

MCODS(132,...) - Current temperature compensation

Description: Provides the **current temperature compensation** in <Buffer>

If -1 is specified for <Channel>, the temperature compensations of all system axes are provided.

Otherwise, the command provides the temperature compensations for the specified channel.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(132,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-223: Syntax MCODES(132,...)

MCODS(133,...) - Current lead screw error compensation

Description: Provides the **current lead screw error compensation** in <Buffer>

If -1 is specified for <Channel>, the lead screw error compensations of all system axes are provided.

Otherwise, the command provides the lead screw error compensations for the specified channel.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(133,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-224: Syntax MCODES(133,...)

MCODS(134,...) - Current cross compensation

Description: Provides the **current cross compensation** in <Buffer>

If -1 is specified for <Channel>, the cross compensations of all system axes are provided.

Otherwise, the command provides the cross compensations for the axes of the specified channel.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(134,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-225: Syntax MCODES(134,...)

MCODS(135,...) - Current compensation offset

Description: Provides the **current compensation offset** in <Buffer>.

For the compensation offset, it is the total of the active, effective compensations.

If -1 is specified for <Channel>, the offset for the system axes are provided.

Otherwise, the command provides the offset for the axes of the specified channel.

Update: Cyclic

Data type of <Buffer>: Double, array

Syntax: **MCODS(135,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-226: Syntax MCODES(135,...)

MCODS(142,...) - Programmed path feeds

Description: Some NC functions can be programmed with two feeds, e.g., G94. Using MCODES(142,...), different feeds can be determined.

The feeds are read from the channel <Channel> and are returned in the REAL array <Buffer>. The <Size> provides the number of elements in the array.

The currently applicable programmed feed index is provided with MCODES(143,...). MCODES(24,...) always provides only the first programmed feed, i.e., the first element that is also provided with MCODES(142,...).

Update: After each block switching

Data type of <Buffer>: Real, array

Syntax: **MCODS(142,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-227: Syntax MCODES(142,...)

MCODS(143,...) - Active programmed path feed

Description: This command provides the index of the currently active programmed path feed. This path feed array can be determined with MCODES(142, ...). Since the active programmed feed can be switched by a PLC interface signal, the index of the programmed path feed can change (multiple times) even during a single block.

Update: After change

Data type of <Buffer>: Integer

Syntax: **MCODS(143,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-228: Syntax MCODES(143,...)

MCODS(145,...) - Peak value and maximum value of axis velocity

Description: Returns peak value, maximum value and nominal value of the current axis velocity of the specified channel in <Buffer>. If -1 is specified for <Channel>, the axis velocity values of all system axes are provided. Otherwise, the command provides the current axis velocity values of the specified channel. Three values consisting of peak value, maximum value and nominal value are transferred for each axis. Divide the peak and the maximum value by the nominal value to determine a value in percent for the peak value and the maximum value.



The control first provides the values for the velocity and acceleration axis load diagnostics if these diagnostics are activated via system data entry (/SysNcDgnCtrl/AxesLoadDgn/VelLoadDgn/DgnMode = 2, .../AccLoadDgn/DgnMode = 2).



Switch off the velocity and acceleration load diagnostics using the system data entry (.../VelLoadDgn/DgnMode = 0, .../AccLoadDgn/DgnMode = 0) if these diagnostics are not required anymore! This impedes unnecessary loads of the interpolator interrupt.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax: **MCODS(145,<Channel>,<Version>,<Buffer>,<Size>)**

Tab. 7-229: Syntax MCODES(145,...)

MCODS(146,...) - Peak value and maximum value of axis acceleration

Description: Returns peak value, maximum value and nominal value of the current axis acceleration of the specified channel in <Buffer>. If -1 is specified for <Channel>, the axis acceleration values of all system axes are provided. Otherwise, the command provides the current axis acceleration values of the specified channel. Three values consisting of peak value, maximum value and nominal value are transferred for each axis. Divide the peak and the maximum value by the nominal value to determine a value in percent for the peak value and the maximum value.



The control first provides the values for the velocity and acceleration axis load diagnostics if these diagnostics are activated via system data entry (/SysNcDgnCtrl/AxesLoadDgn/VelLoadDgn/DgnMode = 2, .../AccLoadDgn/DgnMode = 2).



Switch off the velocity and acceleration load diagnostics using the system data entry (.../VelLoadDgn/DgnMode = 0, .../AccLoadDgn/DgnMode = 0) if these diagnostics are not required anymore! This impedes unnecessary loads of the interpolator interrupt.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax:

MCODS(146,<Channel>,<Version>,<Buffer>,<Size>)

Tab. 7-230: *MCODS(146,...) syntax*

MCODS(147,...) - Actual axis position of the CNC

Description: Seen from the CNC, it provides the actual positions of all **feed and auxiliary axes** in ascending and channel-independent order in <Buffer>:

- for linear axes in mm

- for rotary axes in degree

"Set program position" (e. g. SetPos) is **not** included in the values.



The values provided by MCODES(147..) differ from MCODES(35..) by the axis error corrections LSEC, CCOMP and the temperature compensation.

Update: Cyclic

Data type of <Buffer>: Real, array

Syntax:

MCODS(147,-1,,<Version>,<Puffer>,<Size>)

Tab. 7-231: *Syntax MCODES(147,...)=*

Programming example for MCODES

Example: *Auxiliary functions*

```
:
N10 M3 S1234 T5678
WAIT
10 DIM GROUPID$(64)
15 DIM SYNTAX$(256)
17 ERSION=0
18 CHAN%=1
20 REM find group numbers of M3 and T
25 SYNTAX$="T M3" +CHR$
30 ERR_VAR%=MCODS (94,CHAN%,VERSION,GROUPID$,64,SYNTAX$)
```

```

31 PRN#(0,"94 Error:", ERR VAR%)
32 PRN#(0,"T M3:",ACS(MID$(GROUPID$,1,1),
    ACS(MID$(GROUPID$,2,1)))
33 REM read groups 2 and 1
36 GROUPID$=CHR$(2)+CHR$(1)+CHR$(0):
37 REM read groups 2 and 1
50 VERSION=0
58 CHAN%=1
60 ERR VAR%=MCODS(95,CHAN%,VERSION,SYNTAX$,256,GROUPID$)
65 PRN#(0,"95 Error:", ERR VAR%)
70 RN#(0,"Group 2 1 : ", SYNTAX$)
71 REM SYNTAX$ contains e.g. "S1234 M3"

```

MCOPS (General Part)

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Motion Control Process Service** - Motion Control Process Services

Calls Motion Control process services of the NCS via CPL. Thus, channels in the CNC can be controlled.

Syntax:

MCOPS (<Fct>, <Channel> [[, [<P1>] [, [<P2>] [, [<P3>]]]], <P4>])	
<Fct>	Integer expression Indicates the function to be executed. The following table describes all available functions.
<Channel>	Integer expression Indicates the channel to be subject to the function.
<P1> ... <P4>	Optional parameters depending on <Fct>. Sequences of commas are permitted. However, a comma before a closing parenthesis is not permitted.

Tab. 7-232: MCOPS syntax



In the following tables, some of the parameters specified in the syntax are integer constants. Instead of these constants, integer variables can also be programmed. However, they the specified value when the command is called has to be assigned to then.

MCOPS(1,...) - Cancel distance to go

Description: **Motion Control Process Service(1,...)** - Motion Control Process Service(1,...): Canceling distance to go

Calls the Motion Control process service of the NCS to "cancel the distance to go".

- After triggering "Cancel distance to go", all prepared NC blocks including the remaining of the current block are cancelled and prepared again.

CPL blocks or CPL parts are not considered:

Example:

The CPL variable POS has the value 10 for preparation. The NC word X[POS] is interpreted as X10 following "Cancel distance to go" even though POS can have a completely different value at that time.

Any possible corrections values are considered:

- In the display, the indicated end point is set to the current position and causes a simultaneous deletion of the indicated distance to go. The <Channel> then changes to the NC ready state (iCh_NCRReady PLC input signal).

- Following the NC start (PLC output signal qCh_NCStart), the program is continued at the point of interruption considering the new correction values.

Example for the use of MCOPS(1,<Channel>): After the correction tables have been changed and the new values also apply to the previously prepared blocks.

Restriction:

The returned error code 0 does not ensure the successful execution of the function but only its acceptance. Errors during "Cancel distance to go" are not taken into consideration.

Syntax:

MCOPS(1,<Channel>)	
<Channel>	Integer expression Indicates the channel to be subject to the function

Tab. 7-233: Syntax MCOPS(1,...)

MCOPS(2,...) - Reset

Description: **Motion Control Process Service(2,...)** - Motion Control Process Service(2,...): Reset

Calls the Motion Control process services of the NCS for "reset".

To initiate an overall reset: <Channel> = -2

- The channel does temporarily not accept any more tasks such as program selection or operation mode switching.
- The interpolator is stopped.
- Tasks for the channel - that have not yet been processed - are cancelled.
- The main program is deselected.
- Modified machine parameters that do not require a startup are accepted, e.g. parameter CHAN/Ch/Ini/ChResetState (MP 7060 00020) (software limit switch).
- Error and warnings that this channel initiated are deleted.
- The interpolator is restarted.
- The switched-on state for the reset (machine parameter 7060 00020) is accepted, i.e. the corresponding modal states are applied.
- The channel sends the interface signal 0.2 "Reset executed" and is again ready to receive new tasks.

Syntax:

MCOPS(2,<Channel>[,<ResetVariant>])	
<Channel>	Integer expression Indicates the channel to be subject to the function. <Channel> = -2: A total reset is initiated.
<ResetVariant>	Integer expression Specifies the behavior of the function. The following list contains all defined behavioral patterns. Each pattern is preceded by an ID. To set a certain behavior, the corresponding ID has to be provided in <Reset variant>. If several behavioral patterns are to be combined, the total of all corresponding IDs in ResetVariant has to be transferred to the function. Until now, the list includes only one element: Code number: 2: Automatic program reselection is suppressed if active.

Tab. 7-234: Syntax MCOPS(2,...)

Example:

10 ERR_VAR=MCOPS(2,2,2)

;reset in second channel without automatic program reselection.

MCOPS3,...) - Searching the block

Description:**Motion Control Process Service(3,...)** - Motion Control Process Service(1,...): Search block*Calls the Motion Control process services of the NCS to "search a block":*

- <Starting block> and <End block> are transferred as string expressions. The following conventions apply when searching for a <Starting block> and an <End block>:
- Processing starts with the <Starting block> and end with the <End block>.
If the <Starting block> is missing or if the block is not found, processing starts at the start of the program.
If the <End block> is missing or if the block is not found, processing stops at the end of the program.
- The NC status changes after "Search block" to READY.

Syntax:

MCOPS(3,<Channel> [, <StartingBlock>] [, <EndBlock>])	
<Channel>	Integer expression Indicates the channel to be subject to the function.

<Starting block>	String expression
<End block>	- Spaces , <Tab>, <LF> at the beginning of an NC block are ignored. - If the <StartingBlock> or the <EndBlock> starts with a digit and if the expression is not found in the searched program, the system search again for the expression, but this time with a preceding "N" character. Thus, e.g. "50" also finds the NC block "N50X100". - If the <Starting block> or the <End block> end with a numerical value, the expression in the program can only be found if not other numerical value is added immediately after the first, e.g. "G1X10" does not find the NC block "G1X100". - If the <Starting block> or <End block> ends with a letter, the expression in the searched program is only found if a space follows immediately. e.g.: For example, "50A" finds the NC block "50A =1" but not "50A=1".

Tab. 7-235: Syntax MCOPS(3,...)

Example: 10 ERR_VAR=MCOPS(3,2,"N50","N100")

;Initiates "Search block" in channel 2. The main program is to be processed starting at N50 until N100 (inclusive).

MCOPS(4,...) - Selecting the program or manual data input

Description: **Motion COntrol Process Service(4,...)** - Motion Control Process Service(4,...): Select program or manual data input

Calls the Motion Control process services of the NCS to select a program or a manual data input:

In the programmed <Channel>, it selects a program or a string for processing using the manual data input mode.

The program or the string are selected automatically without a link run.

Syntax:

MCOPS(4,<Channel>[[,<String>],[<StartingBlock>],[<EndBlock>]]],<Selection-Type>])	
<Channel>	Integer expression Indicates the channel to be subject to the function.

<String>	String expression <i>Depending on the <Selection type>, the system interprets the parameter as:</i> • path name (including part program name) of a part program to be selected (max. 100 characters) or • If 32 is specified in the <Selection type>: as an NC block (size max. 512 bytes incl. the terminating 0-byte) that is to be processed using the "manual data input" operation mode, or • If 32+4096 is specified in <Selection type>: as several NC blocks to be processed using the "manual data input" mode. Several NC blocks are separated by NewLine ("\\n", Hex 0x0A). The maximum size of all NC blocks must not exceed 4096 bytes including the terminating 0 byte. • If 32768 is specified in <Selection type>: The name or path name is provided for the task "SETINT" for asynchronous subroutines.

<Starting block>	String expression
<End block>	The <Starting block> and the <End block> specify the starting and end blocks in the part program for processing. If the system interprets <String> as manual data input, <Start block> and <End block> are ignored. • Spaces , <Tab>, <LF> at the beginning of an NC block are ignored. • If the <StartingBlock> or the <EndBlock> starts with a digit and if the expression is not found in the searched program, the system search again for the expression, but this time with a preceding "N" character. Thus, e.g. "50" also finds the NC block "N50X100". • If the <Starting block> or the <End block> end with a numerical value, the expression in the program can only be found if not other numerical value is added immediately after the first, e.g. "G1X10" does not find the NC block "G1X100". • If the <Starting block> or <End block> ends with a letter, the expression in the searched program is only found if a space follows immediately. e.g.: For example, "50A" finds the NC block "50A =1" but not "50A=1". Specifications are possible for asynchronous subroutines if <Selection type>: = 32768. They can be logged in, switched off, restarted, deleted and started. • In case of prescriptions for asynchronous subroutines (<SelectionType>= 32768), the number ($1 \leq \text{Number} \leq 8$) is provided in the <StartBlock> as a string (e. g. "1") and <end block>. Tasks for asynchronous subroutines: "SETINT" log in "DISABLE" switch off "ENABLE" switch on again "CLRINT" delete "START" start
<Selection type>	Integer expression Specifies the behavior of the function. The following list contains all defined behavioral patterns. If a certain behavior is to be set. If a certain behavior is to be set, this ID has to be provided in <Selection type>. If several behavioral patterns are to be combined, the total of all the corresponding IDs in the <Selection type> has to be transferred to the command. If <SelectionType> 2 (wait until the NC state switches to READY) is entered and the program to be selected does not exist or cannot be executed, error message 6 is returned. In all other cases, it is not checked whether the program can be executed. The command returns 0 (no error occurred). A corresponding runtime error is only generated by subsequent linking.

Values for the selection type:	Meaning:
1	During the selection, the system performs a link run. If not existent, a link table is generated for the selected main program. Link tables are required if subroutine calls or CPL commands exist in the program.
2	The system acknowledges the validity of a selection when the NC state switches to READY. Normally, the selection is acknowledged without waiting for the NC state READY.
32	The system interprets <String> as manual data input block.
64	Before the command selects the specified program or manual data input block, an active program or manual data input block is deselected.
256	Required to traverse axes in "jog" mode or if motion are to be performed in the "jog in workpiece coordinates" mode.
512	Required to start axes in the "approach reference point" mode.
2048	An already active program is replaced by the newly selected one. As a result, all modal states remain. In manual data input, the old character sequence is replaced by the new one.
4096	Entering buffered NC blocks is applied together with ID 32. While preceding blocks are still executed, others can already be specified.
32768	Using this flag, asynchronous subroutines can be controlled (see <Starting block> and <End block>).
65536	Block pre-run Settings for block pre-run can be made using this flag. Data is specified as XML string in StartBlock. They are enclosed by <SearchRun> and </SearchRun>. Thus, the following XML string is transferred for the input of the target position 100: <SearchRun> <DesPos> 100 </DesPos> </SearchRun>

131072	Other blocks follow. Is effective together with manual data input (32) and input of buffered NC blocks (4096). A block look-ahead does not stop if the block buffer is empty (and generates a down ramp), but waits until blocks are input again.
262144	Overwrite: If feed hold is present at an active program, the NC block is transferred to the channel in <String>. The channel switches to the overwriting mode and more blocks can be specified and processed. If specified additionally with 2048, already specified blocks can be modified, deleted or a syntax error can be reset. This flag can specify NC blocks for overwriting. <i>It is distinguished between:</i> - Overwriting in automatic modes With the first specification, it is switched to an internal overwriting mode in the automatic modes if feed hold is present. Further NC blocks can then be specified. - Overwriting after block pre-run at the end of the Init adjustment program If the NC is in the "Wait for manual data input" state at the end of the Init adjustment program, this ID can be used to specify NC blocks for the manual adjustment. This command is only effective together with block pre-run (65536).

Tab. 7-236: MCOPS(4,...)

Examples:

```
10 ERR_VAR=MCOPS(4,1,"sekt.cnc","N50","N100",1)
;Program selection of the program "sekt.cnc" in channel 1, including block
search and linking.
20 ERR_VAR=MCOPS(4,1,/usr/"user/p1.cnc")
;Program selection of "p1.cnc" in channel 1 without block search and linking.
30 ERR_VAR=MCOPS(4,1,"F1000G1X500",,,32)
;Selects the block "F1000G1X500" in channel 1 using manual data input.
40 ERR_VAR=MCOPS(4,1,"S1000",,,262144)
;Activates feed hold in the secondary mode "Overwrite" and selects the block
"S1000". It is processed with NC start.
50 Flags=262144 + 2048
60 ERR_VAR=MCOPS(4,1,"STA(SP1)",,,Flags)
An incorrect or already specified block is reset and the block "STA(SP1)" is
selected.
```



If not explicitly specified by choosing the selection type, no link run is used for the selection.

MCOPS(5,...) - Deselecting the program

Description: **Motion Control Process Service(5,...)** - Motion Control Process Service(5,...): Deselect program

Calls the Motion Control process services of the NCS for program deselection:

Deselects a selected program or manual data input block in the programmed <Channel>.

Syntax:

MCOPS(5,<Channel>, <DeselectionVariant>)	
<Channel>	Integer expression Indicates the channel to be subject to the function.
<DeselectionVariant>	Integer expression Specifies the behavior of the function. The following list contains all defined behavioral patterns. Each pattern is preceded by an ID. To set a certain behavior, the corresponding ID has to be provided in <DeselectionVariant>. If several behavioral patterns are to be combined, the total of all corresponding IDs in <DeselectionVariant> has to be transferred. Until now, the list has only one ID: 2: If automatic program reselection is active, it can be suppressed with the value 2 in this program deselection.

Tab. 7-237: Syntax MCOPS(6,...)

Example: 10 ERR_VAR=MCOPS(5,2,2)
;Program deselection in channel 2 without automatic program reselection.

MCOPS(6,...) - Starting the program

Description: **Motion Control Process Service(6,...)** - Motion Control Process Service(9,...):
Start program

Calls the Motion Control process services of the NCS for program start:
Starts a selected program or a selected manual data input block in the programmed <Channel>.

Syntax:

MCOPS(6,<Channel>)	
<Channel>	Integer expression Indicates the channel to be subject to the function.

Tab. 7-238: Syntax MCOPS(6,...)

MCOPS(7,...) - Specifying the operation mode

Description: **Motion Control Process Service(7,...)** - Motion Control Process Service(7,...):
Specify operation mode

Calls the Motion Control process services of the NCS to specify the operation mode:

Thus, only one operation mode is specified for the programmed channel.

The operation mode can be changed only under the following conditions:

- The channel interface signal "qCh_OpModePlc" (PLC operation mode) may not be set
 - No program or block is selected for the NC
- or -
- Switch only between automatic modes, continuous block, program block, single step or single block.

Syntax:

MCOPS(7,<Channel>,<OperationMode>)	
<Channel>	Integer expression Indicates the channel to be subject to the function.
<Operation mode>	Integer expression - Jog mode: Axes can be jogged (+/-). Also refer to MCOPS(4..) under <Selection type>: Characteristic value 256. - Approach reference point: The axes can be started using signals "Manual+ " or "Manual-". Also refer to MCODS(4,...) under <Selection type>: ID 512. - Block pre-run: Part programs can be processed up to a target block without motion. - Manual data input: Individual blocks can be specified for machining. - Automatic (continuous block): Part programs are processed completely. - Automatic (program block): The blocks of a part program are individually prepared and processed with "NC start". - Program block (single step): The blocks of a part program are prepared in relation to each other, but processed individually with "NC start". Even internally generated NC blocks are individually processed. - Teaching: Axes can be jogged (+/-) or traversed with the handwheel. Individual blocks can be specified for machining. - Not assigned - Program block (single block): In contrast to the single step, all blocks internally generated due to a programmed NC block are transferred to the interpolator for processing at each NC start. - Return to contour: Axes can be moved manually away from the contour and subsequently approached. - CPL debugger: Single blocks are processed as blocks in the part program - CPL debugger: All blocks up to the next breakpoint are processed. - Jogging in workpiece coordinates: Axes can be jogged in workpiece coordinates.

Tab. 7-239: Syntax MCOPS(7,...)

Example: 10 ERR_VAR=MCOPS(7,2,5)
;Operation mode change in the second channel to automatic (continuous block).

MCOPS(8,...) - Changing return to contour strategy

Description: **Motion COntrol Process Service(8,...)** - Motion Control Process Service(8,...): Change return to contour strategy

Calls the Motion Control process services of the NCS to specify the return to contour strategy:

Thus, the return to contour strategy is specified in the programmed <Channel>.

Syntax:	MCOPS(8,<Channel>,<Mode>,<ApproachPoint>)
<Channel>	Integer expression Indicates the channel to be subject to the function.
<Mode>	Integer expression <i>Options:</i> • 1: Automatic return to contour • 2: Return to contour with single block • 3: Manual return to contour
<Approach point>	Integer expression <i>Options:</i> • 1: to the starting point • 2: To the end point • 3: to the interruption point

Tab. 7-240: Syntax MCOPS(8,...)

MCOPS(9,...) - Completing return to contour recording

Description: **Motion COntrol Process Service(9,...)** - Motion Control Process Service(9,...): Complete return to contour recording

Calls the Motion Control process services of the NCS to specify the return to contour recording:

Completes the return to contour recording in the programmed <Channel>. Jog motions are thus no longer recorded.

Syntax	MCOPS(9,<Channel>)
<Channel>	Integer expression Indicates the channel to be subject to the function.

Tab. 7-241: Syntax MCOPS(9,...)

MCOPS(10,...)

Description: Does not apply to MTX

MCOPS(11,...) - Channel restart

Description: **Motion COntrol Process Service(11,...)** - Motion Control Process Service(11,...): Channel reset

Calls the Motion Control process services of the NCS to "restart the channel". This functionality restarts the channel at the beginning of the selected main

program. There is no reset. Thus, the switch-on state is also not applied at reset.

If the program is for example waiting for a permanent variable ("WPV"), waiting is canceled.



The modal states are not modified! Exception: Modally effective subroutines/cycles.

In the main program, all modal functionalities used during the program have to be reset, since all active modal NC functions remain active at channel restart.

The command MCOPS(11,...) only returns after the channel restart or if an error occurred.

Error messages Error messages are returned as return value. Possible error messages are specified in [chapter "Error return values" on page 541](#).

Error code 2: A channel restart task is already executed in this channel.

Error code 37: Channel restart failed due to an error.

Error code 38: The current operation mode does not permit a channel restart. Valid are only operation modes in which a main program was selected. That means that the manual data input and the buffered NC block specification are not valid for example. An asynchronous subroutine may also not be active.

- Sequence:**
- The channel does temporarily not accept any more tasks such as program selection or operation mode switching.
 - The interpolator is stopped.
 - Tasks for the channel - that have not yet been processed - are canceled.
 - The active program is reset to the beginning of the main program.
 - Local and global CPL variables are reset.
 - Modal subroutines are disabled.
 - The interpolator is restarted.
 - The channel switches to the "NC ready" state (PLC input signal iCh_NCRReady).

Syntax:

MCOPS(11,<Channel>)	
<Channel>	Integer expression Indicates the channel to be subject to the function.

Tab. 7-242: MCOPS(11,...) syntax

Example: 10 ERR_VAR=MCOPS(11,1)
;Channel restart in first channel.

Error return values

Description: All command calls provide a return value for checking purposes and error handling. This value can be assigned to an "Integer" or "Real" variable.

Syntax examples: ERR_VAR% = MCOPS (. . .)
ERR_VAR% = MCODS (. . .)



Wrong program reactions possible!

If called commands return an error code, actions important for further program execution were either not performed or not performed completely.

We strongly recommend that you check the program (e.g. using CASE) after a command is called to find out whether the command could be executed without errors. The remaining program behavior depends on the type and severity of the error.

The following return values are defined:

- 0: No error occurred
- 1: The specified channel does not exist.
- 2: The command cannot be executed, since the specified channel is currently assigned (the current state does not permit this action).
- 3: An initiated reset in the specified channel has not yet been completed.
- 4: The specified program name is too long (currently not in use).
- 5: The command requests approached reference points.
- 6: The specified program does not exist or cannot be executed.
- 7: With buffered NC block specification, writing to the buffer was interrupted. A second instance tried to write to the buffer at the same time.
- 8: The command cannot be executed in the current operation mode.
- 9: The channel cannot be started, since the state is not READY.
- 10: The command cannot be executed, since no program is selected.
- 11: The specified program cannot be selected, since the channel state does not permit this (e.g. state of block preparation and interpolator is "RUNNING").
- 12: Currently not in use
- 13: The operation mode cannot be switched, since the channel state does not permit this.
- 14: The target of "Search block" was not found.
- 15: "Search block" is not possible, since - although the channel state is READY - processing the main program has already been started (e.g. program is at M0).
- 16: The number of axes is too high when specifying external zero offsets.
- 17: The number of axes ZO groups is too high when specifying external zero offsets.
- 18: The specified syntax is unknown.
- 19: Invalid index when specifying an external tool correction.
- 20: The number of corrections when specifying an external the tool correction is too high (possibly together with the correction index).
- 21: Invalid format specification when providing an external tool correction.
- 22: Invalid edge position with specification of an external tool correction.
- 23: Invalid correction group
- 24: Addressed axis is not available.
- 25: During NC block default with automatic start, a runtime error was detected, e.g. a syntax error.

26:	When a buffered NC block is specified, there was a buffer overflow.
27:	The coordinate filter specification is incorrect.
28:	Specification for the virtual NC is not permitted, since the state is not frozen.
29:	Lack of memory in the server. Requested data could not be stored.
30:	An invalid value has been specified for the Ncs function. Thus, the server cannot provide data.
31:	Incorrect specifications for asynchronous subroutine.
32:	Incorrect specifications for block pre-run.
33:	The correction value could not be transferred, since the last correction value has not yet been applied by the interpolator.
34:	Transferred client buffer too small.
35:	The processed call chain is invalid.
36:	A value was programmed for the tool correction. The amount of the value is either so high or low that it is not reasonable anymore.
37:	Channel restart failed due to an error.
38:	Channel restart is invalid due to the current operation mode and the current channel state.
100:	The magic number of the telegram is incorrect.
101:	NCS communication is not proper.
102:	The specified function is not available in this software version.
103:	An internal error occurred (currently not in use).



If the calling program also contains NC blocks, the block preparation is generally performed before processing at the machine. If a process service is requested or a machine state queried with the MCOPS or MCODS command at the time of block preparation, the prerequisites at the machine are not yet fulfilled.

However, this problem only applies to commands that access exactly the channel in which they are executed themselves.

In this case, use the "WAIT" command in the line preceding the call. This stops block preparation until all the blocks before "WAIT" have actually been processed.

Examples of programming

Example 1: Request SAV state and interpolator of channel 2 immediately

10 DIM BUF% (2)	Create array
20 VERSION=0	Provide data immediately
30 ERR_VAR%=MCODS (32, 2, VERSION, BUF%, 2)	Command call

BUF%(1) contains the SAV state. BUF%(2) contains the IPO state.

The current version number of the data is displayed in VERSION (important for example 2).

Example 2: Wait until SAV state of channel 2 switches to "inactive"

<Code of example 1>

CPL commands

Program:

```

10  INAKTIV = 1
20  WHILE BUF%(1) <> INAKTIV DO
30    ERR_VAR% = MCODS(32,2,VERSION,BUF%,2)
40  END

```

Example 3: *Display the axis names in the MSG window*

```

:
30  VERSION=0
40  DIM AXNAME$(512)
50  REM Request all axis names
60  ERR=MCODS(33,-1,VERSION,AXNAME$,512)
70  IF ERR=0 THEN
80    REM Determine number of axes
90    DIM AXNMB%(3)
100   VERSION=0
110   ERR=MCODS(45,-1,VERSION,AXNMB%,3)
120   ANZ=AXNMB%(2)
130  ENDIF
140  IF ERR<>0 THEN
150    PRN#(0,"Error occurred: ",ERR)
160  ELSE 170 REM Displaying the axis names
180    FOR I%=0 TO (ANZ-1)
190      NAME$=MID$(AXNAME$,I%*9+1,8)
200      IF ASC(NAME$)<>0 THEN
210        REM Axis name defined
220        PRN#(0,I%+1,". Axis name: ",NAME$)
230      ENDIF
240    NEXT
250  ENDIF
N260 M30

```

Example 4: *Channel / wait status*

```

10  CHAN%=1
20  VERSION%=0
30  STATES%=0
40  ERR=MCODS(87,CHAN%,VERSION%,STATES%,1)
50  IF ERR=0 THEN
60    MASKE%=1
70    WHILE MASKE% <= STATES% DO
80      CASE (STATES% AND MASKE%) OF
90        LABEL 1:PRN#(0,"Dwelling time")
100       LABEL 2:PRN#(0,"Acknowledgement required for help function")
110       LABEL 4:PRN#(0,"Read in blocker")
120       LABEL 8:PRN#(0,"Feed in channel equal to 0")
130       LABEL 16:PRN#(0,"Program Stop using M0/M1")
140       LABEL 32:PRN#(0,"Feed Stop in channel")
150       LABEL 64:PRN#(0,"Feed blocker in channel or in channel axis")
160       LABEL 128:PRN#(0,"Read in blocker from customer")
170       LABEL 256:PRN#(0,"Synchronized motion stop between channels (ASTOP, ...)")
180       LABEL 512:PRN#(0,"Waiting for axis during axis exchange (G511)")
190       LABEL 1024:PRN#(0,"Waiting for permanent variable (WPV)")
200       LABEL 2048:PRN#(0,"Waiting for interface signal to active point of time (WAITA, ...)")
210       LABEL 4096:PRN#(0,"Waiting for If signal (WAIT(BITIF(...))) or")
220         PRN#(0," (WAIT(,ZEIT%)) in block preparation")
230       LABEL 8192:PRN#(0,"Motion Control data service (MCODS(...))")
240     ENDCASE
250     MASKE%=MASKE%*2
260   END
270 ENDIF
M30

```

Example 5: *Axis-channel assignment*

```

10  REM Program queries number of axes in system
15  REM and number of axis of channel axis. By means of the

```

```

20 REM information, the channel axes are set to
30 REM position 0 and then traversed to position <channel number>
40 REM
50 CHAN%=SD(8) : REM Individual channel number
60 IDCHAX%=43 : REM Function type for assignment axis - channel
70 IDMAXAX%=45 : REM Function type for number of axes
80 DIM BUF%(16): REM Buffer for assignment axis - channel
90 SIZE%=16
100 ANZ%=0 : REM max. index of system axes
120 REM Determine channel axes of active channel
130 VERSION=0
140 ERR=MCODS (IDCHAX%,CHAN%,VERSION,BUF%,SIZE%)
150 IF ERR=0 THEN
160     VERSION=0
170     ERR=MCODS (IDMAXAX%,CHAN%,VERSION,ANZ%,1)
180     IF ERR=0 THEN
190         FOR I%=1 TO ANZ%
200             IF BUF%(I%) = CHAN% THEN
N210                 F1000 [AXP(I%,0,0)]; Traverse channel axes to 0
220             ENDIF
230         NEXT
240         FOR I%=1 TO ANZ%
250             IF BUF%(I%) = CHAN% THEN
N260                 M0
N270                 WAIT
N280                 F1000 [AXP(I%,CHAN%,0)]; Traverse channel axes to CHAN%
290             ENDIF
300         NEXT
310     ENDIF
330 ENDIF
N310 M30

```

7.4.10 General XML tables

XTAB

STRUCT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Xml TABLE** - Read/write access to any XML table.

In contrast to the special commands for ZO, D-correction and placement tables, this command also provides access to any substructures of a table if a corresponding data type was defined.

Syntax:

XTAB (<Table>,<Substructure>)	
<Table>	Name of the XML table, if applicable including path information Of no path is specified, the table in the root directory is used.
<Substructure>	Desired substructure, provided as XPATH expression

Tab. 7-243: *XTAB syntax*

Example:

```

10 X!=XTAB( "/database/PM1.pmt", "/PMT/Set[1]/G154/Corr/Trans/
XWCS")

20 XTAB( "/database/PM1.pmt", "/PMT/Set[1]/G154/Corr/Trans/
XWCS")=X!+1.0

```

Increment value in table by 1

XTABCR

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Xml TABLE CReate** - Creating a general xml table

XTABCR (<Table>,<Type>[,<TableTemplate>])	
<Table>	Name of the XML table, if applicable including path specification Of no path is specified, the table in the root directory is used.
<Type>	Table type (XML root tag) <i>The following table types are supported:</i> • /ZOT : Zero offset table • /DCT : Table for D-corrections • /PMT : Table for placements
<Table template>	Name of the template XML table, if applicable including path. The specified <Type> has to correspond to the type of the <Table template>. Of no path is specified, the table in the root directory is used.
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK -1: Parameter error -13: Data could not be written -14: XML table exists -15: XML table not found -16: Invalid file extension -17: Invalid table type If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-244: XTABCR syntax

Creates an XML table with no data in the MTX file system.

Syntax:**Example:**

```
10      XTABCR("/database/ZO2.zot","/ZOT","/database/
ZO1.zot")
```

The zero offset table ZO2.zot is created in the directory /database. The table /database/ZO1.zot is used as table template.

7.4.11 Tool database

DBLOAD

Description:

Data Base LOAD - Loading database

Parts of a database table or a complete database table from a file can be read into the database.

The <DataSelection> parameter allows to control whether all data of the data block or only tool-specific or place-specific data is read.

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Syntax:

DBLOAD (<DbTabelle>,<Key1>,<Key2>>,<Filename>[,<DataSelection>])	
<DbTable>	Root tag of the database table
<Key1>	Database key 1
<Key2>	Database key 2
<DataSelection>	Optional (Default: 0): 0: All data of the data block is read 1: Only tool-specific data is read 2: Only place-specific data is read
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-245: DBLOAD syntax

Example:

```
10 DBLOAD ("/DBT1",1,1,"/dbdaten.txt",1,ERRNO)
```

The tool-specific data of the data block (1,1) is read in from the "dbdaten.txt" file.

DBMOVE

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Data Base Move - Data block can be moved in a database

Implementation of a complete data block or the segments of a data block in a database table.

The parameter <DataSelection> can be used to control whether all data of the data block or only tool-specific or place-specific data is implemented.

Syntax:

DBMOVE ((<DbTab>, <SrcKey1>, <SrcKey2>, <DestKey1>, <DestKey2> [, [<Mode>] [,<DataSelection>]])	
<DbTab>	Root tag of the XML file.
<SrcKey1>	Key of source data set
<SrcKey2>	2. Key of source data block
<DestKey1>	1. Key of target data set

<DestKey2>	2. Key of target data block
<Mode>	optional (Default: 0) 0: The selected data of the source data block is transferred to the target data block and the corresponding data of the source data block is subsequently zeroed. 1: The selected data of the source data block is copied to the target data block. The content of the source data block remains. 2: The selected data of the source data block and the target data block are exchanged.
<DataSelection>	optional (Default: 0) 0: All data is moved 1: Tool-specific data is moved 2: Place-specific data is moved
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error -7: Data could not be read -13: Data could not be written -20: Incorrect root tag -21: File access not possible -22: Invalid key -23: Data block is locked If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-246: DBMOVE syntax

Syntax example:

DBMOVE (" /DBT1", 1, 1, 2, 2) The contents of the data block (1,1) are moved to the data block (2,2).

DBSAVE

Description:

Data Base LOAD - Saving database

Use DBSAVE to save a database table completely or in parts into a file.

Syntax:

DBSAVE (<DbTab>,<Key1>,<Key2>,<FileName>[,<Mode>])							
<DbTab>	Root tag of the XML file						
<Key1>	1: Key of the data set						

<Key2>	2. Key of data block
<FileName>	File name
<Mode>	optional (Default: 0) 0: Data is written to the file "FileName". If the file already exists, its content is overwritten 1: Data is attached to the existing file "FileName".
ERRNO	CPL variable. If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK -1: Parameter error -7: Data could not be read -13: Data could not be written -20: Incorrect root tag -21: File access not possible -22: Invalid key -23: Data block is locked If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-247: DBSAVE syntax

If both keys have the value -1, all data blocks are written to the file. If the first key has a value not equal to 1 and the second key the value -1, all data blocks whose first key has the specified value are written into the file.

Syntax example:

```
DBSAVE ("/DBT1", 1, 1, "/dbdaten.txt")
```

The content of the data set (1,1) is saved to the "dbdaten.txt" file.

DBSEA

STRUCT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Data Base SEArch - Search data block

The CPL command returns the header of the first data block that corresponds to the search condition. In this case, the variable <SearchRes> returns 1.

The data block search starts with the defined data block using

<StartKey1> and <StartKey2>

defined data block.

If one of the two starting keys has a value of -1, the search starts at the first data block of the database table.

Syntax:

DBSEA (<DbTable>,<StartKey1>,<StartKey2>,<SearchCond>,<SearchRes>,<ResVar>)]	
<DbTable>	Name of the database table

<StartKey1>	Key1 of starting data block
<StartKey2>	Key2 of starting data block
<SearchCond>	Search condition as string (refer to chapter "DBSEAX" on page 550)
<SearchRes>	Search result, integer variable type 0: No data block found that corresponds to the search criterion. 1: Data block found that corresponds to the search criterion.
<ResVar>	Variable of type integer If <ResVar> is entered, no runtime error results in case of an access error. The following return values are possible: 0: Access OK 1: Error while accessing 2: Insufficient memory 3: Invalid variable type 4: Search criterion faulty If no <ResVar> is specified, a runtime error results in case of an access error.

Tab. 7-248: DBSEA syntax

Example:

```

10    SD.DBRec.Hd=DBSEA("DBT1",-1,-1,"K1=1",FOUND%)
20    WHILE FOUND%=1 DO
30    SD.DBRec.Hd=DBSEA("DBT1", SD.DBRec.Hd.K1, SD.DBRec.Hd.K2, "K1=1",FOUND%)
40    END

```

The following system data is required:

Program:

```

<Variable Storage="volatile">
  <Name>DBRec</Name>
  <Type>DBT1Rec_t</Type>
</Variable>

```

DBSEAX

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Data Base SEARch eXtended** - Extended search for data blocks

Searches for one or more data blocks in a tool database table. The command provides a list of data block headers corresponding to the search criterion. The headers are stored in a system data array <HeaderArr> that is sorted according to K1 and K2. The parameter <ErgSize> is used to specify the maximum number of data block headers to be searched. The return value of DBSEAX provides the number of data blocks found.

The data block search starts with the defined data block using

<StartKey1> and <StartKey2>

defined data block.

If one of the two starting keys has a value of "-1", the search starts at the first data block of the database table.

Syntax:

DBSEAX (<DbTabelle>,<StartKey1>,<StartKey2>,<SearchCond>,<HeaderArr>,<ArrSize>)	
<DbTable>	Name of the database table
<StartKey1>	Key1 of starting data block
<StartKey2>	Key2 of starting data block
<SearchCond>	Search criterion as string (see below)
<HeaderArr>	System data of type data block header (e.g. DBT1Hd_t) If several data block are to be searched, it has to be an array. The array is indexed by incrementing the last index in the SD, e.g. SD.HdArr[1,1] - The second index in HdArr is incremented. SD.SDArr[1,1].RecArr[1].Hd - The RecArr index is incremented
<ArrSize>	Maximum number of searched data block headers Note: <HeaderArr> has to be sufficiently dimensioned to accept this number of headers.
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69) If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-249: DBSEAX syntax

Search criterion (<SearchCond>)

Maintain the following syntax:

- A list of individual conditions / criteria, which have to be fulfilled, is provided.
- A normal single condition looks as follows:
<Tag name><Relational operator><Value> with
 - <Relational operator>:
 - "=" equal to
 - ">" greater than

CPL commands

- ">=" greater than or equal to
- "<" less than
- "<=" less than or equal to
- "<>" not equal to

Only SKQ as relational operator for SKQ: "=" and "<> ",
e.g. SKQ=4711

- A single bitmask condition looks as follows:

<Tag name>=<Value>:<Mask>

- <Value>: Value the relevant bits required.
- <Mask>: Defining the relevant bits.

For example: BQ1=0x307000:0xF0FF00 with:

Only queries regarding equality are possible.

Values and masks are interpreted:

- As hexadecimal number if they start with "0x" or "0X",
- As octal number if they start with "0"
- As decimal number in all other cases.

Within <Value>, preceding or following spaces are not analyzed. If a string is to be specified in which these spaces are important, the substring has to be enclosed inverted commas (e.g.: SKQ=' ').

To use an inverted comma in a substring enclosed in inverted commas, the character "\" has to precede the inverted comma (e.g.: SKQ='Achim \'s')

- The individual conditions are separated by a ",".
- All individual conditions of a list have to refer to different <Tag> names.
- Only predefined tag names may be used.

Boundary conditions**Syntax examples:**

```
K1=1
IKQ2=3
BQ2=0x1:0x1
```

All data blocks have to be found that fulfill a search criterion. For this purpose, the search has to be repeated as often as necessary to ensure that the number of data blocks found is lower than the maximum number of searched data blocks. For every new search, the starting keys have to be set to the key of the latest found data block.

Program:

```
10 KEY1%=-1
10 KEY2%=-1
30 ARRSIZE%=10
31 REPEAT
33   FOUND% = DBSEAX("DBT1",KEY1%,KEY2%,"K1>0",SD.HdArr[1],ARRSIZE%,ERRNO)
35   IF ERRNO=0 THEN
37     IF FOUND%>=1 THEN
59       FOR I%= 1 TO FOUND%
43         PRN#(0,Sector: ",SD.HdArr[I%].K1,", Place: ",SD.HdArr[I%].K2)
46       NEXT I%
49       KEY1%= SD.HdArr[FOUND%].K1
51       KEY2%= SD.HdArr[FOUND%].K2
53   ENDIF
55 ELSE
57   PRN#(0,"Access error on tool database!")
61   ENDIF
63 UNTIL (FOUND%<ARRSIZE%) OR (ERRNO<>0)
```

The following system date is required:

Program:

```
<Variable Dimension="10" Storage="volatile">
  <Name>HdArr</Name>
  <Type>DBT1Hd_t</Type>
</Variable>
```

DBTAB

STRUCT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Data Base TABLE** - Read-only or write access to a database

Can read a complete data set or a substructure of a tool database table into a CPL variable or write from the variable to the data set.

Syntax:

DBTAB (<DbTable>,<key1>,<key2>[,<ResVar>])	
<DbTable>	Name of the database table with substructures (Xpath string). A "." or "/" can be used as separator between structure components.
<Key1>	Database key 1
<Key2>	Database key 2
<ResVar>	Variable of type integer If <ResVar> is entered, no runtime error results in case of an access error. The following return values are possible: 0: Access OK 1: Error while accessing 2: Insufficient memory 3: Invalid variable type If no <ResVar> is specified, a runtime error results in case of an access error.

Tab. 7-250: DBTAB syntax

Example:

```
10 SV.A=DBTAB("DBT1.Rec",1,1)
20 D!=DBTAB("DBT1.Rec.UD.Ed.Geo.L1",1,1)
30 SV.A.UD.Ed.Geo.L1 = 10
40 DBTAB("DBT1.Rec",1,1) = SV.A
```

DBTABX

STRUCT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Data Base TABLE eXtended** - Extended access to a database table

Can read a complete data set or a substructure of a tool database table into a CPL variable or write from the variable to the data set. Parameter <DataSelection> can be used to control whether all data of the data block or only tool- or place-specific data are written.

Syntax:

DBTABX (<DbTabelle>,<Key1>,<Key2>[,<DataSelection>])	
<DbTable>	Name of the database table with substructures (XPath string). A "." or "/" can be used as separator between structure components.
<Key1>	Database key 1
<Key2>	Database key 2
<DataSelection>	In case of write access, the following values can be written to <DataSelection>: 0: All data of the data block is written. 1: Only tool-specific data is written. 2: Only place-specific data is written. If <DataSelection> is not specified, "0" is used as default value. <DataSelection> is of no relevance in case of read-only access.
ERRNO	CPL variable. If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-251: DBTABX syntax

Example:

```

10 SV.A = DBTABX("DBT1.Rec",1,1)
20 D! = DBTABX("DBT1.Rec.UD.Ed.Geo.L1",1,1)
30 SV.A.UD.Ed.Geo.L1 = 10
40 DBTABX("DBT1.Rec",1,1,1,ERRNO) = SV.A
50 IF ERRNO <> 0 THEN
60 SETERR("tool data block could not be written")
70 ENDIF

```

DBTABXL

STRUCT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Data Base TABLE eXtended Lock - Extended database access with locking

Can read a complete data set or a substructure of a tool database table into a CPL variable or write from the variable to the data set. In case of read-only access, the data block is locked. When writing the data block, this lock is removed again.

Syntax:

DBTABXL (<DbTabelle>,<Key1>,<Key2>[,<DataSelection>])	
<DbTable>	Name of the database table with substructures (XPath string). A "." or "/" can be used as separator between structure components.
<Key1>	Database key 1
<Key2>	Database key 2
<DataSelection>	In case of write access, the lock is removed irrespective of <DataSelection>. The following values are possible: 0: All data of the data block is written 1: Only tool-specific data is written 2: Only place-specific data is written 3: No data is written If <DataSelection> is not specified, 0 is used as default value in case of write access. <DataSelection> is of no relevance in case of read-only access.
ERRNO	CPL variable If the CPL variable ERRNO is specified at any position in the parameter list, no runtime error is generated. The error is provided by a corresponding value of the variable. The following return values are possible: 0: Access OK <0: Error (for a detailed error description, refer to chapter 3.14.5 "Variable ERRNO for evaluating errors in CPL functions ERRNO" on page 69). If the CPL variable ERRNO is not specified, a runtime error is generated in case of an access error.

Tab. 7-252: DBTABXL syntax

Example:

```

10 SV.A = DBTABXL("DBT1.Rec",1,1)
20 D! = DBTABXL("DBT1.Rec.UD.Ed.Geo.L1",1,1)
30 SV.A.UD.Ed.Geo.L1 = 10
40 DBTABXL("DBT1.Rec",1,1) = SV.A

```

7.4.12 Errors and warnings

SETERR

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: SET ERROR - Outputting runtime error

After ERRNO has been analyzed, the CPL command SETERR can be used to generate a runtime error. This cancels the part program at this point. The text entered into the parameter <ErrorString> is always output irrespective of the set language.

Using the optional parameter <Mode>, it can be controlled in the current channel, whether the part program name and part program line are addition-

ally output. In case of outputs for a different channel, only the specified text is output

Syntax:

SETERR (<ErrorString> [, [<Channel>] [,<Mode>]]) with	
<Error string>	Text output as error with error number 3371.
<Channel>	Channel number of the channel in which the error is to be output. If <Channel> is not specified, the current channel is used as default value.
<Mode>	Specifies, whether the part program name and the part program line are output in the current channel. 0: No output of part program information 1: Output of part program name and part program line. If <Mode> is not specified, 0 is used as default value.
ERRNO	CPL variable that can be programmed at any position in the parenthesis. When ERRNO is programmed, no runtime error is output. Return values are: 0: Access OK -1: Parameter error -5: Channel does not exist

Tab. 7-253: SETERR syntax

Example: Read the position of channel axis X in the first channel

```
10 POS = ACS("X",1,1,ERRNO)
20 IF ERRNO <> 0 THEN
30 SETERR("position could not be read")
40 ENDIF
```

SETWARN

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

SET WARNING - Outputting runtime warning

After ERRNO has been analyzed, the CPL command SETWARN can be used to generate a warning. The part program continues running. The text specified in the parameter <WarningString> is always output irrespective of the language set.

Using the optional parameter <Mode>, it can be controlled in the current channel, whether the part program name and part program line are additionally output. In case of outputs for a different channel, only the specified text is output.

Syntax:

SETWARN (<WarningString> [, [<Channel>], <Mode>]]) with	
<Warning string>	Text output as WARNING with error number 3372.

<Channel>	Channel number in which the warning is to be output. If <Channel> is not specified, the current channel is used as default value.
<Mode>	Specifies, whether the part program name and the part program line are output in the current channel. 0: No output of part program information 1: Output of part program name and part program line. If <Mode> is not specified, 0 is used as default value.
ERRNO	CPL variable that can be programmed at any position in the parenthesis. When ERRNO is programmed, no runtime error is output. Return values are: 0: Access OK -1: Parameter error -5: Channel does not exist

Tab. 7-254: SETWARN syntax

CLRWARN

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **CLear WARNING** - Deleting warnings

Use the CLRWARN command to clear all warnings of a channel generated by SETWARN.

Syntax:

CLRWARN([<Channel>]) with	
<Channel>	Channel number in which all the warnings generated by SETWARN are to be cleared. If <Channel> is not specified, the current channel is used as default value.
ERRNO	CPL variable that can be programmed at any position in the parenthesis. When ERRNO is programmed, no runtime error is output. Return values are: 0: Access OK -1: Parameter error -5: Channel does not exist

Tab. 7-255: CLRWARN syntax

Example:

```
10 CLRWARN(,ERRNO)
20 IF ERRNO <> 0 THEN
30 SETWARN("warnings not cleared")
40 ENDIF
```

GETERR

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **GET ERROR** - Querying errors

This command queries the current errors in a CPL program. It contains the current error number, channel number of the error and the corresponding error category.

Any occurring error and its elements is saved to an array. The maximum number of errors to be accepted in the array is limited by the dimensioning (DIM) of the parameter <error no.>.

The command GETERR provides the following return values:

- -1; Command could not be executed
- ≥0: Number of the current errors in <channel>

Syntax:

GETERR (<Channel>,[<Category>],<ErrorNo>[,<Number>])	
<Channel>	Channel no of the queried channel -1: All channels >0: Channel number
<Category>	0: All warnings and errors (default) 1: low-priority system error 2: Control or drive error 3: Interpolator error 4: Hardware error 5: ICL error 6: Part program error 7: Runtime warnings 8: MZA messages: Errors 9: MZA messages: Warnings 10: MZA messages: Notes
<Error number>	Result variable: Two-dimensional integer array with at least three elements in the second dimension (DIM <ErrorNo>% (x,3)), default value: 0. The command provides the current error numbers from the <channel> in descending temporal order. <i>Description of 3 elements of the second dimension:</i> • <Error no.>(x,1): Error number • <Error no.>(x,2): Error channel (-1 = channel-comprehensive) • <Error no.>(x,3): Error category (if declared via DIM command). 0 = Unknown category Values other than <category> Example: DIM ERRNO% (100, 3).

Tab. 7-256: Syntax 1 GETERR



Only the variable name without dimension/index may be entered.

<Number>	Integer variable (default value: 1) Specifies the number of errors to be read. 1: Default value > 0: Parameter values are not checked for validity, i.e. if the number of the data to be read exceeds the array dimension, no part program error is generated. Example: <Number> =120, but DIM ERRNO% (100, 3). In this case, 20 errors are not read.

Tab. 7-257: Syntax 2 GETERR

Examples: *Queries the last part program error in channel 2*

```
10 DIM ERR_FELD%(5,3): REM integer array with 5 elements
20 REM queries last parts program error in channel 2
30 CHAN%=2: CATEGORY%=6
40 ERG%=GETERR(CHAN%,CATEGORY%,ERR_FELD%,1)
```

Result analysis for 5 array elements

```
10 DIM ERR_FELD%(5,3): REM integer array with 5 elements
20 CHAN%=2: CATEGORY%=0
25 REM queries warnings and error in channel 2
30 ERG%=GETERR(CHAN%,CATEGORY%,ERR_FELD%,5)
40 FOR I%=1 TO RES%
    50 IF ERR_FELD%(I%,3)=6 THEN
        60 PRN#(0,"parts program error: ",ERR_FELD%(I%,1))
        70 ENDIF
80 NEXT I%
```

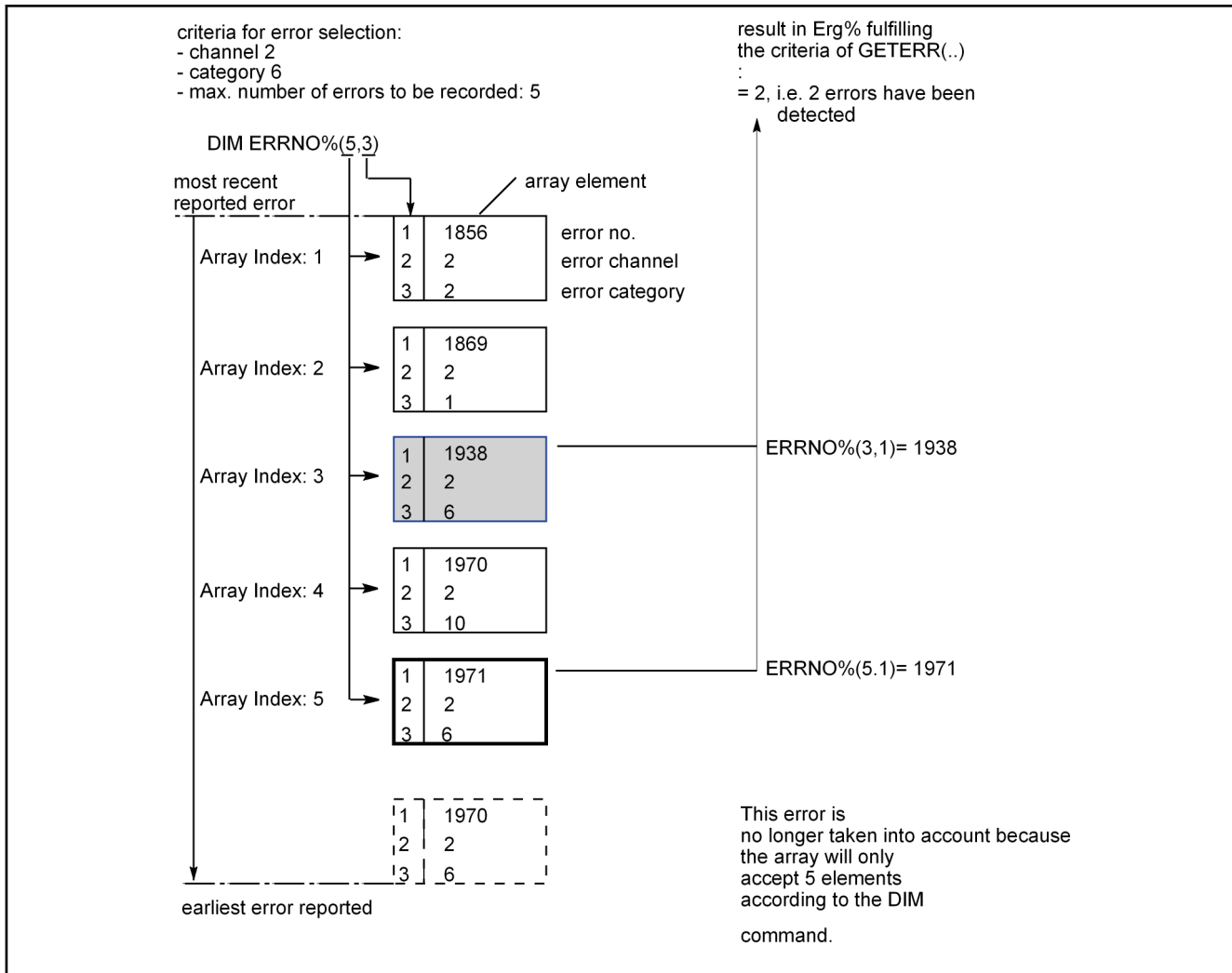


Fig. 7-6: Example

As a result, the part program errors 1938 and 1971 are displayed in the MSG window. Variable Erg% contains the value 2.



GETERR can be used for example to record the chronological occurrence of one or more errors, thus permitting the actual cause of the errors to be traced.

7.5 Files and directories

7.5.1 General information

Description

Files are containers for data. During the CPL program, data can be read from files or written to files. As a result, measured values can be saved first and then displayed or printed. In the CNC, files are managed in the file system. They are organized in a hierarchic structure. Individual files are accessed using directory paths. Files can be managed using various operation modes. To provide read-only or write access to data, the corresponding file always has to be opened first (see OPENW, OPENR commands). As soon as access to the data is no longer required, the file is closed (see CLOSE command).

File names

Description: Note to the following conventions for the file name:

- Maximum length = 30 characters.

There is no differentiation between the name and any possible file name extension. All letters, numbers as well as the special characters "." and "_" are valid.

File names of part programs may not exceed 28 characters, since, during linking, the NC generates a file whose name consisting of the original file name plus two additional characters.



Use the special character "\$" only for internally generated files. External file names (via user interface or DNC) must not contain a "\$".

- A distinction is made between upper-case and lower-case letters.

Examples: File names

P123456789.PRG

P12_Data_Dial

P12_DATA_DIAL

- File names "." and ".." are not valid, since they are already used internally.
- File names have to be unique within a directory. However, files with the same name can exist in different directories.

Random file structure

Description: A random file has a **component** (records) with a **fixed, defined length**. This permits random, direct access to any component in the file. Dividing the random file into records with a fixed length permits direct access to a certain record. As for sequential files, the data is saved as ASCII characters. This permits the familiar access using the editor as well as reading random files in and out.

The advantage of the random file is that required data can be accessed faster. In addition, the data of a record can be processed and/or modified without changing the structure of the remaining file. Records that are not completely filled with data are filled with blanks up to the defined length.

If an attempt is made to insert a STRING variable into a random file and this variable is longer than that of the record, the record is filled up to the defined length with the first characters of the STRING variable and the remaining characters are ignored.

When the file is read, its end is identified by "EOF". The REWRITE and CLOSE instructions are used as for sequential files.

A random file can also be sequentially accessed.

Sequential file structure

Description: A sequential file contains a sequence of **components** (records) that can have a **variable length**. If a certain record is searched in a sequential file, the search has start at the beginning of file. Direct access is not possible. When a record length is changed in a sequential file, all subsequent records have to be moved.

In contrast to random files, sequential files contain records of different lengths (max. length = 1024 characters). The end of a record is labeled by an <LF> which is not counted as part of the length. After the last record of a file, an <ETX><LF> is inserted representing an EOF pointer. An EOF pointer indicates the end of usable data (<ETX>) in a file.

7.5.2 Directories

DIRINF

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **DIRectory INformation** - Management of a directory

DIRINF allows to query management data of the file system of the MTX CNC system in the CPL program. The command provides a value of type "integer" (for negative results, the real result is greater than the maximum integer value, i.e. 2.147.483.647).

The syntax changes depending on the value of the parameter <Index1>.

Syntax:

for <Index1> = 1, 2 or 3: DIRINF (<Index1>[, <Index2>]) for <Index1> = 4 or 6: DIRINF (<Index1>, [<Index2>], <ResVar>[, <FileNumber>]) for <Index1> = 5, 7 or 8: DIRINF (<Index1>, [<Index2>], <ResultVar>[])	
<Index 1>	Constant of type integer Command mode: 1: Available memory 2: Assigned memory 3: Number of files in the directory 4: Name of file in directory with index <File number> 5: Directory name of the active main program 6: Name of mount directory with index <File number> 7: Directory name of the active main program or subroutine 8: Name of calling program
<Index 2>	String type Additional mode: Directory name (default: current directory)
<Event var>	Type dimensioned string variable, only for the command modes 4 - 8. Provides the name of a file in the specified directory or the directory name after the command has been executed.
<File number>	"Integer" type, only for the command modes 4 and 6 In <FileNumber>, the sequential number of the file in the directory is specified. Its name is provided in <ResVar>. If the sequential number is invalid, "NUL" is returned in <ResVar>.

Tab. 7-258: DIRINF syntax

Return value:	<Command mode 1>	Free memory in bytes
	<Command mode 2>	Assigned memory in bytes
	<Command mode 3>	Number of files in the directory
	<Command mode 4>	0 (<Result var> contains file names) - or - 1 (<Event var> is ZERO)
	<Command mode 5>	0 (<ResultVar> contains directory names) - or - 1 (<Event var> is ZERO)
	<Command mode 6>	0 (<ResultVar> contains directory names) - or - 1 (<Event var> is ZERO)
	<Command mode 7>	0 (<ResultVar> contains directory names) - or - 1 (<Event var> is ZERO)
	<Command mode 8>	0 (<Result var> contains file names) - or - 1 (<Event var> is ZERO)

Example: ; All file names of the current directory in
; show MSG window

```

10 DIM FILENAME$(30)
20 DIM DIRNAME$(80)
30 LJUST
40 ERG%= DIRINF(5,,DIRNAME$)
45 IF ERG%=0 THEN
50 ANZ_FILES%= DIRINF(3,DIRNAME$)
60 FOR LNR%= 1 TO ANZ_FILES%
70 ERG%= DIRINF(4,DIRNAME$,FILENAME$,LNR%)
80 PRN#(0,LNR%,"":",FILENAME$)
90 NEXT LNR%
95 ENDIF
M30

```

Example: ; Read all mount directories of the control

```

10 LJUST
20 DIM T$(100)
30 FOR I%= 1 TO 100
40 ERG%= DIRINF(6,,T$,I%)
50 IF ERG%=0 THEN
60 PRN#(0,I%,"":",T$)
70 ELSE
80 PRN#(0,"No other mount directory available!")
90 I%=100
100 ENDIF
110 NEXT I%
M30

```

DIRCR

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **DIRectory CReate** - Creating directory

DIRCR allows to create a new directory in the CPL program. The command provides a value of type "integer".

Syntax:

DIRCR (<Directory>)							
<Directory>		Directory name with complete path as string expression. If the path is not specified, the current directory is positioned in front of the string expression.					

Tab. 7-259: DIRCR syntax

Return value: 0: Directory was created

1: Creating directory not possible

Example: 10 I% = DIRCR("/usr/user/test")

M30

DIRDEL

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **DIRectory CReate** - Deleting directory

DIRDEL allows to delete an empty directory in the CPL program. The command provides a value of type "integer".

Syntax:

DIRDEL (<Directory>)							
<Directory>		Directory name with complete path as string expression.					

Tab. 7-260: DIRDEL syntax

Return value: 0: Directory was deleted

1: Deleting directory not possible. Formatting as for DIRCR.

Example: 10 I% = DIRDEL("/usr/user/test")

M30

7.5.3 File operations

FILENO

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **FILE Number** - Logic file number

Logic file numbers are required to access files. These can assume values between 1 and 9 (i.e. up to nine files can be opened at the same time via CPL). Command FILENO allows to query the next valid logic file number. If the command returns "-1", no free logic file number is available. The return value is of type "integer".

Example:

```

10 LOG_NO% = FILENO
20 IF (LOG_NO% <> (-1)) THEN
30 OPENW(LOG_NO%, "/user/usr/Test")
40 PRN# (LOG_NO%, "Date: ", DATE)
50 CLOSE (LOG_NR%)

```

```

60 ELSE
70 PRN#(0,"No free log. file number available!")
80 ENDIF
M30

```

OPENW, OPENR

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **OPEN Write** - Opening file for write access

OPEN Read - Opening file for read-only access

The command used to open a file depends on the desired type of access:

- Write access: **OPENW**
- Read-only access: **OPENR**

If the file to be opened does not yet exist, it is created while opening.

Files which are already open for writing can also be opened for read-only access via the "OPENR" command. An open file cannot be opened again for writing.

To open a random file, an additional parameter is used which returns the length of the records in the file in bytes (1 byte = length of a character). In all other respects, the command structure corresponds to the one of the sequential file.

After an OPENR command, the file pointer is positioned at the first file record which can then be read. After an OPENW command, the file pointer is positioned on the EOF pointer, i.e. after the last record of the file.

Syntax:

OPENW (<n>,<FileName>[,<EmptyParam>][,<RecordLength>])	
OPENR (<n>,<FileName>[,<RecordLength>])	
<n>	Logic number under which be used to address the file. Values from 1 to 9 can be selected. The logic number has to be programmed as INTEGER expression. A logic number may not be assigned for reading and writing a file simultaneously. A maximum of nine different files can be opened at the same time. If the range of values is not adhered to, the following error message is displayed: "INVALID LOGICAL NUMBER".
<FileName>	String expression: File name; optionally with complete path. Without path specification, it is searched for the file in the current directory.
<empty parameter>	(not assigned)
<Record length>	Number of bytes of a record Value range: 1..1024 If the value range is not adhered to, the error message "INVALID COMPONENT LENGTH" is displayed.

Tab. 7-261: Syntax of read-only and write access

Examples:

```

50 OPENW(1,"P500","This is my best program")
40 A$="P500" : B1$="This is my best program"
50 OPENW(9,A$,B1$)

```

```
50 OPENW(7,"PDaten_Mes.DAT","Save measurement data")
```

When the file is opened for writing, a check is made to ensure that the random structure has been maintained. If the structure has been damaged by the editor, the following error message is displayed: "INVALID COMPONENT LENGTH".

```
10 OPENW(2,"P200",,10)
20 FOR I% = 1 TO 3
30 PRN#(2,"TEST")
40 NEXT I%
50 CLOSE(2)
Result: "P2"
TESTE <LF>
TESTE <LF>
TESTE <LF>
<ETX><LF>
```

When the file is opened for reading, it is checked whether the random structure has been maintained. All components have to have the identical length as specified in the OPENR command.

```
1 OPENW(2,"P200",,"TEST",10)
2 PRN#(2,"ABC")
3 CLOSE(2)
4 OPENR(1,"P2",5)
5 CLOSE(1)
```

It is checked whether the record length of the file "P2" is 5. However, the record length of this file is 10.

P1:

```
N10 G1F10000X1000Y1000Z1000
1 A$="01234567890123456789"
2 B$="TEST"
N20 X0
M30
```

P2:

```
1 OPENW(1,"PMess_PRG",,"RANDOMDATEI",10)
2 OPENR(2,"P1")
3 DIM A$(30)
4 FOR I% = 1 TO 5
5 INP#(2,A$)
6 PRN#(1,A$)
7 NEXT
8 CLOSE(2)
9 CLOSE(1)
```


RESULT: PMess_PRG:

```
N10 G1F100<LF>
1 A$="0123<LF>
2 B$="TEST<LF>
N20 X0 <LF>
M30 <LF>
<ETX><LF><LF>
```

If the structure has been damaged by the editor, the following error message is displayed: "INVALID COMPONENT LENGTH".

REWRITE

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **RE WRITE** - Overwriting file content

If data is already present in the opened file, the new data is normally attached to the existing data while writing. A file can, however, be overwritten with "REWRITE" without deleting the content that is no longer required. While overwriting, the range reserved in the OPENW command remains available in the part program memory.

Syntax:

REWRITE (<n>)	
<n>	Logical file number (value range 1 ... 9)

Tab. 7-262: REWRITE syntax

Before overwriting the file, open it.

CLOSE

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: "CLOSE" - Closes a file.

Up to nine files can be opened at the same time. If nine files are open and accessing one more file is required, close one file first.

Thus, open files are generally to be closed immediately upon completion of read-only or write operations.

Syntax:

CLOSE (<n>)	
<n>	1 to 9: Logic number of the file to be closed.

Tab. 7-263: Close syntax

Example:

```
:
90 DIM A$(35)
100 XPOS = MCS(1)
110 YPOS = MCS(2)
120 OPENW(1,"P5",500,"ACHSPOS")
130 REWRITE(1)
140 PRN#(1,"X-axis",XPOS,YPOS,"Y-axis",YPOS)
150 CLOSE(1)
```

```

160 OPENR(1,"P5")
170 INP#(1,A$)
180 CLOSE(1)
:
```

In the example, the current positions of the X and y-axis are transferred to variables (lines 90 to 110). File 1 is then opened and stored as part program P5 (line 120).

The file is subsequently written or overwritten and then closed (lines 140 to 150). The file is then opened for reading and the content is assigned to the A\$ variable. It is closed again after read-only access (lines 160 to 180).

ERASE

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **ERASE**- Deleting file

Erases files in the current directory.

Syntax:

ERASE (<FileName>)	
<FileName>	File name with complete path as string expression. Without path specification, it is searched for the file in the current directory. The CPL command provides the return value as an integer value.

Tab. 7-264: Erase syntax

The "ERASE" command can return the following values as an integer value to an assigned INTEGER variable or to loops or queries (WHILE, IF, etc.):

- 0:** File erased
 - 1:** File not erased, since it does not exist
 - 2:** File not erased, since it is erase-protected
 - 3:** File not erased, since it is active
 - 4:** <File name> was not deleted, since <File name> is not a file, but a directory
 - 5:** File not erased, since it is open
- If a file cannot be erased, a warning is issued and the program execution continues.

Examples:

```

10 IF ERASE("P1") <> 0 THEN ...
10 I% = ERASE("P1")
10 WHILE ERASE("P1") <> 0 DO ...
```

```

10 OPENW(1,"P2",200)
11 OPENW(2,"P3",200)
20 PRN#(1,"TEST1 FUER ERASE")
21 PRN#(2,"TEST2 FUER ERASE")
31 CLOSE(1) 32 CLOSE(2)
40 ERASE("P2")
43 A$="P3"
44 ERASE(A$)
```

FILECOPY

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **FILE COPY** - Copy file

FILECOPY is used to copy a file to the CPL program.

Syntax:

FILECOPY (<FileSource>, <FileTarget>)													
<SourceFile>	File name with the complete path of the source file as string expression. Without path specification, it is searched for the file in the current directory.												
<TargetFile>	File name with the complete path of the target file as string expression. Without path specification, the file is created in the current directory.												
ERRNO	CPL variable that can be programmed at any position. When ERRNO is programmed, no runtime error is output. The return values are: <table> <tr> <td>0:</td><td>Access OK</td></tr> <tr> <td>-8:</td><td>Source file name including path is too long</td></tr> <tr> <td>-9:</td><td>Access to source file not possible</td></tr> <tr> <td>-10:</td><td>Target file name including path too long</td></tr> <tr> <td>-11:</td><td>File name (source or target) invalid</td></tr> <tr> <td>-12:</td><td>Copying not possible</td></tr> </table>	0:	Access OK	-8:	Source file name including path is too long	-9:	Access to source file not possible	-10:	Target file name including path too long	-11:	File name (source or target) invalid	-12:	Copying not possible
0:	Access OK												
-8:	Source file name including path is too long												
-9:	Access to source file not possible												
-10:	Target file name including path too long												
-11:	File name (source or target) invalid												
-12:	Copying not possible												

Tab. 7-265: FILECOPY syntax

Example: *Program:*

```
10 FILECOPY("/usr/user/Test.txt", "/usr/user/Test.bak", ERRNO)
20 IF ERRNO = 0 THEN
30 PRN#(0,"copying ok")
40 ENDIF
```

7.5.4 Editing the file

PRN#

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description **PRN#** - Writing to a file

The variable type can be freely selected. Indexed variables and character arrays can also be used. Double-precision REAL expressions can also be programmed as any CPL expressions.

If the result of an expression is to be output providing a format, at least one of the expressions has to be of type "STRING". The format can be specified using "#" and "." in this format string. The results are entered at the position of the format instruction specified with "#". The first format instruction in a STRING expression refers to the first subsequent expression which can be output with a format specification. Boolean expressions cannot be formatted. The number of all programmed format entries has to be lower than or equal to the number of expressions to be output. If this condition is not fulfilled, the unnecessary "#" are displayed. Without format specification, an expression is output in standard.

	Default formats	Number of places
INTEGER number	9 places	9 max.
Single-precision REAL number	Four leading and three trailing decimal places	7 max.
Double-precision REAL number	Nine leading and six trailing decimal places	15 max.

If the output of an expression exceeds 1024 characters, the following error message is displayed: "BLOCK EXCEEDS 1024 BYTES".

If the result cannot be output in the specified format, the warning "PRN FORMAT INCORRECT" is returned, and "*" characters are output instead of the incorrect format.

If "#" are to be created in the file, no formatable expression may follow after the string within the PRN# instruction. The "#" character can also be output with CHR\$(35).

A line feed can be initiated during output with CHR\$(13), i.e. the further output of the PRN# command is continued in the next line (i.e. in the next record).

Other control characters can be transferred with the CHR\$() command, e.g. when outputting via a serial interface.

Syntax:

PRN# (<n> , [<Expression>][, <Expression>][, <Expression>][, ...];)	
<n>	1 to 9: Logic file number to be used for writing. 0: The output is diverted to the monitor (as for the message programming via MSG command).
<Expression>	Any alphanumeric characters (text in inverted commas), format strings or variables whose content is to be saved/displayed.
;	Suppresses automatic attaching of <CR><LF>. If a record is overwritten via the PRN# command, the following applies: PRN# command with semicolon: If the length of the data to be newly written is shorter than the length of the old data, the new data is inserted and the rest of the old data is retained. PRN# command without semicolon: If the length of the new data to be written is shorter than the length of the old data, the new data is inserted and the rest of the old data is overwritten with blanks.

Tab. 7-266: PRN# syntax

Examples: PRN# command with semicolon

```

1 OPENW (2, "PProg123.PRG", 200, 35)
2 PRN# (2, "TEST1 FOR PRN COMMAND WITH SEMICOLON")
3 PRN# (2, "TEST2 FOR PRN COMMAND WITH SEMICOLON")
4 PRN# (2, "TEST3 FOR PRN COMMAND WITH SEMICOLON")

```

```

6 SEEK(2,1)
7 PRN#(2,"OVERWRITE");
8 CLOSE(2)

```

RESULT in PProg123.PRG:

```

OVERWRITE COMMAND WITH SEMICOLON      <LF>
TEST2 FOR PRN COMMAND WITH SEMICOLON   <LF>
TEST3 FOR PRN COMMAND WITH SEMICOLON   <LF>
<ETX><LF>

```

PRN# command without semicolon

```

1 OPENW(2,"P2",1000,36) 1 REWRITE(2)
2 PRN#(2,"TEST1 FOR PRN COMMAND WITHOUT SEMICOLON")
3 PRN#(2,"TEST2 FOR PRN COMMAND WITHOUT SEMICOLON")
4 PRN#(2,"TEST3 FOR PRN COMMAND WITHOUT SEMICOLON")
6 SEEK(2,1)
7 PRN#(2,"OVERWRITE")
8 CLOSE(2)

```

RESULT in P2:

```

OVERWRITE      <LF>
TEST2 FOR PRN COMMAND WITHOUT SEMICOLON <LF>
TEST3 FOR PRN COMMAND WITHOUT SEMICOLON <LF>
<ETX><LF>

```

<ETX><LF> is inserted after the last block of the file.

The following error message is displayed if the length of the block exceeds 1024 characters: "BLOCK EXCEEDS 1024 BYTES".

If a sequential file is written and the end of file is reached, the file is copied automatically and the reserved range increased by the assigned length if sufficient memory is available in the part program memory.

Since this requires a lot of memory very quickly, it is advisable to reserve a sufficiently large file length when creating the file with "OPENW".

```

1 OPENW(1,"P2",300,"TEST PRN COMMAND")
2 A$="TEST"
3 B$="FOR"
4 C$="PRN COMMAND"
5 PRN#(1,A$)
6 PRN#(1,B$)
7 PRN#(1,C$)
8 PRN#(1,A$;)
9 PRN#(1,B$;)
10 PRN#(1,C$;)
11 CLOSE(1)

```

Result:

P2:

```
TEST<LF>
FOR<LF>
PRN COMMAND<LF>
TESTFORPRN COMMAND<LF><ETX><LF>

10 DIM E$(50)
20 OPENW(1,"P2",300,"TEST2")
30 A% = 5000
40 R = 1.231
50 B! = 4/3
60 D$ = "ABCDE"
70 E$ = "CDEFGHI"
80 PRN#(1,"10";)
90 PRN#(1,"#####", "###.###", "#.#####", A%, R, B!, D$, E$)
95 CLOSE(1)

P2:
10  5000  1.2311.33333ABCDECDEFGHI<LF>
      A%      R      B!      D$      E$
```

LJUST, NJUST

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Left JUSTify** - Left-justified formatting
No Justify - No formatting

With LJUST , it is switched to left-adjusted data output. It is effective up to the end of the program run for all data outputs.

With NJUST, it can be switched back to the formatted output.

Up to seven places (four predecimal and three postdecimal places) are available for the "REAL" data type and up to nine places are available for the "INTEGER" data type when data is output to files. Leading and trailing zeros are suppressed. This also applies to the left-justified output.

Since blanks between the NC address and the value are suppressed, LJUST allows to use CPL to directly create NC programs which can be executed in "PROCESSING" mode.

INP#

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **INPut#** - Reading from file

With the INP# instruction, the ASCII data in an open file in record form can be read and assigned to one or more variables. This command only has an effect on files opened with "OPENR(..)".

Syntax:	INP# (<n>, <Variable>[, <Variable>][...][:])	
	<n>	1 to 9: Logic number of a file to be used for reading.

<Variable>	Variable under which the read data is stored.
;	If a semicolon is programmed, the file pointer remains in the record until the end of the record is reached. The next record is subsequently opened. Reading does not, however, automatically take place. If no semicolon is programmed, it is switched automatically to the next record.

Tab. 7-267: *INP# syntax*

The variable type can be freely selected. Indexed variables and character arrays can also be used. If a value other than TRUE or FALSE is assigned to a logic variable, this variable is valued at "NUL".

For the variable type INTEGER or REAL (simple or double precision), the characters "0" .. "9", preceding signs "-", "+", preceding zeros or spaces are converted into INTEGER or REAL values. If another character is assigned to an INTEGER or REAL variable, "NUL" is assigned to the variable. If "NUL" is assigned to a variable, the position within the file does not change.

If the value assigned to an INTEGER or a REAL variable is too high, a corresponding error message is displayed:

"INVALID INTEGER VALUE"

"INVALID FLOAT VALUE"

Examples: **INP# instruction**

P2:

ABC 123456789 ABC

P3:

```
1 OPENR(2,"P2") 2 DIM C$(3) 3 DIM D$(3) 4 INP#(2,I
%,J,L?,C$,K%,D$) 5 CLOSE(2)
```

RESULT:

I% = NUL

J = NUL

L? = NUL

C\$ = "ABC"

K% = 123456789

D\$ = "ABC"

Read record from file

```
1 OPENW(1,"P2",200,"Test",22)
2 PRN#(1,"-12TEST1.23V12ABCD2.4A")
3 PRN#(1,"-12TEST1.23V12ABCD2.4A")
4 PRN#(1,"-12TEST1.23V12ABCD2.4A")
5 CLOSE(1)
6 DIM A$(3)
7 DIM C$(5)
8 DIM D$(4)
9 DIM E$(4)
10 DIM G$(25)
```

CPL commands

```

11 DIM H$(7)
12 DIM I$(7)
13 DIM J$(25)
14 DIM R(1,2)
15 OPENR(2,"P2",22)
16 INP#(2,B%,D$,R(1,1),MID$(E$,1,1),R(1,2),A$,C$)
17 INP#(2,G$)
18 INP#(2,H$;)
19 INP#(2,I$;)
20 INP#(2,J$)
21 CLOSE(2)

```

RESULT:

```

B% = -12
D$ = "TEST"                                , since the maximum length of character
                                              array = 4

R(1,1) = 1.230
E$ = "V"
R(1,2) = 12.000
A$ = "ABC"                                , since the maximum length of character
                                              array = 3

C$ = "D2.4A"
G$ = "-12TEST1.23V12ABCD2.4A"
H$ = "-12TEST"                            , since the maximum length of character
                                              array = 7

I$ = "1.23V12"                            , since the maximum length of character
                                              array = 7

J$ = "ABCD2.4A"

```

SEEK

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **SEEK** - Position the file pointer

Positions the file pointer at a certain position of an open file. The file can be a sequential or a random file.

With sequential files, the file has to be opened with the command "OPENR(..)". For random files, the command "OPENW(..)" is also permitted.

Syntax:

SEEK (<n>, <k>[, <o>])	
<n>	Logic file number in which the file pointer is to be positioned. Value range 1 to 9 If the value range is not adhered to, the error message "INVALID FILE NUMBER" is displayed.

<k>	Record number of a random file or byte number of a sequential file. The file pointer is positioned on <k>. Value range: 0 to latest existing record or 0 to latest existing byte. The record with the EOF pointer is assumed as latest existing record. At 0, positioning is on the EOF pointer. The "INVALID COMPONENT" error message is displayed if the range of values is not adhered to or if the specified record does not exist.
<o>	Record offset Indicates at which byte in a record it is to be positioned. Value range: 1 ... Record length + 1 If the record offset is not programmed for random files, the file pointer is positioned at the first byte of the record <k>. If the value range is not adhered to, the error message "INVALID PARAMETER" is displayed. This parameter is only valid for random files. The "INVALID PARAMETER" error message is displayed if this parameter is programmed although it is a sequential file (which has been opened for reading).

Tab. 7-268: *SEEK syntax***Examples: SEEK and sequential file**

```

1      DIM A$(1):LJUST:OPENW(1,"P271",130,"TEST"):FOR I%=1
      TO 10:
      PRN#(1,"!/-/!/-/!/-/!/-/!/-/!/-/!/-/!/"):NEXT:
      CLOSE(1):OPENR(2,"P271"):FOR I%=1 TO FILE-
      SIZE(2,2)-28: IF NOT (EOF(2)) THEN SEEK(2,I
      %):INP#(2,A$) ENDIF: IF (EOF(2)) THEN PRN#(0,"###",I
      %, ". BYTE: <EOF>"): ELSE PRN#(0,"###",I%, ". BYTE:
      < ",A$, ">") ENDIF: NEXT I%:CLOSE(2)

```

M30

SEEK and random file

```

1      OPENW(1,27272,200,"TEST",1024):LJUST
2      FOR I%= 1 TO 10
3      PRN#(1,I%, ". Record")
4      NEXT
5      SEEK(1,3,4) : REM positioned to the fourth byte in the third record
6      PRN#(1,'Overwriting of the third record from byte 4 from this text")
7      SEEK(1,11):PRN#(1,"11. record")
8      SEEK(1,11,5):PRN#(1,"@@")
9      SEEK(1,0):PRN#(1,"<EOF>")
10     SEEK(1,0,1):PRN#(1,"neues <EOF>")
11     CLOSE(1)

```

FILEPOS

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **FILE POSition** - Determining file pointer position

The "FILEPOS()" command returns the record number of the current record of a random file. This record can subsequently be accessed. It is also possible to determine the record offset within the current record of a random file or the offset from the current byte that can be accessed for a sequential file. The file can therefore be a sequential or a random file.

Offset refers to the number of bytes from the beginning of file up to the current byte in a file. The record offset specifies the byte to be positioned on in a record. The record offset starts with value 1 (= first byte in a record) and can assume max. the value of the record length + 1 (last byte in this record is <LF>). The value 1 is returned if located on the EOF pointer.

Syntax:

FILEPOS (<n>[, <Mode>])	
<n>	1 to 9: Logic file number in which the position of the file pointer is to be read. If the value range is not adhered to, the error message "INVALID FILE NUMBER" is displayed.
<mode>	For random files: Value range 1 to 3 <mode> = 1: Provides the offset to the current byte which can be read or written. <mode> = 2: Provides the record number of the current record which can be read or on which can be written. The result is as follows if located on the EOF pointer: Number of records + 1. <mode> = 3: This command provides the record offset in the current record which can be read or on which can be written. The record offset starts at value 1 (⇒ first byte in this record) and can have the maximum value of the record length + 1 (⇒ last byte in this record is <LF>). 1 is returned and reading from the file is invalid if positioned on the EOF pointer. <mode> not programmed: Provides the record number of the current record which can be read or on which can be written. The result is as follows if located on the EOF pointer: Number of records + 1. For sequential files: Value range 1 <mode> = 1 or not programmed: Provides the offset to the current byte which can be read or written.

Tab. 7-269: FILEPOS syntax



If the range of values of <mode> is not adhered to, the following error message is displayed: "INVALID PARAMETER".

Examples: FILEPOS and sequential file

```

1 OPENW(1,"P2",200,"TEST")
2 FOR I%= 1 TO 10
3 PRN#(1,"TEST FOR FILEPOS")
4 NEXT
5 CLOSE(1)
6 OPENR(1,"P2")
7 SEEK(1,3)
8 POS% = FILEPOS(1)
9 POS1% = FILEPOS(1,1)
11 SEEK(1,0) : REM POSITIONED ON END OF FILE
12 POS2% = FILEPOS(1)
13 POS3% = FILEPOS(1,1)
14 CLOSE(1)

```

Result:

POS% = 3	→ Byte number
POS1% = 3	→ Byte number
POS2% = 171	→ Byte number
POS3% = 171	→ Byte number

FILEPOS and random file*Program:*

```

1 OPENW(1,"P2",200,"TEST",1024)
2 FOR I% = 1 TO 10
3 PRN#(1,"TEST FOR FILEPOS")
4 NEXT
5 SEEK(1,3,2)
6 POS% = FILEPOS(1)
7 POS1% = FILEPOS(1,1)
8 POS2% = FILEPOS(1,2)
9 POS3% = FILEPOS(1,3)
10 PRN#(1,"Overwrite 3rd record from byte 2 with this text")
11 SEEK(1,0) : REM Positioned at end of file
12 POS% = FILEPOS(1)
13 POS1% = FILEPOS(1,1)
14 POS2% = FILEPOS(1,2)
15 POS3% = FILEPOS(1,3)
16 CLOSE(1)

```

Result:

POS% = 3	→ Record number of the record of the current position
POS1% = 258	→ Byte number
POS2% = 3	→ Record number of the record of the current position
POS3% = 2	→ Position in the third record
POS% = 11	→ Record number of the record of the current position
POS1% = 1281	POS1% = 1281
POS2% = 11	→ Record number of the record of the current position
POS3% = 1	→ Position in the third record

EOF

BOOL	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **End Of File** - File end ID

The "EOF" command can be used to query whether the end of file is reached.

The EOF command returns the logic value "TRUE" if the end of file is reached while reading. Otherwise, "FALSE" is returned.

Example:

```

: 9 DIM A$(10)
10 OPENR(1,"P",444) : I%=0
11 WHILE NOT (EOF(1)) DO
12 INP#(1,A$)
13 I%=I%+1
14 END
15 CLOSE(1)
M30

```

7.5.5 File properties

FILEACCESS

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **FILE ACCESS** - File of access rights

With FILEACCESS, it is possible to determine in the CPL program whether a file exists and which access rights are assigned.

Syntax:

FILEACCESS (<FileName>)	
<FileName>	File name with complete path as string expression. Without path specification, it is searched for the file in the current directory. The CPL command provides the return value as an integer value.
	-1 :
	0:
	Other- wise:
	Bit0:
	Bit1:
	Bit2:
	Bit3:
	Bit4:
File does not exist	
File without access rights	
Binary-coded access rights:	
Execution possible (X)	
Writing permitted (W)	
Reading permitted (R)	
File is a directory (D)	
File is an active program (A)	

Tab. 7-270: FILEACCESS syntax

An **active program** is a file

- executed as program in a channel
- executed as subroutine of a program and
- opened via a CPL command.

If the **access rights** of a zero offset or tool correction table currently used in a running part program are queried by the CPL command "FILEACCESS", Bit5 is **not** set.

Example: 10 I% = FILEACCESS ("/usrfep/test.cnc")

FILEDATE

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **FILE DATE** - Date

In the CPL program, the date of a file can be determined with FILEDATE.

An access error does not generate a part program error. The command provides an empty string instead.

Syntax:

FILEDATE (<FileName>[, <DataSelection>])	
<FileName>	File name with complete path as string expression. Without path specification, it is searched for the file in the current directory. The CPL command provides a string expression as return value.
<DataSelection>	Integer variable for the command mode (default = 1): 1 = date of the file, format: tt:mm:jj 2 = Time of the file, format: hh:mm:ss

Tab. 7-271: FILEDATE syntax

Example:

```

10   DIM DATUM$(10)
20   DATUM$ = FILEDATE("/usr/user/Test.txt",1)
30   IF LEN(DATUM$)>0 THEN
40     PRN#(0,"File date: ",DATUM$)
50   ENDIF

```

FILESIZE

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **FILE SIZE** - File size

Provides the file size or the limit up to which it has already been written to a file. The file can be a sequential or a random file. This command only has an effect on files opened with "OPENR(..)".

Syntax:

FILESIZE (<n>[, <k>])	
<n>	1 to 9 Logic number of the file of which the size is to be determined. If the range of values is not adhered to, the following error message is displayed: "INVALID FILE NUMBER".
<k>	For random files: Value range 1 to 4 For sequential files: Value range 1 to 2 <k> = 1: Total memory size (in bytes) used by a file. <k> = 2: Memory size (in bytes) used from the start of the data range up to the EOF pointer (excluding the size of the EOF pointer). <k> = 3: Maximum number of records in a file. This result depends on the record length used to open the file. <k> = 4: Number of records from the beginning of the file up to the EOF pointer. This result depends on the record length used to open the file. <k> not programmed: as <k> = 1. If the range of values for <k> is not adhered to, the following error message is displayed: "INVALID PARAMETER".

Tab. 7-272: FILESIZE syntax

Examples: FILESIZE and sequential file

```

1      OPENW(1,2,1000)
2      FOR I%= 1 TO 20
3      PRN#(1,"TESTE FILESIZE")
4      NEXT
5      CLOSE(1)
6      OPENR(2,2)
7      A%=FILESIZE(2)
9      B%=FILESIZE(2,1)
10     C%=FILESIZE(2,2)
11     CLOSE(2)

```

The INTEGER variable A% includes the value: 302

The INTEGER variable B% includes the value: 302

The INTEGER variable C% includes the value: 300

FILESIZE and random file

```

1      OPENW(1,"P2",1000,10)
2      FOR I%= 1 TO 20
3      PRN#(1,"TESTE FILESIZE")
4      NEXT
5      CLOSE(1)
6      OPENR(2,2,10)
7      A%=FILESIZE(2)
9      B%=FILESIZE(2,1)
10     C%=FILESIZE(2,2)
10     D=FILESIZE(2,3)
10     E%=FILESIZE(2,4)
11     CLOSE(2)

```

The INTEGER variable A% includes the value: 222

The INTEGER variable B% includes the value: 222

The INTEGER variable C% includes the value: 220

The INTEGER variable D% includes the value: 20

The INTEGER variable E% has the value: 20

7.6 Character string processing

7.6.1 Basics

DIM

---	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Dimension** - Dimensioning character string

To create a character array, index a character variable using a DIM instruction. Thus, character arrays with up to 1,024 characters can be created (value range of index: 1 to 1,024). If the value range is not adhered to, the error message "INVALID ARRAY LIMIT" is displayed.

Examples: 1 DIM VWX\$(14)

In this example, the character array VWX\$, consisting of 14 individual character variables, is created. Strings with up to 14 characters in length can therefore be stored in VWX\$.

1 DIM ABC\$(1)	Character array for a string with a maximum length of 1 character.
2 DIM BCDE\$(10)	Character array for a string with a maximum length of 10 characters.

INSTR

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description **In String** - Searching in character string

Searches for a character string using a specified character sequence.

Syntax:

INSTR (<CharacterString>,<StringExpression>[,<StartingPoint>])	
	Starts searching for a <CharacterString> in a <StringExpression> from the <StartingPoint> and returns the position of the first character of the found <CharacterString> in the <StringExpression> as INTEGER value. 0 is returned if the <CharacterString> is not found. The <CharacterString> can be programmed as a string expression. The range of values for the 3rd parameter is between 1 to 1024. The error message "INVALID PARAMETER" is displayed if the range of values is not adhered to.

*Tab. 7-273: INSTR syntax***Example:**

```

1 DIM A$(8)
2 DIM B$(16)
3 A$="A" : MID$(A$,2)="UVWXYZ"
4 B$="ABCDEF UVWXYZ GH"
5 POS1%=INSTR(MID$(A$,2),B$,4)
6 POS2%=INSTR(MID$(A$,2,4),B$,10)
7 POS3%=INSTR(MID$(A$,2),B$)

```

The INTEGER variable POS1% includes the value 8.

The INTEGER variable POS2% includes the value 0.

The INTEGER variable POS3% includes the value 8.

LEN

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Length** - Length of a character string**Syntax:**

LENLEN (<StringExpression>)	
	Provides the number of characters of <StringExpression>. The result is an INTEGER value. If the <StringExpression> is empty, LEN returns 0. If the <StringExpression> is not defined, LEN returns -1.

*Tab. 7-274: LEN syntax***Example:**

```

1 DIM XYZ$(10)
2 XYZ$="ABC"
3 I%=LEN(XYZ$)
4 XYZ$=""
5 J%=LEN(XYZ$)
6 XYZ$=NUL
7 K%=LEN(XYZ$)

```

The INTEGER variable I% has a value of 3.

The INTEGER variable J% has a value of 0.

The INTEGER variable K% include the value -1.

Comparing STRING expressions

Description: It is also possible to compare STRING expressions, also STRING constants, STRING variables and character arrays with one another.

Example:

```

1 DIM A$(10)
2 DIM B$(15)
3 A$="ANTON"
4 B$= "WILLI"
5 C$="ABCDE"
6 D$="VWXYZ"
7 IF A$ < B$ THEN ...
8 IF MID$(A$,2,3) = MID$(B$,1,3) THEN ...
9 Z?=A$ <> "TEST"
10 IF "A" <= "C" THEN ...
11 IF C$ > D$ THEN ...
12 IF A$ = C$ THEN ...
13 IF "TE" < MID$(D$,2,2) THEN ...

```

The contents of the STRING expressions are checked for alphabetical order according to the ASCII code of the individual characters.

7.6.2 Type conversion

ASC

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **ASCII** - Determines the ASCII code of a character

Syntax:

ASC (<CharacterString>)	
	Provides the ordinal number of the first character (ASCII code) of <CharacterString> as INTEGER value. If <String> is empty or not defined, ASC returns -1. <String> has to be a string expression. ASC is the inversion of CHR\$.

Tab. 7-275: ASC syntax

Example:

```

10 DIM A$(1)
20 A$ = "ABC"
30 B$ = "BCD"
40 I% = ASC(A$)
50 J% = ASC(B$)
60 A$ = ""
70 K% = ASC(A$)

```

The INTEGER variable I% includes the value 65.

The INTEGER variable J% includes the value 66.

The INTEGER variable K% include the value -1.

```
80 A$ = NUL
90 L% = ASC (A$)
```

The INTEGER variable L% includes the value of -1.

CHR\$

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **CHaRacter** - Determine ASCII characters
CHR\$ is the reversing command of ASC.

Syntax:	CHR\$ (<IntegerExpression>)
	Converts <Integer expression> into the respective ASCII character.

Tab. 7-276: CHR\$ syntax

All ASCII character decimal priorities are displayed in the "ASCII character block" table in the appendix of this documentation.

Example:

```
10 DIM A$(1)
20 I% = 65
40 A$ = CHR$(I%)
```

The string variable A\$ includes the value 65.

STR\$

BOOL	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description **STRing** - Converting number into a character string

Syntax:	STR\$ ([<FormatString> ,]<Value>)
	Converts the numeric expression <Value> into a character string that can be assigned to only one character array. If assigned to a string variable, a runtime error is caused. <Value> can be an INTEGER or REAL expression of single or double precision. If <Format string> is programmed, the string can be provided already formatted. "#" indicates digits and "." indicates decimal points. Provided in standard format without <FormatString>.

Tab. 7-277: STR\$ syntax

	Default formats	Number of places
INTEGER number	9 places	9 max.
Single-precision REAL number	Four leading and three trailing decimal places	7 max.
Double-precision REAL number	Nine leading and six trailing decimal places	15 max.

Example:

```

10 DIM A$(50)
20 DIM B$(21)
30 A$=STR$( "Integer=##.###", 37/3) REM content of character array A$:
                                         "Number=12.333"
40 B$=STR$(2.5)                        Content of character array B$:
                                         "2.500"

```

VAL

Description **VALue** - Converting number to character string

Syntax:

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

VAL (<StringExpression>)	
	Returns the numerical value for a <StringExpression>. If the string contains a character other than a leading space, the leading "+" or "-" signs, the numbers 0 to 9 or the decimal point ".", the conversion is performed up to this (other) character. Leading spaces and zeros are ignored for purposes of value formation. If none of the characters above are displayed, "NUL" is returned. If the string contains a decimal point, the result may only be assigned to a REAL or double-precision REAL variable. Assignment to an INTEGER variable in this case would lead to an "INVALID ASSIGNMENT" error message.

Tab. 7-278: VAL syntax

Example:

```

1 I% = VAL("1.23DE")
2 K% = VAL("123DE")
3 J% = VAL("ABC")
4 R = VAL("-1.23DE")
5 Z = VAL("+ 000001234TEST4365")
6 X = VAL("ABC1.23DE")
7 D! = VAL("1234567.234567")

```

Line 1 causes an error message, since an assignment to an INTEGER variable is to be performed.

The value of the INTEGER variable K% is 123.

The digits 1, 2, 3 are converted into an INTEGER number.

The "D" character aborts the conversion, since it cannot belong to an INTEGER number. The following characters are not handled further.

The value of INTEGER variable "J%" is NUL, i.e. the variable is not assigned. The character "A" interrupts the processing of the <StringExpression>.

The value of the REAL variable "R" is -1.23. The "-" character is detected as sign for the REAL number. The digit 1, the "." character and the digits 2 and 3 are converted into a REAL number. The "D" character aborts the conversion, since it cannot belong to a REAL number. The "E" character is not handled further.

The value of the REAL variable "Z" is 1234. The "+" character is identified as sign for the REAL number. The following spaces as well as the leading zeros

are ignored for purposes of value formation. The digits 1, 2, 3 and 4 are converted into a REAL number. The "T" character aborts the conversion, since it cannot belong to a REAL number. The remaining characters are not handled further.

The REAL variable "X" is NUL, i.e. not assigned. The conversion is aborted when the character "A" is detected. The value of the double-precision REAL variable "D!" is 1234567.234567.

7.6.3 Changing the character string

Assigning a STRING expression to a character array

Description: If the STRING expression contains fewer characters than the selected range of the character array, the remaining range is regarded as not assigned. This remaining range is not included in the character array length.

Examples:

```
1 DIM XYZ$(16)
2 XYZ$=" " Content of the character array XYZ$: " "
Length of character array XYZ$: 1
```

If the length of the STRING expression exceeds the maximum length of the character array during assignment, the surplus characters are discarded.

```
1 DIM XYZ$(3)
2 XYZ$="ABCDEF"
Content of character array XYZ$: "ABC"
Length of character array XYZ$: 3 ---> Maximum length
```

```
1 DIM XYZ$(16)
2 A$="DAS "
3 B$="IS A TEST"
4 C$="An Egg"
5 MID$(XYZ$,1,4)=A$
```

CONTENT OF THE CHARACTER ARRAY

Program:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
D	A	S													

length = 4 field is not used

```
6 MID$(XYZ$,5,6)=B$
```

CONTENT OF THE CHARACTER ARRAY

Program:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
D	A	S		I	S	T		E	I						

```
| length = 10                                |field is not used
```

```
7 MID$(XYZ$,5,12)=B$
```

CONTENT OF THE CHARACTER ARRAY

Program:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10|11|12|13|14|15|16|
|---|
| D | A | S |   | I | S | T |   | E | I | N |   | T | E | S | T |
|---|
| length = 16; field is used completely      |
```

```
8 MID$(XYZ$,9,8)=C$
```

CONTENT OF THE CHARACTER ARRAY

Program:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10|11|12|13|14|15|16|
|---|
| D | A | S |   | I | S | T |   | E | I | N |   | T | E | S | T |
|---|
| length = 16; field is used completely      |
```

```
9 XYZ$=MID$(XYZ$,1,4)
```

CONTENT OF THE CHARACTER ARRAY

Program:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10|11|12|13|14|15|16|
|---|
| D | A | S |   |   |   |   |   |   |   |   |   |   |   |   |
|---|
| length = 4      | field is not used      |
```

The character array was cut off.

Linking STRING expressions

Description: Multiple STRING expressions can be linked by the "+" character. The result has to be assigned to a character array. The nesting depth for the linking of STRING expressions is 3. If this depth is exceeded, the following error message is returned: "RUNTIME ERROR 2153 - NESTING TOO DEEP".

Example: **Link also in CPL commands**

```
1 DIM A$(3)
2 DIM B$(3)
3 A$ = "ABC"
4 B$ = "DEF"
5 C$ = "GH"
6 D$ = "JKL"
7 OPENW(1,"P2",130,"TEST Linking",10)
8 PRN#(1,A$+B$)
```

CPL commands

```

9 PRN# (1,A$+C$)
10 PRN# (1,C$+D$)
11 PRN# (1,A$+C$+"TEST")
12 PRN# (1,"UVW"+"XYZ")
13 CLOSE (1)

```

Content of file P2:

ABCDEF	<LF>
ABCGH	<LF>
GHJKL	<LF>
ABCGHTEST	<LF>
UVWXYZ	<LF><ETX><LF>

Link texts using STRING expressions

```

10 DIM A$ (100)
20 DIM B$ (100)
30 DIM C$ (10)
40 DIM D$ (20)
51 DIM E$ (30)
52 DIM F$ (30)
53 DIM G$ (30)
54 DIM H$ (30)
55 DIM I$ (30)
60 A$="DAS "
70 B$="IST EIN TEST"
80 MID$ (C$,1,6)=A$ + B$
90 MID$ (D$,1,10)=MID$ (A$,1,1) + MID$ (B$,1,2)
92 E$=A$ + MID$ (B$,1)
93 X$="ABC"
94 Y$="DE"
95 F$=X$ + Y$ 96 G$=X$ + A$
97 H$=X$ + A$ + "TEST"
98 I$="TES" + "T1"

```

Content of A\$:	"THIS "	Length 4
Content of B\$:	"IS A TEST"	Length 12
Content of C\$:	"THIS IS"	Length 6
Content of D\$:	"DIS"	Length 3
Content of E\$:	"THIS IS A TEST"	Length 16
Content of F\$:	"ABCDE"	Length 5
Content of G\$:	"ABCDAS "	Length 7

Content of H\$: "ABCDAS TEST" Length 11
 Content of I\$: "TEST1" Length 5

The following programming causes errors:

1 DIM A\$(3):A\$ = "ABC":B\$ = "CD":C\$ = "EF"	
2 D\$ = A\$ + B\$	Invalid assignment to an non-dimensioned STRING variable
3 D\$ = B\$ + C\$	
4 D\$ = A\$ + B\$ + "TEST"	
5 D\$ = "TEST" + "TEST1"	

STR\$

```
1 DIM A$(50) : DIM B$(21)
2 A$ = STR$("A$ = ##.###", (37/3)) : B$ = STR$(2.5)
```

Content of character array A "A\$ = 12.333"; \$:	Character array "2.500" B\$:

VAL

```
1 DIM FOLGE$(20) : FOLGE$="X-VALUE -0001.234 MM"
2 XR=VAL(MID$(FOLGE$,7)) : Z%=VAL(MID$(FOLGE$,7,6))
3 Y%=VAL(MID$(FOLGE$,15,5)) : X%=VAL(MID$(FOLGE$,18))
```

Content of the REAL variable XR:	-1.234
Content of INTEGER variable Z%:	-1
Content of INTEGER variable Y%:	34
Content of INTEGER variable X%:	NUL

LEN

```
1 DIM Z$(10)
2 Z$ = "TEST"
3 S$ = "TEST"
4 A% = LEN("TEST")
5 B% = LEN(Z$)
6 C% = LEN(S$)
7 D% = LEN("TEST"+Z$+S$)
```

Content of INTEGER variable A%:	4
Content of INTEGER variable B%:	4
Content of INTEGER variable C%:	4
Content of INTEGER variable D%:	12

MID\$ command with read-only access

CPL commands

```

10 DIM A$(4)
20 DIM B$(10)
30 DIM C$(10)
40 DIM D$(10)
50 DIM E$(10)
55 DIM F$(10)
60 A$ = "ABCD"

```

```

70 B$ = MID$(A$,2,2)          --> B$ = "BC"
80 C$ = MID$(A$,2,5)          --> C$ = "BCD"
95 E$ = MID$(A$,5,1)          --> E$ = NUL
97 F$ = MID$(A$,2)             --> F$ = "BCD"
98 F$ = MID$(F$,1,1)          --> F$ = "B"

```

MID\$ command with write access

```

10 DIM A$(4)
20 DIM B$(10)
30 DIM C$(10)
40 DIM D$(10)
60 A$ = "ABCD"
70 B$ = "1234567890"
80 C$ = "EFGHIJKLMN"
85 D$ = A$                    --> D$ = "ABCD"
90 MID$(D$,2,3) = B$          --> D$ = "A123"
95 MID$(D$,5,1) = C$          --> D$ = "A123E"
97 MID$(D$,4) = B$            --> D$ = "A121234567"

```

TRIM\$

```

1 DIM XYZ$(16)
2 XYZ$ = "XWERT = 0.123 "
3 A$ = MID$(XYZ$,8)
4 B$ = TRIM$(MID$(XYZ$,8))
5 C$ = TRIM$(MID$(XYZ$,8),"L")
6 D$ = TRIM$(MID$(XYZ$,8),"R")

```

Content of STRING variable A\$:	" 0.123 "
Content of the STRING variable B\$:	"0.123"
Content of STRING variable C\$:	"0.123 "
Content of STRING variable D\$:	" 0.123"

TRIM\$

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Trim - Removing leading or trailing spaces

When a character field range is assigned to a string variable or character field, "TRIM\$()" returns a character string without preceding (→ Index L) or following (→ Index R) spaces. The "TRIM" command without index hides preceding and trailing spaces. If it is linked in the TRIM command (e.g. TRIM\$(A\$+B\$)), the result may be assigned only to a character array.

Syntax:

```
TRIM$ (<CharacterString>)
TRIM$ (<CharacterString> ,"L")
TRIM$ (<CharacterString> ,"R")
```

Tab. 7-279: TRIM\$ syntax

Example:

```
1 A$ = " ABCDEF "
2 B$ = TRIM$ (A$ ,"L")
3 C$ = TRIM$ (A$ ,"R")
4 D$ = TRIM$ (A$)
5 PRN# (1, ">", A$, "<")
6 PRN# (1, ">", B$, "<")
7 PRN# (1, ">", C$, "<")
8 PRN# (1, ">", D$, "<")
```

Results in the following lines in the file with the logic number 1 opened with write access:

```
> ABCDEF <
>ABCDEF <
> ABCDEF<
>ABCDEF<
```

MID\$

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Reading **Mid** character in character string

The result can be transferred to a dimensioned or to a non-dimensioned character variable:

- A dimensioned character variable receives the complete partial string defined by the MID\$ command.
- A non-dimensioned character variable receives only the starting address and length of the defined partial string. If the string expression, from which the partial string was removed, changes, the non-dimensioned character variable changes accordingly.

If it is linked (e.g. MID\$ (A\$+B\$, 2, 3)) in the MID command, the result can be assigned only to a character array.

Syntax:

```
MID$ (<STR Expression> , <StartingPoint>[, <NumberOfCharacters>])
<STR expression> String expression from which parts are to be removed.
```

<Starting point>	Specifies the position in the character array <StringExpression> from where the characters are to be removed.
<no. of characters>	Determines the number of characters to be removed. If <NumberOfCharacters> is not programmed, all characters up to the length end of the character array are removed.

Tab. 7-280: MIDS\$ syntax

The range of values for the 2nd and 3rd parameters encompasses integer values from 1 to 1024. If the range of values is not adhered to, the error message "INVALID PARAMETER" is displayed.

"NUL" is returned if a part of the character array, which is not yet assigned, is accessed.

Example: *Program:*

```

1 DIM A$(10)
2 DIM B$(5)
3 A$="ABCDEFGH IJ"
4 B$=MID$(A$,2,5)
5 C$=MID$(A$,2,5)
6 REM The variables B$ and C$ both have the following content: BCDEF
7 A$="QRSTUVWXYZ"
8 REM The variable B$ has the following content: BCDEF
   The variable C$ has the content: RSTUV

```

MCASE\$

CHAR	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description: **Mixed Case** - Converting upper-case and lower-case letters

The command converts a string expression in upper-case and lower-case letters. The conversion result is provided to the command via the return value.

Syntax:

MCASE\$ (<StringExpression>, <DataSelection>)	
<StringExpression>	String expression to be converted.
<DataSelection>	String expression "U": Conversion to upper-case letters "L": Conversion to lower-case letters
ERRNO	CPL variable If the CPL variable ERRNO is programmed at any position in the parameter list, the command does not generate any internal runtime error. The error is returned by a corresponding return value of the variable. The following return values are possible: 0: Access OK. -1: Parameter error If the CPL variable ERRNO is not entered, a runtime error is generated when an parameter error occurs.

Tab. 7-281: MCASE\$ syntax

Example:

```

1 DIM A$(20)
2 DIM B$(20)
3 A$="Hello world"

```

4 B\$=MCASE\$(A\$,"U")
 5 REM B\$ has the following content: "HALLO WELT"

Special features and restrictions: Assigning the return value to a non-dimensioned character variable causes a runtime error.

7.6.4 Programming examples

Description: A string expression can be assigned to a string variable.

Examples: **Program string variables (without previous dimensioning)**

```
1 A$="ABCDE"
2 B$=CHR$(10)
```

String variables (without previous dimensioning) behave like references or pointers:

Program:

```
10 DIM B$(10)
20 A$ = "Blue"
30 B$ = "Yellow"
40 PRN#(0,A$,B$) : REM output: YellowBlue
50 A$ = B$       : REM A$ refers to B$
60 B$ = "XXXX"
70 PRN#(0,A$,B$) : REM output: XXXXXXXX
```

During read-only access, parts of the string variables can be accessed via the MID\$ command:

```
1 A$="ABCDEFGHJKLMNOP"
2 B$=MID$(A$,2,1)
3 C$=MID$(A$,4,4)
```

The following programming cause errors:

```
4 MID$(A$,1,4)="ABCD"
4 A$=MID$(A$,1,3) + MID$(A$,4,1)
4 A$=B$ + A$
```

To continue processing a dimensioned character array, it is necessary to specifically access one or more interrelated characters. Only then it is possible to assign a character array or a part of the character array to a string variable or to another character array.

Read-only and write access to a part of a character array is performed via the MID\$ command. If only the character array name is specified, the complete character array is addressed.

Read character array

If the <n>th character of the character array is to be accessed, proceed as follows (n is less than or equal to the length of the character array and the number of characters of the array):

```
1 DIM VWX$(13)
2 VWX$="TEST TEST TES"
3 A$ = MID$(VWX$,12,1)
4 I%=12
5 A$=MID$(VWX$,I%,1)
```

The twelfth character ("E") of the VWX\$ character array is assigned to the A\$ string variable.

Write on character array

If the content of a string variable is to be assigned to the character array or a part of the character array, the assignment has to be changed.

Example: Partial writing on a character array

```
1 DIM XYZ$ (15)
2 B$="ABCDE"
3 MID$ (XYZ$, 1, 5) =B$
4 MID$ (XYZ$, 6, 5) =B$
```

1 to 10 The content of the B\$ string variable is assigned to the characters of the XYZ\$ character array.

The following programming would lead to the error message "CHARACTER FIELD NOT ASSIGNED" because the 1st to 5th characters of the character field are not yet assigned:

```
1 DIM XYZ$ (15)
2 B$="ABCDE"
4 MID$ (XYZ$, 6, 5) =B$
```

Example: Partial writing on a character array

```
1 DIM XYZ$ (100)
2 B$= "ABCDE"
3 MID$ (XYZ$, 1, 10) =B$
```

Content of the string variable B "ABCDE"

\$:

Content of array variable XYZ "ABCDE"

\$:

The array variable has the length 5. The remaining 95 characters are not assigned.

If the string variable is shorter than the character array, the character array XYZ\$ is written only in the length of the string variable. When assigning this character array to a STRING variable, it is not the entire character array (defined via DIM instruction) which is assigned, but only the range on which was previously written (→ length of the character array).

Example:

```
1 DIM XYZ$ (100)
3 MID$ (XYZ$, 1, 10) ="ABCDE"
4 MID$ (XYZ$, 6, 3) ="T"
```

The content of the XYZ\$ array variable after block 3 is: "ABCDE". The array variable length is 5. The remaining 95 characters are not assigned and therefore not part of the length.

The content of the XYZ\$ array variable after block 4 is: "ABCDET". The array variable has the length 6. The remaining 94 characters are not assigned and therefore not part of the length.

Example: Overwrite character array

```
1 DIM XYZ$ (100)
3 MID$ (XYZ$, 1, 10) ="1234567890"
```

```
4 MID$(XYZ$, 3, 3) = "T"
```

The content of the XYZ\$ array variable after block 3 is "1234567890". The array variable length is 10.

The content of the XYZ\$ array variable after block 4 is "12T4567890".

The array variable length is 10. The character "3" is overwritten by "T". The characters "4" and "5" are retained.

Example: Invalid access to the character array

```
1 DIM XYZ$(100)
```

```
3 MID$(XYZ$, 1, 6) = "ABCDEF"
```

```
5 MID$(XYZ$, 9, 5) = "TESTE"
```

The content of the XYZ\$ array variable after block 3 is "ABCDEF".

The array variable length is 6.

After block 5, an attempt is made to assign a constant to the 9th up to 13th components of the character array. This, however, results in the error message "CHARACTER FIELD NOT ASSIGNED" because the 7th and 8th components are not assigned.

If the whole character array is to be accessed, entering the variable name is sufficient.

7.7 Communicating with a client

7.7.1 MMC

INT	READ	WRITE	FUNCT	PROC	WAIT	ERRNO	!

Description:

Man Machine Communication

Sends information on the program runtime from a subroutine to a client and waits for the result from this client. This is performed using CPL variables, which can send values from the part program as well as return values to the part program.

The part program is stopped at program runtime at the point where the MMC command is received.

There are the following processing options:

- If no client that it is able to process the MMC command has logged in, the corresponding return value (=1) is assigned and the processing of the part program continues.
- If a suitable client for processing the data of the MMC command is available, there is an assignment between the part program and the client. After the client has sent a reply, the corresponding return value is set and the execution of the part program continues.

The MMC command can have up to 20 CPL variables as parameters. The name and the values of these variables are transmitted to the client.

Syntax:

MMC (<CPL-Var1>[, <Cpl-Var2>....[, <Cpl-VarN>]....])	
<CPL-Var1>...	CPL variables, N = max. 20
<CPL-VarN>	The client can write new values to the CPL variables specified in the MMC command. The CPL variables specified in the MMC command can be used in the part program.

Tab. 7-282: MMC syntax

The MMC command provides the following return values as result:

0:	OK
1:	No client present
2:	Error in client
9:	Client completed

Example:

```

10  DIM PROGNAME$ (50)
20  PROGNAME$="WinProg"
30  INTPAR%=1
40  REALPAR=1.1
50  I%=MMC (PROGNAME$, INTPAR%, REALPAR)

```

;CPL variables PROGNAME\$, INTPAR% and REALPAR are provided with their values to the client.

;The block preparation of the part program is only continued in line 60 until a corresponding 'finished' message is received.

```

60  IF I%=0 THEN
70  IF INTPAR%=2 THEN
80  ...
90  ELSE
100 ...
110 ENDIF
120 ENDIF

```

8 Appendix

8.1 Tabular overview on NC functions

8.1.1 Alphanumerically arranged according to long form

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
()	-	-	Comment: Skip content in parentheses
(MSG	-	-	Output message text
/	-	-	Skip block
//	-	-	Comment
;	-	-	Comment: Skip rest of line
ABS	-	CPL	ABS(<Input value>)
AC(...)	-	-	Local absolute dimension programming , e.g. X=AC(10)
ACOS	-	CPL	<Function value> = ACOS(<input value>): Application of the arc cosine function to the <input value>.
ACS	-	CPL	ACS(<axis>[,<selection type>[,<channel>]]): Provides the current command position of an axis.
ActRadFact	ARF		ActRadFact(<axis name> <factor> {,<axis name> <factor>}...) Program effective radius factors
Adjust	ADJ	-	ADJ(<NCF-Name>): Adjust modal NC function groups.
AND	-	CPL	<expression1> AND <expression2>: Binary operation of two BOOLEAN or INTEGER expressions
APOS	-	CPL	APOS(<axis selection>): Transfers the current actual axis value based on the machine zero point.
Area	ARA	-	Area(<BNr>,<Sta>{,<Mod>,{<P1>},{<P2>},{<D1>},{<D2>}}): Defines, activates or deactivates up to ten rectangular, two-dimensional dead zones or working ranges with axis-parallel boundaries.
ASC	-	CPL	ASC(<character string>)
ASIN	-	CPL	<Function value> = ASIN(<input value>): Application of arc cosine function on <input value>.
ASPCLR	-	-	ASPCLR(<Sr no.>) Asynchronous subroutines: log off
ASPDIS	-	-	ASPDIS(<Sr no.>) Asynchronous subroutines: switch off
ASPENA	-	-	ASPENA(<SR no.>) Activating asynchronous subroutines

Appendix

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
ASPRTP	-	-	ASPRTP(<SR no.>,<Point>) Asynchronous subroutines: Defining point of return to contour
ASPSET	-	-	ASPSET(<Up no.>,<Up name>{,<Flags>}) Asynchronous subroutines: register
ASPSTA	-	-	ASPSTA(<SR no.>,<channel no.>)) Asynchronous subroutines: trigger
AssLogName	ALN	-	ALN(...): Assign logical axis name
ASTOPA	-	-	ASTOPA[<ChannelNo.>, <Cond.1> {,<Cond.2>}{...{,<Cond.8>}}...] Channel synchronization command: The channel to be controlled is stopped as long as all conditions are met. The conditions refer to the axis positions e.g. "Z" > 20.
ASTOPO	-	-	ASTOPO[<ChannelNo.>, <Cond.1> {,<Cond.2>}{...{,<Cond.8>}}...] Channel synchronization command: The channel to be controlled is stopped as long as one condition is met. The conditions refer to the axis positions, e.g. "Z" > 20
ATAN	-	CPL	<Function value> = ATAN(<input value>): Application of arc tangent function on the <input value>.
ATAN2	-	CPL	<Function value> = TAN2(Inputvalue1 , Inputvalue2) This function can be used to determine the angle in the correct quadrant
ATBWD	-	-	AtBwd(<AxCoord>,<Coord>) Calibrate axis kinematics: Backward transformation
ATCAL	-	-	ATCAL(<File>,<OptData>,<Mask> {,<Info>}{,<NumIt>}) Calibrate axis kinematics: Optimize parameters.
ATFWD	-	-	ATFWD(<Coord>,<AxKoord>{,<ParData>}) Calibrate axis kinematics: Convert parameters
ATGET	-	-	ATGET(<ParData>{,<ATrafNo>}) Calibrate axis kinematics: Read parameters from NC
ATPUT	-	-	ATPUT(<ParData>{,<ATrafNo>}) Calibrate axis kinematics: Write parameters to NC
ATrans	ATR	Program coordinate offset	ATR(X..., Y..., Z): Additive program coordinate offset ON ATR(): Additive program coordinate offset OFF
AUXFUNC	-	-	Execute active auxiliary functions of all groups.

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
AxAcc	AAC	Axis acceleration	AAC(X..., Y..., Z): Enable AAC(1): Switch on with saved settings AAC(): Disable Program axis acceleration
AxAccSave	AAS	-	Save current axis acceleration
AxCompTab	-	-	AxCompTab(<CAX>,<table>): Exchange axis error compensation tables (LSEC and CCOMP)
AxCouple	AXC	Axis coupling	AXC(<Master>,<Slave1>(…),<Slave2>(…),…): Activate axis coupling AXC(): Deactivate axis coupling
AXADR\$	-	CPL	AXADR\$(<Index> [, [<selection type>] [, <channel number>]]) Axis and coordinate addresses
AXINF	-	CPL	Provides the system axis index for a channel or system axis as return value.
AxisReplace	AXR		AXR(<KAN1>,<PAN2>) Replace axes
AxisToSpindle	ATS	-	ATS(<Axisname>): Switch axis to spindle mode.
AxisToSpindle-Wait	ATSW	-	AxisToSpindleWait(<PAN> <PAI>{,<PAN> <PAI>}...) Switch off C-axis mode for spindles and wait
AxJerk	AJK	-	AxJerk(<values>) Change maximum axis jerk AxJerk() or AxJerk(0) Activates the axis jerk again from the machine parameters
AXO	-	CPL	AXO((<axis selection>[,<selection type>]): Transfers an active SetPos offset for a coordinate.
AXP	-	CPL	AXP(axis number>,<path information>): This function permits plane-independent part and measuring programs (program AXP instead of the address values).
AxVel	AVE	-	AxVel(<values>): Change maximum axis velocity
AxVelSave	AVS	-	Buffer maximum axis velocity
BCD	-	CPL	<BCD value> = BCD(<binary value>): Convert from BCD to binary format.

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
BcsCorr	BCR	Workpiece position correction	BCR({<XW offset>},{<YW offset>}{,<ZW offset>},{<Angle1>}{,<Angle2>}{,<Angle3>}}): Enable BCR(): Disable Placement: Tool position correction.
BIN	-	CPL	<Binary value> = BIN(<BCD value>): Convert from binary to BCD format.
BITIF	-	CPL	BITIF(<BitSignal>[,<Index>[,<IF Unit>]]): Access to the digital interface between NC and PLC.
BlkNmb	BNB	-	BlkNmb(<No>): Limits the number of program blocks read in and viewed by block preparation.
BSTOPA	-	-	BSTOPA[<ChannelNo.>, <Cond.1> {,<Cond.2>}{...{,<Cond.8>}}...] Channel synchronization command: The channel to be controlled is stopped as long as all conditions are met. The conditions refer to basic workpiece positions, e.g., "Z" > 20.
BSTOPO	-	-	BSTOPA[<ChannelNo.>, <Cond.1> {,<Cond.2>}{...{,<Cond.8>}}...] Channel synchronization command: The channel to be controlled is stopped as long as one condition is met. The conditions refer to basic workpiece positions, e.g., "Z" > 20.
CALL	-	CPL	CALL <Program number> [<Transfer parameter1>,...] [DIN]: Subroutine call from a CPL program
CASE	-	CPL	CASE <integer expression> OF LABEL <Int. constant> [,<other int. constant>] [: <instruction>]<instruction> LABEL ... OTHERWISE <instruction><instruction> ENDCASE Conditional selection from several alternatives.
ChkNxtBlk	CNB	-	ChkNxtBlk (1) Switch on continuous block monitoring ChkNxtBlk (0) Switch off continuous block monitoring
ChLength	CHL	Chamfers/roundings	CHL(<chamfer length>): Enable CHL(): Disable Insert transitional chamfers with defined chamfer length.
CHR\$	-	CPL	CHR\$(<integer expression>): Provides a character whose ordinal number in the ASCII table is equal to the value transferred via the parameter <INTEGER expression>.

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
ChSection	CHS	Chamfers/roundings	CHS(<chamfer segment>): Enable CHS(): Disable Insert transitional chamfers with defined chamfer segment.
CLOCK	-	CPL	<Function value> = CLOCK: Query of time counter in milliseconds.
CLOSE	-	CPL	CLOSE(<n>): Closes an opened file after reading or writing operations are completed.
CLRWARN	-	CPL	Delete the warnings of a channel generated with SETWARN.
COF	-	CPL	COF(<axis selection>[,<selection type>]): Provides the latest programmed contour shift of a coordinate for the current channel.
COFFS	-	-	Contour offset for 3D radius correction
Collision	CLN	Collision monitoring	CLN(1): Enable CLN(CollErr.. LA... DLA... DEF): Switch on with parameterization CLN(): Disable Collision monitoring for 2D milling cutter path correction.
ConfG0	CG0	Interpolation	ConfG0(ASY<status>) ConfG0(ASY1) Enable asynchronous G0 interpolation ConfG0(ASY0) Disable asynchronous G0 interpolation
ConstFeed	CFD	V-profile	Constant velocity
CornerLineDetection	CLD	-	Automatic detection of corners and straight lines
Coord	CRD	-	CRD(x): Switch on axis transformation CRD(): Switch off axis transformation, e.g. end face transformation
COS	-	CPL	<Function value> = COS(<input value>): Cosine function for the <input value>.
CoupleSplineTab	CST	-	CST(STAB(...)) Generate spline coupling table
CurveParamInterpolation	CPI	Interpolation	CurveParamInterpolation(1) Switch on curve parameter interpolation CurveParamInterpolation(0) Switch back to path length interpolation
D	D	D-correction	Dx: Selecting D-correction (1-99 tools) D0: Deselecting D-correction

Appendix

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
DATE	-	CPL	<String variable> = DATE: Assigns the date as DD.MM to a <STRING variables> .
DBLOAD	-	CPL	DBLOAD(<DbTable>,<Key1>,<Key2>>, <File name>[,<Mode>]): Parts of a database table or a complete database table from a file can be read into the database.
DBMOVE	-	CPL	DBMOVE(<DbTab>,<SrcKey1>,<SrcKey2>, <DestKey1>,<DestKey2>[,<Mode>]): Implementation of a data set in a database table.
DBSAVE	-	CPL	DBSAVE(<DbTab>,<Key1>,<Key2>, <File name>[,<Mode>]): Use DBSAVE to save a database table completely or in parts into a file.
DBSEA	-	CPL	DBSEA (<DbTab>,<Key1>,<Key2>, <SearchCond>,<SearchRes>[,<ResVar>]): Searches data sets in a database table.
DBSEAX	-	CPL	DBSEAX(<DbTable>,<Key1>,<Key2>, <SearchCond>,<ErgArr>,<ErgSize>): Searches data sets which correspond to the search criterion in a tool database table.
DBTAB	-	CPL	DBTAB(<DbTable>,<Key1>,<Key2> [,<ResVar>]) Reads a complete data set or a substructure of a database table to a CPL variable or writes back from this variable.
DBTABX	-	CPL	DBTABX(<DbTable>,<Key1>,<Key2> [,<Mode>]): Can read a complete data set or a substructure of a tool database table into a CPL variable or write from the variable to the data set.
DBTABXL	-	CPL	DBTABXL(<DbTable>,<Key1>,<Key2>[,<Mode>]): Can read a complete data set or a substructure of a tool database table into a CPL variable or write from the variable to the data set.
DCOFFS	-	-	Change rate of the contour offset for the 3D radius correction
DCT	-	CPL	DCT(<value selection>,<Data block> [,<Table>] [,<Unit>]): Read and write access to any D-correction table or to external tool correction values.
DcTSel	DCS	-	DCS({<Path>}<FileName>): Selection of a D-correction table
DefAxis	DAX	-	Create default axis configuration

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
DefSpindle	DSP	-	Reset channel spindles to MP
DefTangTrans	DTT	Transition angle	DTT(<Transition angle>): Activating DTT(): Deactivating Definition of tangential contour transition
DiaProg	DIA	Diameter/radius	Diameter programming
DIM	-	CPL	DIM <Variable name>(<array size1> [, <Array size2>]): Specification of the array size (dimensioning) of ARRAY variables with INTEGER constants.
DIRCR	-	CPL	DIRDEL(<Directory>) Creates a new directory.
DIRDEL	-	CPL	DIRDEL(<Directory>) Deletes an empty directory.
DIRINF	-	CPL	DIRINF(4, [<Index2>], <ResultVar> [, <file number>]): Calls management data of the file system.
DistCtrl	DCR	-	DCR(1): Distance control for digitizing ON according to the machine parameters. DCR(<Fct>): ON with individual configuration data DCR(): Distance control OFF
DPC	-	CPL	DPC(<Selection>[, <Selection type>]): Provides the latest programmed parameter of workpiece position correction "BcsCorr" of a coordinate (offset values and turning angles) for the current channel.
ED	ED	ED-correction	ED x: Selection of external tool correction (1..16 tools/edges) ED0: Deselection of external tool correction
EndPosCouple	EPC	-	EPC(<M-Coord>, <S-Coord>, <Factor>): End position coupling ON EPC(): End position coupling OFF
EOF	-	CPL	EOF(<n>): Check for end of file
ERASE	-	CPL	ERASE(<File name>): Erases files
ERRNO	-	CPL	Error message of various CPL functions.
EXP	-		EXP(<Exponent> [, <Basis>]) Calculates the exponential function regarding an arbitrary programmed basis or a default basis e (e=2.718281828...) if no <Basis> (optional parameter) is specified.
F	F	-	F-address for G93, G94, G95

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
F2 address	F2		F2=<value> The syntax is "F2=". A blank is not allowed between "F2" and "=".
FA	FA	-	Velocity for asynchronous axes
FALSE	-	CPL	<Boolean variable> = FALSE: Truth value of a BOOLEAN variable
FeedAd	FAD	Feed ad.	FAD(1) : Remove axis FROM feed computing FAD() : Axis in feed computing
FeedForward	FFW	Feed forward	FFW(X..., Y..., Z) : Enable FFW() : Disable Enabling feed forward
FeedGrp	FDG	Feed computation	FeedGrp(1) or FeedGrp Feed computation: Feed group
FeedGrpType	FDGT	Feed computation	FeedGrpType(SMOOTH) or FeedGrpType(1) Feed computation: Feed group type
FILEACCESS	-	CPL	FILEACCESS(<file name>): Determines whether a file exists and its access rights.
FILECOPY	-	CPL	Copy files
FILEDATE	-	CPL	FILEDATE(<File name>[,<Mode>]): Determines date/time of a file.
FILENO	-	CPL	Provides the next valid logical file number.
FILEPOS	-	CPL	FILEPOS(<n>[,<mode>]): Provides the record number of the current record and the record offset of a random file. Provides the current byte position of the file pointer for sequential files.
FILESIZE	-	CPL	FILESIZE(<n>[,<k>]): Provides file size or the limit up to which it has already been written to a file.
FlyMeas	FME	Flying Measurement	FME(MpiAxis< i>) X... Y... Z...: Travel with flying measurement
FOR NEXT	-	CPL	FOR <counting var.>=<start count> [STEP <step size>] TO <end count><routine>NEXT [<count variable>]: Loop construction with automatically updating counter.
FsMove	FSM	-	Move to fixed stop (Fs = fixed stop)
FsProbe	FSP	-	Measuring at fixed stop

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
FsReset	FSR	-	Cancel fixed stop
FsTorque	FST	-	Torque for fixed stop
G	-	-	User-defined local subroutines
G0	G0	Interpolation	Linear interpolation (rapid traverse)
G0(..)	G0(..)	Interpolation	Rapid traverse with options Optional parameters: - NIPS: without exact stop - IPS1: exact stop window 1 - IPS2: exact stop window 2 - IPS3: Exact stop window 3 (only decelerate to V = 0) - POL/POLAR: with polar coordinate programming, e.g. G0(POL) X50 A45 B10, with A, B = polar angle 1/2
G1	G1	Interpolation	Linear interpolation (feed)
G2	G2	Interpolation	Clockwise circular interpolation (incl. helical-N)
G2(POL)	G2(..)	Interpolation	Clockwise circular interpolation (incl. helical-N) with polar coordinate programming
G3	G3	Interpolation	Counterclockwise circular interpolation (incl. helical-N)
G3(POL)	G3(..)	Interpolation	Counterclockwise circular interpolation (incl. helical-N) with polar coordinate programming
G4	G4	-	G4(F..): Dwell time in seconds G4(S...): Dwell time in spindle rotations. Reference spindle is the main spindle (MainSp)
G5	G5	Interpolation	Tangential circle entry (incl. helical N)
G6	G6	Interpolation	Splines, nurbs
G8	G8	Path slope	Path slope ON with path SHAPE according to machine parameters
G8(SHAPE...)	-	Path slope	Path slope ON with programmed Path SHAPE
G9	G9	Path slope	Path slope OFF (without SHAPE)
G9(..)	G9(..)	Path slope	G9(X..., Y): Path slope OFF with axis-by-axis programmed SHAPE order G9(ASHAPE): Path slope OFF with axis-by-axis SHAPE order according to machine parameters G9(SHAPE ...): Path slope OFF with programmed path SHAPE G9(SIN ...): Path slope OFF with sin ² -shaped SHAPE (possible SHAPE order: 5, 10, 15, 20, 40)

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
G140	-	3D radius correction	3D radius correction OFF
G141	-	3D radius correction	3D radius correction to the left of the contour
G142	-	3D radius correction	3D radius correction to the right of the contour
G152.1	G152	IP bank 1	Placement: Inclined plane, programmable, bank 1
G152.2-G152.5	-	IP bank 2-5	Placement: Inclined plane, programmable, bank 2-5
G153	-	SE	Placement: Inclined plane, all OFF
G153.1-G153.5	-	IP bank 1-5	Placement: Inclined plane, bank 1-5 OFF
G154.1	G154	IP bank 1	Placement: 1. Inclined plane, bank 1 ON
G154.2-G154.5	-	IP bank 2-5	Placement: 1. Inclined plane, bank 2-5 ON
G155.1	G155	IP bank 1	Placement: 2. Inclined plane, bank 1 ON
G155.2-G155.5	-	IP bank 2-5	Placement: 2. Inclined plane, bank 2-5 ON
G156.1	G156	IP bank 1	Placement: 3. Inclined plane, bank 1 ON
G156.2-G156.5	-	IP bank 2-5	Placement: 3. Inclined plane, bank 2-5 ON
G157.1	G157	IP bank 1	Placement: 4. Inclined plane, bank 1 ON
G157.2-G157.5	-	IP bank 2-5	Placement: 4. Inclined plane, bank 2-5 ON
G158.1	G158	IP bank 1	Placement: 5. Inclined plane, bank 1 ON
G158.2-G158.5	-	IP bank 2-5	Placement: 5. Inclined plane, bank 2-5 ON
G159.1	G159	IP bank 1	Placement: 6. Inclined plane, bank 1 ON
G159.2-G159.5	-	IP bank 2-5	Placement: 5. Inclined plane, bank 2-5 ON
G16	-	Plane selection	No plane
G17	-	Plane selection	Plane selection XY
G17(...),G18(...); G19(...)	-	Plane selection	Advanced plane switching G17/18/19(<axis1>,<axis2>,<axis3>): The axes in parentheses open the WCS and receive the meaning X, Y and Z. Subsequently, the programmed plane is selected. G17/18/19(): Reset the workpiece coordinate system to the default setting and select the programmed plane.
G18	-	Plane selection	Plane selection ZX
G19	-	Plane selection	Plane selection YZ
G20	-	Plane selection	Free plane selection (independent of WCS) The placements influence the WCS. The plane is defined irrespective of the WCS.
G33	-	Thread	Thread cutting
G40	-	Path correction	Milling cutter path correction OFF
G41	-	Path correction	Milling cutter path correction to the left of the workpiece

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
G42	-	Path correction	Milling cutter path correction to the right of the workpiece
G43	-	Insertion strategy	Insertion strategy: arc
G44	-	Insertion strategy	Insertion strategy: intersection
G45	-	Velocity of milling cutter	Velocity of the milling cutter contact point
G46	-	Velocity of milling cutter	Velocity of milling cutter center
G47	-	Tool correction	Switch on tool length correction
G47(..)	-	Tool correction	G47(<L1-coord>,<L2-coord>,<L3-coord>)G47(ActPlane): Tool length correction ON when switching the correction assignment. G47(): Tool length correction ON with correction assignment according to machine parameters.
G48	-	Tool correction	Tool length correction OFF
G53	-	-	All zero offsets OFF
G53.1-G53.5	-	ZO bank 1-5	Zero offsets, bank 1-5 OFF
G54.1	G54	ZO bank 1	1. Zero offset, bank 1 ON
G54.2-G54.5	-	ZO bank 2-5	1. Zero offset, bank 2-5 ON
G55.1	G55	ZO bank 1	2. Zero offset, bank 1 ON
G55.2-G55.5	-	ZO bank 2-5	2. Zero offset, bank 2-5 ON
G56.1	G56	ZO bank 1	3. Zero offset, bank 1 ON
G56.2-G56.5	-	ZO bank 2-5	3. Zero offset, bank 2-5 ON
G57.1	G57	ZO bank 1	4. Zero offset, bank 1 ON
G57.2-G57.5	-	ZO bank 2-5	4. Zero offset, bank 2-5 ON
G58.1	G58	ZO bank 1	5. Zero offset, bank 1 ON
G58.2-G58.5	-	ZO bank 2-5	5. Zero offset, bank 2-5 ON
G59.1	G59	ZO bank 1	6. Zero offset, bank 1 ON
G59.2-G59.5	-	ZO bank 2-5	6. Zero offset, bank 2-5 ON
G61		Exact stop	Exact stop ON
G61(IPS...)	-	Exact stop	Exact stop ON with exact stop window IPS1, IPS2 or IPS3
G62	-	Exact stop	Exact stop OFF
G63	-	-	G63(M3/M4, S.../H...) F... Z: Tapping without compensating chuck
G70	-	Inch/metric	Inch programming affects the programmed path and length, feeds and accelerations.
G71	-	Inch/metric	Metric programming

Appendix

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
G74	-	-	G74 X1 Y1 Z1 ...: Approach reference point coordinates
G74(Home)	-	-	G74(HOME) X1 Y1 Z1: Approach reference point (real referencing, also for asynchronous axes)
G75	-	-	Travel to touch probe (motion abort)
G76	-	-	Approach to fixed machine position (machine coordinates)
G77	-	-	G77 <coord 1><Mode> <coord n><Mode> ... F<value> Alternative syntax: REPOS Asynchronous subroutines: Reposition individual coordinates
G78	-	-	G78 {{{-}<Coordinate1>},{-}<Coordinate2>},{-}<Coordinate3>}}: Correction switching ON
G79	-	-	Correction switching OFF
G90	-	Abs/Rel	Absolute dimension programming
G91	-	Abs/Rel	Relative dimension programming
G93	-	Feed programming	Time programming
G94	-	Feed programming	Feed programming (per min)
G94(...)	-	Feed programming	G94{{DF <value>},{ DS1 <value>, DS2 <value>, ...}}): Incremental velocity programming with acceleration adaptation.
G95	-	Feed programming	Feed programming (per rotation)
G96	-	Spindle programming	G96{{{<reference axis>,<actuation point>}}): Constant cutting velocity
G97	-	Spindle programming	Direct speed programming
GCT(0)	-	-	Switch off straightness and angle error compensation
GCT(1)	-	-	Switch on straightness and angle error compensation
GetAxis	GAX	-	GAX(...): Apply free axis to channel
GETERR	-	CPL	GETERR(<channel> [,<category>], <Error>-No> [,<Number>]): Provides error number, channel number and error category for the current errors.
GetSpindle	GSP	-	GetSpindle (<SysSpNo> <SysSpName>,<ChanSpNo> <ChanSp-Name>,{...}): Create channel spindles

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
GMSG	-	-	(GMSG<Information text>) Program notes into the user interface.
GoAhead	GOA	-	Jump forward
GoBack	GOB	-	Jump backward
GoCond	GOC	-	Conditional jump
GOTO	-	CPL	GOTO <Jump target>: Unconditional program jumps to line number, block number or label.
GoTo	-	-	Unconditional jump
HsBlkSwitch	HSB	-	HSB(HS<X>=<Y>): Block switching on-the-fly via high-speed signal
HsBlkSwitch(..HS STOP=..)	HSB	-	HSB(...,HSSTOP=..): Block switching with abort via high-speed signal
HWOC	-	-	HWOC({CHAN<channel no>},CRDNO <coordno>,{ STEP<Incr>}): Online correction in workpiece coordinates ON HWOCDIS({CHAN<channel no>}): Online correction OFF; save values HWO(): Online correction OFF; delete values
IC(...)	-	-	Local relative dimension programming e.g. X=IC(5)
IF ENDIF	-	CPL	IF <condition> THEN <routine> [ELSE <Optional routine>] ENDIF: Conditional jump to a routine or alternative routine.
InitMeas	IME	-	IME(MpiAxis i): Initialize "Flying Measurement"
INP#	-	CPL	INP#(<n>,<Variable>[,<Variable>] [...] [:]): Read access to a file
INSDEP	-	-	3D radius correction plunging depth
INSTR	-	CPL	INSTR(<string sequence>,<string expression>[,<starting point>]): Searches a <String expression> from <starting point> after a <string sequence>and outputs their initial position as INTEGER value.
INT	-	CPL	<Integer number>=INT(<Real expression>): Converts a <real expression> into an <integer number> by skipping the number of decimals.
InvTime	ITIM		ITIM <value> Inverse time programming

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
IPS1 IPS2 IPS3s	-	-	IPS1: Fine exact stop window size IPS2: Rough exact stop window size IPS3: Infinite exact stop window size
JerkAxList	JAL	-	Determine system axes to specify the optimum SHAPE filter in case of an active jerk limitation
JerkControl	JKC	-	Jogging in workpiece coordinates
JogWCSSelect	-	-	Selects coordinates to jog in workpiece coordinates (setup mode).
KinSplitDisable	KSD	-	KinSplitDisable(0) Enabling kinematic NC block splitting KinSplitDisable(1) Disabling kinematic NC block splitting
KvProg	KVP	CO progr.	COP(Y..., Y..., Z): CO programming ON KVP(): CO programming OFF
LCP	-	-	Lag compensation One or more additional axis/axes is/are coupled to (or decoupled from) its/their master(s) defined in the configuration.
LCTS	-	-	Linear circular spline conversion LCTS(1): Enable LCTS(0): Disable
LEN	-	CPL	LEN(<String expression>): Provides the number of characters in a STRING expression.
LEN	-	-	LEN=<value>: Splits the traversing block into several sections of equal length.
LFCnf	LFC	-	LFC({LL(...)}): Parameterize laser power control
LFP	LFP	-	LFP, LFP(1), LFP({LL(...)}): Laser power control ON LFP(0): Laser power control OFF
LinDownFeed	LND	V-profile	Linear deceleration
LinearToSpline	LTS	-	Linear to spline switching LinearToSpline(1): Enable LinearToSpline(0): Disable
LinModZp	LMZ	-	LMZ{{LinModAxis i}}: Set the linear modulo axis to zero (Zp = zero point)
LinUpFeed	LNU	V-profile	Linear acceleration
LJUST	-	CPL	Switch to left-justified data output; effective until the end of program run.
M	M	-	User-defined local subroutines

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
M0	M0	-	Program stop
M1	M1	-	Conditional program stop
M19, M119, M219	-	-	Orient spindle
M2	M2	-	End of main program
M3, M103,M203M13, M113,M213	-	-	Spindle clockwise rotation Spindle clockwise rotation with coolant ON
M30	M30	-	End of main program
M4, M104,M204,M14, M114,M214	-	-	Spindle counterclockwise Spindle counterclockwise rotation with coolant ON
M40, M140	-	-	Automatic gear range selection
M41 ... M44,M141 ... M241 ... M244	-	-	Manual gear range selection
M48, M148,M248	-	-	Disengage gear
M5, M105,M205	-	-	Spindle stop
MainSp	MSP	-	Select main spindle, e.g. for G33 and G95
MCA	-	CPL	MCA(<Block>,<Index>[,<channel>]): Transfers the contents of a single MACODA parameter.
MCODS	-	CPL	MCODS(<type>,<channel>,<version>,<buffer>,<size>[,<P1>]): Calls the Motion Control Data Services of the NCS. Thus, data and states can be read from the NC.
MCOPS	-	CPL	MCOPS(<Fct>,<channel>[[,<P1>][,<P2>], [<P3>]],<P4>)) Calls the Motion Control process services of the NCS. This allows channels in the NC to be controlled.
MCS	-	CPL	MCS(<coordinate>[,<selection type>[,<channel>]]): Provides the current machine position (MCS) for a machine coordinate without straightness and angle error correction.
MID\$	-	CPL	MID\$(<string expression>,<starting point> [,<no. of characters>]): Takes part of the STRING expression and outputs it as text. The result can be transferred to a STRING variable or to an appropriately dimensioned character array. MID\$(<character field>,<starting point>[,<number of characters>]): Overwrites parts of a character array.
Mirror	MIR	Mirroring	MIR(X1, Y1,Z1,...): Mirroring coordinates ON MIR(): Mirroring coordinates OFF

Appendix

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
MMC	-	CPL	MMC(<CPL-Var1>[,<CPL-Var2>... [,<CPL-VarN>]...]): Sends information on the program runtime from a subroutine to a client and waits for the result from this client.
MOC			MOC(MP<probe> CI<customer input> HS<HighSpeed>{, ES<edge>}, RE<response>{, AS<Asup>}{, ER<error>}{, <channel axis name1>{, <channel axis name2> ...}}) Measuring on a contour
Modulo	MOD	-	Modulo calculation for incrementally traversed endless axes
MPEDGE			MPEDGE(MP1 MP2{,ES1 ES2}) Change edge configuration of touch probe
MSG	-	-	(MSG<Information text>) Program notes in user interface
NCF	-	CPL	NCF(<NC function>) Transfers the syntax of the active NC function in the NC modal group of <NC-function>.
Nibble	NIB	Punching	NIB(NUM...): Nibble processing ON NIB(): Nibble processing OFF
NJUST	-	CPL	Earlier reset from left-justified data output to formatted output.
NOT	-	CPL	NOT <expression>: Negation of a BOOLEAN expression or bit-wise negation of an INTEGER expression.
NUL	-	CPL	<Variable> = NUL: Delete variable
NUM	-	-	NUM=<value>: Splits the traversing block into a defined number of sections of equal length.
O	O	-	O(...): Tool orientation motion
OFFSTOPA	-	-	Channel synchronization command: Deletes the programmed "AND stop conditions" in the control channel (ASTOPA, BSTOPA, WSTOPA).
OFFSTOPO	-	-	Channel synchronization command: Deletes the programmed "OR stop conditions" in the control channel (ASTOPO, BSTOPO, WSTOPO).
Omega	-	-	Omega address
OPENR	-	CPL	OPENR(<n>,<Prog-Name>[,<Record length>]): Opens file for following read-only access.

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
OPENW	-	CPL	OPENW(<n>,<Progr-Name>[,<empty param.>][,<Comment>][,<Record length>]]) Opens file for following write access.
OR	-	CPL	<expression> OR <expression>: Binary operation of two BOOLEAN or INTEGER expressions with the OR operation.
OvrDis	OVD	Feed 100%	Feed potentiometer OFF
OvrEna	OVE	Feed 100%	Feed potentiometer ON
P	P	-	Subroutine call (also directly without P)
PL	-	-	Programming spline parameter length
PathAcc	PAC	Path acceleration	PAC(ACC... UP..., DOWN ...): Enable PAC(): Disable Path acceleration programming, either separately for acceleration and deceleration or combined.
PCS	-	CPL	PCS(<coordinate>[,<selection type>]): Provides the latest programmed absolute position of a coordinate.
PCSPROBE	-	CPL	PCSPROBE (<coordinate>[,<selection type>]): Reads the measured value for each coordinate
PDIM	-	CPL	PDIM <ParameterName>(<ArraySize>) Has to be used if a subroutine: - is to be called with a string constant as transfer parameter and - the calling program is selected without linking.
PLC	-	CPL	PLC(<type>,<empty param.>,<address>,<size>): This function is used to access the operands of the PLC.
PMT	-	CPL	PMT(<coord. selection>,<PI code>[,<PIBank>][,<Table>][,<Unit>]]): Read/write access to an xml placement table.
PmTSel	PMS	-	PMS(<Placement table name>): Selection of placement table
PMV	-	CPL	PMV(<coord. selection>[,<PI bank>]): Provides the effective placement values for a coordinate.
PolarPol	POP	PolarPol	POP(X...,Y...,Z...): Set pole POP(): Pole at origin Pole for polar coordinate programming.
PoleSet	PLS	Set pole	PLS(X..., Y..., Z...): Set pole PLS(): Pole at origin Pole for mirroring/rotation input help.

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
PosDepHSOut	PHS	-	PHS(<mode>{,<distance>}{,<duration>}): Programmable position-dependent high-speed output
PosMode	PMD	Positioning type	PMD(A..., B...): Switch positioning type PMD(): Positioning type acc. to machine parameters Positioning type for endless axes
PPOS	-	CPL	PPOS(<axis selection>[,<axis type>]): Queries the axis position for the switching point of the touch probe.
PrecProg	PRP	Accuracy programming	PRP(DIST... EPS...): Accuracy programming ON PRP(): Accuracy programming OFF
PREP	-	-	PREP(<BlkCnt> { , SILENT }) Program processing depending on the block preparation
PRN#	-	CPL	PRN#(<n>[,<expression>][,<expression>] [,<expression>][,...][.]): Write access on a file.
PROBE	-	CPL	PROBE(<axis selection>[,<axis type>]): Queries the axis position for the switching point of the touch probe related to the axis zero point coordinates of the MCS.
PtBlkEnd	PTE	Punching stroke	PTE(X..., Y..., Z): Stroke trigger at block end
PtDefault	PTD	Punching stroke	Stroke trigger according to parameterization
PtInpos	PTI	Punching stroke	PTI(X..., Y..., Z): Stroke trigger at Inpos (Pt = punch time)
PtpMove	PTP	-	In case of an active axis transformation, a point-to-point traversing motion (PTP) is executed.
Punch	PUN	Punching	PUN(NUM...): Punch processing ON PUN(): Punch processing OFF
RadProg	RAD	Diameter/radius	Radius programming
RadialAcc	RAC	-	RadialAcc (<value>): Setting maximum radial acceleration RadialAcc(0): Canceling radial acceleration limit
RecordRevVec	RRV	-	RRV(...) Record retract vector
RedTorque	RDT	-	RDT(<axis1>{,<axis2>{, ...}}): Torque specification for torque reduction
ReentContBlk	RCB	-	Continuing Program with Block Pre-Run after Program Abort
REM	-	CPL	REM <CommentText> Program comments

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
RemAxis	RAX	-	RAX(...): Remove axis from channel
RemLogName	RLN	-	RLN(...): Remove logical axis name
RemSpindle	RSP	-	RemSpindle (<SysSpNo> <SysSpName> <ChanSpName , {...}) Remove channel spindles
REPEAT	-	CPL	REPEAT <Routine> UNTIL <Condition>: Loop construction with query of the abort condition after the routine has been processed for the first time.
REPEAT	RPT		RPT<repetition factor> Block repetition
REPOSDEF	-	-	Defines return to contour feed and/or point of return to contour locally (in an asynchronous subroutine).
REPOSTP	-	-	REPOSTP(<point>) Asynchronous subroutines: Define point of return to contour in asynchronous subroutines.
RetTo	-	-	Subroutine return to any program block
REV	-		REV(<EventNo> { , <EventNo> ...}): A bit event is deleted at runtime.
REWRITE	-	CPL	REWRITE(<n>): Overwrite existing file.
Rotate	ROT	Rotate	ROT(<angle>): Coordinate rotation ON ROT(): Coordinate rotation OFF
ROTAX	-	-	Define rotary axis for vector orientation
ROUND	-	CPL	<Integer number> = ROUND(<Real expression>): Convert REAL expression into INTEGER number by rounding up or down.
RoundEps	RNE	Chamfers/roundings	RNE(<value>): Rounding ON RNE(): Rounding OFF Rounding with defined contour shift.
Rounding	RND	Chamfers/roundings	RND(<value>): Rounding ON RND(): Rounding OFF Rounding with defined rounding radius
S	S	-	S<number>=<value> S<value>: Program spindle speed
Scale	SCL	Scaling	SCL(X..., Y..., Z): Coordinate scaling ON SCL(): Coordinate scaling OFF

Appendix

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
SCL	-	CPL	SCL(<SCL selection>[,<selection>[,<selection type>]]): Provides the latest programmed parameter of the PLS and ROT functions (polar coordinates, scaling factors and rotating angle) for the current channel.
SCS3	-	CPL	SCS3(<Axis index>,<EID-No>[,<ERRNO>]): Read-only access to Sercos drive parameters with extended ident number.
SCSL3	-	CPL	SCSL3(<Axis index>,<EID-No><File name>[,<ERRNO>]): Create file for Sercos parameter lists with extended ident number.
SD	-	CPL	SD(<group>[,<Index1>[,<Index2>[,<Index3>]]]): Reads active system data of the NC control.
SDLOAD	-	CPL	SDLOAD(<FileName>,<Root-Element_1>[,...,<Root-Element_n>]) Read system data from file
SDR	-	CPL	SDR(<group>[,<Index1>[,<Index2>]]): Read active system data of the NC control in REAL format.
SDSAVE	-	CPL	SDSAVE(<FileName>,<Root-Element_1>[,...,<Root-Element_n>])
SEEK	-	CPL	SEEK(<n>,<k>[,<o>]): Positions file pointer on the <k>th record of a random file or on the kth byte of a sequential file.
SelCrdCouple	SCC	-	SCC(SC<channel>, CL(Q1>,<Z1>,{...})): Selected additional coordinate coupling ON SCC(): All coordinate couplings OFF
SelCrdCoupleTab	SCCT	-	Selective additive coordinate coupling with table
SETERR	-	CPL	Generate a runtime error after evaluating ERRNO.
SetPos	PLC	-	Set program position
SETWARN	-	CPL	Generate a warning after evaluating ERRNO.
SEV	-		SEV(<EventNo> {, <EventNo> ...}): A bit event is triggered (set) at runtime.
Shift	SHT	-	SHT(X..., Y ..., Z): Contour shift ON SHT(): Contour shift OFF
Sin2DownFeed	S2D	V-profile	Sine ² -shaped deceleration
Sin2UpFeed	S2U	V-profile	Sine ² -shaped acceleration
SinDownFeed	SND	V-profile	Sine-shaped deceleration
SinUpFeed	SNU	V-profile	Sine-shaped acceleration

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
Smax	SMX	-	SMX(<value>): Maximum speed for spindle.
Smin	SMN	-	SMN(<value>): Minimum speed for spindle.
SpAdjSp	-	-	SpAdjSp Adjust spindles in block pre-run init adjustment program
SpAdjStr	-	-	SpAdjStr<Spindle> Determining spindle syntax for block pre-run adjustment program
SpAdmin	SPA	-	SPA(Si=0 1): Conditionally release or apply a reserved spindle by another channel.
SpCouple_Wait	SPC_WA IT	-	SPC_WAIT(CP=1..4): Wait for synchronous mode of the specified couple group.
SpCoupleConfig	SPCC	-	SPCC(CP=1..4, MA=<Master>, Si=1, Sj=1, ...): Define coupling group, delete or add following spindles, cancel coupling group.
SpCoupleDist	SPCD	-	SPCD(S1=<distance>, S2=<distance>, ...): Define coupling distance for spindle couplings.
SpCoupleErrWin	SPCE	-	SPCE(S1=<window>, S2=<window>, ...): Define synchronous run error window for Define spindle couplings.
SpCouplePosOffs	SPCP	-	SPCP(S1=<offset>, S2=<offset>, ... {POSVEL<speed>}): Define angular offset for the active spindle coupling group.
SpCouplePos- Offs_Wait	SPCP_W AIT	-	SPCP_WAIT(CP=1..4): Wait for angular offset for the programmed coupling group.
SpCoupleSync- Win	SPCS	-	SPCS(S1=<window>, S2=<window>, ...): Define synchronization window spindle couplings.
SPG	-	-	SPG<group>(<numbers>) SPGALL(0): Defining/deleting spindle groups
SplineCornering	SCO	-	SplineCornering(E<Distance>) SplineCornering(L1=<L1>, L2=<L2>): Rounding corners with splines
SplineCornerin- gLocal	SCOL	-	Rounding corners with splines (local)
SplineDef	SPD	-	Specifying spline variant
SpindleToAxis	STA	-	STA(...) Switching spindle to axis mode.

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
SpindleToAxis-Wait	STAW	-	SpindleToAxisWait(<PAN> <PAI>{ ,<PAN> <PAI>}...) Switch on C-axis mode for spindles and wait
Split	SPT	-	Split({<Mode>},{<partial length>}): Splits traversing blocks into several sections if they exceed a certain length.
SpindleWait	SPWAIT	-	SpindleWait(<Spindle> { ,<Condition>}) Waits at the end of each block until the respective spindle reached the desired state.
SpindleWaitMode	SWM	-	SpindleWaitMode(<Spindle>,<Switch on> { , <Strategy>): Waiting strategy for the function "Path motion and spindle functions"
SpMode	SPM	-	SPM(S1=0 1, S2=0 1, ...): Manually switch spindle drive interface between velocity mode and position mode
SPOS	-	CPL	SPOS(<axis selection>): Transfers the current axis command value of a physical axis.
SPV	-	-	SPV [<Perm.CPL-Var.> = <simple CPL expression>] Channel synchronization command: The permanent CPL variable is written at runtime.
SPVE	-	-	SPV E[<Perm.CPL var.> = <CPL expression>] Channel synchronization command: The permanent CPL variable is written at runtime. The CPL expression can include CPL commands, but is already terminated during block preparation time.
SQRT	-	CPL	<Function value> = SQRT(<input value>): Apply square root function to <input value>.
SSD	-	CPL	SSD[<SystemDate> =<simple CPL expression>] The value to be assigned is determined at runtime.
SSDE	-	CPL	SSDE[<System date> =<CPL expression>] The value to be assigned is determined at preparation time.
SSDQ	-	-	SSDQ(M <Message> { , Q <XPath>} { , C <XPath>}) Writing to a system data queue
SSpAdm	-	-	SSpAdm(<SSpNo.> <SSpName>, <Mode> { , <SSpNo.> <SSpName>, <Mode>} ...): Release/apply reserved spindle
SSpAdmOff	-	-	SSpAdmOff(<SSpNo.> <SSpName>, <Channel no.> { , <Channel no.> } ...) SSpAdmOff(<SSpNo.> <SSpName>): Switch off spindle motion management for specified channels.

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
SSPG	-	-	SSPG<group>=<value>: Set speed for spindle group, see also S<value>
SSpGear	-	-	SSpGear(<SSpNo.> <SSpName>, <step>{ ,<SSpNo.> <SSpName>, <step>} ...): Change gear range
SSpMax	-	-	SSpMax(<SSpNo.> <SSpName>, <speed>{ ,<SSpNo.> <SSpName>, <speed>} ...): Limiting maximum speed
SSpMin	-	-	SSpMin(<SSpNo.> <SSpName>, <speed>{ ,<SSpNo.> <SSpName>, <speed>} ...): Limiting minimum speed
SSpMode	-	-	SSpMode(<SSpNo.> <SSpName>, <Mode> { ,<SSpNo.> <SSpName>, <Mode>} ...): Switching between position and speed interface
SSpMove	SSPM	-	SSpMove(<SSpNo.> <SSpName>, <task>{ ,<SSpNo.> <SSpName>, <task>} ...): Motion programming
SSpOri	SSPO	-	SSpOri(<SSpNo.> <SSpName>, <angle>{ ,<SSpNo.> <SSpName>, <angle>} ...): Spindle orientation
SSpSpeed	SSPS	-	SSpAdmOff(<SSpNo.> <SSpName>, <Speed> { , <SSpNo.> <SSpName>, <Speed>} ...): Speed programming
StatToolOri	STO	-	StatToolOri({<φ>} {,<θ>} {,<ψ>}) Parameterize static tool orientation
StepModeDisable	SMD	-	StepModeDisable({{Variable={}{Value}}}{,}{Variable={}{Value}}}): Used to process parts of an NC program in one of the operation modes in one step: Single Step, Single Block, Program Block and Debug Step. That means that the blocks are processed without con- firming NC start
StepModeEnable	SME	-	StepModeEnable No parameters can be programmed. The block - in which the SME is programmed - is one of the blocks the NC start is suppressed in.
STR\$	-	CPL	STR\$([<Format string>,<value>): Converts the numeric expression <Value> into a character string that can be assigned to only one character array. If <Format string> is programmed, the character string can be output as formatted.
TAN	-	CPL	<Function value> = TAN(<input value>): Apply tangent function to <input value>.

Appendix

NC function name		Group	Description/explanation
Long form	Short form	("-" refers to: Applies block by block)	
TangTool	TTL	Tool orient.	TTL(TAX..., SYM..., ANG..., IA..., PLC...): Tangential tool guidance ON TTL(): Tangential tool guidance OFF
TangToolOri	TTO	Tangential tool orient.	TTO(SYM..., ANG...): Tangential tool orientation ON TTO(): Tangential tool orientation OFF
TappRet1	-	-	Switches the spindle(s) - running again in spindle speed mode after control reset (control startup) - to position mode.
TappRet2	-	-	Initiates the actual retract motion. with the programmed F-value.
TappSp	TSP	-	TSP(CAXi, ..., GRPj, ...): Spindle selection for tapping without compensating chuck.
TargetAxisConf	TAX		TAX or TargetAxisConf Axis configuration adjustment
TCM(, ,)	-	-	TCM({<xVal>},{<yVal>},{<zVal>}): Traversing in the TCS
TcsDef	TCS	-	Explicitly specify the position of the tool coordinate system (implicit tool length correction combined with corresponding axis transformation).
TCV	-	CPL	TCV(<value selection>[,<correction selection>]): Provides the latest programmed tool correction values.
ThreadSet	TST	-	Thread cutting, additional functions
TIME	-	CPL	<String variable> = TIME: Assigns the time to a STRING variable as HH.MM.SS.
Trans	TRS	Program coordinate offset	TRS(X ..., Y ..., Z): Program coordinate offset ON TRS(): Program coordinate offset OFF
TRIM\$	-	CPL	TRIM\$(<CharacterString>) TRIM\$(<character string>,"L") TRIM\$(<character string>,"R"). For the assignment of a character array range to a STRING variable or a character array, it provides a character string without a preceding (index "L") or following (index "R") space. If there is no index, both preceding and following spaces are hidden.
TRUE	-	CPL	<Boolean variable> = TRUE: Truth value of a BOOLEAN variable
VAL	-	CPL	VAL(<string expression>): Returns the numeric value of the STRING expression.
VARINF	-	-	VARINF(<Variable name>) Name of the structured system date or permanent variable.

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
VERSINF\$	-	CPL	VERSINF\$(<Index1>[,<Index2>]): Calls management data, e.g. the software version.
VirtAxisPos	VAP	-	Sets the axis position of virtual synchronous axes in the current channel.
VREC_START	-	-	Start diagnostic recording for velocity control
VREC_STOP	-	-	Exit diagnostic recording.
WAIT	-	-	WAIT in NC block (like CPL WAIT)
WAIT	-	CPL	WAIT (without parameter): Stops block preparation until all the blocks programmed before WAIT are processed. WAIT (,<idle time>): Stops block processing until a specified period elapsed. WAIT(BITIF(...)): Stops block preparation until a certain state occurs at the PLC-NC bit interface.
WAITA	-	-	WAITA[BITIF(<Parameter>){=<State>} {,...}{,<Timeout>}] Channel synchronization command: When the program is at runtime, wait until all queried interface signals are given or until the timeout has elapsed.
WaitAxis	WAX	-	WAX(...): Wait until the axis is released and then apply it to the channel.
WAITO	-	-	WAITO[BITIF(<Parameter>){=<State>} {,...}{,<Timeout>}] Channel synchronization command: When the program is at runtime, wait until one of the queried interface signals is given or until the timeout has elapsed.
WCS	-	CPL	WCS(<coordinate>[,<selection type>[,<channel>]]): Provides the current workpiece position without online correction values for a coordinate.
WEV	-		WEV(<EventNo> {, EventNo ...}): With the WEV function, it is waited at runtime (or the continuous block switching is impeded) until a certain bit event occurs.
WHILE	-	CPL	WHILE <condition> DO <routine>END Loop construction with query of the abort condition before the loop is iterated for the first time.
WPV	-	-	WPV[<Perm.CPL var.><Comparison op.> <simpl. CPL expression>{,<Timeout>}] Channel synchronization command: When the program is at runtime, wait until the expression is met or until the timeout has elapsed.

Appendix

NC function name		Group ("-" refers to: Applies block by block)	Description/explanation
Long form	Short form		
WPVE	-	-	WPVE[<Perm.CPL var.><Comparison op.> <CPL expr.>{,<Timeout>}] Channel synchronization command: When the program is at runtime, wait until the expression is met or until the timeout has elapsed. The CPL expression can include CPL commands, but is already terminated during block preparation time.
Writeld3	WID3	-	WID3(S-0-0104.0.0, X..., Y..., Z): Writing Sercos parameters with extended ident number
WSTOPA	-	-	WSTOPA[<channel no.>,<cond.1>{,<cond.2>} {...,<cond.8>}...] Channel synchronization command: The channel to be controlled is stopped as long as all the conditions are met. The conditions refer to the workpiece positions, e.g. "Z" > 20.
WSTOPO	-	-	WSTOPO[<channel no.>,<cond.1>{,<cond.2>} {...,<cond.8>}...] Channel synchronization command: The channel to be controlled is stopped as long as one of the conditions is met. The conditions refer to workpiece positions, e.g., "Z" > 20.
XOR	-	CPL	<expression> XOR <expression>: Binary operation of two BOOLEAN or INTEGER expressions with the EXCLUSIVE OR operation.
XTAB	-	CPL	XTAB(<Table>,<Substructure>): Read and write access to any XML table of type DCT, ZOT, PMT or GCT.
XTABCR	-	CPL	XTABCR(<Table>,<Type> [,<TableTemplate>]): Creates an XML table with no data in the MTX file system.
ZOCDEL	-	CPL	ZOCDEL(<table>,<position>[,<channel>]): Deletes an axis from an xml zero offset table.
ZOCINS	-	CPL	ZOCINS(<table>,<position>,<axis name> [,<AxisType>][,<AxisChannel>][,<PosChannel>]]): Inserts a new axis in front of an existing column in an XML zero offset table.
ZOT	-	CPL	ZOT(<column selection>,<ZO code> [,<ZOBank>][,<Table>][,<Unit>]]): Read and write access to any XML zero offset table.

NC function name		Group (“-” refers to: Applies block by block)	Description/explanation
Long form	Short form		
ZOTCR	-	CPL	ZOTCR(<table>,<channel/template>): Creates an XML zero offset table with no correction sets in the MTX file system.
ZoTSel	ZOS	-	ZOS(<ZO name>): Selection of a zero offset table.
ZOV	-	CPL	ZOV(<axis selection>[.<ZO bank>]): Provides the effective ZO values for an axis/machine coordinate.

Tab. 8-1: Alphanumerically arranged according to long form

8.2 ASCII character set

ASCII stands for "American Standard Code for Information Interchange". Characters 00 to 31 and character 127 are control signals

Dec.	Hex	ASCII
0	00	NUL
1	01	SOH
2	02	STX
3	03	ETX
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB

Appendix

Dec.	Hex	ASCII
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	SP
33	21	!
34	22	"
35	23	#
36	24	\$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=

Dec.	Hex	ASCII
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	y
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	↑ (^)
95	5F	→ (→)
96	60	'
97	61	a
98	62	b
99	63	c

Dec.	Hex	ASCII
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	y
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	
125	7D	}
126	7E	~
127	7F	DEL

Highlighted in yellow Characters skipped by default when read in.

Tab. 8-2: ASCII character set

8.3 Additional key codes

Key code (dec.)	Description
134	CURSOR UP
135	CURSOR DOWN
136	CURSOR RIGHT
137	CURSOR LEFT
139	PLANE RETURN

Key code (dec.)	Description
141	SOFTKEY1
142	SOFTKEY2
143	SOFTKEY3
144	SOFTKEY4
145	SOFTKEY5
146	SOFTKEY6
147	SOFTKEY7
148	SOFTKEY8

Tab. 8-3: Additional key codes

8.4 Optimizing the machining time

8.4.1 Measures directly influencing processing velocity

Introduction

These measures directly affect the processing velocity.

Path slope G8/G9 at exact stop OFF "G62"

Without the "path slope", (G9) the control performs a complete up and down slope (velocity ramp) to $v = 0$ at the start and end of a traversing block.

Using "path slope", (G8) the control attempts to generate a velocity that is as constant as possible being as high as of the programmed feed, even at the block transition. This reduces the processing time.

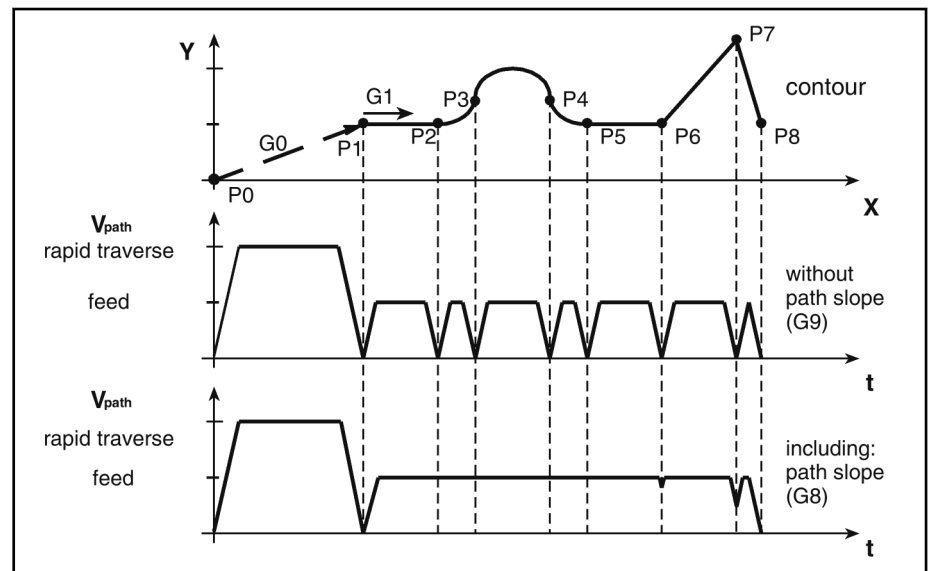


Fig. 8-1: Path slope

In order for G8 to be active, the exact stop has to be switched off (G62).

Inpos window at exact stop ON "G61"

"Exact stop ON" is only used where really required to wait for reaching the exact position.

Note that the more time is required at the block transitions to reach the inpos window, the smaller the inpos range and the lower the quality of control of the respective position control loop is set.

G61 (IPS1 IPS2 IPS3)	
with	
IPS1	Exact stop ON Wait for fine positioning window. At the end of the block, the control first reduces the path velocity to $v = 0$. Only when this positioning window was reached for all axes involved, the next block is traversed.
IPS2	Exact stop ON Wait for rough positioning window. "Inpos range 2 active" is displayed on the channel interface (also see the documentation "MTX PLC Interface"). At the end of the block, the control first reduces the path velocity to $v = 0$. Only when this positioning window was reached for all axes involved, the next block is traversed.
IPS3s	Exact stop ON Deceleration to $v = 0$ at end of block. At the end of the block, the control reduces the path velocity to $v = 0$. The next block is then traversed to a positioning window without being checked.

Tab. 8-4: Optional parameters G61

The latest activated positioning window for contour mode remains effective for "exact stop ON".

Inpos window at G0

G61 affects only motion in the feed. It has no effect on rapid traverse motions.

The inpos window with G0(NIPS) can be disabled for G0.

Continuous block transitions

Smooth, continuous transitions are passed faster than those with corners. Smoothed transitions can be achieved using the following functions:

- SCO (corner rounding with splines)
- RND (corner rounding with radius)

NC blocks

Generally, every programmed NC block is transmitted to the interpolator and requires a certain execution period, even though no traversing motion was programmed (at least one interpolation cycle).

It is not useful to program non-traversing information with traversing information in one block.

Example:

Reducing interpolation cycles

Instead of

N10 G91

N20 G1 F1000

N30 X100 Y50

is to be written

N10 G91 G1 X100 Y50 F1000

Sercos communication

Each ident number writing () requires processing time and causes the interpolation to decelerate to 0.

Freeform surface machining

The NC blocks for the freeform surfaces are generally provided with very short path lengths. To avoid an excessive reduction of the path velocity, it is recommended to enable the "Skip NC block" functionality using the machine parameters.

The parameters "Max. number of blocks for NC block buffer (706000120)" and "Perc. number of blocks for NC block buffer (706000130)" specify the maximum number of blocks to be skipped. The control averages the maximum velocity of each block via a window with a set block number. If there is at least one longer block with a higher maximum velocity within one window, this block increases the maximum velocities of the shorter neighboring blocks, but only if the limit values are not violated.

8.4.2 Measures indirectly influencing the processing velocity

Introduction

These measures only affect the processing velocity if the processing is executed at the block cycle time limit.

Short syntax

Each programmed character in the part program needs a certain runtime. Therefore, the so-called short syntax is better than the long form with regard to the block interpretation.

Example:

Short and long syntax

FFW < FeedForward

SCCT < SelCrdCoupleTab

Modal functions

Modal functions are only to be programmed when they change.

Whenever possible, the modification of the modal functions is to be programmed with the traversing information in one NC block.

Example:

Modal functions

N10 G90

N20 1 X50 F1000

N30 G90 G1 X70 Y20 F1000

N40 G90 G1 X100 Y50 F1000

N50 G91

N60 G1 X-100 F1000

N70 G91 G0 Y-50

Example:

```

Ideal modal functions
N10 G90 G1 X50 F1000
N20 X70 Y20
N30 X100 Y50
N40 G91 X-100
N50 G0 Y-50

```

Comments

- Longer comment sections are time-optimized and to be skipped using a GOTO command.

```

10 GOTO .KOPF
;=====
; Refer to the documentation prior to executing the
NC program
;
; ...
; PRG Header
;=====
20 .HEAD

```
- If single lines are to be labeled as comments, a ";" is to be used at the beginning of a line.

WAIT command

Generally, CPL sections can only be separated from the preceding DIN sections using "WAIT" if the CPL section accesses the current values of the interpolation (when querying current positions).

If only programmed values are read or the data is written in the CPL section, no additional "WAIT" is required since the CPL runs in a synchronized manner in the block preparation.

"WAIT" is also not necessary after the CPL section, if CPL data is written to be processed later on by a DIN section (e.g. tool corrections).

The following functions are provided with an internal WAIT functionality

ATBWD	Calibrate axis kinematics: Backward transformation
ATCAL	Calibrate axis kinematics: Optimize parameters
ATFWD	Calibrate axis kinematics: Forward transformation
ATGET	Calibrate axis kinematics: Read parameters from NC
ATPUT	Calibrate axis kinematics: Write parameters to NC
DCR(0)	Disabling distance control for digitization
EPC(0)	Disabling end position coupling
FSM	Move to fixed stop
FSP	Measuring at fixed stop
FSR	Resetting "move to fixed stop"
G74(HOME)	Approach reference point

G75, G75.1, G75.2	Measuring probe
HSB	Block switching with abort via high-speed signal
HWOC(0)	Disabling online correction in workpiece coordinates
LCP	Lag compensation
SCC(0)	Disabling selective, additive coordinate coupling
SCCT(0)	Disabling selective, additive coordinate coupling with table
SCPOWAIT	New spindle coupling offset enabled
SCWAIT	Waiting for synchronous spindle coupling
SPC_WAIT	Waiting for synchronous mode
SPCP_WAIT	Waiting for angular offset
TappRet1, TappRet2	Retract from tapping
TSP	Enabling tapping for individual spindles
VAP	Virtual drives

File access

Access to mounted files is always slower than access to data in the internal file system of the control. Store frequently called programs or cycles under "usr/user".

Waiting for permanent CPL variable

If waiting on a certain event, use the commands "WPV" and "SPV" first before checking in the inner of the CPL loop.

Example:

```
WPV, SPV
10 WPV[@1=1]
20 SPV[@2=3]
```

Version parameters for "MCODS"

If the version variable content is 0 at the MCODES call, the function provides the requested data immediately. Every data modification increments the version. If the MCODES is called with a version content $\neq 0$, the function waits for the following data modification.

Example:

```
MCODS call provides data immediately
10 VERSION=0
20 REPEAT
30 ERR = MCODES(32,1,VERSION,SAVSTATE%,1)
40 UNTIL SAVSTATE%=6
```

8.4.3 Processing delay due to faulty programming

Introduction

Special programming syntax can cause unexpected programming delay. Due to the time behavior of the programmed functions, an event can already be

processed even though the program waits for this event in a different channel. The program does not continue.

Applying a spindle as axis to channel

. Generally, when switching from "Spindle to axis" (STA) or vice versa (ATS), WAIT is to be programmed without parameter according to the respective command.

Example:

Applying spindle as axis

N10 STA(C)

N20 WAIT

N30 WAX(C)

Programming rotary axis via speed specification

A rotary axis is programmed via the speed specification and returned to the channel. WAIT has to be entered in front of the WAX.

Example:

Rotary axis with speed specification

N10 RAX(C1)

N20 C1=S(500)

N30 G4 F5

N40 C1=S(0)

N50 WAIT

N60 WAX(C1)

CPL loops

CPL loops aborted by an external event are to be provided with the instruction "WAIT(<WaitingTime>)". Thus, tasks with lower priorities (position display) are blocked.

Example:

CPL loop

10 REPEAT

20...

30 WAIT(,200)

40 UNTIL SD(9)=0

9 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts offer you advice and assistance should you have any queries. You can contact us **24/7**.

Service Germany

Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	+49 9352 40 5060
Fax:	+49 9352 18 4941
E-mail:	service.svc@boschrexroth.de
Internet:	http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide

Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information

To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

Index

Symbols

<Axis>=S<Value>.....	188
\$<ChannelNumber>	27

0 ... 9

3D tool radius correction.....	170

A

AAC	221
AAS	221
Abbreviations.....	7
About this documentation.....	1
Information representation.....	7
Validity of the documentation.....	1
ABS.....	74
Absolute dimension programming.....	155
Absolute value of the input value.....	74
ABSR0T2EUL.....	445, 446
AC(...).....	155
Access operands.....	486
Access the digital interface.....	484
Access to XML tables.....	545
ACN.....	324
ACOS.....	76
ACP.....	324
ACS.....	425
Activate C-axis mode for spindles and wait.....	381
Activate jerk limitation.....	281
Activate zero offset tables.....	415
Activating D-correction tables.....	250
Activating placement tables.....	317
Active functions	
NCF.....	487
ActRadFact.....	196
Additional conditions.....	17
Path information with additional condition.....	17
Program words.....	17
Address parameter list.....	39
Address names, unambiguity.....	39
Parameter addresses, permitted.....	40
Parameter values, permitted.....	40
Using several parameters.....	41
Using the same variable names.....	40
ADJ.....	197
ADR_PARA).....	40
ADSCLR.....	201
ADSDIS.....	201
ADSENA.....	200
ADSLAB.....	201
ADVA.....	202
AdvanceAppend.....	202
Advanced plane switching.....	127
Axis name assignment.....	128
AdvanceSubstitute.....	203
ADVS.....	203

AFC.....	228
AJK.....	231
AJS.....	231
Align spindle/position spindle.....	184
ALN.....	213
AND.....	76
Angular offset with active coupling.....	369
Angular offset, waiting for.....	370
anwperm.dat.....	64
APOS.....	426, 427
Apply axis.....	271
Apply axis, wait, if required.....	412
Apply spindle as axis to channel.....	632
Applying axis setting from MP.....	251
Approach axis position, machine-fixed.....	152
Approach reference point.....	150
Approach reference point coordinates	149
Approach to machine-fixed axis position.....	152
ARA.....	205
Area.....	205
Area monitoring.....	203
Activate, deactivate and release one or	
all monitoring areas at once.....	205
Dead areas.....	204
Define, activate, deactivate and release	
one monitoring area.....	205
Workspaces.....	204
ARF.....	196
ARRAY variable.....	66
Array variable (ARRAY).....	66
ASC.....	583
ASCII character set.....	623
ASIN.....	76
ASPCLR.....	209
ASPDIS.....	209
ASPENA.....	210
ASPRTP.....	210
ASPSET.....	211
ASPSTA.....	212
Assign logical axis name.....	212
Assign STRING expression.....	586
Assign STRING expression to a character	
array.....	586
AssLogName.....	213
ASTOPA.....	91
ASTOPO.....	91
Asynchronous axes	
Velocity.....	190
Asynchronous subroutine	
Triggered via program.....	212
Asynchronous subroutines	
Define point of return to contour.....	210
Define point of return to contour in asyn-	
chronous subroutine.....	346
Defining the repositioning behavior.....	346

Index

Log off.....	209	Spline parameter length.....	117
Log on.....	211	Spline point weight for checkpoints.....	117
Switch off.....	209	B-spline approximation.....	117
Switch on.....	210	Coordinate programming.....	117
Asynchronous subroutines: Reposition indi-		Parameter length.....	118
vidual coordinates.....	152	Tolerance value.....	118
AsynchrShapeOrder, ASO.....	121	B-spline approximation parameters.....	119
AsynchShapeOrder.....	122	Backing up the program starting point for re-	
ATAN.....	76	entry after program cancelation	341
ATAN2.....	76	Backward jump to a NC standard NC block	
ATBWD.....	213	GoBack (GOB).....	51
ATCAL.....	214	Backward transformation.....	213
ATFWD.....	216	BAP.....	119
ATGET.....	217	Basic CPL elements	
ATPUT.....	219	Code characters.....	78
ATR.....	407	Constants.....	78
ATrans.....	406	Basics of character string processing	
ATS.....	233	Character string processing.....	582
ATSW.....	233	Dimension.....	581
Automatic detection of corners and lines.....	241	Search in character string.....	581
c2-constant cubic spline.....	242	BC Type.....	119
Automatic gear stage selection.....	186	BCD.....	77
Automatic syntax conversion to an address		BCR.....	237
parameter list.....	43	BCS2WCS.....	448
AUXFUNC.....	220	BcsCorr.....	237
Available functions, NS coupling		BIN.....	77
MCODS, function overview.....	490	BITIF.....	484
AVE.....	235	Blank lines in the program code.....	28
AVS.....	235	BlkNmb.....	82
AxAcc.....	221	Block execution	
AxAccSave.....	221	CPL elements	13
AXADR\$.....	419	Block numbers.....	27
AXC.....	225	CPL blocks.....	27
AxCompTab.....	223	Standard NC blocks.....	27
AxCouple.....	225	Block switching on the fly via high-speed	
Axes and coordinates.....	417	signal.....	273
AXINF.....	421	Block switching on-the-fly via high-speed	
axis coupling		signal.....	274
constant coupling factor.....	225	Block switching with stop via high-speed sig-	
constant offset.....	224	nal.....	276
Axis coupling.....	224	Event-related stop of the traversing mo-	
Axis feed coupling.....	228	tion.....	276
Axis programming.....	112, 114	Block transition without deceleration.....	252
AxisFeedCouple.....	228	BNB.....	82
AxisToSpindle.....	233	BOOL variable.....	66
AxisToSpindleWait.....	233	BsaPar.....	119
AxJerk.....	231	BSTOPA.....	92
AxJerkSave.....	231	BSTOPO.....	92
AXO.....	441		
AXP.....	423	C	
AXR.....	234	C1- and C2-constant cubic spline.....	115
AxVel.....	235	Coordinates programming.....	115
AxVelSave.....	235	Spline parameter length.....	116
		Starting and end boundary conditions.....	116
B		Calibrate axis kinematics.....	213
B-spline (NURBS).....	116	Backward transformation.....	213
Coordinate programming.....	116	Read parameters from NC.....	217

Write parameters to the NC.....	218	CLOSE.....	567
Calibrating axis kinematics		CLRWARN.....	557
Forward transformation.....	216	CNB.....	238
Optimize parameters.....	214	Code characters.....	78
CALL command.....	38	COF.....	442
Canceling the stop condition.....	94	COFFS.....	172
OFFSTOPA.....	94	Collision.....	243
OFFSTOPO.....	94	Collision monitoring	242
CASE.....	56	Command words, reserved.....	70
CASE-LABEL...LABEL-OTHERWISE-END- CASE.....	56	Commands - NC program.....	17
Center point programming.....	107	Comments.....	630
CFD.....	292	Comments in a part program.....	28
CG0.....	244	Comments in the CPL block.....	28
Chamfer programming.....	239	Communicating with a client.....	595
Chamfer between two abutting circle seg- ments.....	239	Compare STRING expressions.....	583
Chamfer between two abutting straight lines.....	239	Compensation, volumetric	407
Chamfer length.....	239	Complaints.....	8
Length of the chamfer segment.....	239	Conditional jump to a standard NC block.....	51
Change character string		Conditional program abort.....	181
Convert upper-case and lower-case let- ters.....	592	Conditional program halt.....	181
Link STRING expressions.....	587	ConfG0.....	244
Change character strings		Configure G0 interpolation.....	244
Assign STRING expression.....	586	Configuring dynamics.....	404
Read characters in character string.....	591	Configuring retract data.....	404
Remove spaces.....	591	Constant cutting velocity.....	166
Change gear range.....	376	Constants.....	78
Change maximum path acceleration.....	316	Character string constants.....	78
Channel ID.....	27	Floating-point constants (DOUBLE).....	78
Channel number.....	27	Integer constants (INT).....	78
CHAR variable.....	67	String constant.....	78
Character string constant.....	78	ConstFeed.....	292
Character string processing		Continuous block transitions.....	628
Basics.....	581	Contour move programming.....	129
Change character strings.....	586	Contour shift, programmed.....	363
Compare STRING expressions.....	583	Contour transitions for milling cutter path correction.....	137
Type conversion.....	583	G43 - Arc.....	137
ChkNxtBlk.....	238	G44 - Intersection.....	138
CHL.....	240	Control data.....	453
ChLength.....	240	Active functions.....	487
CHLL.....	240	Active modal auxiliary functions.....	487
CHR\$.....	584	Control version.....	453
CHS.....	240	Error and warnings.....	555
ChSection.....	240	General XML tables.....	545
CHSL.....	240	Machine parameters.....	456
Circle entry, tangential.....	110	PLC interface.....	484
Circular interpolation.....	104	Sercos interface.....	479
Circular/helical/helical-N interpolation.....	104	Simple system data.....	456
Optional parameters.....	108	Time recording.....	453
Radius programming.....	106	Tool database.....	546
Circular/Helical/Helical-N interpolation		Control data and processes	
Center point programming.....	107	NC interface (NCS).....	488
CLD.....	241	Control version	
CLN.....	243	VERSINF\$.....	453
CLOCK.....	453	Conversion between numerical systems.....	77
		BCD format.....	77
		BCD-coded numbers in binary value	77
		BIN format.....	77

Index

Binary value in BCD format.....	77	CPL instruction: FOR-STEP-TO-NEXT.....	58
Coord.....	246	CPL instruction: REPEAT-UNTIL.....	59
Coordinate and axis positions		CPL instruction: WHILE-DO-END.....	59
Axis command position.....	425	CPL variable ERRNO.....	69
Read actual axis values.....	426	CRD	246
Read command position of a system axis....	429	Create channel spindles.....	272
Read machine coordinates.....	427	Criticism.....	8
Read program coordinates.....	428	CST.....	248
Read workpiece position.....	430	Curve Parameter Interpolation.....	249
Coordinate and axis values		CurveParamInterpolation.....	250
Measuring units.....	418	Customer Feedback.....	8
Coordinate programming.....	114, 116	Customer-specific velocity profile.....	250
Coordinates programming.....	112, 115	Cutter center point - G46.....	139
CornerLineDetection.....	241	Cutter contact point - G45	139
Correction of tool length	139	Cutting velocity, constant.....	166
Correction switching OFF.....	154		
Correction switching ON.....	154	D	
COS.....	76	D.....	193
CoupleSplineTab.....	248	D-correction.....	192
Coupling distance slave spindle.....	368	DATE.....	454
CPL command: CASE-LABEL...LABEL- OTHERWISE-ENDCASE.....	56	DAX.....	251
CPL commands.....	72, 417, 441	DBLOAD.....	546, 547
Axes and coordinates.....	417	DBMOVE.....	547
Axis and coordinate addresses.....	419	DBSAVE.....	548
Axis information.....	421	DBSEA.....	549
Character string processing.....	581	DBSEAX.....	550, 551
Communicate with client.....	595	DBTAB.....	553
Control data and processes.....	453	DBTABX.....	553, 554
Conversion between numerical systems.....	77	DBTABXL.....	554, 555
Coordinate and axis positions.....	424	DC.....	324
Files and directories.....	560	DCB.....	257
Input helps.....	448	DcBreak.....	257
Logic operations.....	76	DCC.....	257
Mathematical operations.....	73	DcCont.....	257
Read measured values.....	431	DCOFFS.....	172
Relational operations.....	77	DCR.....	256
Spindle addresses.....	434	DCS.....	250
Spindles.....	434	DCT.....	450, 451
Tool corrections.....	449	DcTSel.....	250
Value assignments.....	72	Decision and branch instructions.....	55
Variable axis address.....	423	CPL command: CASE-LABEL...LABEL- OTHERWISE-ENDCASE.....	56
Workpiece position correction place- ments (inclined plane).....	442	CPL instruction: IF-THEN-ELSE-ENDIF.....	57
Zero point offsets.....	436	Skip block.....	55
CPL elements.....	13	Default axis configuration.....	251
CPL instruction: FOR-STEP-TO-NEXT.....	58	DefAxis.....	251
CPL instruction: IF-THEN-ELSE-ENDIF.....	57	Definable permanent variables.....	62
CPL instruction: REPEAT-UNTIL.....	59	Define (activate) a coupling group.....	366
CPL instruction: WHILE-DO-END.....	59	Define point of return to contour in asynchro- nous subroutine.....	346
CPL jump to any program block.....	53	Define pole.....	318
CPL language elements.....	14	Define reflection point/rotation point.....	322
CPL loops.....	632	Define spline type.....	385
CPL parameter list.....	42	Define/remove spindle groups.....	371
CPL program portions.....	13	Defining the repositioning behavior.....	346
CPL programming.....	12	DefSpindle.....	251
CPL repeat commands.....	58	DefTangTrans.....	252
CPL repeat instructions			

Delayed response to drive errors.....	254	ED compensation	193
Deleting a bit event.....	85	Edit file	
Denominator polynomial programming.....	113	Determine file pointer position.....	576
DERDelay.....	254	File end ID.....	578
Determine file size		Position file pointer.....	574
FILESIZE and random file.....	580	Read.....	572
FILESIZE and sequential file.....	580	Write.....	569
Determining system axes to specify the optimum SHAPE filter in case of enabled jerk limitation.....	280	Editing file	
Determining the spindle syntax for the block pre-run adjustment program	365	Format.....	572
DIA.....	253	Enable C-axis mode for spindles.....	380
Diagnostics of velocity control.....	409	Enable retraction.....	405
Diameter programming.....	252	Enable/apply reserved spindle.....	366, 375
DiaProg.....	253	End position coupling.....	258
Digitalization		EndPosCouple.....	258
Distance control.....	256	EOF.....	578
DIM.....	581	EPC.....	258
dimension of the working plane.....	255	ERASE.....	568
Dimension of the working plane.....	254, 255	ERRNO variable.....	69
DIN language elements.....	14	Error and error categories "GETERR"	
DIN programming.....	12	Query last part program error in channel 2.....	559
DIRCR.....	564	Errors and error categories "GETERR"	
DIRDEL.....	564	Result analysis for 5 array elements.....	559
Direct speed programming.....	166	Errors and warnings	
Directories		Delete warnings.....	557
Create.....	564	Query.....	558
Delete.....	564	Runtime error.....	555
Directory information.....	562	Runtime warning.....	556
DIRINF.....	562	Event-related feed.....	274
Disable management of spindle motion for specific channels.....	375	Exact stop ON.....	627
Disabling kinematic NC block splitting.....	283	Exact stop ON/OFF.....	145
Disabling the velocity limit due to the sum of radial and tangential acceleration	283	Execute active auxiliary functions of all groups.....	220
Distance control for digitalization	256	Execution of block	
DistCtrl.....	256	DIN - CPL language parts.....	14
Dividing the traversing block: Length of path segment.....	286	Exit program.....	181
Dividing the traversing block: Number of path segments	305	Exiting diagnostics recording.....	257
DN	113	Exponential function.....	74
Documentation		F	
Change record.....	1	F.....	189
DOUBLE variable.....	66	F-address.....	189
DPC.....	443	FA.....	190
DREC_START.....	258	FAD.....	260
DREC_STOP.....	258	FDG.....	263
Drives, virtual.....	409	FDGT.....	264
DSP.....	251	Feed.....	102
DTT.....	252	Feed 100%.....	315
Dwell time.....	109	Feed and speed programming.....	187
E		F-address.....	189
EBC.....	118	F2-address F2.....	189
ED.....	194	Velocity of asynchronous axes.....	190
		Feed computation: Feed group.....	263
		Feed computation: Feed group type.....	264
		Feed computation: Hide axes.....	259
		Feed correction.....	139
		Cutter center point G46.....	139

Index

Cutter contact point G45.....	139	FSP.....	267
Feed forward.....	260, 262	FsProbe.....	267
Feed programming (per min).....	157	FSR.....	269
Feed programming per revolution.....	163	FsReset.....	269
Feed programming per rotary axis revolution...	166	FST.....	269
FeedAd.....	260	FsTorque.....	269
Feedback.....	8		
FeedForward.....	261	G	
FeedGrp.....	263	G codes	
FeedGrpType.....	264	C1- and C2-constant cubic spline.....	115
FFW.....	261	G-codes.....	101
FFWA.....	262	3D tool radius correction.....	170
File access.....	631	Absolute velocity programming with ac-	
File access rights.....	11	celeration adaptation.....	162
File operation		Absolute velocity programming with path-	
Copy.....	569	proportional velocity adaptation.....	158
File operations		Absolute velocity programming with time-	
Close.....	567	proportional velocity adaptation.....	159
Delete.....	568	Advanced plane switching.....	127
Logic file number.....	564	Approach reference point coordinates.....	149
Open.....	565	Asynchronous subroutines: Reposition	
Overwrite.....	567	individual coordinates.....	152
File properties		AsynchrShapeOrder.....	121
Access rights.....	578	B-spline approximation parameters.....	119
Date.....	579	B-splines (NURBS) (spline type 3).....	116
File size.....	579	Circular/helical/helical-N interpolation.....	104
File system and file protection.....	11	Compensation switching.....	153
FILEACCESS.....	578	Constant cutting velocity.....	166
FILECOPY.....	569	Contour move programming.....	129
FILEDATE.....	579	Contour transitions for milling cutter path	
FILENO.....	564	correction.....	137
FILEPOS.....	576	Direct speed programming.....	166
Files and directories.....	560	Dwell time.....	109
Directories.....	562	Exact stop ON/OFF	145
Edit file.....	569	Feed correction.....	139
File names.....	561	Feed programming.....	157
File operations.....	564	Feed programming per revolution.....	163
File properties.....	578	Feed programming per rotary axis revolu-	
Random file structure.....	561	tion.....	166
Sequential file structure.....	561	Free plane switching.....	128
Files names.....	561	Hermite spline.....	114
FILESIZE.....	579, 580	Inch programming.....	148
Floating point constant (DOUBLE).....	78	Incremental velocity programming with	
Floating point variable (DOUBLE).....	66	acceleration adjustment.....	164
Floating point variable (REAL).....	66	Jerk-limited velocity control.....	121
Flying measurement.....	265	Linear interpolation in the feed.....	102
Initialization.....	278	Local absolute dimension programming.....	155
FlyMeas.....	265	Local relative dimension programming.....	155
FME.....	265	Metric programming.....	149
Following error.....	145	Milling cutter path correction.....	135
FOR-STEP-TO-NEXT.....	58	No plane.....	125
Forward jump to a standard NC block.....	50	Path slope OFF.....	120
Forward transformation.....	216	Path slope ON.....	120
Free selection of planes (independent of the		Placement: Inclined plane.....	173
WCS).....	128	Plane switching.....	126
Freeform surface machining.....	629	Programmable zero point offset.....	141
FSM.....	269	Relative dimension programming.....	155
FsMove.....	269		

Spline programming.....	111	G54.5.....	143
Spline with coefficient programming.....	112	G55.1.....	143
Starting and end boundary conditions.....	118	G55.5.....	143
Straight line interpolation in rapid tra- verse.....	101	G56.1.....	143
Tangential circle entry.....	110	G56.5.....	143
Tangential circle exit.....	110	G57.1.....	143
Tapping without compensation chuck.....	147	G57.5.....	143
Thread cutting.....	131	G58.1.....	143
Time programming.....	156	G58.5.....	143
Tool length correction.....	139	G59.1.....	143
Traveling to probe.....	150	G59.5.....	143
Traverse to fixed machine axis position.....	152	G61.....	146, 628
Traverse to reference point.....	150	G62.....	146, 627
G-Codes		G63.....	147
Incremental velocity programming with acceleration adaptation	160	G63.2.....	147
G0.....	101	G70.....	148
G0 interpolation configuration.....	244	G71.....	149
G1.....	102, 103	G74.....	149
G2 circular path in clockwise rotation.....	104	G74(HOME).....	150
G3 circular path in clockwise rotation.....	104	G75.....	150
G4.....	109	G76.....	152
G5.....	111	G77.....	152
G6.....	112	G78.....	154
G8.....	121	G79.....	154
G8(SHAPE...).....	123	G90.....	155
G9.....	121	G91.....	155
G9(ASHAPE...).....	123	G93.....	156
G9(SHAPE...).....	123	G94.....	157
G9(X... , Y... , ...).....	121	G94 (F-value/S-value).....	160
G12.....	110	G94 (LINS).....	158
G16.....	126	G94 (LINT).....	159
G17.....	126	G94 (TF value/TS value).....	162
G17(...).....	128	G95.....	163
G18.....	126	G95(F-value/S-value)	164
G18(...).....	128	G95(FRA=<Axis>).....	166
G19.....	126	G96.....	166
G19(...).....	128	G97.....	166
G20.....	129	G140.....	172
G33.....	132	G141.....	172
G40.....	136	G142.....	172
G41.....	136	G151.....	177
G42.....	137	G152.....	178
G43.....	138	G153.....	177
G43 - Arc.....	137	G153 - G159.....	179
G44.....	138	GAX.....	271
G44 - Intersection.....	138	GCT.....	271
G45.....	139	GCT(0).....	271
G46.....	139	GCT(1).....	271
G47.....	140	Gear idle selection.....	187
G48.....	140	Gear stage selection	
G52.....	141	Automatic.....	186
G53. bis G59.....	143	Manual.....	186
G53.1.....	143	General language syntax	
G53.5.....	143	Programming effective radius factors Ac- tRadFact, ARF.....	196
G54.1.....	143	General XML tables	
G54.1 to G59.1.....	144	Access.....	545
		Create.....	545

Index

Generating a positive random integer number....	74
GeoComp.....	271
GetAxis.....	271
GETERR.....	558
GetSpindle.....	272
Global variables.....	62
GMSG.....	29
GOA.....	50
GoAhead.....	50
GOB.....	51
GoBack.....	51
GOC.....	51
GoCond.....	51
GOTO.....	53
GOTO command.....	630
GSP.....	272

H

Helical interpolation.....	104
Helical-N interpolation.....	105
Helpdesk.....	633
Hermite spline.....	114
Axis programming.....	114
Coordinate programming.....	114
Programming spline parameter lengths.....	115
User-defined velocity profile.....	115
High-level language syntax.....	195
High-speed output	
Position-dependent.....	322
High-speed signal, block switching on-the-fly.....	273
High-speed signal, block switching with stop....	276
Hotline.....	633
HSB.....	273
HSB(..,HSSTOP=..)	276
HsBlkSwitch.....	273
HsBlkSwitch(..,HSSTOP=..)	276
HWOC.....	277
HWOCDIS.....	277

I

IC(...)	155
IF-THEN-ELSE-ENDIF.....	57
IME.....	278
INCH.....	148
Inch programming.....	148
Inclined plane.....	173
Influence channel interface signal.....	405
Information representation	
Names and abbreviations.....	7
Safety instructions.....	7
Symbols used.....	7
Initialization "Flying measurement".....	278
InitMeas.....	278
INP#.....	572
Inpos window at exact stop ON G61.....	627
Inpos window at G0.....	628

Input helps

SCL.....	448
Input tool: Mirroring.....	294
Mirroring an orientation vector.....	296
Input tool: Programmed contour shift.....	363
Input tool: Rotate.....	351
Input tool: Rotating	
Rotating an orientation vector.....	352
Input tool: Scaling.....	357
INSDEP.....	172
INSTR.....	581, 582
Instructions - NC program	
Path function with coordinate specification....	17
Path functions.....	17
INT.....	74
Integer constant (INT).....	78
Integer variable.....	66
Integer variable (INT).....	66
Intended use	
Areas of application.....	9
Introduction.....	9
Use cases.....	9
Interrupt program.....	180
InvTime.....	279
IPS1.....	280
IPS2.....	280
IPS3.....	280
ITIM.....	279

J

JAL.....	280
Jerk.....	125
Jerk-limited velocity control.....	121
Relation SHAPE order - jerk.....	125
Resulting SHAPE order.....	124
SHAPE for asynchronous axes.....	122
SHAPE for path mode.....	123
SHAPE for positioning mode.....	123
JerkAxList.....	280
JerkControl.....	282
JKC.....	282
Jog mode in workpiece coordinates.....	282
JogWCSSelect.....	282
Jump instructions.....	48
Jumps in program run	
Decision/branch instructions.....	31
Jump instructions.....	30
Repeat instructions.....	31
Subroutine calls.....	30
Jumps in Program Run.....	30

K

Key codes.....	626
Keywords.....	70
KinLimAcc.....	283
KinSplitDisable.....	283
KV programming.....	284

KVP.....	284
KvProg.....	284

L

Label programming	
In a CPL block (DIN).....	50
In a standard NC block (DIN).....	49
Label programming and jump instructions.....	48
Backward jump to a standard NC block	
GoBack (GOB).....	51
Conditional jump to a standard NC block	
GoCond (GOC).....	51
CPL jump (GOTO) to any program block.....	53
Forward jump to a standard NC block	
GoAhead (GOA).....	50
Labels in standard and CPL blocks.....	49
Subroutine return to any program block-	
RetTo.....	54
Unconditional jump to a standard NC	
block GoTo.....	52
Labels in CPL blocks.....	50
Labels in standard NC blocks.....	49
Lag compensation.....	285
LagCompensation.....	285
Laser power control	
Path velocity-dependent.....	287
LCP.....	285
LCTS.....	290
LEN.....	286, 582
LFC.....	288
LFConf.....	288
LFP.....	288
Limit maximum speed.....	376
Limit minimum speed.....	377
Limitation of speed.....	363
LinDownFeed.....	292
Linear interpolation.....	102
Linear interpolation in rapid traverse	
Optional parameters.....	101
Linear interpolation in the feed.....	102
Basic function.....	103
Optional parameters.....	103
Linear orientation motion with axis program-	
ming.....	306
Linear orientation motion with coordinate	
programming.....	308
Linear to spline switching.....	289
Linear traversing motions in the tool coordi-	
nate system TCS.....	393
Linear-circular-spline-conversion.....	289
LinearToSpline.....	289
Link STRING expressions.....	587
Link also in CPL commands.....	587
Link texts using STRING expressions.....	588
MID\$ command with read-only access.....	589
MID\$ command with write access.....	590
Linking	

Link table.....	14
Linking with DIN/CPL ID.....	16
Linking, NC programs.....	14
Linking in the background.....	15
Linking with DIN/CPL ID.....	16
Linking, subroutines.....	16
Linking, subroutines.....	16
Postlinking.....	16
LinModZp.....	290
LinUpFeed.....	292
LJUST.....	572
LJUSTi.....	572
LMZ.....	290
LND.....	292
LNU.....	292
local.....	26
Local absolute dimension programming.....	155
Local inch programming.....	148
Local relative dimension programming.....	155
Local subroutines.....	34
Local variables.....	61
Logic operations	
AND function AND.....	76
EXCLUSIVE OR function XOR.....	77
NOT function NOT.....	76
OR function OR.....	76
Logic variable (BOOL).....	66
Logical operations.....	76
LTS.....	289

M

M-codes.....	180
Automatic gear selection.....	186
Conditional program abort.....	181
Exit program.....	181
Gear idle selection.....	187
Interrupt program.....	180
Manual gear selection.....	186
Modulo speed setting.....	187
Omega address (feed).....	191
Orient/position spindle.....	184
Program spindle speed.....	191
Speed setting.....	187
Spindle clockwise rotation.....	182
Spindle clockwise rotation and coolant	
ON.....	182
Spindle counterclockwise rotation.....	183
Spindle counterclockwise rotation and	
coolant ON.....	183
Spindle stop.....	184
M0.....	181
M00.....	181
M1.....	181
M01.....	181
M2.....	181
M02.....	181
M3.....	182

Index

M4.....	183	Simple functions.....	73
M5.....	184	SIN.....	76
M13.....	182	Skipping the digits after the decimal point....	74
M14.....	183	SQRT.....	75
M19.....	185	Square root of an input value.....	75
M30.....	181	TAN.....	76
M40.....	186	Trigonometric function.....	75, 76
M41...44.....	186	Maximum axis acceleration	
M48.....	187	Buffer.....	221
M103.....	182	Change.....	221
M104.....	183	Maximum axis jerk	
M105.....	184	Buffer.....	231
M113.....	182	Change.....	231
M114.....	183	Maximum axis velocity	
M119.....	185	Buffer.....	235
M140.....	186	Change.....	235
M141...144.....	186	Maximum radial acceleration.....	337
M148.....	187	MCA.....	456
M203.....	182	MCASE\$.....	592
M204.....	183	MCODS.....	488, 489
M205.....	184	Active additive program coordinate offset....	523
M213.....	182	Active channel axis names.....	509
M214.....	183	Active coordinate mirroring.....	525
M219.....	185	Active offset coordinates.....	525
M240.....	186	Active path feed.....	528
M241...244.....	186	Active placement data.....	522
M248.....	187	Active program coordinate offset.....	523
machdef.dat.....	65	Active radius correction.....	496
Machine parameters		Active scaling factors.....	525
MCA.....	456	Active syntax of auxiliary function groups....	518
Machining velocity, direct influence		Active tool correction.....	508
Continuous block transitions.....	628	Active zero point offset data.....	519
Freeform surface machining.....	629	Active ZO values.....	496
Inpos window at G0.....	628	Actual axis position of the CNC:.....	529
NC blocks.....	628	Actual axis position of the drives:.....	502
Sercos communication.....	629	Actual axis values in WCS.....	503
Macro programming.....	47	Actual spindle position.....	521
with parameters.....	47	Actual spindle torque.....	522
Macros.....	47	Angle of rotation.....	524
Macros with parameters.....	47	Automatic program reselection.....	513
Main spindle switching.....	294	Auxiliary function requiring an acknowl-	
MainSp.....	294	edgment active.....	503
Manual gear stage selection.....	186	Axis assignment: Channel.....	504
Mathematical operation		Axis command position.....	493
Exponential function.....	74	Axis command position (integer).....	494
Mathematical operations.....	73	Axis coordinate end points.....	519
ABS.....	74	Axis coordinates in ACS0.....	521
Absolute value of the input value.....	74	Axis coordinates in ACS1.....	514
ACOS.....	76	Axis mode.....	506
ASIN.....	76	Axis names.....	502
ATAN.....	76	Axis potentiometer.....	507
ATAN2.....	76	Basic coordinates of tool tip (BCS-Tcp).....	518
COS.....	76	Basic coordinates, actual values.....	514
EXP.....	74	Basic coordinates, command values.....	513
INT.....	74	Block pre-run adjustment status.....	526
RANDOM.....	74	Block transfer enabled.....	504
ROUND.....	75	Channel mode.....	500
Rounding up and down.....	75	Channel reset.....	526

Channel spindle: Group assignment.....	512	Programmed coordinate end points.....	514
Channel waiting states.....	516	Programmed coordinate values.....	503
Control device type.....	508	Programmed end position.....	498
Coordinate end points.....	514	Programmed path feed.....	498
Coordinate names.....	515	Programmed path feeds.....	528
Current axis speed.....	524	Programmed spindle speed.....	498
Current axis velocity.....	524	Programming example.....	529
Current compensation value.....	527	Radius correction number.....	496
Current cross compensation.....	527	Reference for coordinates.....	515
Current lead screw compensation.....	527	Reference point approached.....	499
Current temperature compensation.....	527	Return to contour.....	505
Customer-specific data.....	504	SAV mode.....	501
Default axis assignment: Channel.....	509	Skip block.....	513
Default setting of channel axis names.....	510	Spindle axis number.....	520
Diameter programming.....	518	Spindle gear range.....	511
Distance to go of axes in ACS.....	516	Spindle list speed.....	502
Distance to go of coordinates in WCS.....	516	Spindle motion function.....	511
Dwell time active.....	503	Spindle names.....	520
End position including input helps.....	497	Spindle performance.....	522
Feed inhibit.....	507	Spindle potentiometer.....	495
Feed potentiometer.....	495	Spindle speed.....	495
Gear switching, automatic/manual.....	512	Spindle system number.....	520
Group numbers of auxiliary functions.....	518	Spindles: Gear switching.....	512
Ident number from cyclic axis telegram.....	511	Status of block pre-run re-entry mark.....	525
Inch/metric.....	497	Status of online correction in WCS.....	517
InPos.....	495	Test mode.....	499
INPOS status for coordinates.....	515	Total of active program coordinate offset.....	524
IPO mode.....	501	Total of placements.....	523
Jog velocities.....	499	Total of zero point offsets.....	519
Lag.....	494	Travel command.....	505
Machine coordinate names.....	521	Workpiece coordinates.....	513
Machine coordinates.....	514	Workpiece position correction.....	523
Main program: Name and path.....	499	MCODS call.....	631
Manufacturer version.....	508	MCOPS.....	530
Maximum axis velocity.....	526	Cancel distance to go.....	530
Maximum spindle speed.....	497	Change return to contour recording.....	540
Measuring unit for coordinates.....	515	Change return to contour strategy.....	540
Measuring unit of axes.....	507	Channel restart.....	540
Measuring unit of axes (default setting).....	510	Program deselection.....	537
Minimum spindle speed.....	498	Program start.....	538
Motor type.....	509	Reset.....	531
Name of tool correction table.....	496	Search block.....	532
Name of ZO table.....	496	Select manual data input.....	533
Notes (messages).....	499	Select program.....	533
Number of axes.....	505	Specify operation mode.....	538
Number of channels.....	504	MCS.....	427
Number of coordinates.....	516	Measurement at fixed stop.....	266
Number of machine coordinates.....	521	Measurement unit	
Number of spindles.....	520	Inch.....	148
Online correction in WCS.....	517	Measuring unit	
Optional stop.....	513	Metric.....	149
Path and name of active placement table.....	522	Measuring units	
Path feed.....	494	Provided axis and coordinate positions	418
Peak value and maximum value of axis acceleration.....	529	Metric programming.....	149
Peak value and maximum value of axis velocity.....	528	MID\$.....	591
Programmed axis speed.....	524	Milling cutter path correction.....	135
		Contour transitions.....	137
		MIR.....	295, 297

Index

mirror.....	295	Retrace forward program block extension...	202
Mirroring.....	294	NC functions - Basics.....	11
MMC.....	595	NC functions - overview.....	597
MOB.....	297	NC functions with general language syntax	
MOC.....	298	Activate jerk limitation.....	281
MOD.....	301	Activate zero offset tables.....	415
modal - program words		Activating D-correction tables.....	250
self-sustaining.....	26	Activating placement tables.....	317
Modal - Program words.....	25	Additive program coordinate offset.....	405
Modal function groups - overview.....	97	Adjust axis configuration TargetAxisConf, TAX.....	400
Modal functions.....	629	Adjust modal NC functions.....	197
Modal subroutines.....	43	Adjust spindles in the block pre-run init adjustment program.....	364
Modulo.....	301	Angular offset with active coupling.....	369
Modulo axis, zeroing.....	290	Apply axis.....	271
Modulo calculation for incrementally traveling endless axes.....	301	Apply axis, wait, if required.....	412
For all endless axes of the channel.....	301	Applying axis settings from MP.....	251
For individual endless axes of the channel.....	301	Area monitoring.....	203
Modulo range.....	301	Assign logical axis name.....	212
Modulo offset for endless axis.....	302	Asynchronous subroutines.....	209, 346
Modulo velocity specification.....	187	Automatic detection of corners and lines.....	241
Motion information.....	11	Axis coupling.....	224
Motion programming.....	377	Axis feed coupling.....	228
Motion stop until axis position is reached.....	90	B-spline approximation.....	236
ASTOPA.....	90	Block repetition.....	345
ASTOPO.....	91	Block transition with stop via high-speed signal.....	276
Motion stop until basic workpiece position is reached.....	91	Block transition without deceleration.....	252
BSTOPA.....	92	Chamfer programming.....	239
BSTOPO.....	92	Change maximum axis acceleration.....	221
Motion stop until workpiece position is reached.....	93	Change maximum axis jerk.....	231
WSTOPA.....	93	Change maximum axis velocity.....	235
WSTOPO.....	93	Change maximum path acceleration.....	316
Move to fixed stop.....	268	Change touch probe edge configuration.....	302
Moving to fixed stop		CO programming.....	284
Moving to fixed stop.....	268	Collision monitoring.....	242
Reset fixed stop.....	269	Create channel spindle.....	272
Torque reduction fixed stop.....	268	Curve parameter interpolation.....	249
MPEDGE.....	302	Define (activate) coupling group.....	366
MSG.....	29	Define reflection/rotation point.....	322
MSP.....	294	Define spline type.....	385
		Define/remove spindle groups.....	371
N		Delayed response to drive errors.....	254
NC block synchronization.....	78	Determining system axes to specify the optimum SHAPE filter in case of enabled jerk limitation.....	280
active point in time.....	79	Diagnostics of velocity control.....	409
Block preparation.....	79	Diameter programming.....	252
Synchronization function of block preparation.....	79, 82	Dimension of the working plane.....	254
Write system date at execution time.....	94	Disabling kinematic NC block splitting.....	283
NC blocks.....	628	Distance control for digitalization.....	256
NC function with general language syntax		Enable C-axis mode for spindles.....	380
Determining the spindle syntax for block pre-run adjustment program.....	365	Enable C-axis mode for spindles and wait.....	381
System spindle programming.....	375	Enable/transfer reserved spindle.....	366
NC function with high-level language syntax		End position coupling.....	258
		Exiting diagnostics recording.....	257

Feed 100%.....	315	Selected additional coordinate coupling.....	359
Feed computation: Feed group type.....	264	Selecting axis transformation.....	245
Feed forward.....	260	Setting the program position	362
Flying block transition via high-speed signal.....	273	Special functions for thread cutting.....	404
Flying measurement.....	265	Speed limitation.....	363
G0 interpolation configuration.....	244	Spindle selection for tapping without compensation chuck.....	399
Initialization "Flying measurement".....	278	Spline coupling table.....	247
Inverse time programming.....	279	Starting diagnostics recording.....	257
Jog mode in workpiece coordinates.....	282	Stroke release time.....	329
Lag compensation.....	285	Suppress single step modes.....	390
Linear to spline switching.....	289	Switch off C-axis mode for spindles.....	233
Loading axis error correction table.....	222	Switch off C-axis mode for spindles and wait.....	233
main spindle switching.....	294	Synchronization window.....	368
Maximum radial acceleration.....	337	Synchronize system axis coupling.....	355
Measurement at fixed stop.....	266	Synchronous window.....	370
Measuring on a contour.....	298	Tangential tool control.....	393
Modulo calculation for incrementally traveling endless axes.....	301	TCS definition in program coordinates.....	401
Modulo offset for endless axes.....	302	Torque reduction.....	339
Moving to fixed stop.....	268	Travel in TCS.....	393
Next block monitoring with buffered NC block input.....	238	Velocity profiles (ramp function).....	291
Nibbling.....	303	Wait for spindle programming at end of block.....	382
Online correction in workpiece coordinates.....	277	Waiting for angular offset.....	370
Orientation Programming.....	305	Waiting for synchronous mode.....	371
Parameterizing static tool orientation.....	388	Writing into a system data queue.....	372
Path velocity-dependent laser power control.....	287	Writing Sercos parameters.....	413
Position-dependent high-speed output.....	322	Zero modulo axis (linear endless axis).....	290
Positioning type for endless axes.....	324	NC functions with high level language syntax	
Precision programming.....	326	Programmable distance-dependent signal.....	200
Program continuation with block pre-run after program abort.....	340	NC functions with high-level language syntax	
Program coordinate offset.....	405	Additional function to orient/position spindles.....	380
Program processing depending on block preparation.....	327	Backing up the program starting point for re-entry after program cancelation	341
Programmable path split.....	386	Calibrate axis kinematics.....	213
Programming polar coordinates.....	318	Coupling distance of slave spindle.....	368
PTP traversing motion.....	333	Customer-specific velocity profile.....	250
Punching.....	335	Disabling the velocity limit due to the sum of radial and tangential acceleration	283
Radius programming.....	252	Divide the traversing block: Number of path segments.....	305
Recording the reverse vector.....	338	Dividing traversing block: Length of path segments.....	286
Remove (deactivate) the coupling group.....	366	Execute active auxiliary functions of all groups.....	220
Remove axis from axis group.....	343	Feed computation: Hide axes.....	259
Remove channel spindles.....	345	Feed computing: Feed group.....	263
Remove logical axis name.....	344	Feed forward asynchronous axes.....	262
Replace axes.....	234	Input tool: Mirroring.....	294
Resetting channel spindles to MP.....	251	Input tool: Rotate.....	351
Retract from tapping.....	399	Input tool: Scaling.....	357
Round corners with splines.....	383	Linear-circular-spline-conversion.....	289
Rounding of corners.....	353	Measuring on a block.....	297
Save maximum axis acceleration.....	221	Placement: Workpiece position correction.....	236
Save maximum axis jerk.....	231		
Save maximum axis velocity.....	235		
Select a waiting strategy	382		
Select exact stop window.....	279		

Index

Programmed contour shift.....	363	non self-sustaining.....	26
Retrace backward program extension.....	350	non self-sustaining.....	26
Retrace backward program replacement.....	351	Non-modal - Program words.....	26
Retrace forward program replacement.....	203	NOT.....	76
Retrace switching lock LockRetrace, LOCKRETRACE	349	Notes in the user interface.....	29
Retrace, RETRACE.....	347	channel-specific.....	29
Sharpening internal corners	246	Notes on the user interface	
Spindle: Switching between position and speed interface.....	387	Channel-independent.....	29
Straightness and angle error compensa- tion.....	271	NUL.....	73
Suppressing the collision monitoring.....	361	NUM.....	305
Tangential tool orientation.....	396	NURBS.....	116
Tool-specific spindle limitations.....	379		
Tool-specific spindle limits.....	392	O	
Virtual drives.....	409	O().....	311
NC functions with syntax according to DIN 66025.....	99	OFFSTOPA	94
Feed and speed programming.....	187	OFFSTOPO	94
G-codes.....	101	Omega.....	191
M-codes.....	180	Omega address (feed).....	191
Tool correction.....	192	Online correction in workpiece coordinates.....	277
NC interface (NCS)		Open file	
Data services.....	488	OPENR.....	565
Process services.....	530	OPENW.....	565
Programming examples.....	543	OPENR.....	565
NC program		OPENW	565
Additional conditions.....	17	Operations, logical.....	76
Basic components.....	16	Optimize machining time.....	627
Commands.....	17	Optional parameters	
Program block.....	16	Circular/helical/helical-N interpolation.....	108
NC programming.....	12	Linear interpolation in rapid traverse.....	101
NC-functions with high-level language syntax		Linear interpolation in the feed.....	103
Volumetric compensation.....	407	OR.....	76
NCF.....	487	Orientation programming.....	305, 306
NCS coupling		Linear orientation motion with axis pro- gramming.....	306
Error return values.....	541	Linear orientation motion with coordinate programming.....	308
Programming example - Axis channel as- signment.....	544	Tensor orientation.....	312
Programming example - Channel/waiting state.....	544	Vector orientation.....	309
Programming example - Output axis names in the MSG window.....	544	Orientation vector programming.....	113
Programming example - Wait until SAV state of channel 2 switches to inactive.....	543	OVD.....	315
Programming examples - Request SAV state and interpolator of channel 2 imme- diately.....	543	OVE.....	316
Next block monitoring with buffered NC		OvrDis.....	315
block input	238	OvrEna.....	316
NIB.....	303		
Nibble	303	P	
Nibbling.....	303	P.....	32
NJUST.....	572	PAC.....	316
No plane.....	125	Parameter transfer to subroutine	
non modal - program words		Conversion to an address parameter list.....	43
local.....	26	Parameter transfer to subroutines.....	39
		Parameterizing static tool orientation.....	388
		Path segmentation, programmable.....	386
		Path slope G8 at exact stop OFF AUS G62.....	627
		Path slope G9 at exact stop OFF G62.....	627
		Path slope OFF.....	120
		Path slope ON.....	120
		Path velocity-dependent laser power control....	287

PathAcc.....	316	Apply spindle as axis to channel.....	632
PCS.....	428	CPL loops.....	632
PCSPROBE.....	431	Program rotary axis via speed specifica- tion.....	632
PDIM.....	42	Processing velocity, direct influence.....	627
Permanent variables.....	62	Inpos window for at exact stop ON.....	627
PHS.....	322	Path slope G8/G9 at exact stop OFF G62.....	627
PL.....	114, 115, 116, 117	Processing velocity, indirect influence.....	629
Placement: Inclined plane.....	173	Comments.....	630
Configure after activation of inclined plane perpendicular to tool.....	177	File access.....	631
Implicit activation in the part program.....	175	Modal functions.....	629
Inclined plane vertical to tool.....	176	Short syntax.....	629
Programming directly in program line as parameter.....	177	Version parameters for "MCODS".....	631
Programming together with placement ta- bles.....	178	Waiting for permanent CPL variable.....	631
placement: Workpiece position correction.....	174	Processing velocity, indirect influences	
Chain of calculation.....	237	WAIT command.....	630
Placement: Workpiece position correction.....	236	Program block - NC program.....	16
Basic rotation around the z-coordinate.....	237	Program continuation with block pre-run af- ter program abort.....	340
Plane switching.....	126	Program design	
Assignment irrespective of axes	127	Jumps in program run.....	30
Plane switching, advanced.....	127	Program design, elements.....	27
PLC.....	485, 486	Blank lines in the program code.....	28
PLC interface.....	484	Block numbers.....	27
Bit interface.....	484	Channel ID.....	27
Operand access.....	485	Comments in the CPL block.....	28
PLS.....	322	Comments in the part program.....	28
PMD.....	324	Notes on the user interface.....	29
PMS.....	317	Provide comments for program lines.....	28
PMT.....	444	Remarks in the standard NC block.....	28
PmTSel.....	317	Program end.....	25, 181
PMV.....	443	Program spindle speed.....	191
Polar coordinate programming		Program stop.....	180
Define pole.....	318	Program words.....	18
PolarPol.....	318	as parameter.....	22
PoleSet.....	322	of NC functions.....	19
POP.....	318	Partial words.....	18
PosDepHSOut.....	322	Program attributes.....	22
Position and contour offset		Separators between two partial words.....	24
Position offset.....	441, 442	Program words - effect.....	25
Position and coordinate offset.....	441	Non-modal.....	26
Position file pointer		Program words - Effect	
SEEK and random file.....	575	modal.....	25
SEEK and sequential file.....	575	Program words as parameter.....	22
Position-dependent high-speed output.....	322	Program words as parameters	
Positioning type for endless axes.....	324	Parameter with following parameter list.....	22
PosMode.....	324	Parameters with following value.....	22
Postlinking.....	16	Program words of NC functions.....	19
PPOS.....	432	CPL expressions in parameter lists.....	20
Precision programming.....	326	Parameter lists with specific syntax ele- ments.....	21
PrecProg.....	327	Parameter lists without specific syntax el- ements.....	20
PREP.....	328	Parameters in a parameter list.....	20
PRN#.....	570	Programmable path segmentation.....	386
PROBE.....	433	Programmable zero offset.....	141
Probe, traveling to	150	Programmed contour shift.....	363
Processing delay due to faulty programming....	631		
Processing delay, faulty programming			

Index

Programming		
SHAPE for path mode.....	123	
SHAPE for positioning mode.....	123	
Programming attributes.....	22	
Programming examples - Remove preceding		
or following spaces.....	593	
Overwrite character array.....	594	
Program string variables (without previ-		
ous dimensioning).....	593	
Read character array.....	593	
Write on character array.....	594	
Programming options, DIN and CPL.....	12	
Programming polar coordinates.....	318	
In absolute dimensions.....	320	
In incremental dimensions.....	321	
Polar angle 1.....	319	
Polar angle 2.....	319	
Programming polar coordinates.....	318	
Radius value.....	319	
Programming spline parameter lengths....	114, 115	
Provide remarks for program lines.....	28	
PRP.....	327	
PT2EUL.....	446	
PtBlkEnd.....	330	
PTD.....	331	
PtDefault.....	331	
PTE.....	330	
PTI.....	332	
PtInpos.....	332	
PTP.....	333	
PTP traversing motion.....	333	
PUN.....	336	
Punch.....	336	
Punching.....	335	
PW.....	117	
R		
RAC.....	338	
RAD.....	253	
RadialAcc.....	338	
Radius programming.....	106, 252	
RadProg.....	253	
RANDOM.....	75	
Random file structure.....	561	
RAX.....	344	
RCB.....	340	
RDT.....	339	
Read file		
INP# instruction.....	573	
Read record from file.....	573	
Read file pointer position		
FILEPOS and random file.....	577	
FILEPOS and sequential file.....	577	
Read measured values.....	431	
Measured value in axis coordinates.....	432	
Measured value in program coordinates.....	431	
Measured value of the drive.....	433	
REAL variable.....	66	
Recording the retract vector.....	338	
RecordRevVec.....	339	
RedTorque.....	339	
ReentContBlk.....	340	
ReEntryRef.....	342	
Reference system LCS.....	167	
Reference system MCS.....	167	
Reference system PCS.....	167	
Relational operations.....	77	
Relative dimension programming.....	155	
REM.....	28	
Remarks in the standard NC block.....	28	
RemAxis.....	344	
RemLogName.....	344	
Remove (deactivate) coupling group.....	366	
Remove axis from axis group.....	343	
Remove logical axis name.....	344	
Remove preceding or following spaces		
Programming examples.....	593	
Removing channel spindles.....	345	
RemSpindle.....	345	
REPEAT.....	345	
REPEAT-UNTIL.....	59	
REPOSDEF.....	346	
Reposition individual coordinates	152	
REPOSTP.....	347	
Reserved command words.....	70	
Keywords.....	70	
Resetting channel spindles to MP	251	
Resulting SHAPE order.....	124	
Retrace.....	347, 348	
Retrace backward program extension.....	350	
Retrace backward program replacement.....	351	
Retrace forward program extension.....	202	
Retrace forward program replacement.....	203	
Retrace switching lock.....	349	
Retract from tapping.....	399	
Retract after power failure.....	399	
Retract from tapping.....	399	
RetTo.....	54	
REV.....	85	
REVA.....	350	
ReverseAppend.....	350	
ReverseSubstitute.....	351	
REVS.....	351	
REWRITE.....	567	
RLN.....	344	
RND.....	354	
RNDL.....	354	
RNE.....	353	
ROT(...).	352	
ROT2EUL.....	445	
Rotary axis programmed via speed specifi-		
cation.....	632	
Rotate(...).	352	
Rotating.....	351	
Rotating a contour.....	351	

ROTAX()	311
ROUND	75
Round corners	
with radius specification	354
Round corners with splines	
Modal	383
RoundEps	353
Rounding	354
Rounding corners	
with radius specification	354
Rounding corners with splines	
Local	384
rounding of corners	
specifying the deviation	353
Rounding of corners	
specifying the deviation	353
Rounding up and down	75
RRV	339
RSP	345

S

S	191
S-attribute	188
S2D	292
S2U	292
SACSYNC	355
SBC	118
Scale(...)	358
Scaling	357
SCC	360
SCCT	360
SCL	448, 449
SCL(...)	358
SCO	384
SCOL	384
SCS3	479
SCS3L	481
SD	456, 457
Accuracy programming	468
Additive program coordinate offset	
(ATrans)	467
Block pre-run: Target position	470
Channel axis coupling	468
Channel axis number of a system axis	462
Channel of system axis	463
Channel spindle number of system spindle	466
Channel spindle numbers of a spindle group	466
Coordinate meaning	461
Current channel	458
Direction of spindle rotation	459
Feed group	469
Init program, triggering event	463
Length correction - Assigned coordinates	458
Motion function of channel spindles	466
National language	460

Number of axes in channel	461
Number of axes in current channel	461
Override	457
Plane configuration - Assigned channel	
axes	459
Point to return to contour	464
Processing method	460
Program coordinate offset (Trans +	
ATrans)	464
Program coordinate offset (Trans)	465
Programming example	469
Speed	457
Speed of channel spindle	466
Spindle override	465
Spline data	463
System axis number of a channel axis	462
System axis number of a channel axis in	
the current channel	462
System axis number of a channel spindle	467
System axis number of system spindle	467
System spindle number of a channel	
spindle	467
System spindle number of a system axis	467
Tangential tool guidance	465
Test without motion	460
Touch probe	458
Velocity	457
Working areas and dead areas	465
SDLOAD	477
SDR	470
Accuracy programming	474
Additive program coordinate offset	
(ATrans)	474
Channel axis coupling	474
Epsilon environment around the north pole	474
Orienting position of a channel spindle	474
Override	470
Program coordinate offset (Trans +	
ATrans)	471
Program coordinate offset (trans)	473
Re-entry points	473
Return to contour (WCS) after Asup	472
Return to contour after Asup	472
Speed of channel spindle	473
Spindle override	473
Spline data	471
Tangential tool guidance	473
Velocity	471
SDSAVE	478
SEEK	574
SelCrdCouple	360
SelCrdCoupleTab	360
Select a waiting strategy	382
Selected additional coordinate coupling	359
SelCrdCouple, SCC	359
With table	360

Index

Selected additional coordinate coupling		
SelCrdCouple, SCC.....	359	
Selected additional coordinate coupling with table.....	360	
Selecting axis transformation.....	245	
Points of action.....	245	
Selecting the exact stop window.....	279	
Selection of planes, free.....	128	
self-sustaining.....	26	
Separators between 2 partial words		
Axis and coordinate designations.....	24	
Separators between two partial words.....	24	
Sequential file structure.....	561	
Sercos communication.....	629	
Sercos interface		
Read Sercos III parameters.....	479	
Read Sercos parameters lists.....	481	
Writing Sercos parameters.....	482	
Service hotline.....	633	
Set parameters of the B-spline approximation.....	236	
SETERR.....	555, 556	
SetPos.....	362	
Setting the program position.....	362	
Setting the stroke release time to default.....	331	
Setting-up correction.....	236	
SETWARN.....	556	
SEV.....	85	
SHAPE for asynchronous axes.....	122	
Sharpening internal corners.....	246	
Shift.....	363	
Short syntax.....	629	
SHT.....	363	
Simple mathematical functions.....	73	
Simple system data		
SD.....	456	
SDR.....	470	
SIN.....	76	
Sin2DownFeed.....	292	
Sin2UpFeed.....	292	
SinDownFeed.....	292	
Single Step suppression.....	390	
SinUpFeed.....	292	
Skip block for standard NC blocks.....	55	
Skipping the digits after the decimal point.....	74	
Slave spindle		
Angular offset with active coupling.....	369	
Coupling distance.....	368	
SMax.....	364	
SMD.....	390	
SME.....	391	
SMin.....	364	
SMN.....	364	
SMX.....	364	
SND.....	292	
SNU.....	292	
SPA.....	366	
SpAdjStr.....	365	
SpAdmin.....	366	
SPADR\$.....	434	
SPC_WAIT.....	371	
SPCC.....	367	
SPCD.....	368	
SPCE.....	369	
SpCouple_Wait.....	371	
SpCoupleConfig.....	367	
SpCoupleDist.....	368	
SpCoupleErrWin.....	369	
SpCouplePosOffs.....	369	
SpCouplePosOffs_Wait.....	370	
SpCoupleSyncWin.....	370	
SPCP.....	369	
SPCP_WAIT.....	370	
SPCS.....	370	
SPD.....	385	
Special functions for thread cutting.....	404	
Configuring dynamics.....	404	
Configuring retract data.....	404	
Enable retraction.....	405	
Influence channel interface signal.....	405	
Switch spindle mode.....	405	
Speed limitation.....	363	
Speed programming.....	378	
Speed programming, direct.....	166	
Speed specification.....	187	
Flying positioning.....	188	
SPG.....	372	
SPGALL.....	372	
SPHOME.....	380	
Spindle clockwise rotation.....	182	
Spindle clockwise rotation and coolant ON.....	182	
Spindle counterclockwise rotation.....	183	
Spindle counterclockwise rotation and coolant ON.....	183	
Spindle coupling		
Angular offset with active coupling.....	369	
Coupling distance of slave spindle.....	368	
Synchronous mode error window.....	368	
Synchronous mode window.....	370	
Waiting for angular offset.....	370	
Waiting for synchronous mode.....	371	
Spindle orientation.....	378	
Spindle selection for tapping without compensation chuck.....	399	
Spindle stop.....	184	
Spindle: Switching between position and speed mode.....	387	
SpindleHome.....	380	
Spindles.....	434	
SpindleToAxis.....	380	
SpindleToAxisWait.....	381	
SpindleWait.....	382	
SpindleWaitMode.....	383	
Spline		
C1-constant cubic.....	115	
C2-constant cubic.....	115	

Spline coupling table.....	247	StepModeEnable.....	391
Spline parameter length.....	116, 117	STO.....	388
Spline point weight for checkpoints with b-splines.....	117	SToolLim.....	392
Spline programming.....	111	STR\$.....	584
Spline type 0.....	112	Straight line interpolation in rapid traverse "G0".....	101
Spline type 1.....	114	Straightness and angle error compensation.....	271
Spline type 1 and 2.....	115	Activation.....	271
Spline type 3.....	116	String constant.....	78
Spline type 4.....	117	STRING variable.....	67
Spline type definition.....	385	Stroke release time.....	329
Spline types.....	111, 385	Set stroke release time to default.....	331
Spline with coefficient programming.....	112	Stroke release time (Inpos window).....	332
Axis programming.....	112	Stroke release time (interpolation end point).....	329
Coordinates programming.....	112	Stroke release time (Inpos window).....	332
Denominator polynomial programming.....	113	Stroke release time (interpolation end point)....	329
Orientation vector programming.....	113	Setting the stroke release times.....	330
Programming spline parameter lengths.....	114	Subsequent stroke release.....	329
SplineCornering.....	383	Stroke release time (interpolation end time).....	329
SplineCorneringLocal.....	384	Early stroke release.....	329
SplineDef.....	385	Structured variables.....	65
Split.....	386	Subroutine.....	
SPM.....	387	Nesting depth.....	45
SpMode.....	387	Subroutine call in CPL via CALL.....	38
SPOS.....	429	DIN identifier in the program to be called.....	38
SPS.....	362	Link process with DIN.....	38
SPT.....	386	Subroutine call self-defined modal.....	37
SPV.....	89	Subroutine call self-defined with G- and M-codes.....	36
SPV command.....	631	Subroutine call with P-address.....	32
SPVE.....	89	Subroutine call without P address.....	33
SPWAIT.....	382	Subroutine end.....	25
SQRT.....	75	Subroutine return to any program block.....	54
Square root of an input value.....	75	subroutines.....	
SSD.....	95	call without P address.....	33
SSDE.....	95	Subroutines.....	31
SSDQ.....	372	Call in CPL via CALL command.....	38
SSDQInit.....	374	Call self-defined with G and M-codes.....	36
SSpAdm.....	375	Call with P-address.....	32
SSpAdmOff.....	375	Error recovery.....	45
SSPG.....	191	Local subroutines.....	34
SSpGear.....	376	Locally effective.....	32
SSPM.....	378	Modally effective.....	32
SSpMax.....	376	Self-defined modal call.....	37
SSpMin.....		Transferring parameters to subroutines.....	39
SSPMIN.....	377	Suggestions.....	8
SSpMode.....	377	Support.....	633
SSpMove.....	378	Suppressing the collision monitoring.....	361
SSPOO.....	378	Switch between position/speed interface.....	377
SSpOri.....	378	Switch off C-axis mode and wait.....	233
SSPS.....	378	Switch off C-axis mode for spindles.....	233
SSpSpeed.....	378	Switch spindle mode.....	405
SSpToolLim.....	379	Switching spindle to axis.....	632
STA.....	380	Switching spindles in C-axis mode.....	380
Standard programming.....	12	SWM.....	383
Starting and end boundary conditions.....	118	Synchronization functions at block execution time.....	
StatToolOri.....	388	Bit event.....	84
STAW.....	381		
StepModeDisable.....	390		

Index

Cancel stop conditions by the end of program (M30).....	94	Tangential tool guidance.....	393
Canceling the stop conditions.....	94	Position of the tool guidance.....	394
Delete a bit event.....	85	Setting angle.....	394
Motion stop until axis position is reached.....	90	Tangential tool orientation.....	397
Motion stop until basic workpiece position is reached.....	91	TangTool.....	394
Motion stop until workpiece position is reached.....	93	TangToolOri.....	397
Trigger a bit event.....	85	Tapping.....	147
Waiting for bit event.....	83	Tapping without compensation chuck.....	147
Write a permanent CPL variable.....	89	Spindle selection.....	399
Synchronization functions at the block execution time		Tapping, retract.....	399
Waiting for states at the PLC NC bit interface.....	86	TappRet1.....	399
Waiting for the value of a permanent CPL variable.....	87	TappRet2.....	399
Synchronization functions of block preparation		TappSp.....	400
WAIT (<idle time>).....	80	Target groups.....	1
WAIT (without parameters).....	79	TCM(...).....	393
Synchronization functions of the block preparation		TCS.....	402
BikNmb (BNB).....	82	TCS definition in program coordinates.....	401
WAIT (BITIF(...)).....	80	TcsDef.....	402
Synchronize system axis coupling.....	355	TCV.....	450
Synchronizing functions for block execution.....	82	Tensor orientation.....	312
Synchronous mode error window.....	368	Column vectors (Ox, Oy, Oz).....	312
Synchronous mode window.....	368, 370	O(...).....	313
System data		Phi, theta, psi.....	312
System data of structured types.....	475	ROTAX(...).....	312
System data of structured types		Thread cutting.....	131
Access.....	475	Face thread.....	133
Load structured system data.....	477	Longitudinal thread.....	133
Save structured system data.....	478	Multi-start threads.....	134
Variable information.....	476	Sequential threads.....	134
Variable value.....	476	Tapered face thread.....	134
System data, structured types		Thread cutting, special functions.....	404
Assignment rules for basic types.....	476	Thread pitch.....	147
System information.....	70	ThreadSet.....	404
System spindle programming.....	375	ThreadSet(...).....	404
Speed programming.....	378	ThreadSet(DYN).....	404
System spindle programming SSp...		ThreadSet(ROn).....	405
Change gear range.....	376	ThreadSet(SPC).....	405
Disable management of spindle motion for specific channels.....	375	ThreadSet(TCI).....	405
Enable/apply reserved spindle.....	375	Time programming.....	156
Limit maximum speed.....	376	Time recording.....	453
Limit minimum speed.....	377	Date.....	454
Motion programming.....	377	System time.....	453
Spindle orientation.....	378	Time.....	455
Switch between position/speed interface.....	377	Times of interpretation.....	14
		Too database	
T		Read-only or write access.....	553
TAN.....	76	Tool coordinate system (TCS)	
Tangential circle entry.....	110	Tool correction.....	153
Tangential circle exit.....	110	tool correction	
		ED compensation "ED".....	193
		Tool correction.....	192
		D-correction.....	192
		Tool corrections.....	449
		Active tool correction values.....	450
		Tool correction table and correction registers.....	450
		Tool database	
		Extended access.....	553

Extended access with locking.....	554	CPL variable ERRNO for evaluating errors...	69
Extended search.....	550	Variable groups.....	61
Load.....	546	Variable names.....	60
Move data block.....	547	Variable types.....	66
Save.....	548	Variables within standard NC program-	
Search data block.....	549	ming.....	68
Tool guidance, tangential.....	393	Variable types.....	66
Tool length correction.....	139	Array variable (ARRAY).....	66
Tool orientation, static, parameterizing.....	388	CHAR variables.....	67
Torque reduction.....	339	Floating point variable (DOUBLE).....	66
Trans.....	406	Floating point variable (REAL).....	66
Transferring parameters to modal subrou-		Integer variable (INT).....	66
tines.....	43	Logical variable (BOOL).....	66
Transferring parameters to subroutines		Overview.....	68
Address parameter list.....	39	Variables in standard NC programming.....	68
CPL parameter list.....	42	VARINF.....	476
Transferring parameters to subroutines.....	43	VARVAL.....	476, 477
Travel in TCS.....	393	VCPEna.....	407
Traveling to probe.....	150	VCPLoad.....	408
Trigger a bit event.....	85	VEC2EUL.....	447
Triggering diagnostics recording.....	257	Vector orientation.....	309
TRIM\$.....	591	O(...).....	310
TRS.....	406	Orientation coordinates.....	309
TSP.....	400	phi, theta, psi.....	309
TST.....	404	ROTAX(...).....	310
TTL.....	394	Velocity of asynchronous axes.....	190
TTO.....	397	velocity profiles	
Type conversion		Definition of separate.....	291
ASCII characters.....	584	Velocity profiles (ramp function).....	291
ASCII code.....	583	Acceleration interpolators.....	292
Character conversion.....	584	Constant feed interpolator, effect.....	292
Number conversion.....	585	Deceleration interpolators.....	292
		Oscillation cycle of the U axis.....	293
U		Velocity programming with acceleration adaptation	
Unconditional jump to a standard NC block.....	52	Absolute.....	162
Unintended use.....	10	Incremental.....	160, 164
Consequences, non-liability.....	9	VERSINF\$.....	453
Use case LeapFrog		Versions parameter of the MCODS.....	631
Use case LeapFrog.....	44	VirtAxisPos.....	409
User-defined velocity profile.....	115	Virtual drives.....	409
		Volumetric compensation.....	407
V		VREC_START.....	409
Value assignment.....	72	VREC_STOP.....	411
Assignment.....	72		
Between variables.....	73	W	
BOOL variable.....	72	WAIT.....	79
NUL.....	73	WAIT (without parameters).....	79
REAL variable.....	72	WAIT command.....	630
VAP.....	409	Wait for a delete a bit event.....	84
Variable groups.....	61	Wait for spindle programming at block end.....	382
Definable permanent variables.....	62	WAIT with waiting period.....	80
Global variables.....	62	WAIT(<idle time>).....	80
Local variables.....	61	WAIT(BITIF(...)).....	81
Permanent variables.....	62	WAITA.....	86
Structured variables.....	65	WaitAxis.....	412
Variable names.....	60	Waiting for bit event.....	83
Variable programming.....	60	Waiting for permanent CPL variable.....	631

Index

- Waiting for states on the PLC NC bit interface
 - WAITA..... 86
 - WAITO..... 86
 - Waiting for states on the PLC NC bit interface: WAITA / WAITO..... 86
 - Waiting for synchronous mode..... 371
 - Waiting for the value of a permanent CPL variable..... 87
 - WPV..... 87
 - WPVE..... 87
 - WAITO..... 86
 - WAX..... 412
 - WCS..... 430
 - WCS2BCS..... 447, 448
 - WEV..... 83
 - WHILE-DO-END..... 60
 - WID command..... 629
 - WID3..... 413, 414
 - wmhpem.dat..... 64
 - Workpiece coordinate system (WCS)
 - tool correction..... 153
 - Workpiece coordinates
 - Jogging..... 282
 - Workpiece position correction..... 442
 - Workpiece position correction and placements (inclined plane)
 - Active placement values..... 443
 - Compute Euler angles..... 445, 446, 447
 - Convert point..... 447, 448
 - Placement table..... 444
 - Workpiece position correction..... 442
 - Workpiece position correction placements (inclined plane)..... 442
 - WPV..... 87
 - WPV command..... 631
 - WPVE..... 87
 - WREV..... 84
 - Write on file
 - PRN# command with semicolon..... 570
 - PRN# command without semicolon..... 571
 - Write parameters to the drive
 - Spindles and asynchronous axes (WriteId3, WID3)..... 413
 - Synchronous axes (Writeld3, WID3)..... 414
 - Write system date at execution time
 - SSD..... 94
 - SSDE..... 95
 - Writeld3..... 414
 - WritID3..... 413, 414
 - Writing extended ident numbers of an ID list
 - to a drive
 - Writeld3, WID3..... 414
 - Writing into a system data queue..... 372
 - LastFilled..... 373
 - LastGet..... 373
 - Writing permanent CPL variables..... 89
 - SPV..... 89
 - SPVE..... 89
 - Writing Sercos parameters
 - For one or several drives (spindles and asynchronous axes)..... 413
 - For one or several drives (synchronous axes)..... 414
 - Values of a parameter list..... 414
 - Writeld3, WID3..... 413
 - Writing extended ident numbers of an ID list..... 414
 - WSCS..... 482
 - Writing system date at execution time..... 94
 - Writing values of a parameter list to a drive
 - Writeld3, WID3..... 414
 - WSCS..... 482
 - WSTOPA..... 93
 - WSTOPO..... 93
- X**
- XOR..... 77
 - XTAB..... 545
 - XTABCR..... 545
- Z**
- Zero offsets..... 142
 - Zero point offsets..... 436
 - Active ZO values..... 441
 - Create ZO table..... 436
 - Delete axis in ZO table..... 438
 - Insert axis in ZO table..... 436
 - Read and write access to a ZO table..... 438
 - Zeroing modulo axis (linear endless axis)..... 290
 - ZO..... 142
 - ZOCDEL..... 438
 - ZOCINS..... 436, 437
 - ZOS..... 415
 - ZOT..... 438, 439
 - ZOTCR..... 436
 - ZoTSel..... 415
 - ZOV..... 441

Notes

Bosch Rexroth AG

P.O. Box 13 57

97803 Lohr a.Main, Germany

Bgm.-Dr.-Nebel-Str. 2

97816 Lohr a.Main, Germany

Phone +49 9352 18 0

Fax +49 9352 18 8400

www.boschrexroth.com/electrics



R911342634