SIEMENS

SINUMERIK

SINUMERIK 840D sl / 828D Job Planning

Programming Manual

Valid for

Control
SINUMERIK 840D sl / 840DE sl
SINUMERIK 828D

Software	Version
CNC software	4.7 SP2

10/2015
6FC5398-2BP40-5BA3

Preface
Fundamental safety instructions 1
Flexible NC programming 2
File and Program Management 3
Protection zones 4
Special motion commands 5
Coordinate transformations (frames) 6
Transformations 7
Kinematic chains 8
Collision avoidance with kinematic chains 9
Tool offsets 10
Path traversing behavior 11
Axis couplings 12
Synchronized actions 13
Oscillation 14
Punching and nibbling 15
Grinding 16
Additional functions 17
User stock removal programs 18
Programming cycles externally 19
Tables 20
Appendix A

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER
indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING
indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION
indicates that minor personal injury can result if proper precautions are not taken.

NOTICE
indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

Order number: 6FC5398-2BP40-5BA3
℗ 11/2015 Subject to change

Copyright © Siemens AG 1995 - 2015.
All rights reserved

Preface

SINUMERIK documentation

The SINUMERIK documentation is organized in the following categories:

- General documentation
- User documentation
- Manufacturer/service documentation

Additional information

You can find information on the following topics at www.siemens.com/motioncontrol/docu:

- Ordering documentation/overview of documentation
- Additional links to download documents
- Using documentation online (find and search in manuals/information)

Please send any questions about the technical documentation (e.g. suggestions for improvement, corrections) to the following address:

docu.motioncontrol@siemens.com

My Documentation Manager (MDM)

Under the following link you will find information to individually compile OEM-specific machine documentation based on the Siemens content:

www.siemens.com/mdm

Training

For information about the range of training courses, refer under:

- www.siemens.com/sitrain
  SITRAIN - Siemens training for products, systems and solutions in automation technology
- www.siemens.com/sinutrain
  SinuTrain - training software for SINUMERIK

FAQs

You can find Frequently Asked Questions in the Service&Support pages under Product Support. http://support.automation.siemens.com

SINUMERIK

You can find information on SINUMERIK under the following link:

www.siemens.com/sinumerik

Target group

This publication is intended for:

- Programmers
- Project engineers

Benefits

With the programming manual, the target group can develop, write, test, and debug programs and software user interfaces.

Standard scope

This Programming Manual describes the functionality of the standard scope. Extensions or changes made by the machine tool manufacturer are documented by the machine tool manufacturer.

Other functions not described in this documentation might be executable in the control. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

Furthermore, for the sake of clarity, this documentation does not contain all detailed information about all product types and cannot cover every conceivable case of installation, operation or maintenance.

Technical Support

You will find telephone numbers for other countries for technical support in the Internet under http://www.siemens.com/automation/service&support

Information on structure and contents

Programming Manual, Fundamentals/Job Planning

The description of the NC programming is divided into two manuals:

1. **Fundamentals**
   This "Fundamentals" Programming Manual is intended for use by skilled machine operators with the appropriate expertise in drilling, milling and turning operations. Simple programming examples are used to explain the commands and statements which are also defined according to DIN 66025.
2. **Job planning**
   The Programming Manual "Advanced" is intended for use by technicians with in-depth, comprehensive programming knowledge. By virtue of a special programming language, the SINUMERIK control enables the user to program complex workpiece programs (e.g. for free-form surfaces, channel coordination, ...) and makes programming of complicated operations easy for technologists.

Availability of the described NC language elements

All NC language elements described in the manual are available for the SINUMERIK 840D sl. The availability regarding SINUMERIK 828D should be taken from Table "Operations: Availability for SINUMERIK 828D (Page 761)".

Table of contents

Preface **3**

1 Fundamental safety instructions **17**

1.1 General safety instructions 17

1.2 Industrial security 17

2 Flexible NC programming **19**

2.1 Variables 19
2.1.1 System data 19
2.1.2 Predefined user variables: Arithmetic parameters 22
2.1.2.1 Channel-specific arithmetic parameters (R) 22
2.1.2.2 Global arithmetic parameters (RG) 23
2.1.3 Predefined user variables: Link variables 25
2.1.4 Definition of user variables (DEF) 27
2.1.5 Redefinition of system data, user data, and NC commands (REDEF) 33
2.1.6 Attribute: Initialization value 36
2.1.7 Attribute: Limit values (LLI, ULI) 39
2.1.8 Attribute: Physical unit (PHU) 40
2.1.9 Attribute: Access rights (APR, APW, APRP, APWP, APRB, APWB) 43
2.1.10 Attribute: Data class (DCM, DCI, DCU) - only SINUMERIK 828D 47
2.1.11 Overview of definable and redefinable attributes 48
2.1.12 Definition and initialization of array variables (DEF, SET, REP) 50
2.1.13 Definition and initialization of array variables (DEF, SET, REP): Further Information 54
2.1.14 Data types 56
2.1.15 Check availability of a variable (ISVAR) 57
2.1.16 Read attribute values / data type (GETVARPHU, GETVARAP, GETVARLIM, GETVARDFT, GETVARTYP) 58

2.2 Indirect programming 63
2.2.1 Indirectly programming addresses 63
2.2.2 Indirectly programming G codes 65
2.2.3 Indirectly programming position attributes (GP) 66
2.2.4 Indirectly programming part program lines (EXECSTRING) 69

2.3 Arithmetic functions 70

2.4 Comparison and logic operations 72

2.5 Precision correction on comparison errors (TRUNC) 74

2.6 Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND) 75

2.7 Priority of the operations 77

2.8 Possible type conversions 77

2.9 String operations 78
2.9.1 Type conversion to STRING (AXSTRING) 79
2.9.2 Type conversion from STRING (NUMBER, ISNUMBER, AXNAME) 80
2.9.3 Concatenation of strings (<<) 81

2.9.4 Conversion to lower/upper case letters (TOLOWER, TOUPPER)....82
2.9.5 Determine length of string (STRLEN)....83
2.9.6 Search for character/string in the string (INDEX, RINDEX, MINDEX, MATCH)....83
2.9.7 Selection of a substring (SUBSTR)....84
2.9.8 Reading and writing of individual characters....85
2.9.9 Formatting a string (SPRINT)....87

2.10 Program jumps and branches....94
2.10.1 Return jump to the start of the program (GOTOS)....94
2.10.2 Program jumps to jump markers (GOTOB, GOTOF, GOTO, GOTOC)....95
2.10.3 Program branch (CASE ... OF ... DEFAULT ...)....98

2.11 Repeat program section (REPEAT, REPEATB, ENDLABEL, P)....99

2.12 Check structures....105
2.12.1 Conditional statement and branch (IF, ELSE, ENDIF)....107
2.12.2 Continuous program loop (LOOP, ENDLOOP)....108
2.12.3 Count loop (FOR ... TO ..., ENDFOR)....109
2.12.4 Program loop with condition at start of loop (WHILE, ENDWHILE)....110
2.12.5 Program loop with condition at the end of the loop (REPEAT, UNTIL)....111
2.12.6 Program example with nested check structures....112

2.13 Program coordination (INIT, START, WAITM, WAITMC, WAITE, SETM, CLEARM)....112

2.14 Interrupt routine (ASUB)....118
2.14.1 Function of an interrupt routine....118
2.14.2 Creating an interrupt routine....119
2.14.3 Assign and start interrupt routine (SETINT, PRIO, BLSYNC)....120
2.14.4 Deactivating/reactivating the assignment of an interrupt routine (DISABLE, ENABLE)....122
2.14.5 Delete assignment of interrupt routine (CLRINT)....122
2.14.6 Fast retraction from the contour (SETINT LIFTFAST, ALF)....123
2.14.7 Traversing direction for fast retraction from the contour 125
2.14.8 Motion sequence for interrupt routines....128

2.15 Axis replacement, spindle replacement (RELEASE, GET, GETD)....129

2.16 Transfer axis to another channel (AXTOCHAN)....133

2.17 Activate machine data (NEWCONF)....134

2.18 Write file (WRITE)....135

2.19 Delete file (DELETE)....139

2.20 Read lines in the file (READ)....140

2.21 Check for presence of file (ISFILE)....142

2.22 Read out file information (FILEDATE, FILETIME, FILESIZE, FILESTAT, FILEINFO)....143

2.23 Roundup (ROUNDUP)....144

2.24 Subprogram technique....146
2.24.1 General information....146
2.24.1.1 Subprogram....146
2.24.1.2 Subprogram names....147
2.24.1.3 Nesting of subprograms....148
2.24.1.4 Search path....149
2.24.1.5 Formal and actual parameters....149
2.24.1.6 Parameter transfer....150

2.24.2 Definition of a subprogram....151
2.24.2.1 Subprogram without parameter transfer....151
2.24.2.2 Subprogram with call-by-value parameter transfer (PROC)....152
2.24.2.3 Subprogram with call-by-reference parameter transfer (PROC, VAR)....154
2.24.2.4 Save modal G functions (SAVE)....156
2.24.2.5 Suppress single block execution (SBLOF, SBLON)....157
2.24.2.6 Suppress current block display (DISPLOF, DISPLON, ACTBLOCNO)....163
2.24.2.7 Identifying subprograms with preparation (PREPRO)....166
2.24.2.8 Subprogram return M17....166
2.24.2.9 RET subprogram return....167
2.24.2.10 Parameterizable subprogram return jump (RET ...)....168
2.24.2.11 Parameterizable subprogram return jump (RETB ...)....175
2.24.3 Subprogram call....179
2.24.3.1 Subprogram call without parameter transfer....179
2.24.3.2 Subprogram call with parameter transfer (EXTERN)....180
2.24.3.3 Number of program repetitions (P)....183
2.24.3.4 Modal subprogram call (MCALL)....184
2.24.3.5 Indirect subprogram call (CALL)....186
2.24.3.6 Indirect subprogram call with specification of the calling program part (CALL BLOCK ... TO ...)....187
2.24.3.7 Indirect call of a program programmed in ISO language (ISOCALL)....188
2.24.3.8 Call subprogram with path specification and parameters (PCALL)....189
2.24.3.9 Extend search path for subprogram calls (CALLPATH)....189
2.24.3.10 Execute external subprogram (840D sl) (EXTCALL)....191
2.24.3.11 Execute external subprogram (828D) (EXTCALL)....194
2.25 Macro technique (DEFINE ... AS)....197

3 File and Program Management....201

3.1 Program memory....201
3.1.1 Program memory in the NCK....201
3.1.2 External program memory....203
3.1.3 Addressing program memory files....205
3.1.4 Search path for subprogram call....209
3.1.5 Interrogating the path and file name....210
3.2 Working memory (CHANDATA, COMPLETE, INITIAL)....211

4 Protection zones....215

4.1 Defining the protection zones (CPROTDEF, NPROTDEF)....215
4.2 Activating/deactivating protection zones (CPROT, NPROT)....218
4.3 Checking for protection zone violation, working area limitation and software limit switches (CALCPOSI)....222

5 Special motion commands....227

5.1 Approaching coded positions (CAC, CIC, CDC, CACP, CACN)....227
5.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)....228
5.3 Spline group (SPLINEPATH)....238
5.4 Activating/deactivating NC block compression (COMPON, COMPCURV, COMPCAD, COMPSURF, COMPOF)....239

5.5 Polynomial interpolation (POLY, POLYPATH, PO, PL)....240
5.6 Settable path reference (SPATH, UPATH)....245
5.7 Measuring with touch-trigger probe (MEAS, MEAW)....248
5.8 Axial measurement (MEASA, MEAWA, MEAC) (option)....251
5.9 Special functions for OEM users (OMA1 ... OMA5, OEMIPO1, OEMIPO2, G810 ... G829)....262
5.10 Feedrate reduction with corner deceleration (FENDNORM, G62, G621)....263
5.11 Programmable end of motion criteria (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)....264

6 Coordinate transformations (frames)....267

6.1 Coordinate transformation via frame variables....267
6.1.1 Predefined frame variable ($P_CHBFRAME, $P_IFRAME, $P_PFRAME, $P_ACTFRAME)....269
6.2 Value assignments to frames....272
6.2.1 Assigning direct values (axis value, angle, scale)....272
6.2.2 Reading and changing frame components (TR, FI, RT, SC, MI)....274
6.2.3 Calculating with frames....275
6.2.4 Definition of frame variables (DEF FRAME)....277
6.3 Coarse and fine offsets (CTRANS, CFINE)....278
6.4 External zero offset ($AA_ETRANS)....279
6.5 Set actual value with loss of the referencing status (PRESETON)....280
6.6 Set actual value without loss of the referencing status (PRESETONS)....282
6.7 Frame calculation from three measuring points in space (MEAFRAME)....283
6.8 NCU global frames....286
6.8.1 Channel-specific frames ($P_CHBFR, $P_UBFR)....287
6.8.2 Frames active in the channel....288

7 Transformations....293

7.1 General programming of transformation types....293
7.1.1 Orientation movements for transformations....295
7.1.2 Overview of orientation transformation TRAORI....299
7.2 Three, four and five axis transformation (TRAORI)....301
7.2.1 General relationships of universal tool head....301
7.2.2 Three, four and five axis transformation (TRAORI)....304
7.2.3 Variants of orientation programming and initial setting (ORIRESET)....305
7.2.4 Programming the tool orientation (A..., B..., C..., LEAD, TILT)....306
7.2.5 Face milling (A4, B4, C4, A5, B5, C5)....313
7.2.6 Reference of the orientation axes (ORIWKS, ORIMKS):....314
7.2.7 Programming orientation axes (ORIAXES, ORIVECT, ORIEULER, ORIRPY, ORIRPY2, ORIVIRT1, ORIVIRT2)....316
7.2.8 Orientation programming along the peripheral surface of a taper (ORIPLANE, ORICONCW, ORICONCCW, ORICONTO, ORICONIO)....318
7.2.9 Specification of orientation for two contact points (ORICURVE, PO[XH]=, PO[YH]=, PO[ZH]=)....321

7.3 Orientation polynomials (PO[angle], PO[coordinate])....323
7.4 Rotations of the tool orientation (ORIROTA, ORIROTR, ORIROTT, ORIROTC, THETA)....325
7.5 Orientations relative to the path....327
7.5.1 Orientation types relative to the path....327
7.5.2 Rotation of the tool orientation relative to the path (ORIPATH, ORIPATHS, angle of rotation)....328
7.5.3 Interpolation of the tool rotation relative to the path (ORIROTC, THETA)....330
7.5.4 Smoothing of orientation characteristic (ORIPATHS A8=, B8=, C8=)....332
7.6 Compression of the orientation (COMPON, COMPCURV, COMPCAD, COMPSURF)....333
7.7 Smoothing the orientation characteristic (ORISON, ORISOF)....336
7.8 Kinematic transformation....337
7.8.1 Activate face end transformation (TRANSMIT)....337
7.8.2 Activate cylinder surface transformation (TRACYL)....338
7.8.3 Activate skew angle transformation (TRAANG)....341
7.8.4 Oblique plunge-cutting on grinding machines (G5, G7)....342
7.9 Activate concatenated transformation (TRACON)....343
7.10 Cartesian PTP travel....344
7.10.1 Activating/deactivating Cartesian PTP travel (PTP, PTPG0, PTPWOC, CP)....344
7.10.2 Specify the position of the joints (STAT)....345
7.10.3 Specify the sign of the axis angle (TU)....349
7.10.4 Example 1: PTP and TRAORI....351
7.10.5 Example 2: PTPG0 and TRANSMIT....352
7.11 Constraints when selecting a transformation....353
7.12 Deselecting a transformation (TRAFOOF)....354
8 Kinematic chains....357
8.1 Deletion of components (DELOBJ)....357
8.2 Index determination by means of names (NAMETOINT)....360
9 Collision avoidance with kinematic chains....361
9.1 Check for collision pair (COLLPAIR)....361
9.2 Requesting a recalculation of the collision model (PROTA)....362
9.3 Setting the protection zone status (PROTS)....363
9.4 Determining the clearance of two protection zones (PROTD)....364
10 Tool offsets....367
10.1 Offset memory....367
10.2 Additive offsets....370
10.2.1 Selecting additive offsets (DL)....370
10.2.2 Specify wear and setup values ($TC_SCPxy[t,d], $TC_ECPxy[t,d])....371
10.2.3 Delete additive offsets (DELDL)....372
10.3 Special handling of tool offsets....373
10.3.1 Mirroring of tool lengths....375
10.3.2 Wear sign evaluation....375

10.3.3 Coordinate system of the active machining operation (TOWSTD, TOWMCS, TOWWCS, TOWBCS, TOWTCS, TOWKCS)....376
10.3.4 Tool length and plane change....379

10.4 Online tool offset....380
10.4.1 Defining a polynomial function (FCTDEF)....380
10.4.2 Write online tool offset continuously (PUTFTOCF)....381
10.4.3 Write online tool offset, discrete (PUTFTOC)....382
10.4.4 Activate/deactivate online tool offset (FTOCON/FTOCOF)....383

10.5 Activate 3D tool offsets (CUT3DC..., CUT3DF...)....383
10.5.1 Activating 3D tool offsets (CUT3DC, CUT3DCD, CUT3DF, CUT3DFS, CUT3DFF, ISD)....383
10.5.2 3D tool offset peripheral milling, face milling....386
10.5.3 3D tool offset Tool shapes and tool data for face milling....388
10.5.4 3D tool offset Offset on the path, path curvature, insertion depth (CUT3DC, CUT3DCD, ISD)....389
10.5.5 3D tool offset Inside/outside corners and intersection procedure (G450/G451)....391
10.5.6 3D tool offset 3D circumferential milling with limitation surfaces....393
10.5.7 3D tool offset Taking into consideration a limitation surface (CUT3DCC, CUT3DCCD)....393

10.6 Tool orientation (ORIC, ORID, OSOF, OSC, OSS, OSSE, ORIS, OSD, OST)....397

10.7 Free assignment of D numbers, cutting edge numbers....403
10.7.1 Free assignment of D numbers, cutting edge numbers (CE address)....403
10.7.2 Free assignment of D numbers: Checking D numbers (CHKDNO)....403
10.7.3 Free assignment of D numbers: Rename D numbers (GETDNO, SETDNO)....404
10.7.4 Free assignment of D numbers: Determine T number to the specified D number (GETACTTD)....405
10.7.5 Free assignment of D numbers: Invalidate D numbers (DZERO)....405

10.8 Toolholder kinematics....406

10.9 Tool length compensation for orientable toolholders (TCARR, TCOABS, TCOFR, TCOFRX, TCOFRY, TCOFRZ)....411

10.10 Online tool length compensation (TOFFON, TOFFOF)....413

10.11 Cutting data modification for tools that can be rotated (CUTMOD)....417

11 Path traversing behavior....423

11.1 Tangential control....423
11.1.1 Defining coupling (TANG)....423
11.1.2 Activating intermediate block generation (TLIFT)....424
11.1.3 Activating the coupling (TANGON)....425
11.1.4 Deactivating the coupling (TANGOF)....427
11.1.5 Deleting a coupling (TANGDEL)....427

11.2 Feedrate characteristic (FNORM, FLIN, FCUB, FPO)....428

11.3 Acceleration behavior....434
11.3.1 Acceleration mode (BRISK, BRISKA, SOFT, SOFTA, DRIVE, DRIVEA)....434
11.3.2 Influence of acceleration on following axes (VELOLIMA, ACCLIMA, JERKLIMA)....436
11.3.3 Activation of technology-specific dynamic values (DYNNORM, DYNPOS, DYNROUGH, DYNSEMIFIN, DYNFINISH)....438

11.4 Traversing with feedforward control (FFWON, FFWOF)....439

11.5 Programmable contour accuracy (CPRECON, CPRECOF)....440

11.6 Program sequence with preprocessing memory (STOPFIFO, STARTFIFO, FIFOCTRL, STOPRE) 442
11.7 Influencing the stop events by stop delay areas (DELAYFSTON, DELAYFSTOF)....444
11.8 Prevent program position for SERUPRO (IPTRLOCK, IPTRUNLOCK)....446
11.9 Repositioning to the contour (REPOSA, REPOSL, REPOSQ, REPOSQA, REPOSH, REPOSHA, DISR, DISPR, RMIBL, RMBBL, RMEBL, RMNBL) 448
11.10 Influencing the motion control....457
11.10.1 Percentage jerk correction (JERKLIM)....457
11.10.2 Percentage velocity correction (VELOLIM)....458
11.10.3 Program example for JERKLIM and VELOLIM....460
11.11 Programmable contour/orientation tolerance (CTOL, OTOL, ATOL)....461
11.12 Tolerance for G0 motion (STOLF)....464
11.13 Block change behavior with active coupling (CPBC)....466
12 Axis couplings....467
12.1 Coupled motion (TRAILON, TRAILOF)....467
12.2 Curve tables (CTAB)....471
12.2.1 Define curve tables (CTABDEF, CATBEND)....471
12.2.2 Check for presence of curve table (CTABEXISTS)....477
12.2.3 Delete curve tables (CTABDEL)....478
12.2.4 Locking curve tables to prevent deletion and overwriting (CTABLOCK, CTABUNLOCK)....479
12.2.5 Curve tables: Determine table properties (CTABID, CTABISLOCK, CTABMEMTYP, CTABPERIOD)....480
12.2.6 Read curve table values (CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABSSV, CTABSEV, CTAB, CTABINV, CTABTMIN, CTABTMAX)....481
12.2.7 Curve tables: Check use of resources (CTABNO, CTABNOMEM, CTABFNO, CTABSEGID, CTABSEG, CTABFSEG, CTABMSEG, CTABPOLID, CTABPOL, CTABFPOL, CTABMPOL)....486
12.3 Axial master value coupling (LEADON, LEADOF)....487
12.4 Electronic gear (EG)....493
12.4.1 Defining an electronic gear (EGDEF)....493
12.4.2 Switch-in the electronic gearbox (EGON, EGONSYN, EGONSYNE)....494
12.4.3 Switching-in the electronic gearbox (EGOFS, EGOFC)....497
12.4.4 Deleting the definition of an electronic gear (EGDEL)....498
12.4.5 Rotational feedrate (G95) / electronic gear (FPR)....499
12.5 Synchronous spindle....499
12.5.1 Synchronous spindle: Programming (COUPDEF, COUPDEL, COUPON, COUPONC, COUPOF, COUPOFS, COUPRES, WAITC)....500
12.6 Generic coupling (CP...)....509
12.7 Master/slave coupling (MASLDEF, MASLDEL, MASLON, MASLOF, MASLOFS)....516
13 Synchronized actions....521
13.1 Definition of a synchronized action....521

14 Oscillation....523
14.1 Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE, OSB)....523
14.2 Oscillation controlled by synchronized actions (OSCILL)....527
15 Punching and nibbling....535
15.1 Activation, deactivation....535
15.1.1 Punching and nibbling on or off (SPOF, SON, PON, SONS, PONS, PDELAYON, PDELAYOF, PUNCHACC)....535
15.2 Automatic path segmentation....540
15.2.1 Path segmentation for path axes....542
15.2.2 Path segmentation for single axes....544
16 Grinding....547
16.1 Activate/deactivate grinding-specific tool monitoring (TMON, TMOF)....547
17 Additional functions....549
17.1 Axis functions (AXNAME, AX, SPI, AXTOSPI, ISAXIS, AXSTRING, MODAXVAL)....549
17.2 Replaceable geometry axes (GEOAX)....551
17.3 Axis container (AXCTSWE, AXCTSWED, AXCTSWEC)....555
17.4 Wait for valid axis position (WAITENC)....557
17.5 Programmable parameter set changeover (SCPARA)....558
17.6 Check scope of NC language present (STRINGIS)....559
17.7 Interactively call the window from the part program (MMC)....563
17.8 Program runtime/part counter....565
17.8.1 Program runtime/part counter (overview)....565
17.8.2 Program runtime....565
17.8.3 Workpiece counter....569
17.9 Process DataShare - Output to an external device/file (EXTOPEN, WRITE, EXTCLOSE):....570
17.10 Alarms (SETAL)....574
17.11 Extended stop and retract (ESR)....576
17.11.1 NC-controlled ESR....577
17.11.1.1 NC-controlled retraction (POLF, POLFA, POLFMASK, POLFMLIN)....577
17.11.1.2 NC-controlled stopping....580
17.11.2 Drive-integrated ESR....581
17.11.2.1 Configuring drive-integrated stopping (ESRS)....581
17.11.2.2 Configuring drive-integrated retraction (ESRS)....582
18 User stock removal programs....585
18.1 Supporting functions for stock removal....585
18.2 Generate contour table (CONTPRON)....585
18.3 Generate coded contour table (CONTDCON)....591
18.4 Determine point of intersection between two contour elements (INTERSEC)....595

18.5 Execute the contour elements of a table block-by-block (EXECTAB)....596
18.6 Calculate circle data (CALCDAT)....597
18.7 Deactivate contour preparation (EXECUTE)....599
19 Programming cycles externally....601
19.1 Technology cycles....601
19.1.1 Introduction....601
19.1.2 Technology-specific overview....602
19.1.3 HOLES1 - row of holes....604
19.1.4 HOLES2 - hole circle....604
19.1.5 POCKET3 - milling a rectangular pocket....606
19.1.6 POCKET4 - milling a circular pocket....609
19.1.7 SLOT1 - longitudinal slot....611
19.1.8 SLOT2 - circumferential slot....614
19.1.9 LONGHOLE - elongated hole....616
19.1.10 CYCLE60 - engraving cycle....618
19.1.11 CYCLE61 - Face milling....621
19.1.12 CYCLE62 - contour call....623
19.1.13 CYCLE63 - Milling contour pocket....624
19.1.14 CYCLE64 - Predrilling contour pocket....626
19.1.15 CYCLE70 - thread milling....627
19.1.16 CYCLE72 - Path milling....629
19.1.17 CYCLE76 - rectangular spigot milling....633
19.1.18 CYCLE77 - circular spigot milling....635
19.1.19 CYCLE78 - Drill thread milling....637
19.1.20 CYCLE79 - multi-edge....639
19.1.21 CYCLE81 - drilling, centering....641
19.1.22 CYCLE82 - drilling, counterboring....642
19.1.23 CYCLE83 - deep-hole drilling....645
19.1.24 CYCLE84 - tapping without compensating chuck....648
19.1.25 CYCLE85 - reaming....651
19.1.26 CYCLE86 - boring....652
19.1.27 CYCLE92 - cut-off....653
19.1.28 CYCLE95 - contour cutting....655
19.1.29 CYCLE98 - thread chain....657
19.1.30 CYCLE99 - thread turning....661
19.1.31 CYCLE435 - calculate dressing tool position....666
19.1.32 CYCLE495 - form-truing....666
19.1.33 CYCLE800 - swiveling....668
19.1.34 CYCLE801 - grid or frame....671
19.1.35 CYCLE802 - arbitrary positions....673
19.1.36 CYCLE830 - deep-hole drilling 2....675
19.1.37 CYCLE832 - High-Speed Settings....681
19.1.38 CYCLE840 - tapping with compensating chuck....684
19.1.39 CYCLE899 - Milling open slot....687
19.1.40 CYCLE930 - groove....690
19.1.41 CYCLE940 - undercut forms....692
19.1.42 CYCLE951 - stock removal....695
19.1.43 CYCLE952 - contour grooving....698
19.1.44 CYCLE4071 - longitudinal grinding with infeed at the reversal point....704
19.1.45 CYCLE4072 - longitudinal grinding with infeed at the reversal point and cancel signal. 705

19.1.46 CYCLE4073 - longitudinal grinding with continuous infeed....709
19.1.47 CYCLE4074 - longitudinal grinding with continuous infeed and cancel signal....710
19.1.48 CYCLE4075 - surface grinding with infeed at the reversal point....713
19.1.49 CYCLE4077 - surface grinding with infeed at the reversal point and cancel signal....716
19.1.50 CYCLE4078 - surface grinding with continuous infeed....719
19.1.51 CYCLE4079 - surface grinding with intermittent infeed....721
19.1.52 Supplementary conditions....723
19.1.52.1 Technology scaling in cycle screen forms....723
19.2 Measuring cycles....725
20 Tables....727
20.1 Operations....727
20.2 Operations: Availability for SINUMERIK 828D....761
20.3 Currently set language in the HMI....786
A Appendix....789
A.1 List of abbreviations....789
A.2 Documentation overview....798
Glossary....799
Index....821

Fundamental safety instructions 1

1.1 General safety instructions

⚠ WARNING

Risk of death if the safety instructions and remaining risks are not carefully observed

If the safety instructions and residual risks are not observed in the associated hardware documentation, accidents involving severe injuries or death can occur.

- Observe the safety instructions given in the hardware documentation.
- Consider the residual risks for the risk evaluation.

⚠ WARNING

Danger to life or malfunctions of the machine as a result of incorrect or changed parameterization

As a result of incorrect or changed parameterization, machines can malfunction, which in turn can lead to injuries or death.

- Protect the parameterization (parameter assignments) against unauthorized access.
- Respond to possible malfunctions by applying suitable measures (e.g. EMERGENCY STOP or EMERGENCY OFF).

1.2 Industrial security

Note

Industrial security

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens' products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit this address (http://www.siemens.com/industrialsecurity).

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit this address (http://support.automation.siemens.com).

⚠ WARNING

Danger as a result of unsafe operating states resulting from software manipulation

Software manipulation (e.g. by viruses, Trojan horses, malware, worms) can cause unsafe operating states to develop in your installation which can result in death, severe injuries and/or material damage.

- Keep the software up to date.
  You will find relevant information and newsletters at this address (http://support.automation.siemens.com).
- Incorporate the automation and drive components into a holistic, state-of-the-art industrial security concept for the installation or machine.
  You will find further information at this address (http://www.siemens.com/industrialsecurity).
- Make sure that you include all installed products into the holistic industrial security concept.

Flexible NC programming 2

2.1 Variables

The use of variables from the areas system data and user data, especially in conjunction with arithmetic functions and check structures, enables highly flexible NC programs and cycles to be written.

- System data
  The system data contains the variables predefined in the system. These variables have a defined meaning. They are primarily used by the system software. The user can read and write these variables in NC programs and cycles. Example: Machine data, setting data, system variables.
  Although the meaning of a system data item is fixed, the user can modify its properties within certain limits by redefinition. See "Redefinition of system data, user data, and NC commands (REDEF) (Page 33)"
- User data
  The user data contain those variables defined by the user whose meaning is exclusively defined by the user. They are not evaluated by the system.
  The user data are divided into:
  - Predefined user variables
    Predefined user variables are variables that have already been defined in the system and whose number is parameterized in the machine data. The user can make changes to the properties of these variables. See "Redefinition of system data, user data, and NC commands (REDEF) (Page 33)."
  - User-defined variables
    User-defined variables are variables that are defined by the user and are not created by the system until runtime. Their number, data type, visibility, and all other properties are defined exclusively by the user.
    See "Definition of user variables (DEF) (Page 27)"

2.1.1 System data

The system data contain the variables that are predefined in the system and enable access to the current parameter settings of the control, as well as to machine, control, and process states, in NC programs and cycles.

Preprocessing variables

Preprocessing variables are system data that are read and written during preprocessing, in other words, at the instant at which the block containing the variable is interpreted. Preprocessing variables do not trigger preprocessing stops.

Main run variables

Main run variables are system data that are read and written during the main run, in other words, at the instant at which the block containing the variable is executed. The following are main run variables:

- Variables that can be programmed in synchronized actions (read/write)
- Variables that can be programmed in the NC program and trigger preprocessing stops (read/write)
- Variables that can be programmed in the NC program and whose value is calculated during preprocessing but not written until the main run (main run synchronized: write only)

Prefix system

To distinguish system data from other data, their names are usually preceded by a prefix comprising the $ sign followed by one or two letters and an underscore.

$ + 1st letter	**Meaning: Data type**
System data that are read/written during preprocessing	
$M	Machine data [1]
$S	Setting data, protection areas [1]
$T	Tool management data
$P	Programmed values
$C	Cycle variables of ISO envelope cycles
$O	Option data
R	R-parameters (arithmetic parameters) [2]
System data that are read/written during the main run	
$$M	Machine data [1]
$$S	Setting data [1]
$A	Current main run data
$V	Servo data
$R	R-parameters (arithmetic parameters) [2]
[1] Whether machine and setting data is treated as preprocessing or main run variables depends on whether they are written with one or two $ characters. The notation is freely selectable for the specific application. [2] When an R-parameter is used in the part program/cycle as a preprocessing variable, the prefix is omitted, e.g. R10. When it is used in a synchronized action as a main run variable, a $ sign is written as a prefix, e.g. $R10.	

2nd letter	**Meaning: Visibility**
N	NCK-global variable (**N**CK)
C	Channel-specific variable (**C**hannel)
A	Axis-specific variable (**A**xis)

Supplementary conditions

Exceptions in the prefix system

The following system of variables deviate from the prefix system specified above:

- $TC_...: Here, the 2nd letter C does not refer to channel-specific system variables but to toolholder-specific system variables (TC= tool carrier).
- $P_ ...: Channel-specific system variables

Use of machine and setting data in synchronized actions

When machine and setting data is used in synchronized actions, the prefix can be used to define whether the machine or setting data will be read/written synchronous to the preprocessing run or the main run.

If the data remains unchanged during machining, it can be read synchronous to the preprocessing run. For this purpose, the machine or setting data prefix is written with a $ sign:

```
ID=1 WHENEVER $AA_IM[z] < $SA_OSCILL_REVERSE_POS2[Z]-6 DO $AA_OVR[X]=0
```

If the data changes during machining, it must be read/written synchronous to the main run. For this purpose, the machine or setting data prefix is written with two $ signs:

```
ID=1 WHENEVER $AA_IM[z] < $$SA_OSCILL_REVERSE_POS2[Z]-6 DO $AA_OVR[X]=0
```

Note

Writing machine and setting data

When writing an item of machine or setting data, it is important to ensure that the access level which is active when the part program/cycle is executed permits write access and that the data is set to take "IMMEDIATE" effect.

References

A list of the properties of all system variables appears in:

List Manual, System Variables

See also

Variables (Page 19)

2.1.2 Predefined user variables: Arithmetic parameters

2.1.2.1 Channel-specific arithmetic parameters (R)

Channel-specific arithmetic parameters or R parameters are predefined user variables with the designation R, defined as an array of the REAL data type. For historical reasons, notation both with array index, e.g. `R[10]`, and without array index, e.g. `R10`, is permitted for R parameters.

When using synchronized actions, the $ sign must be included as a prefix, e.g. `$R10`.

Syntax

When used as a preprocessing variable:
`R<n>`
`R[<expression>]`

When used as a main run variable:
`$R<n>`
`$R[<expression>]`

Meaning

`R`:	Identifier when used as a preprocessing variable, e.g. in the part program	
`$R`:	Identifier when used as a main run variable, e.g. in synchronized actions	
	Type:	REAL
	Range of values:	For a non-exponential notation: ± (0.000 0001 ... 9999 9999) **Note:** A maximum of 8 decimal places are permitted
		For an exponential notation: ± ($1*10^{-300}$ ... $1*10^{+300}$) **Note:** • Notation: <Mantisse>EX<exponent> e.g. 8.2EX-3 • A maximum of 10 characters are permitted including sign and decimal point.
`<n>`:	Number of the R parameter	
	Type:	INT
	Range of values:	0 - MAX_INDEX **Note** MAX_INDEX is calculated from the parameterized number of R-parameters: MAX_INDEX = (MD28050 $MN_MM_NUM_R_PARAM) - 1
`<expression>`:	Array index Any expression can be used as an array index, as long as the result of the expression can be converted to the INT data type (INT, REAL, BOOL, CHAR).	

Example

Assignments to R-parameters and use of R-parameters in mathematical functions:

```
Program code                Comment
R0=3.5678                   ; Assignment in preprocessing
R[1]=-37.3                  ; Assignment in preprocessing
R3=-7                       ; Assignment in preprocessing
$R4=-0.1EX-5                ; Assignment in the main program run: R4 = -0.1 * 10^-5
$R[6]=1.874EX8              ; Assignment in the main program run: R6 = 1.874 * 10^8
R7=SIN(25.3)                ; Assignment in preprocessing

R[R2]=R10                   ; Indirect addressing using R-parameter
R[(R1+R2)*R3]=5             ; Indirect addressing using math. expression

X=(R1+R2)                   ; Traverse axis X to the position resulting from the sum
                            of R1 and R2
Z=SQRT(R1*R1+R2*R2)         ; Traverse axis Z to the square root position (R1^2 + R2^2)
```

See also

Variables (Page 19)

2.1.2.2 Global arithmetic parameters (RG)

Function

In addition to the channel-specific R parameters, the user has access to global R parameters. They exist once within the control unit and can be read and written from all channels.

Global R parameters are used, for example, to transfer information from one channel to the next. Another example concerns global settings that should be evaluated for all channels, such as the overhang of the raw part from the spindle.

The global R parameters are read and written from the user interface or in the NC program during the preprocessing. Synchronous actions and technology cycles cannot be used.

Note

No synchronization between the channels when reading and writing global R parameters.

Because the reading and writing is performed during the preprocessing, the point in time when a written value from one channel becomes active in another channel is not defined.

Example:

In channel 1, a loop runs with a global R parameter as loop counter. Channel 2 writes a value to this global R parameter; this causes a loop abort in channel 1. All loops that can be interpreted in the preprocessing in channel 1 are however still executed. The number of loops is not defined and depends on the channel loading, etc.

The user must implement a synchronization between the channels as application, e.g. with WAIT flags!

Syntax

Writing in the NC program

```
RG[<n>]=<value>
RG[<expression>]=<value>
```

Reading in the NC program

```
R...=RG[<n>]
R...=RG[<expression>]
```

Meaning

RG :	Default name of the NC address for **global** R parameters **Note:** The name of the NC address can be set via MD15800 $MN_R_PARAM_NCK_NAME	
<n>:	Number of the global R parameter	
	Type:	INT
	Range of values:	0 ... MAX_INDEX **Note** MAX_INDEX is calculated from the parameterized number of global R parameters: MAX_INDEX = (MD18156 $MN_MM_NUM_R_PARAM_NCK) - 1
<expression>:	Any expression can be used as an array index, as long as the result of the expression can be converted to the INT data type (INT, REAL, BOOL, CHAR).	

<value>:	Value of the global R parameter	
	Type:	REAL
	Range of values:	For a non-exponential notation: ± (0.000 0001 ... 9999 9999) **Note:** A maximum of eight decimal places are permitted
		For an exponential notation: ± ($1*10^{-300}$ ... $1*10^{+300}$) **Note:** • Notation: <mantissa>EX<exponent> e.g. 8.2EX-3 • A maximum of ten characters are permitted including sign and decimal point.

2.1.3 Predefined user variables: Link variables

Link variables can be used in the context of the "NCU-Link" function for cyclic data exchange between NCUs which are linked on a network. They facilitate data-format-specific access to the link variables memory. The link variables memory is defined both in terms of size and data structure on a system-specific basis by the user / machine manufacturer.

Link variables are system-global user variables which can be read and written in part programs and cycles by all NCUs involved in a link if link communication has been configured. Unlike global user variables (GUD), link variables can also be used in synchronized actions.

On systems without an active NCU link, link variables can be used locally on the control as additional global user variables alongside global user variables (GUD).

Syntax

```
$A_DLB[<index>]
$A_DLW[<index>]
$A_DLD[<index>]
$A_DLR[<index>]
```

Meaning

$A_DLB:	Link variable for BYTE data format (1 byte)	
	Data type:	UINT
	Range of values:	0 ... 255
$A_DLW:	Link variable for WORD data format (2 bytes)	
	Data type:	INT
	Range of values:	-32768 ... 32767
$A_DLD:	Link variable for DWORD data format (4 bytes)	
	Data type:	INT
	Range of values:	-2147483648 ... 2147483647

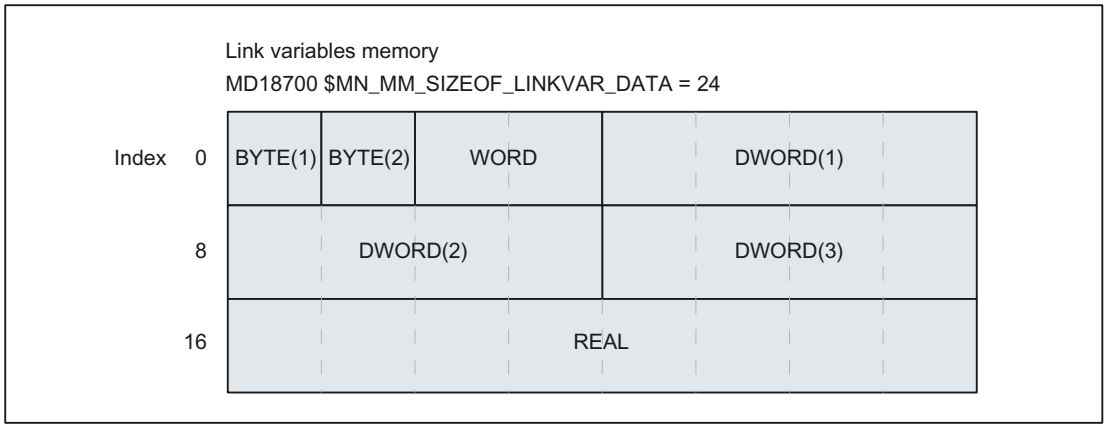
$A_DLR:	Link variable for REAL data format (8 bytes)	
	Data type:	REAL
	Range of values:	±(2,2*10^{-308} … 1,8*10^{+308})
<index>:	Address index in bytes, counted from the start of the link variable memory	
	Data type:	INT
	Range of values:	0 - MAX_INDEX **Note** • MAX_INDEX is calculated from the parameterized size of the link variables memory: MAX_INDEX = (MD18700 $MN_MM_SIZEOF_LINKVAR_DATA) - 1 • Only indices may be programmed, so that the bytes addressed in the link variables memory are located on a data format limit ⇒ Index = n * bytes, where n = 0, 1, 2, etc. – $A_DLB[i]: i = 0, 1, 2, ... – $A_DLW[i]: i = 0, 2, 4, ... – $A_DLD[i]: i = 0, 4, 8, ... – $A_DLR[i]: i = 0, 8, 16, ...

Example

An automation system contains two NCUs (NCU1 and NCU2). Machine axis AX2 is connected to NCU1. It is traversed as a link axis of NCU2.

NCU1 writes the actual current value ($VA_CURR) of axis AX2 cyclically to the link variables memory. NCU2 reads the actual current value transferred via link communication cyclically and displays alarm 61000 if the limit value is exceeded.

The data structure in the link variables memory is illustrated in the following figure. The actual current value is transmitted in the REAL value.



NCU1

NCU1 uses link variable $A_DLR[16] to write the actual current value of axis AX2 to the link variables memory cyclically in the interpolation cycle in a static synchronized action.

Program code

```
N111 IDS=1 WHENEVER TRUE DO $A_DLR[16]=$VA_CURR[AX2]
```

NCU2

NCU2 uses link variable $A_DLR[16] to read the actual current value of axis AX2 to the link variables memory cyclically in the interpolation cycle in a static synchronized action. If the actual current value is greater than 23.0 A, alarm 61000 is displayed.

Program code

```
N222 IDS=1 WHEN $A_DLR[16] > 23.0 DO SETAL(61000)
```

See also

Variables (Page 19)

2.1.4 Definition of user variables (DEF)

With the `DEF` command, you can define user-specific variables, or user variables (user data), and assign values to them.

According to the range of validity (in other words, the range in which the variable is visible) there are the following categories of user variable:

- Local user variables (LUD)
  Local user variables (LUD) are variables defined in an NC program that is not the main program at the time of execution. They are created when the NC program is called, and deleted at the end of the NC program or when the NC is reset. Local user variables can only be accessed within the NC program in which they are defined.
- Program-global user variables (PUD)
  Program-global user variables (PUD) are user variables defined in an NC program used as the main program. They are created when the NC program starts, and deleted at the end of the NC program or when the NC is reset. It is possible to access PUD in the main program and in all subprograms of the main program.
- Global user variables (GUD)
  Global user variables (GUD) are NC or channel-global variables which are defined in a data block (SGUD, MGUD, UGUD, GUD4 to GUD9) and are retained following shutdown and restart. GUD can be accessed in all NC programs.

User variables must be defined before they can be used (read/write). The following rules must be observed in this context:

- GUDs must be defined in a definition file, e.g. _N_DEF_DIR/_N_MGUD_DEF.
- PUDs and LUDs must be defined in the definition section of the NC program.
- The data must be defined in a dedicated block.

- Only one data type may be used for each data definition.
- Several variables of the same data type can be defined for each data definition.

Syntax

LUD and PUD

```
DEF <type> <phys_unit> <limit values> <name>[<value_1>, <value_2>,
<value_3>]=<init_value>
```

GUD

```
DEF <range> <pp_stop> <access_rights> <data class> <type>
<phys_unit> <limit values> <name>[<value_1>, <value_2>,
<value_3>]=<init_value>
```

Meaning

DEF:	Command for defining GUD, PUD, LUD user variables	
<range>:	Range of validity, only relevant for GUD:	
	NCK:	NC-global user variable
	CHAN:	Channel-global user variable
<PP_stop>:	Preprocessing stop, only relevant for GUD (optional)	
	SYNR:	Preprocessing stop when reading
	SYNW:	Preprocessing stop when writing
	SYNRW:	Preprocessing stop when reading/writing
<access rights>:	Protection level for reading/writing GUD via NC program or OPI (optional)	
	APRP <protection level>:	Read: NC program
	APWP <protection level>:	Write: NC program
	APRB <protection level>:	Read: OPI
	APWB <protection level>:	Write: OPI
	<protection level>:	Range of values: 0 ... 7
	See "Attribute: Access rights (APR, APW, APRP, APWP, APRB, APWB) (Page 43)"	
<data class>:	Data class assignment (only SINUMERIK 828D)	
	DCM:	Data class M (= Manufacturer)
	DCI:	Data class I (= Individual)
	DCU:	Data class U (= User)
	See "Attribute: Data class (DCM, DCI, DCU) - only SINUMERIK 828D (Page 47)".	

<type>:	Data type:	
	INT:	Integer with sign
	REAL:	Real number (LONG REAL to IEEE)
	BOOL:	Truth value TRUE (1)/FALSE (0)
	CHAR:	ASCII character
	STRING[<MaxLength>]:	Character string of a defined length
	AXIS:	Axis/spindle identifier
	FRAME:	Geometric data for a static coordinate transformation
	See "Data types (Page 56)"	
<phys_unit>:	Physical unit (optional)	
	PHU <unit>:	Physical unit
	See "Attribute: Physical unit (PHU) (Page 40)"	
<limit values>:	Lower/upper limit value (optional)	
	LLI <limit value>:	Lower limit value (lower limit)
	ULI <limit value>:	Upper limit value (upper limit)
	See "Attribute: Limit values (LLI, ULI) (Page 39)"	
<name>:	Name of variable **Note** • Maximum 31 characters • The first two characters must be a letter and/or an underscore. • The $ sign is reserved for system variables and must not be used.	
[<value_1>, <value_2>, <value_3>]:	Specification of array sizes for 1- to max. 3-dimensional array variables (optional) For the Initialization of array variables see "Definition and initialization of array variables (DEF, SET, REP) (Page 50)"	
<init_value>:	Initialization value (optional) See "Attribute: Initialization value (Page 36)" For the Initialization of array variables see "Definition and initialization of array variables (DEF, SET, REP) (Page 50)"	

Examples

Example 1: Definition of user variables in the data block for machine manufacturers

Program code	Comment
%_N_MGUD_DEF	; GUD block: Machine manufacturer
$PATH=/_N_DEF_DIR	
DEF CHAN REAL PHU 24 LLI 0 ULI 10 STROM_1, STROM_2	
;Description	
;Definition of two GUD items: STROM_1, STROM_2	
;Range of validity: Throughout the channel	
;Data type: REAL	

Program code	Comment

```
PP stop: Not programmed => default value = no PP stop
; Phys. unit: 24 = [A]
;Limit values: Low = 0.0, high = 10.0
;Access rights: Not programmed => default value = 7 = key-operated switch position 0
;Initialization value: Not programmed => default value = 0.0

DEF NCK REAL PHU 13 LLI 10 APWP 3 APRP 3 APWB 0 APRB 2 ZEIT_1=12, ZEIT_2=45
;Description
;Definition of two GUD items: ZEIT_1, ZEIT_2
;Range of validity: Throughout the NCK
;Data type: REAL
PP stop: Not programmed => default value = no PP stop
; Phys. unit: 13 = [s]
;Limit values: low = 10.0, high = not programmed => upper definition range limit
;Access rights:
; NC program: Write/read = 3 = end user
;OPI: Write = 0 = Siemens, read = 3 = end user
;Initialization value: ZEIT_1 = 12.0, ZEIT_2 = 45.0

DEF NCK APWP 3 APRP 3 APWB 0 APRB 3 STRING[5] GUD5_NAME = "COUNTER"
;Description
;Definition of one GUD item: GUD5_NAME
;Range of validity: Throughout the NCK
;Data type: STRING, max. 5 characters
PP stop: Not programmed => default value = no PP stop
; Phys. unit: Not programmed => default value = 0 = no phys. unit
;Limit values: Not programmed => definition range limits: Low = 0, high = 255
;Access rights:
; NC program: Write/read = 3 = end user
;OPI: Write = 0 = Siemens, read = 3 = end user
;Initialization value: "COUNTER"
M30
```

Example 2: Global program and local user variables (PUD/LUD)

Program code	Comment
PROC MAIN	; Main program
DEF INT VAR1	;PUD definition
...	
SUB2	;Subprogram call
...	
M30	

Program code	Comment
PROC SUB2	;Subprogram SUB2
DEF INT VAR2	;LUD DEFINITION
...	
IF (VAR1==1)	;Read PUD
VAR1=VAR1+1	;Read & write PUD
VAR2=1	;Write LUD
ENDIF	
SUB3	;Subprogram call
...	
M17	

Program code	Comment
PROC SUB3	;Subprogram SUB3
...	
IF (VAR1==1)	;Read PUD
VAR1=VAR1+1	;Read & write PUD
VAR2=1	;Error: LUD from SUB2 not known
ENDIF	
...	
M17	

Example 3: Definition and use of user variables of data type AXIS

Program code	Comment
DEF AXIS ABSCISSA	; 1st geometry axis
DEF AXIS SPINDLE	;Spindle
...	
IF ISAXIS(1) == FALSE GOTOF CONTINUE	
ABSCISSA = $P_AXN1	
CONTINUE:	
...	
SPINDLE=(S1)	; 1st spindle
OVRA[SPINDLE]=80	;Spindle override = 80%
SPINDLE=(S3)	; 3rd spindle

Supplementary conditions

Global user variables (GUD)

In the context of the definition of global user variables (GUD), the following machine data has to be taken into account:

No.	Identifier: $MN_	Meaning
11140	GUD_AREA_ SAVE_TAB	Additional save for GUD blocks
18118 [1)]	MM_NUM_GUD_MODULES	Number of GUD files in the active file system
18120 [1)]	MM_NUM_GUD_NAMES_NCK	Number of global GUD names
18130 [1)]	MM_NUM_GUD_NAMES_CHAN	Number of channel-specific GUD names
18140 [1)]	MM_NUM_GUD_NAMES_AXIS	Number of axis-spec. GUD names
18150 [1)]	MM_GUD_VALUES_MEM	Memory location for global GUD values
18660 [1)]	MM_NUM_SYNACT_GUD_REAL	Number of configurable GUD of the REAL data type
18661 [1)]	MM_NUM_SYNACT_GUD_INT	Number of configurable GUD of the INT data type
18662 [1)]	MM_NUM_SYNACT_GUD_BOOL	Number of configurable GUD of the BOOL data type
18663 [1)]	MM_NUM_SYNACT_GUD_AXIS	Number of configurable GUD of the AXIS data type
18664 [1)]	MM_NUM_SYNACT_GUD_CHAR	Number of configurable GUD of the CHAR data type
18665 [1)]	MM_NUM_SYNACT_GUD_STRING	Number of configurable GUD of the STRING data type

[1)] For SINUMERIK 828D, MD can only be read!

Program-global user variables (PUD)

Note

Visibility of program-global user variables (PUD)

Program-global user variables (PUD) defined in the main program will only be visible in subprograms if the following machine data is set:

MD11120 $MN_LUD_EXTENDED_SCOPE = 1

If MD11120 = 0 the program-global user variables defined in the main program will only be visible in the main program.

Cross-channel use of an NCK-global user variable of the AXIS data type

An NCK-global user variable of the `AXIS` data type initialized during definition in the data block with an axis identifier can then only be used in other NC channels if the axis has the same channel axis number in these channels.

If this is not the case, the variable has to be loaded at the beginning of the NC program or, as in the following example, the AXNAME(...) function (see "Axis functions (AXNAME, AX, SPI, AXTOSPI, ISAXIS, AXSTRING, MODAXVAL) (Page 549)") has to be used.

Program code	Comment
DEF NCK STRING[5] ACHSE="X"	;Definition in the data block
...	
N100 AX[AXNAME(ACHSE)]=111 G00	; Use in the NC program

2.1.5 Redefinition of system data, user data, and NC commands (REDEF)

The REDEF command changes the attributes of system data, user data, and NC commands. A fundamental condition of redefinition is that it has to post-date the corresponding definition.

Multiple attributes cannot be changed simultaneously during redefinition. A separate REDEF command must be programmed for each attribute to be changed.

If several concurrent attribute changes are programmed, the last change is always active.

Resetting attribute values

The attributes for access rights and initialization time change with REDEF can be reset to their default values by reprogramming REDEF, followed by the name of the variable or the NC language command:

- Access rights: Protection level 7
- Initialization time: No initialization or retention of the current value

Redefinable attributes

See "Overview of definable and redefinable attributes (Page 48)".

Local user variables (PUD/LUD)

Redefinitions are not permitted for local user variables (PUD/LUD).

Syntax

```
REDEF <name> <PP_stop>
REDEF <name> <phys_unit>
REDEF <name> <limit_values>
REDEF <name> <access_rights>
REDEF <name> <init_time>
REDEF <name> <init_time> <init_value>
REDEF <name> <data class>
REDEF <name>
```

Meaning

REDEF:	Command for redefinition of a certain attribute or to reset the "Access rights" and/or "Initialization time" attributes of system variables, user variables and NC language commands	
<name>:	Name of an already defined variable or an NC language command	
<PP stop>:	Preprocessing stop	
	SYNR:	Preprocessing stop when reading
	SYNW:	Preprocessing stop when writing
	SYNRW:	Preprocessing stop when reading/writing
<phys_unit>:	Physical unit	
	PHU <unit>:	Physical unit
	See "Attribute: Physical unit (PHU) (Page 40)". **Note** Cannot be redefined for: • System variables • Global user data (GUD) of the data types: BOOL, AXIS, STRING, FRAME	
<limit values>:	Lower/upper limit	
	LLI <limit value>:	Lower limit value (lower limit)
	ULI <limit value>:	Upper limit value (upper limit)
	See "Attribute: Limit values (LLI, ULI) (Page 39)". **Note** Cannot be redefined for: • System variables • Global user data (GUD) of the data types: BOOL, AXIS, STRING, FRAME	
<access rights>:	Access rights for reading/writing via part program or OPI	
	APX <protection level>:	Execute: NC language element
	APRP <protection level>:	Read: Part program
	APWP <protection level>:	Write: Part program
	APRB <protection level>:	Read: OPI
	APWB <protection level>:	Write: OPI
	<protection level>:	Range of values: 0 ... 7
	See "Attribute: Access rights (APR, APW, APRP, APWP, APRB, APWB) (Page 43)".	
<init_time>:	Point in time at which the variable is reinitialized	
	INIPO:	Power On
	INIRE:	End of main program, NC reset or Power On
	INICF:	NewConfig or end of main program, NC reset or Power On
	PRLOC:	End of main program, NC reset following local change or Power On
	See "Attribute: Initialization value (Page 36)".	

<init_value>:	Initialization value When redefining the initialization value, an initialization time always has to be specified also (see <init_time>). See "Attribute: Initialization value (Page 36)". For the Initialization of array variables, see "Definition and initialization of array variables (DEF, SET, REP) (Page 50)". **Note** Cannot be redefined for system variables, except setting data.	
<data class>:	Data class assignment (only SINUMERIK 828D)	
	DCM:	Data class M (= Manufacturer)
	DCI:	Data class I (= Individual)
	DCU:	Data class U (= User)
	See "Attribute: Data class (DCM, DCI, DCU) - only SINUMERIK 828D (Page 47)".	

Example

Redefinitions of system variable $TC_DPCx in the data block for machine manufacturers

Program code

```
%_N_MGUD_DEF                                ; GUD block: Machine manufacturer
N100 REDEF $TC_DPC1 APWB 2 APWP 3
N200 REDEF $TC_DPC2 PHU 21
N300 REDEF $TC_DPC3 LLI 0 ULI 200
N400 REDEF $TC_DPC4 INIPO (100, 101, 102, 103)
N800 REDEF $TC_DPC1
N900 REDEF $TC_DPC4
M30
```

regarding N100: Write access: OPI = protection level 2, part program = protection level 3

regarding N200: Physical unit [%]

regarding N300: Lower limit value = 0, upper limit value = 200

regarding N400: The array variable is initialized with the four values at POWER ON.

regarding N800 / N900: Reset of the "Access rights" and/or "Initialization time" attribute values

Note

Use of ACCESS files

If ACCESS files are used, the redefinition of access rights has to be relocated from _N_MGUD_DEF to _N_MACCESS_DEF.

Supplementary conditions

Granularity

A redefinition is always applied to the entire variable which is uniquely identified by its name. Array variables do not, for example, support the assignment of different attributes to individual array elements.

2.1.6 Attribute: Initialization value

Definition (DEF) of user variables

During definition, an initialization value can be preassigned for the following user variables:

- Global user variables (GUD)
- Program-global user variables (PUD)
- Local user variables (LUD)

Redefinition (REDEF) of system and user variables

During redefinition, an initialization value can be preassigned for the following variables:

- System data
  - Setting data
- User data
  - R parameters
  - Synchronized action variables ($AC_MARKER, $AC_PARAM, $AC_TIMER)
  - Synchronized action GUD (SYG_xy[ ], where x=R, I, B, A, C, S and y=S, M, U, 4 to 9)
  - EPS parameters
  - Tool data OEM
  - Magazine data OEM
  - Global user variables (GUD)

Reinitialization time

During redefinition, the point in time can be specified at which the variable should be reinitialized, i.e. reset to the initialization value.

- `INIPO` (POWER ON)
  The variable is reinitialized at Power On.
- `INIRE` (reset)
  The variable is reinitialized on NC reset, mode group reset, at the end of the part program (`M02`/`M30`) or at Power On.
- `INICF` (NewConfig)
  The variable is reinitialized on a NewConf request via the HMI, with part program command `NEWCONFIG` or on an NC reset, mode group reset, at the end of the part program (M02/M30) or at Power On.
- `PRLOC` (program-local change)
  The variable is only reinitialized on an NC reset, mode group reset or at the end of the part program (M02/M30) if it has changed during the current part program.
  The `PRLOC` attribute may only be changed in conjunction with programmable setting data (see the table below).

Table 2-1 Programmable setting data

Number	Identifier	G command [1]
42000	$SC_THREAD_START_ANGLE	SF
42010	$SC_THREAD_RAMP_DISP	DITS/DITE
42400	$SA_PUNCH_DWELLTIME	PDELAYON
42800	$SA_SPIND_ASSIGN_TAB	SETMS
43210	$SA_SPIND_MIN_VELO_G25	G25
43220	$SA_SPIND_MAX_VELO_G26	G26
43230	$SA_SPIND_MAX_VELO_LIMS	LIMS
43300	$SA_ASSIGN_FEED_PER_REV_SOURCE	FPRAON
43420	$SA_WORKAREA_LIMIT_PLUS	G26
43430	$SA_WORKAREA_LIMIT_MINUS	G25
43510	$SA_FIXED_STOP_TORQUE	FXST
43520	$SA_FIXED_STOP_WINDOW	FXSW
43700	$SA_OSCILL_REVERSE_POS1	OSP1
43710	$SA_OSCILL_REVERSE_POS2	OSP2
43720	$SA_OSCILL_DWELL_TIME1	OST1
43730	$SA_OSCILL_DWELL_TIME2	OST2
43740	$SA_OSCILL_VELO	FA
43750	$SA_OSCILL_NUM_SPARK_CYCLES	OSNSC
43760	$SA_OSCILL_END_POS	OSE
43770	$SA_OSCILL_CTRL_MASK	OSCTRL
43780	$SA_OSCILL_IS_ACTIVE	OS
43790	$SA_OSCILL_START_POS	OSB
1) This G command addresses the setting data.		

Supplementary conditions

Initialization value: Global user variables (GUD)

- Only INIPO (Power On) can be defined as the initialization time for global user variables (GUD) with the NCK range of validity.
- In addition to INIPO (Power On), INIRE (reset) or INICF (NewConfig) can be defined as the initialization time for global user variables (GUD) with the CHAN range of validity.
- In the case of global user variables (GUD) with the CHAN range of validity and INIRE (reset) or INICF (NewConfig) initialization time, for an NC reset, mode group reset and NewConfig, the variables are only reinitialized in the channels in which the named events were triggered.

Initialization value: FRAME data type

It is not permitted to specify an initialization value for variables of the FRAME data type. Variables of the FRAME data type are initialized implicitly and always with the default frame.

Initialization value: CHAR data type

For variables of the CHAR data type, instead of the ASCII code (0...255), the corresponding ASCII character can be programmed in quotation marks, e.g. "A".

Initialization value: Data type STRING

In the case of variables of the STRING data type, the character string must be enclosed in quotation marks, e.g. ...= "MACHINE_1"

Initialization value: AXIS data type

In the case of variables of the AXIS data type, for an extended address notation, the axis identifier must be enclosed in brackets, e.g. ...=(X3)

Initialization value: System variable

For system variables, redefinition cannot be used to define user-specific initialization values. The initialization values for the system variables are specified by the system and cannot be changed. However, redefinition can be used to change the point in time (INIRE, INICF) at which the system variable is reinitialized.

Implicit initialization value: AXIS data type

For variables of the AXIS data type the following implicit initialization value is used:

- System data: "First geometry axis"
- Synchronized action GUD (designation: SYG_A*), PUD, LUD:
  axis designation from the machine data: MD20082
  $MC_AXCONF_CHANAX_DEFAULT_NAME

Implicit initialization value: Tool and magazine data

Initialization values for tool and magazine data can be defined using the following machine data: MD17520 $MN_TOOL_DEFAULT_DATA_MASK

Note

Synchronization

The synchronization of events triggering the reinitialization of a global variable when this variable is read in a different location is the sole responsibility of the user / machine manufacturer.

See also

Variables (Page 19)

2.1.7 Attribute: Limit values (LLI, ULI)

An upper and a lower limit of the definition range can only be defined for the following data types:

- INT
- REAL
- CHAR

Definition (DEF) of user variables: Limit values and implicit initialization values

If no explicit initialization value is defined when defining a user variable of one of the above data types, the variable is set to the data type's implicit initialization value.

- INT: 0
- REAL: 0.0
- CHAR: 0

If the implicit initialization value is outside the definition range specified by the programmed limit values, the variable is initialized with the limit value which is closest to the implicit initialization value:

- Implicit initialization value < lower limit value (LLI) ⇒
  initialization value = lower limit value
- Implicit initialization value > upper limit value (ULI) ⇒
  initialization value = upper limit value

Examples:

Program code	Comment
DEF REAL GUD1	; Lower limit value = definition range limit ; Upper limit value = definition range limit ; No initialization value programmed ; => Implicit initialization value = 0.0
DEF REAL LLI 5.0 GUD2	; Lower limit value = 5.0 ; Upper limit value = definition range limit ; => Initialization value = 5.0
DEF REAL ULI -5 GUD3	; Lower limit value = definition range limit ; Upper limit value = -5.0 ; => Initialization value = -5.0

Redefinition (REDEF) of user variables: Limit values and current actual values

If the limit values of a user variable are redefined, they change to the extent that the current actual value is outside the new definition range, an alarm will be issued and the limit values will be rejected.

Note

Redefinition (REDEF) of user variables

If the limit values of a user variable are redefined, care must be taken to ensure that the following values are changed consistently:

- Limit values
- Actual value
- Initialization value on redefinition and automatic reinitialization on the basis of INIPO, INIRE or INICF

See also

Variables (Page 19)

2.1.8 Attribute: Physical unit (PHU)

A physical unit can only be specified for variables of the following data types:

- INT
- REAL

Programmable physical units (PHU)

The physical unit is specified as fixed point number: `PHU <unit>`

The following physical units can be programmed:

<unit>	Meaning	Physical unit
0	Not a physical unit	-
1	Linear or angular position [1)2)]	[mm], [inch], [degree]
2	Linear position [2)]	[mm], [inch]
3	Angular position	[degree]
4	Linear or angular velocity [1)2)]	[mm/min], [inch/min], [rpm]
5	Linear velocity [2)]	[mm/min]
6	Angular velocity	[rpm]
7	Linear or angular acceleration [1)2)]	[m/s^2], [$inch/s^2$], [rev/s^2]
8	Linear acceleration [2)]	[m/s^2], [$inch/s^2$]
9	Angular acceleration	[rev/s^2]
10	Linear or angular jerk [1)2)]	[m/s^3], [$inch/s^3$], [rev/s^3]
11	Linear jerk [2)]	[m/s^3], [$inch/s^3$]
12	Angular jerk	[rev/s^3]
13	Time	[s]
14	Position controller gain	[16.667/s]
15	Revolutional feedrate [2)]	[mm/rev], [inch/rev]
16	Temperature compensation [1)2)]	[mm], [inch]
18	Force	[N]
19	Mass	[kg]
20	Moment of inertia [3)]	[kgm^2]
21	Percent	[%]
22	Frequency	[Hz]
23	Voltage	[V]
24	Current	[A]
25	Temperature	[°C]
26	Angle	[degree]
27	KV	[1000/min]
28	Linear or angular position [3)]	[mm], [inch], [degree]
29	Cutting rate [2)]	[m/min], [feet/min]
30	Peripheral speed [2)]	[m/s], [feet/s]
31	Resistance	[ohm]
32	Inductance	[mH]
33	Torque [3)]	[Nm]
34	Torque constant [3)]	[Nm/A]
35	Current controller gain	[V/A]
36	Speed controller gain [3)]	[Nm/(rad*s)]
37	Speed	[rpm]
42	Power	[kW]
43	Current, low	[µA]
46	Torque, low [3)]	[µNm]
48	Per mil	-

<unit>	Meaning	Physical unit
49	-	[Hz/s]
65	Flow rate	[l/min]
66	Pressure	[bar]
67	Volume 3)	[cm^3]
68	Controlled-system gain 3)	[mm/(V*min)]
69	Force controller controlled-system gain	[N/V]
155	Thread lead 3)	[mm/rev], [inch/rev]
156	Change in thread lead 3)	[mm/rev / rev], [inch/rev / rev]
1) The physical unit depends on the axis type: Linear or rotary axis		
2) System of units changeover G70/G71(inch/metric) After changing over the basic system (MD10240 $MN_SCALING_SYSTEM_IS_METRIC) with G70/G71, for read/write operations to system and user variables involving a length, then the values are **not** converted (actual value, default value and limit values) G700/G710(inch/metric) After changing over the basic system (MD10240 $MN_SCALING_SYSTEM_IS_METRIC) with G700/G710, for read/write operations to system and user variables involving a length, then the values **are** converted (actual value, default value and limit values)		
3) The variable is **not** converted to the NC's current measuring system (inch/metric) automatically. Conversion is the sole responsibility of the user/machine manufacturer.		

Note

Level overflow due to format conversion

The internal storage format for all user variables (GUD/PUD/LUD) with physical units of length is metric. Excessive use of these types of variable in the NCK's main run, e.g. in synchronized actions, can lead to a CPU time overflow at interpolation level when the measuring system is switched over, generating alarm 4240.

Note

Compatibility of units

When using variables (assignment, comparison, calculation, etc.) the compatibility of the units involved is not checked. Should conversion be required, this is the sole responsibility of the user / machine manufacturer.

See also

Variables (Page 19)

2.1.9 Attribute: Access rights (APR, APW, APRP, APWP, APRB, APWB)

The following protection levels, which have to be specified during programming, correspond to the access rights:

Access right	Protection level
System password	0
Machine manufacturer password	1
Service password	2
End user password	3
Key-operated switch position 3	4
Key-operated switch position 2	5
Key-operated switch position 1	6
Key-operated switch position 0	7

Definition (DEF) of user data

Access rights (APR.../APW...) can be defined for the following data:

- Global user data (GUD)

Redefinition (REDEF) of system and user data

Access rights (APR.../APW...) can be redefined for the following data:

- System data
  - Machine data
  - Setting data
  - System variable
  - Process data
  - Magazine data
  - Tool data
- User data
  - R parameters
  - Synchronized action variables ($AC_MARKER, $AC_PARAM, $AC_TIMER)
  - Synchronized action GUD (SYG_xy[ ], where x=R, I, B, A, C, S and y=S, M, U, 4 to 9)
  - EPS parameters
  - Tool data OEM
  - Magazine data OEM
  - Global user variables (GUD)

Note

During redefinition the access right can be freely assigned to a variable between the lowest protection level 7 and the dedicated protection level, e.g. 1 (machine manufacturer).

Redefinition (REDEF) of NC language commands

The access or execution right (APX) can be redefined for the following NC language commands:

- G functions / path conditions
  References
  Programming Manual, Fundamentals, Section: G functions / path conditions
- Predefined functions
  References
  Programming Manual, Fundamentals, Section: Predefined functions
- Predefined subprogram calls
  References
  Programming Manual, Fundamentals, Section: Predefined subprogram calls
- DO operation with synchronized actions
- Cycles program identifier
  The cycle must be saved in a cycle directory and must contain a PROC operation.

Access rights in relation to NC programs and cycles (APRP, APWP)

The various access rights facilitate the following with regard to access from an NC program or cycle:

- APRP 0/APWP 0
  - During NC program processing the system password has to be set.
  - The cycle has to be stored in the _N_CST_DIR directory (system).
  - The execution right must be set to system for the _N_CST_DIR directory in MD11160 $MN_ACCESS_EXEC_CST.
- APRP 1/APWP 1 or APRP 2/APWP 2
  - During NC program processing the machine manufacturer or service password has to be set.
  - The cycle has to be stored in the _N_CMA_DIR (machine manufacturer) or _N_CST_DIR directory.
  - The execution rights must be set to at least machine manufacturer for the _N_CMA_DIR or _N_CST_DIR directories in machine data MD11161 $MN_ACCESS_EXEC_CMA or MD11160 $MN_ACCESS_EXEC_CST respectively.
- APRP 3/APWP 3
  - During NC program execution, the end-user password must be set.
  - The cycle has to be stored in the _N_CUS_DIR (user), _N_CMA_DIR or _N_CST_DIR directory.
  - The execution rights must be set to at least end user for the _N_CUS_DIR, _N_CMA_DIR or _N_CST_DIR directories in machine data MD11162 $MN_ACCESS_EXEC_CUS, MD11161 $MN_ACCESS_EXEC_CMA or MD11160 $MN_ACCESS_EXEC_CST respectively.
- APRP 4...7/APWP 4...7
  - During NC program processing the key-operated switch must be set to 3 ... 0.
  - The cycle has to be stored in directory _N_CUS_DIR, _N_CMA_DIR or in directory _N_CST_DIR.
  - The execution rights must be set to at least the corresponding key-operated switch position for the _N_CUS_DIR, _N_CMA_DIR or _N_CST_DIR directories in machine data MD11162 $MN_ACCESS_EXEC_CUS, MD11161 $MN_ACCESS_EXEC_CMA or MD11160 $MN_ACCESS_EXEC_CST respectively.

Access rights in relation to OPI (APRB, APWB)

The access rights (APRB, APWB) restrict access to system and user variables via the OPI equally for all system components (HMI, PLC, external computers, EPS services, etc.).

Note

Local HMI access rights

When changing access rights to system data, care must be taken to ensure that such changes are consistent with the access rights defined using HMI mechanisms.

APR/APW access attributes

For compatibility reasons, attributes APR and APW are implicitly mapped to the attributes APRP / APRB and APWP / APWB:

- APR x ⇒ APRP x APRB x
- APW y ⇒ APWP y APWB y

Access rights using ACCESS files

When using ACCESS files to assign access rights, access rights for system data, user data, and NC commands must only be redefined in ACCESS files. Global user data (GUD) is an exception. For this data, access rights still have to be redefined in the corresponding definition files *_DEF.

For continuous access protection, the machine data for the execution rights and the access protection for the corresponding directories have to be modified consistently.

In principle, the procedure is as follows:

1. Creation of the necessary definition files:
   - _N_DEF_DIR/_N_SACCESS_DEF
   - _N_DEF_DIR/_N_MACCESS_DEF
   - _N_DEF_DIR/_N_UACCESS_DEF
2. Setting of the write right for the definition files to the value required for redefinition:
   - MD11170 $MN_ACCESS_WRITE_SACCESS = <protection level>
   - MD11171 $MN_ACCESS_WRITE_MACCESS = <protection level>
   - MD11172 $MN_ACCESS_WRITE_UACCESS = <protection level>

3. For access to protected elements from cycles, the execution and write rights for cycle directories _N_CST_DIR, _N_CMA_DIR, and _N_CST_DIR have to be modified.
   Execution rights
   - MD11160 $MN_ACCESS_EXEC_CST = <protection level>
   - MD11161 $MN_ACCESS_EXEC_CMA = <protection level>
   - MD11162 $MN_ACCESS_EXEC_CUS = <protection level>

   Write rights
   - MD11165 $MN_ACCESS_WRITE_CST = <protection level>
   - MD11166 $MN_ACCESS_WRITE_CMA = <protection level>
   - MD11167 MN_ACCESS_WRITE_CUS = <protection level>

   The execution right has to be set to at least the same protection level as the highest protection level of the element used.
   The write right must be set to at least the same protection level as the execution right.
4. The write rights of the local HMI cycle directories must be set to the same protection level as the local NC cycle directories.
   References
   Operating Manual

Subprogram calls in ACCESS files

To structure access protection further, subprograms (SPF or MPF identifier) can be called in ACCESS files. The subprograms inherit the execution rights of the calling ACCESS file.

Note

Only access rights can be redefined in the ACCESS files. All other attributes have to continue to be programmed/redefined in the corresponding definition files.

See also

Variables (Page 19)

2.1.10 Attribute: Data class (DCM, DCI, DCU) - only SINUMERIK 828D

To simplify the data handling during the commissioning, series start-up and upgrade of machines and machine series, all system and user data of the NC is divided into data classes.

Data class	Data
S = System	System data provided by Siemens, such as machine and setting data, standard and measuring cycles, definitions (SGUD) and macros (SMAC), etc.
M = Manufacturer (machine manufacturer)	Machine series-specific commissioning data such as manufacturer cycles, definitions (MGUD) and macros (MMAC) and machine data that defines the functional scope of the machine.

Data class	Data
I = Individual (machine-specific)	Machine-specific commissioning data such as compensation data reference point offsets.
U = User	Machine-specific data generated during operation of the machine such as tool data, setting data, part programs, user cycles, definitions (UGUD) and macros (UMAC).

References:
SINUMERIK 828D Commissioning Manual, Turning and Milling; Section "Introduction and use of data classes"

Definition (DEF) of user data

The data class of the data item is implicitly specified through the data class of the file or directory in which the user data is defined. The data class of the data item cannot be changed.

However, for the definition (DEF) of the user data, a different data class to that of the data item can be specified for the **data value**.

The following must apply for the data class of the data item:

Priority of the data class of the data value **≤** priority of the data class of the data item

Example:

The definition of the GUD, which defines a probe, should be in data class M (= Manufacturer) because it is required to run the manufacturer cycles. However, the value of the data item should belong to data class I (= Individual) because the probe type can differ from machine to machine.

MGUD.DEF (data class M)

```
...
DEF CHAN DCI INT CALIPER
...
```

Redefinition (REDEF) of system data

The data class of the system data can be changed through redefinition (REDEF). The redefinition must be performed in a definition file with data class S or M.

When using ACCESS files, the redefinitions may only be performed with the ACCESS files.

The respective data class of the machine, setting and option data as well as the system variables can be found in the

- List Manual, Detailed Machine Data Description, parameter: "Class"
- List Manual, System Variables

2.1.11 Overview of definable and redefinable attributes

The following tables show which attributes can be defined (DEF) and/or redefined (REDEF) for which data types.

System data

Data type	Init. value	Limit values	Physical unit	Access rights	Data class (only 828D)
Machine data	---	---	---	REDEF	REDEF
Setting data	REDEF	---	---	REDEF	---
FRAME data	---	---	---	REDEF	---
Process data	---	---	---	REDEF	---
Leadscrew error comp. (EEC)	---	---	---	REDEF	---
Sag compensation (CEC)	---	---	---	REDEF	---
Quadrant error compensation (QEC)	---	---	---	REDEF	---
Magazine data	---	---	---	REDEF	---
Tool data	---	---	---	REDEF	---
Protection areas	---	---	---	REDEF	---
Toolholder, with orientation capability	---	---	---	REDEF	---
Kinematic chains	---	---	---	REDEF	---
3D protection areas	---	---	---	REDEF	---
Working area limitation	---	---	---	REDEF	---

User data

Data type	Init. value	Limit values	Physical unit	Access rights	Data class
R-parameters	REDEF	REDEF	REDEF	REDEF	---
Synchronized action variable ($AC_...)	REDEF	REDEF	REDEF	REDEF	---
Synchronized action GUD (SYG_...)	REDEF	REDEF	REDEF	REDEF	---
EPS parameters	REDEF	REDEF	REDEF	REDEF	---
Tool data OEM	REDEF	REDEF	REDEF	REDEF	---
Magazine data OEM	REDEF	REDEF	REDEF	REDEF	---
Global user variables (GUD)	DEF/REDEF	DEF	DEF	DEF/REDEF	DEF/REDEF
Local user variables (PUD/LUD)	DEF	DEF	DEF	---	---

See also

Variables (Page 19)

2.1.12 Definition and initialization of array variables (DEF, SET, REP)

A user variable can be defined as a 1- up to a maximum of a 3-dimensional array.

- 1-dimensional: `DEF <data type> <variable name>[<n>]`
- 2-dimensional: `DEF <data type> <variable name>[<n>,<m>]`
- 3-dimensional: `DEF <data type> <variable name>[<n>,<m>,<o>]`

Note

STRING data type user variables can be defined as up to a maximum of 2-dimensional arrays.

Data types

User variables can be defined as arrays for the following data types: BOOL, CHAR, INT, REAL, STRING, AXIS, FRAME

Assignment of values to array elements

Values can be assigned to array elements at the following points in time:

- During array definition (initialization values)
- During program execution

Values can be assigned by means of:

- Explicit specification of an array element
- Explicit specification of an array element as a starting element and specification of a value list (`SET`)
- Explicit specification of an array element as a starting element and specification of a value and the frequency at which it is repeated (`REP`)

Note

FRAME data type user variables cannot be assigned initialization values.

Syntax (DEF)

```
DEF <data type> <variable name>[<n>,<m>,<o>]
DEF  STRING[<string length>] <variable name>[<n>,<m>]
```

Syntax (DEF...=SET...)

Using a value list:

- During definition:
  ```
  DEF <data type> <variable
  name>[<n>,<m>,<o>]=SET(<value1>,<value2>, etc.)
  ```
  Equivalent to:
  ```
  DEF <data type> <variable name>[<n>,<m>,<o>]=(<value1>,<value2>,
  etc.)
  ```

  Note

  `SET` does not have to be specified for initialization via a value list.

- During value assignment:
  ```
  <variable name>[<n>,<m>,<o>]=SET(<VALUE1>,<value2>, etc.)
  ```

Syntax (DEF...=REP...)

Using a value with repetition

- During definition:
  ```
  DEF <data type> <variable name>[<n>,<m>,<o>]=REP(<value>)
  DEF <data type> <variable name>[<n>,<m>,<o>]=REP(<value>,
  <number_array_elements>)
  ```
- During value assignment:
  ```
  <variable name>[<n>,<m>,<o>]=REP(<value>)
  DEF <data type> <variable
  name>[<n>,<m>,<o>]=REP(<value>,<number_array_elements>)
  ```

Meaning

`DEF`:	Command to define variables	
`<data type>`:	Data type of variables	
	Range of values: • for system variables: BOOL, CHAR, INT, REAL, STRING, AXIS • for GUD or LUD variables: BOOL, CHAR, INT, REAL, STRING, AXIS, FRAME	
`<string length>`:	Maximum number of characters for a STRING data type	
`<variable name>`:	Variable name.	
`[<n>,<m>,<o>]`:	Array sizes or array indices	
`<n>`:	Array size or array index for 1st dimension	
	Type:	INT (for system variables, also AXIS)
	Range of values:	Max. array size: 65535 Array index: 0 ≤ n ≤ 65534

<m>:	Array size or array index for 2nd dimension	
	Type:	INT (for system variables, also AXIS)
	Range of values:	Max. array size: 65535 Array index: 0 ≤ m ≤ 65534
<o>:	Array size or array index for 3rd dimension	
	Type:	INT (for system variables, also AXIS)
	Range of values:	Max. array size: 65535 Array index: 0 ≤ o ≤ 65534
SET:	Value assignment using specified value list	
(<value1>,<value2>, etc.):	Value list	
REP:	Value assignment using specified <value>	
<value>:	Value, which the array elements should be written when initializing with REP.	
<number_array_elements>:	Number of array elements to be written with the specified <value>. The following apply to the remaining array elements, dependent on the point in time: • Initialization when defining the array: → Zero is written to the remaining array elements. • Assignment during program execution: → The actual values of the array elements remain unchanged. If the parameter is not programmed, all array elements are written with <value>. If the parameter equals zero, the following apply dependent on the point in time: • Initialization when defining the array: → All elements are pre-assigned zero • Assignment during program execution: → The actual values of the array elements remain unchanged.	

Array index

The implicit sequence of the array elements, e.g. in the case of value assignment using SET or REP, is right to left due to iteration of the array index.

Example: Initialization of a 3-dimensional array with 24 array elements:

```
DEF INT FELD[2,3,4] = REP(1,24)
  FELD[0,0,0] = 1     1. array element
  FELD[0,0,1] = 1     2. array element
  FELD[0,0,2] = 1     3. array element
  FELD[0,0,3] = 1     4. array element
  ...
  FELD[0,1,0] = 1     5. array element
  FELD[0,1,1] = 1     6. array element
  ...
```

```
  FELD[0,2,3] = 1      12. array element
  FELD[1,0,0] = 1      13. array element
  FELD[1,0,1] = 1      14. array element
  ...
  FELD[1,2,3] = 1      24. array element
```

corresponding to:

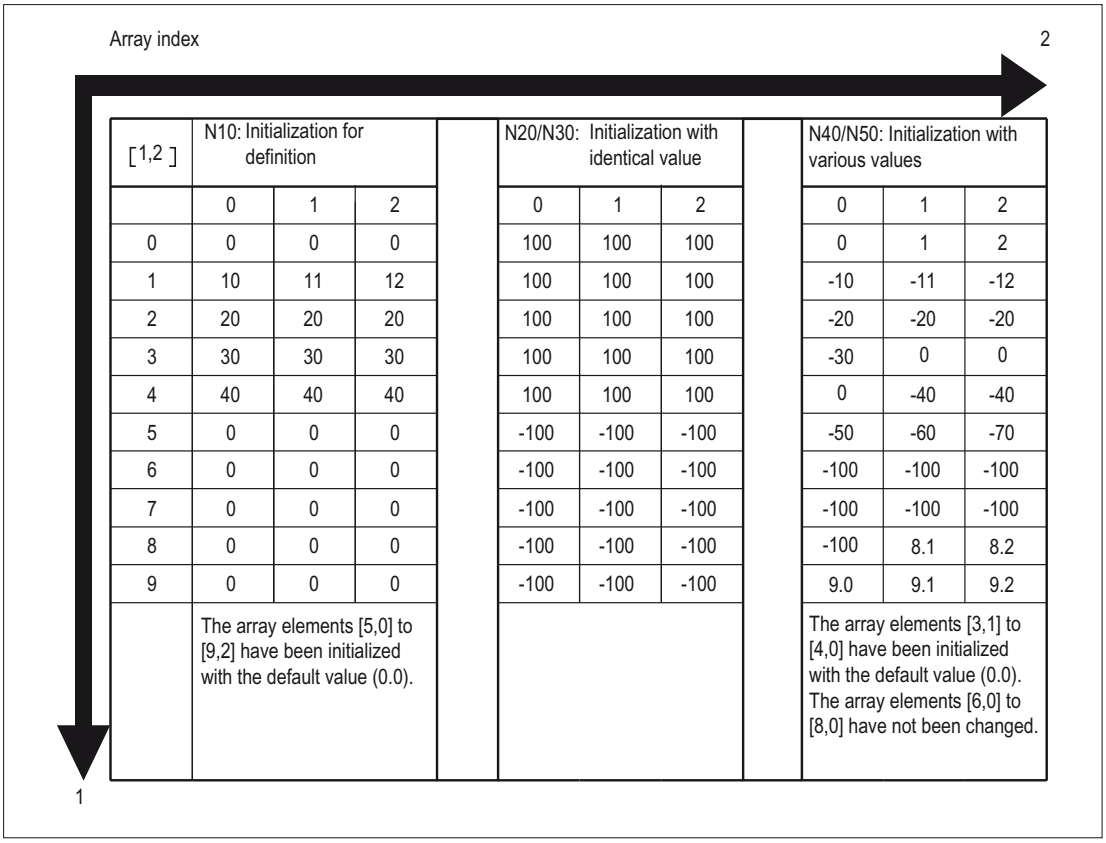
```
FOR n=0 TO 1
  FOR m=0 TO 2
    FOR o=0 TO 3
      FELD[n,m,o] = 1
    ENDFOR
  ENDFOR
ENDFOR
```

Example: Initializing complete variable arrays

For the actual assignment, refer to the diagram.

Program code

```
N10 DEF REAL FELD1[10,3]=SET(0,0,0,10,11,12,20,20,20,30,30,30,40,40,40,)
N20 ARRAY1[0,0] = REP(100)
N30 ARRAY1[5,0] = REP(-100)
N40 FELD1[0,0]=SET(0,1,2,-10,-11,-12,-20,-20,-20,-30, , , ,-40,-40,-50,-60,-70)
N50 FELD1[8,1]=SET(8.1,8.2,9.0,9.1,9.2)
```



[1,2]	N10: Initialization for definition		
	0	1	2
0	0	0	0
1	10	11	12
2	20	20	20
3	30	30	30
4	40	40	40
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
	The array elements [5,0] to [9,2] have been initialized with the default value (0.0).		

N20/N30: Initialization with identical value		
0	1	2
100	100	100
100	100	100
100	100	100
100	100	100
100	100	100
-100	-100	-100
-100	-100	-100
-100	-100	-100
-100	-100	-100
-100	-100	-100

N40/N50: Initialization with various values		
0	1	2
0	1	2
-10	-11	-12
-20	-20	-20
-30	0	0
0	-40	-40
-50	-60	-70
-100	-100	-100
-100	-100	-100
-100	8.1	8.2
9.0	9.1	9.2
The array elements [3,1] to [4,0] have been initialized with the default value (0.0). The array elements [6,0] to [8,0] have not been changed.		

See also

Definition and initialization of array variables (DEF, SET, REP): Further Information (Page 54)

Variables (Page 19)

2.1.13 Definition and initialization of array variables (DEF, SET, REP): Further Information

Further information (SET)

initialization during definition

- Starting with the 1st array element, as many array elements are assigned with the values from the value list as there are elements programmed in the value list.
- A value of 0 is assigned to array elements without explicitly declared values in the value list (gaps in the value list).
- For variables of the AXIS data type, gaps in the value list are not permitted.
- If the value list contains more values than there are array elements defined, an alarm will be displayed.

Value assignment in program execution

In the case of value assignment in program execution, the rules described above for definition apply. The following options are also supported:

- Expressions are also permitted as elements in the value list.
- Value assignment starts with the programmed array index. Values can be assigned selectively to subarrays.

Example:

Program code	Comments
`DEF INT ARRAY[5,5]`	`; Array definition`
`ARRAY[0,0]=SET(1,2,3,4,5)`	`; Value assignment to the first 5 array elements [0,0] - [0,4]`
`FELD[0,0]=SET(1,2, , ,5)`	`; Value assignment with gap to the first 5 array elements [0,0] - [0,4], array elements[0,2] and [0,3] = 0`
`ARRAY[2,3]=SET(VARIABLE,4*5.6)`	`; Value assignment with variable and expression starting at array index [2,3]: [2,3] = VARIABLE [2,4] = 4 * 5.6 = 22.4`

Further information (`REP`)

initialization during definition

- All or the optionally specified number of array elements are initialized with the specified value (constant).
- Variables of the FRAME data type cannot be initialized.

Example:

Program code	Comments
`DEF REAL varName[10]=REP(3.5,4)`	`; Initialize array definition and array elements [0] to [3] with value 3.5.`

Value assignment in program execution

In the case of value assignment in program execution, the rules described above for definition apply. The following options are also supported:

- Expressions are also permitted as elements in the value list.
- Value assignment starts with the programmed array index. Values can be assigned selectively to subarrays.

Examples:

Program code	Comments
`DEF REAL varName[10]`	`; Array definition`
`varName[5]=REP(4.5,3)`	`; Array elements [5] to [7] = 4.5`
`R10=REP(2.4,3)`	`; R-parameters R10 to R12 = 2.4`
`DEF FRAME FRM[10]`	`; Array definition`

Program code	Comments
`FRM[5] = REP(CTRANS (X,5))`	`; Array elements [5] to [9] = CTRANS(X,5)`

See also

Definition and initialization of array variables (DEF, SET, REP) (Page 50)

2.1.14 Data types

The following data types are available in the NC:

Data type	Meaning	Value Range
INT	Integer with sign	-2147483648 ... +2147483647
REAL	Real number (LONG REAL to IEEE)	±(~2,2*10^{-308} … ~1,8*10^{+308})
BOOL	Truth value TRUE (1) and FALSE (0)	1, 0
CHAR	ASCII character	ASCII code 0 to 255
STRING	Character string of a defined length	Maximum of 200 characters (no special characters)
AXIS	Axis/spindle identifier	Channel axis identifier
FRAME	Geometric parameters for static coordinate transformation (translation, rotation, scaling, mirroring)	---

Implicit data type conversions

The following data type conversions are possible and are performed implicitly during assignments and parameter transfers:

from ↓/ to →	REAL	INT	BOOL
REAL	x	o	&
INT	x	x	&
BOOL	x	x	x
x : Possible without restrictions o: Data loss possible due to the range of values being overshot ⇒ alarm; rounding: decimal place value ≥ 0.5 ⇒ round up, decimal place value < 0.5 ⇒ round down &: value ≠ 0 ⇒ TRUE, value== 0 ⇒ FALSE			

See also

Variables (Page 19)

2.1.15 Check availability of a variable (ISVAR)

The predefined ISVAR function can be used to check whether a system/user variable (e.g. machine data, setting data, system variable, general variables such as GUD) is known in the NCK.

Syntax

```
<Result>=ISVAR(<variable>)
```

The transfer parameter `<variable>` can be set up as follows:

- Undimensioned $ variable: `$<identifier>`
- One-dimensional $ variable without array index: `$<identifier>[ ]`
- One-dimensional $ variable with array index: `$<identifier>[<n>]`
- Two-dimensional $ variable without array index: `$<identifier>[,]`
- Two-dimensional $ variable with array index: `$<identifier>[<n>,<m>]`

Meaning

`<result>`:	Return value		
	Data type:	BOOL	
	Range of values:	1	Variable available
		0	Variable unknown
`ISVAR`:	Checks whether system/user variable is known in the NCK		
`<identifier>`:	Name of the system/user variables		
	Data type:	STRING	
`<n>`:	Array index for first dimension		
	Data type:	INT	
`<m>`:	Array index for second dimension		
	Data type:	INT	

The following checks are made in accordance with the transfer parameter:

- Does the identifier exist
- Is it a 1- or 2-dimensional array
- Is the array index permitted

Only if all these checks have a positive result will TRUE (1) be returned. If a check has a negative result or if a syntax error has occurred, it will return FALSE (0).

Examples

```
Program code                              Comment
DEF INT VAR1
DEF BOOL IS_VAR=FALSE
N10 IS_VAR=ISVAR("VAR1")                  ; IS_VAR is in this case TRUE.
```

Program code	Comment
DEF REAL VARARRAY[10,10]	
DEF BOOL IS_VAR=FALSE	
N10 IS_VAR=ISVAR("VARARRAY[,]")	; IS_VAR is in this case TRUE, is a two-dimensional array.
N20 IS_VAR=ISVAR("VARARRAY")	; IS_VAR is TRUE, variable exists.
N30 IS_VAR=ISVAR("VARARRAY[8,11]")	; IS_VAR is FALSE, array index is not permitted.
N40 IS_VAR=ISVAR("VARARRAY[8,8")	; IS_VAR is FALSE, "]" missing (syntax error).
N50 IS_VAR=ISVAR("VARARRAY[,8]")	; IS_VAR is TRUE, array index is permitted.
N60 IS_VAR=ISVAR("VARARRAY[8,]")	; IS_VAR is TRUE, array index is permitted.

Program code	Comment
DEF BOOL IS_VAR=FALSE	
N100 IS_VAR=ISVAR("$MC_GCODE_RESET_VALUES[1]"	; Transfer parameter is a machine data item, IS_VAR is TRUE.

Program code	Comment
DEF BOOL IS_VAR=FALSE	
N10 IS_VAR=ISVAR("$P_EP")	; IS_VAR is in this case TRUE.
N20 IS_VAR=ISVAR("$P_EP[X]")	; IS_VAR is in this case TRUE.

2.1.16 Read attribute values / data type (GETVARPHU, GETVARAP, GETVARLIM, GETVARDFT, GETVARTYP)

The attribute values of system/user variables can be read with the predefined GETVARPHU, GETVARAP, GETVARLIM and GETVARDFT functions, the data type of a system/user variable with GETVARTYP.

Read physical unit

Syntax:
<Result>=GETVARPHU(<name>)

Meaning:

<result>:	Numeric value of the physical unit		
	Data type:	INT	
	Range of values:	See Table in "Attribute: Physical unit (PHU) (Page 40)"	
		In case of fault	
		- 2	The specified <name> has not been assigned to a system parameter or a user variable.
GETVARPHU:	Reading of the physical unit of a system/user variable		

<name>:	Name of the system/user variables	
	Data type:	STRING

Example:

The NCK contains the following GUD variables:

DEF CHAN REAL PHU 42 LLI 0 ULI 10000 electric

```
Program code                    Comment
DEF INT result=0
result=GETVARPHU("elec-         ; Determine the physical unit of the GUD variables.
tric")
IF (result < 0) GOTOF error
```

The value 42 is returned as result. This corresponds to the physical unit [kW].

Note

GETVARPHU can be used, for example, to check whether both variables have the expected physical units in a variable assignment a = b.

Read access right

Syntax:
<Result>=GETVARAP(<name>,<access>)

Meaning:

<result>:	Protection level for the specified <access>		
	Data type:	INT	
	Range of values:	0 ... 7	See "Attribute: Access rights (APR, APW, APRP, APWP, APRB, APWB) (Page 43)".
		In case of fault	
		- 2	The specified <name> has not been assigned to a system parameter or a user variable.
		- 3	Incorrect value for <access>
GETVARAP:	Reading of the access right to a system/user variable		
<name>:	Name of the system/user variables		
	Data type:	STRING	
<access>:	Type of access		
	Data type:	STRING	
	Range of values:	"RP":	Read via part program
		"WP":	Write via part program
		"RB":	Read via OPI
		"WB":	Write via OPI

Example:

Program code	Comment
`DEF INT result=0`	
`result=GETVARAP("$TC_MAP8","WB")`	`; Determine the access protection for the system parameter "magazine position" with regard to writing via OPI.`
`IF (result < 0) GOTOF error`	

The value 7 is returned as result. This corresponds to the key switch position 0 (= no access protection).

Note

GETVARAP can be used, for example, to implement a checking program that checks the access rights expected by the application.

Read limit values

Syntax:
`<Status>=GETVARLIM(<name>,<limit value>,<result>)`

Meaning:

`<Status>`:	Function status		
	Data type:	INT	
	Range of values:	1	OK
		-1	No limit value defined (for variables of type AXIS, STRING, FRAME)
		-2	The specified `<name>` has not been assigned to a system parameter or a user variable.
		-3	Incorrect value for `<limit value>`
`GETVARLIM`:	Reading of the lower/upper limit value of a system/user variables		
`<name>`:	Name of the system/user variables		
	Data type:	STRING	
`<limit value>`:	specifies which limit value should be read out		
	Data type:	CHAR	
	Range of values:	`"L"`:	= lower limit value
		`"U"`:	= upper limit value
`<result>`:	Return of the limit value		
	Data type:	VAR REAL	

Example:

Program code	Comment
DEF INT state=0	
DEF REAL result=0	
state=GETVARLIM("$MA_MAX_AX_VELO","L",result)	Determine the lower limit value for MD32000 $MA_MAX_AX_VELO.
IF (result < 0) GOTOF error	

Read default value

Syntax:
<Status>=GETVARDFT(<name>,<result>[,<index_1>,<index_2>,<index_3>])

Meaning:

<Status>:	Function status		
	Data type:	INT	
	Range of values:	1	OK
		-1	No default value available (e.g. because <result> has the wrong type for <name>)
		-2	The specified <name> has not been assigned to a system parameter or a user variable.
		-3	Incorrect value for <index_1>, dimension less than one (= no array = scalar)
		-4	Incorrect value for <index_2>
		-5	Incorrect value for <index_3>
GETVARDFT:	Reading of the default value of a system/user variable		
<name>:	Name of the system/user variables		
	Data type:	STRING	
<result>:	Return of the default value		
	Data type:	VAR REAL (when reading the default value of variables of the types INT, REAL, BOOL, AXIS)	
		VAR STRING (when reading the default value of variables of the types STRING and CHAR)	
		VAR FRAME (when reading the default value of variables of the type FRAME)	
<index_1>:	Index to the first dimension (optional)		
	Data type:	INT	
	Not programmed means = 0		
<index_2>:	Index to the second dimension (optional)		
	Data type:	INT	
	Not programmed means = 0		

<index_3>:	Index to the third dimension (optional)	
	Data type:	INT
	Not programmed means = 0	

Example:

```
Program code                                    Comment
DEF INT state=0
DEF REAL resultR=0                              ; Variable to accept the default values
                                                of the types INT, REAL, BOOL, AXIS.
DEF FRAME resultF=0                             ; Variable to accept the default values
                                                of the type FRAME


IF (GETVARTYP("$MA_MAX_AX_VELO") <> 4)
GOTOF error
state=GETVARDFT("$MA_MAX_AX_VELO",              ; Determine the default value of the
resultR, AXTOINT(X))                            "X" axis.
IF (result < 0) GOTOF error


IF (GETVARTYP("$TC_TP8") <> 3) GOTOF error
state=GETVARDFT("$TC_TP8", resultR)


IF (GETVARTYP("$P_UBFR")  <> 7) GOTOF error
state=GETVARDFT("$P_UBFR", resultF )
```

Read data type

Syntax:
<Result>=GETVARTYP(<name>)

Meaning:

<result>:	Data type of the specified system/user variables		
	Data type:	INT	
	Range of values:	1	= BOOL
		2	= CHAR
		3	= INT
		4	= REAL
		5	= STRING
		6	= AXIS
		7	= FRAME
		In case of fault	
		< 0	The specified <name> has not been assigned to a system parameter or a user variable.
GETVARTYP:	Reading of the data type of a system/user variable		
<name>:	Name of the system/user variables		
	Data type:	STRING	

Example:

```
Program code                    Comment
DEF INT result=0
DEF STRING name="R"
result=GETVARTYP(name)          ; Determine the type of the R parameter.
IF (result < 0) GOTOF error
```

The value 4 is returned as result. This corresponds to the REAL data type.

2.2 Indirect programming

2.2.1 Indirectly programming addresses

When indirectly programming addresses, the extended address (<index>) is replaced by a variable with a suitable type.

Note

It is not possible to indirectly program addresses for:

- N (block number)
- L (subprogram)
- Settable addresses
  (e.g. X[1] instead of X1 is not permissible)

Syntax

```
<ADDRESS>[<Index>]
```

Meaning

<ADDRESS>[...]:	Fixed address with extension (index)
<index>:	Variable, e.g. for spindle number, axis, ...

Examples

Example 1: Indirectly programming a spindle number

Direct programming:

```
Program code                    Comment
S1=300                          ; Speed in rpm for the spindle number 1.
```

Indirect programming:

Program code	Comment
DEF INT SPINU=1	; Defining variables, type INT and value assignment.
S[SPINU]=300	; Speed 300 rpm for the spindle, whose number is saved in the SPINU variable (in this example 1, the spindle with the number 1).

Example 2: Indirectly programming an axis

Direct programming:

Program code	Comment
FA[U]=300	; Feedrate 300 for axis "U".

Indirect programming:

Program code	Comment
DEF AXIS AXVAR2=U	; Defining a variable, type AXIS and value assignment.
FA[AXVAR2]=300	; Feedrate of 300 for the axis whose address name is saved in the variables with the name AXVAR2.

Example 3: Indirectly programming an axis

Direct programming:

Program code	Comment
$AA_MM[X]	; Read probe measured value (MCS) of axis "X".

Indirect programming:

Program code	Comment
DEF AXIS AXVAR3=X	; Defining a variable, type AXIS and value assignment.
$AA_MM[AXVAR3]	; Read probe measured value (MCS) whose name is saved in the variables AXVAR3.

Example 4: Indirectly programming an axis

Direct programming:

Program code
X1=100 X2=200

Indirect programming:

Program code	Comment
DEF AXIS AXVAR1 AXVAR2	; Defining two type AXIS variables.
AXVAR1=(X1) AXVAR2=(X2)	; Assigning the axis names.
AX[AXVAR1]=100 AX[AXVAR2]=200	; Traversing the axes whose address names are saved in the variables with the names AXVAR1 and AXVAR2

Example 5: Indirectly programming an axis

Direct programming:

Program code
G2 X100 I20

Indirect programming:

Program code	Comment
DEF AXIS AXVAR1=X	; Defining a variable, type AXIS and value assignment.
G2 X100 IP[AXVAR1]=20	; Indirect programming the center point data for the axis, whose address name is saved in the variable with the name AXVAR1.

Example 6: Indirectly programming array elements

Direct programming:

Program code	Comment
DEF INT ARRAY1[4,5]	; Defining array 1

Indirect programming:

Program code	Comment
DEFINE DIM1 AS 4	; For array dimensions, array sizes must be specified as fixed values.
DEFINE DIM2 AS 5	
DEF INT ARRAY[DIM1,DIM2]	
ARRAY[DIM1-1,DIM2-1]=5	

Example 7: Indirect subprogram call

Program code	Comment
CALL "L" << R10	; Call the program, whose number is located in R10 (string cascading).

2.2.2 Indirectly programming G codes

Indirect programming of G codes permits cycles to be effectively programmed.

Syntax

```
G[<group>]=<number>
```

Meaning

G[...]:	G command with extension (index)	
<group>:	Index parameter: G function group	
	Type:	INT

<number>:	Variable for the G code number	
	Type:	INT or REAL

Note

Generally, only G codes that do not determine the syntax can be indirectly programmed.

Only G function group 1 is possible from the G codes that determine the syntax.
The syntax-determining G codes of G function groups 2, 3 and 4 are not possible.

Note

Arithmetic functions are not permitted in the indirect G code programming. If it is necessary to calculate the G code number, this must be done in a separate part program line before the indirect G code programming.

Examples

Example 1: Settable zero offset (G function group 8)

```
Program code                                Comment
N1010 DEF INT INT_VAR
N1020 INT_VAR=2
...
N1090 G[8]=INT_VAR G1 X0 Y0                 ;G54
N1100 INT_VAR=INT_VAR+1                     ; G code calculation
N1110 G[8]=INT_VAR G1 X0 Y0                 ;G55
```

Example 2: Level selection (G function group 6)

```
Program code                                Comment
N2010 R10=$P_GG[6]                          ; Read active G function of G function
                                            group 6
...
N2090 G[6]=R10
```

References

For information on the G function groups, see:
Programming Manual, Fundamentals; Section "G function groups"

2.2.3 Indirectly programming position attributes (GP)

Position attributes, e.g. the incremental or absolute programming of the axis position, can be indirectly programmed as variables in conjunction with the key word GP.

Application

The indirect programming of position attributes is used in **replacement cycles**, as in this case, the following advantage exists over programming position attributes as keyword (e.g. `IC`, `AC`, ...):

As a result of the indirect programming as variable, **no** `CASE` statement is required, which would otherwise branch for all possible position attributes.

Syntax

```
<POSITIONING COMMAND>[<axis/spindle>]=
GP(<position>,<position attribute)
<axis/spindle>=BP(<position>,<position attribute)
```

Meaning

`<POSITIONING COMMAND>[]`:	The following positioning commands can be programmed together with the key word GP: `POS`, `POSA`, `SPOS`, `SPOSA` Also possible: • All axis and spindle identifiers present in the channel: `<axis/spindle>` • Variable axis/spindle identifier `AX`
`<axis/spindle>`:	Axis/spindle that is to be positioned
`GP()`:	Key word for positioning
`<position>`:	Parameter 1 Axis/spindle position as constant or variable
`<position attribute>`:	Parameter 2 Position attribute (e.g. position approach mode as a variable (e.g. $P_SUB_SPOSMODE) or as key word (IC, AC, ...)

The values supplied from the variables have the following significance:

Value	Meaning	Permissible for:
0	No change to the position attribute	
1	AC	`POS`, `POSA`, `SPOS`, `SPOSA`, `AX`, axis address
2	IC	`POS`, `POSA`, `SPOS`, `SPOSA`, `AX`, axis address
3	DC	`POS`, `POSA`, `SPOS`, `SPOSA`, `AX`, axis address
4	ACP	`POS`, `POSA`, `SPOS`, `SPOSA`, `AX`, axis address
5	ACN	`POS`, `POSA`, `SPOS`, `SPOSA`, `AX`, axis address
6	OC	-
7	PC	-
8	DAC	`POS`, `POSA`, `AX`, axis address
9	DIC	`POS`, `POSA`, `AX`, axis address
10	RAC	`POS`, `POSA`, `AX`, axis address
11	RIC	`POS`, `POSA`, `AX`, axis address
12	CAC	`POS`, `POSA`

Value	Meaning	Permissible for:
13	CIC	POS, POSA
14	CDC	POS, POSA
15	CACP	POS, POSA
16	CACN	POS, POSA

Example

For an active synchronous spindle coupling between the leading spindle S1 and the following spindle S2, the following replacement cycle to position the spindle is called using the SPOS command in the main program.

Positioning is realized using the statement in N2230:
SPOS[1]=GP($P_SUB_SPOSIT,$P_SUB_SPOSMODE) SPOS[2]=GP($P_SUB_SPOSIT, $P_SUB_SPOSMODE)

The position to be approached is read from the system variable $P_SUB_SPOSIT; the position approach mode is read from the system variable $P_SUB_SPOSMODE.

```
Program code                                    Comment
N1000 PROC LANG_SUB DISPLOF SBLOF
...
N2100 IF($P_SUB_AXFCT==2)
N2110                                           ; Replacement of the SPOS / SPOSA / M19
                                                command for an active synchronous spindle
                                                coupling
N2185 DELAYFSTON                                ; Start of stop delay area
N2190 COUPOF(S2,S1)                             ; Deactivate synchronous spindle coupling
N2200                                           ; Position leading and following spindles
N2210 IF($P_SUB_SPOS==TRUE) OR
($P_SUB_SPOSA==TRUE)
N2220                                           ; Positioning the spindle with SPOS:
N2230    SPOS[1]=GP($P_SUB_SPOSIT,
$P_SUB_SPOSMODE)
         SPOS[2]=GP($P_SUB_SPOSIT,
$P_SUB_SPOSMODE)
N2250 ELSE
N2260                                           ; Positioning the spindle using M19:
N2270 M1=19 M2=19                               ; Position leading and following spindles
N2280 ENDIF
N2285 DELAYFSTOF                                ; End of stop delay area
N2290 COUPON(S2,S1)                             ; Activate synchronous spindle coupling
N2410 ELSE
N2420                                           ; Query on further replacements
...
N3300 ENDIF
...
N9999 RET
```

Supplementary conditions

- The indirect programming of position attributes is not possible in synchronized actions.

References

Function Manual Basic Functions; BAG, Channel, Program Operation, Reset Response (K1), Section: Replacement of NC functions by subprograms

2.2.4 Indirectly programming part program lines (EXECSTRING)

Using the part program command `EXECSTRING`, it is possible to execute a previously generated string variable as part program line.

Syntax

`EXECSTRING` is programmed in a separate part program line:
`EXECSTRING (<string_variable>)`

Meaning

`EXECSTRING`:	Command to execute a string variable as part program line
`<string variable>`:	Type STRING variable, that includes the actual part program line to be executed

Note

With `EXECSTRING`, all part program constructions that can be programmed in the **program section of a part program**, with the exception of control structures (Page 105), can be extracted. This means that `PROC` and `DEF` statements are excluded as well as the general use in INI and DEF files.

Example

```
Program code                          Comment
N100 DEF STRING[100] MY_BLOCK         ; Definition of string variables to accept the part
                                      program line to be executed.
N110 DEF STRING[10] MFCT1="M7"
...
N200 EXECSTRING(MFCT1 <<              ; Execute part program line "M7 M4711".
"M4711")
...
N300 R10=1
N310 MY_BLOCK="M3"
N320 IF(R10)
```

Program code	Comment
N330 MY_BLOCK = MY_BLOCK << MFCT1	
N340 ENDIF	
N350 EXECSTRING(MY_BLOCK)	; Execute part program line "M3 M7".

2.3 Arithmetic functions

Operator / arithmetic function	Meaning
+	Addition
–	Subtraction
*	Multiplication
/ [1]	Division [1]
DIV [1]	Integer number division [1]
MOD [1]	Modulo division (supplies the remainder of the integer number division) [1]
:	Chain operator for FRAME variables
SIN()	Sine
COS()	Cosine
TAN()	Tangent
ASIN()	Arc sine
ACOS()	Arc cosine
ATAN2(,) [1]	Arc tangent2 [1]
SQRT()	Square root
ABS()	Absolute value
POT()	2. power (square)
TRUNC()	Integer component The accuracy for comparison commands can be set using TRUNC (see "Precision correction on comparison errors (TRUNC) (Page 74)")
ROUND()	Round to integer
LN()	Natural logarithm
EXP()	Exponential function
MINVAL ()	Lower value of two variables (see "Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND) (Page 75)")
MAXVAL ()	Larger value of two variables (see "Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND) (Page 75)")
BOUND ()	Variable value within the defined value range (see "Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND) (Page 75)")
CTRANS()	Offset
CROT ()	Rotation

CSCALE()	Change of scale
CMIRROR()	Mirroring
1) See the paragraph, "Examples"	

Programming

The usual mathematical notation is used for arithmetic functions. Priorities for execution are indicated by parentheses. Angles are specified for trigonometry functions and their inverse functions (right angle = 90°).

Examples

Division: /

(type REAL) = type INT or type REAL) / (type INT or type REAL);

Example: 3 / 4 = 0.75

Integer number division: DIV

(type INT) = (type INT or REAL) / (type INT or REAL);

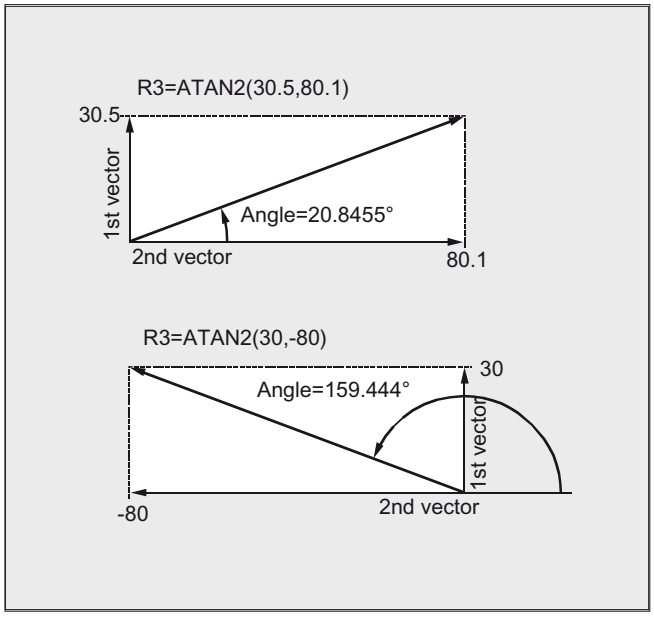
Example: 7 DIV 4.1 = 1

Modulo division (supplies the remainder of the integer number division): MOD

(type REAL) = (type INT or REAL) MOD (type INT or REAL);

Example: 7 MOD 4.1 = 2.9

Arc tangent 2: ATAN2



The arithmetic function ATAN2 calculates the angle of the total vector from two mutually perpendicular vectors.

The result is in one of four quadrants (-180° < 0 < +180°).

The angular reference is always based on the 2nd value in the positive direction.

Programming examples

Program code	Comment
R1=R1+1	; New R1 = old R1 + 1
R1=R2+R3 R4=R5-R6 R7=R8*R9	
R10=R11/R12 R13=SIN(25.3)	
R14=R1*R2+R3	; Multiplication or division takes precedence over addition or subtraction.
R14=(R1+R2)*R3	; Expressions and parentheses are calculated first.
R15=SQRT(POT(R1)+POT(R2))	; Inner parentheses are resolved first: R15 = square root of ((R1^2 + R2^2))
RESFRAME=FRAME1:FRAME2	; FRAME logic operation with chain operator
FRAME3=CTRANS(…):CROT(…)	Value assignment at a FRAME component

2.4 Comparison and logic operations

Comparison operations can be used, for example, to formulate a jump condition. Complex expressions can also be compared.

The comparison operations are applicable to variables of type CHAR, INT, REAL and BOOL. The code value is compared with the CHAR type.
For types STRING, AXIS and FRAME, the following are possible: == and <>, which can be used for STRING type operations, even in synchronous actions.

The result of comparison operations is always of BOOL type.

Logic operators are used to link truth values.

The logical operations can only be applied to type BOOL variables. However, they can also be applied to the CHAR, INT and REAL data types via internal type conversion.

For the logic (Boolean) operations, the following applies to the BOOL, CHAR, INT and REAL data types:

- 0 corresponds to: FALSE
- Not equal to 0 means: TRUE

Bit-by-bit logic operators

Logic operations can also be applied to single bits of types CHAR and INT. Type conversion is automatic.

Programming

Relational operator	Meaning
==	Equal to
<>	Not equal to
>	Greater than
<	Less than

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

Logic operator	Meaning
AND	AND
OR	OR
NOT	Negation
XOR	Exclusive OR

Bit-by-bit logic operator	Meaning
B_AND	Bit-by-bit AND
B_OR	Bit-by-bit OR
B_NOT	Bit-by-bit negation
B_XOR	Bit-by-bit exclusive OR

Note

In arithmetic expressions, the execution order of all the operators can be specified by parentheses, in order to override the normal priority rules.

Note

Spaces must be left between BOOLEAN operands and operators.

Note

The operator `B_NOT` only refers to one operand. This is located after the operator.

Examples

Example 1: Comparison operators

```
IF R10>=100 GOTOF DEST
```

or

```
R11=R10>=100
IF R11 GOTOF DEST
```

The result of the `R10>=100` comparison is first buffered in `R11`.

Example 2: Logic operators

```
IF (R10<50) AND ($AA_IM[X]>=17.5) GOTOF DESTINATION
```

or

```
IF NOT R10 GOTOB START
```

`NOT` only refers to one operand.

Example 3: Bit-by-bit logic operators

```
IF $MC_RESET_MODE_MASK B_AND 'B10000' GOTOF  ACT_PLANE
```

2.5 Precision correction on comparison errors (TRUNC)

The TRUNC command truncates the operand multiplied by a precision factor.

Settable precision for comparison commands

Program data of type REAL is displayed internally with 64 bits in IEEE format. This display format can cause decimal numbers to be displayed imprecisely and lead to unexpected results when compared with the ideally calculated values.

Relative equality

To prevent the imprecision caused by the display format from interfering with program flow, the comparison commands do not check for absolute equality, but rather for relative equality.

Syntax

Precision correction on comparison errors

```
TRUNC (R1*1000)
```

Meaning

TRUNC:	Truncate decimal places

Relative quality of 10^{-12} taken into account for

- Equality: (==)
- Inequality: (<>)
- Greater than or equal to: (>=)
- Less than or equal to: (<=)
- Greater/less than: (><) with absolute equality
- Greater than: (>)
- Less than: (<)

Compatibility

For compatibility reasons, the check for relative quality for (>) and (<) can be deactivated by setting machine data MD10280 $MN_ PROG_FUNCTION_MASK Bit0 = 1.

Note

Comparisons with data of type REAL are subject to a certain imprecision for the above reasons. If deviations are unacceptable, use INTEGER calculation by multiplying the operands by a precision factor and then truncating with TRUNC.

Synchronized actions

The response described for the comparison commands also applies to synchronized actions.

Examples

Example 1: Precision considerations

Program code	Comments
N40 R1=61.01 R2=61.02 R3=0.01	;Assignment of initial values
N41 IF ABS(R2-R1) > R3 GOTOF ERROR	; Jump would have been executed up until now
N42 M30	; End of program
N43 ERROR: SETAL(66000)	
R1=61.01 R2=61.02 R3=0.01	;Assignment of initial values
R11=TRUNC(R1*1000) R12=TRUNC(R2*1000)	; Accuracy correction
R13=TRUNC(R3*1000)	
IF ABS(R12-R11) > R13 GOTOF ERROR	; Jump is no longer executed
M30	; End of program
ERROR: SETAL(66000)	

Example 2: Calculate and evaluate the quotient of both operands

Program code	Comments
R1=61.01 R2=61.02 R3=0.01	;Assignment of initial values
IF ABS((R2-R1)/R3)-1) > 10EX-5 GOTOF ERROR	; Jump is not executed
M30	; End of program
ERROR: SETAL(66000)	

2.6 Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND)

The MINVAL and MAXVAL commands compare the values of two variables. The smaller value (in the case of MINVAL) or the larger value (in the case of MAXVAL) respectively is delivered as a result.

The BOUND command tests whether the value of a test variable falls within a defined range of values.

Syntax

```
<smaller value>=MINVAL(<variable1>,<variable2>)
<larger value>=MAXVAL(<variable1>,<variable2>)
<return value>=<BOUND>(<minimum>,<maximum>,<test variable>)
```

Meaning

MINVAL:	Obtains the **smaller** value of two variables (<variable1>, <variable2>)
<smaller value>:	Result variable for the MINVAL command Set to the smaller variable value.
MAXVAL:	Obtains the **larger** value of two variables (<variable1>, <variable2>)
<larger value>:	Result variable for the MAXVAL command Set to the larger variable value.
BOUND:	Tests whether a variable (<test variable) is within a defined range of values.
<minimum>:	Variable which defines the minimum value of the range of values.
<maximum>:	Variable which defines the maximum value of the range of values.
<return value>:	Result variable for the BOUND command If the value of the test variable is within the defined range of values, the result variable is set to the value of the test variable. If the value of the test variable is greater than the maximum value, the result variable is set to the maximum value of the definition range. If the value of the test variable is less than the minimum value, the result variable is set to the minimum value of the definition range.

Note

MINVAL, MAXVAL, and BOUND can also be programmed in synchronized actions.

Note

Behavior if values are equal

If the values are equal, MINVAL/MAXVAL are set to this equal value. In the case of BOUND the value of the variable to be tested is returned again.

Example

```
Program code                                        Comment
DEF REAL rVar1=10.5, rVar2=33.7, rVar3, rVar4, rVar5, rValMin, rValMax, rRetVar
rValMin=MINVAL(rVar1,rVar2)                         ; rValMin is set to value 10.5.
rValMax=MAXVAL(rVar1,rVar2)                         ; rValMax is set to value 33.7.
rVar3=19.7
rRetVar=BOUND(rVar1,rVar2,rVar3)                    ; rVar3 is within the limits, rRetVar is set to 19.7.
rVar3=1.8
rRetVar=BOUND(rVar1,rVar2,rVar3)                    ; rVar3 is below the minimum limit, rRetVar is set to
                                                    10.5.
rVar3=45.2
rRetVar=BOUND(rVar1,rVar2,rVar3)                    ; rVar3 is above the maximum limit, rRetVar is set to
                                                    33.7.
```

2.7 Priority of the operations

Each operator is assigned a priority. When an expression is evaluated, the operators with the highest priority are always applied first. Where operators have the same priority, the evaluation is from left to right.

In arithmetic expressions, the execution order of all the operators can be specified by parentheses, in order to override the normal priority rules.

Order of operators

From the highest to lowest priority

1.	NOT, B_NOT	Negation, bit-by-bit negation
2.	*, /, DIV, MOD	Multiplication, division
3.	+, –	Addition, subtraction
4.	B_AND	Bit-by-bit AND
5.	B_XOR	Bit-by-bit exclusive OR
6.	B_OR	Bit-by-bit OR
7.	AND	AND
8.	XOR	Exclusive OR
9.	OR	OR
10.	<<	Concatenation of strings, result type STRING
11.	==, <>, >, <, >=, <=	Comparison operators

Note

The concatenation operator ":" for Frames must not be used in the same expression as other operators. A priority level is therefore not required for this operator.

Example: IF statement

```
If (otto==10) and (anna==20) gotof end
```

2.8 Possible type conversions

Function

Type conversion on assignment

The constant numeric value, the variable, or the expression assigned to a variable must be compatible with the variable type. If this is the case, the type is automatically converted when the value is assigned.

Possible type conversions

to	REAL	INT	BOOL	CHAR	STRING	AXIS	FRAME
from							
REAL	yes	yes*	Yes[1])	Yes *	–	–	–
INT	yes	yes	Yes[1])	Yes [2)]	–	–	–
BOOL	yes	yes	yes	yes	yes	–	–
CHAR	yes	yes	Yes [1)]	yes	yes	–	–
STRING	–	–	Yes [4)]	Yes [3)]	yes	–	–
AXIS	–	–	–	–	–	yes	–
FRAME	–	–	–	–	–	–	yes

Explanation

* At type conversion from REAL to INT, fractional values that are >=0.5 are rounded up, others are rounded down (cf. ROUND function).

1) Value <> 0 is equivalent to TRUE; value == 0 is equivalent to FALSE

2) If the value is in the permissible range

3) If only 1 character

4) String length 0 = >FALSE, otherwise TRUE

Note

If conversion produces a value greater than the target range, an error message is output.

If mixed types occur in an expression, type conversion is automatic. Type conversions are also possible in synchronous actions, see Chapter "Motion-synchronous actions, implicit type conversion".

2.9 String operations

Sting operations

In addition to the classic operations "assign" and "comparison" the following string operations are possible:

- Type conversion to STRING (AXSTRING) (Page 79)
- Type conversion from STRING (NUMBER, ISNUMBER, AXNAME) (Page 80)
- Concatenation of strings (<<) (Page 81)
- Conversion to lower/upper case letters (TOLOWER, TOUPPER) (Page 82)
- Determine length of string (STRLEN) (Page 83)
- Search for character/string in the string (INDEX, RINDEX, MINDEX, MATCH) (Page 83)
- Selection of a substring (SUBSTR) (Page 84)

- Reading and writing of individual characters (Page 85)
- Formatting a string (SPRINT) (Page 87)

Special significance of the 0 character

Internally, the 0 character is interpreted as the end identifier of a string. If a character is replaced with the 0 character, the string is truncated.

Example:

```
Program code                                    Comment
DEF STRING[20] STRG="axis . stationary"
STRG[6]="X"
MSG(STRG)                                       ; Supplies the message "axis X sta-
                                                tionary".
STRG[6]=0
MSG(STRG)                                       ; Supplies the message "axis".
```

2.9.1 Type conversion to STRING (AXSTRING)

The function "type conversion to STRING" allows variables of different types to be used as a component of a message (MSG).

When using the << operator this is realized implicitly for data types INT, REAL, CHAR and BOOL (see " Concatenation of strings (<<) (Page 81) ").

An INT value is converted to normal readable format. REAL values convert with up to 10 decimal places.

Type AXIS variables can be converted to STRING using the AXSTRING command.

Syntax

```
<STRING_RES> = << <any_type>
<STRING_RES> = AXSTRING(<axis identifier>)
```

Meaning

<STRING_RES>:	Variable for the result of the type conversion	
	Type:	STRING
<any_type>:	Variable types INT, REAL, CHAR, STRING and BOOL	
AXSTRING:	The AXSTRING command supplies the specified axis identifier as string.	
<axis identifier>:	Variable for axis identifier	
	Type:	AXIS

Note

FRAME variables cannot be converted.

2.9.2 Type conversion from STRING (NUMBER, ISNUMBER, AXNAME)

A conversion is made from STRING to REAL using the NUMBER command. The ability to be converted can be checked using the ISNUMBER command.

A string is converted into the axis data type using the AXNAME command.

Syntax

```
<REAL_RES>=NUMBER("<string>")
<BOOL_RES>=ISNUMBER("<string>")
<AXIS_RES>=AXNAME("<string>")
```

Meaning

NUMBER:	The NUMBER command returns the number represented by the <string> as REAL value.		
<string>:	Type STRING variable to be converted		
<REAL_RES>:	Variable for the result of the type conversion with NUMBER		
	Type:	REAL	
ISNUMBER:	The ISNUMBER command checks whether the <string> can be converted into a valid number.		
<BOOL_RES>:	Variable for the result of the interrogation with ISNUMBER		
	Type:	BOOL	
	Value:	TRUE	ISNUMBER supplies the value TRUE, if the <string> represents a valid REAL number in compliance with the language rules.
		FALSE	If ISNUMBER supplies the value FALSE, when calling NUMBER with the same <string>, an alarm is initiated.
AXNAME:	The AXNAME command converts the specified <string> into an axis identifier. **Note:** If the <string> cannot be assigned a configured axis identifier, an alarm is initiated.		
<AXIS_RES>:	Variable for the result of the type conversion with AXNAME		
	Type:	AXIS	

Example

Program code	Comment
DEF BOOL BOOL_RES	

Program code	Comment
DEF REAL REAL_RES	
DEF AXIS AXIS_RES	
REAL_RES == 1234.9876Ex-7	; BOOL_RES == TRUE
BOOL_RES=ISNUMBER("1234XYZ")	; BOOL_RES == FALSE
REAL_RES=NUMBER("1234.9876Ex-7")	; REAL_RES == 1234.9876Ex-7
AXIS_RES=AXNAME("X")	; AXIS_RES == X

2.9.3 Concatenation of strings (<<)

The function "concatenation strings" allows a string to be configured from individual components.

The concatenation is realized using the operator "**<<**". This operator has STRING as the target type for all combinations of basic types CHAR, BOOL, INT, REAL, and STRING. Any conversion that may be required is carried out according to existing rules.

Syntax

```
<any_type> << <any_type>
```

Meaning

<any_type>:	Variable, type CHAR, BOOL, INT, REAL or STRING
<< :	Operator to chain variables (<any_type>) to configure a character string (type STRING). This operator is also available alone as a so-called "unary" variant. This can be used for explicit type converter to STRING (not for FRAME and AXIS): << <any_type>

For example, such a message or a command can be configured from text lists and parameters can be inserted (for example a block name):
`MSG(STRG_TAB[LOAD_IDX]<<BLOCK_NAME)`

Note

The intermediate results of string concatenation must not exceed the maximum string length.

Note

The FRAME and AXIS types cannot be used together with the operator "**<<**".

Examples

Example 1: Concatenation of strings

Program code	Comment
DEF INT IDX=2	
DEF REAL VALUE=9.654	
DEF STRING[20] STRG="INDEX:2"	
IF STRG=="Index:"<<IDX GOTOF NO_MSG	
MSG("Index:"<<IDX<<"/value:"<<VALUE)	; Display: "Index:2/value:9.654"
NO_MSG:	

Example 2: Explicit type conversion with <<

Program code	Comment
DEF REAL VALUE=3.5	
<<VALUE	; The specified REAL type variable is converted into a STRING type.

2.9.4 Conversion to lower/upper case letters (TOLOWER, TOUPPER)

The "conversion to lowercase/uppercase letters" function allows all of the letters of a string to be converted into a standard representation.

Syntax

```
<STRING_RES>=TOUPPER("<string>")
<STRING_RES>=TOLOWER("<string>")
```

Meaning

TOUPPER:	Using the TOUPPER command, all of the letters in a character string are converted into **uppercase** letters.	
TOLOWER:	Using the TOLOWER command, all of the letters in a character string are converted into **lowercase** letters.	
<string>:	Character string that is to be converted	
	Type:	STRING
<STRING_RES>:	Variable for the result of the conversion	
	Type:	STRING

Example

Because user inputs can be initiated on the user interface, they can be given standard capitalization (uppercase or lowercase):

```
Program code
DEF STRING [29] STRG
...
IF "LEARN.CNC"==TOUPPER(STRG) GOTOF LOAD_LEARN
```

2.9.5 Determine length of string (STRLEN)

The `STRLEN` command determines the length of a character string.

Syntax

```
<INT_RES>=STRLEN("<STRING>")
```

Meaning

`STRLEN`:	The `STRLEN` command determines the length of the specified character string. The number of characters that are not the 0 character, counting from the beginning of the string is returned.	
`<string>`:	Character string whose length is to be determined	
	Type:	STRING
`<INT_RES>`:	Variable for the result of the determination	
	Type:	INT

Example

In conjunction with the single character access, this function allows the end of a character string to be determined:

```
Program code
IF (STRLEN(BLOCK_NAME)>10) GOTOF ERROR
```

2.9.6 Search for character/string in the string (INDEX, RINDEX, MINDEX, MATCH)

This functionality searches for single characters or a string within a string. The function results specify where the character/string is positioned in the string that has been searched.

Syntax

`INT_RES=INDEX(STRING,CHAR)` ; **Result type: INT**

`INT_RES=RINDEX(STRING,CHAR)` ; **Result type: INT**

`INT_RES=MINDEX(STRING,STRING)` ; **Result type: INT**

`INT_RES=MINDEX(STRING,STRING)` ; **Result type: INT**

Semantics

Search functions: It supplies the position in the string (first parameter) where the search has been successful. If the character/string cannot be found, then the value -1 is returned. The first character has position 0.

Meaning

`INDEX`:	Searches for the character specified as second parameter (from the beginning) in the first parameter.
`RINDEX`:	Searches for the character specified as second parameter (from the end) in the first parameter.
`MINDEX`:	Corresponds to the INDEX function, except for the case that a list of characters is transferred (as string) in which the index of the first found character is returned.
`MATCH`:	Searches for a string in a string.

This allows strings to be broken up according to certain criteria, for example, at positions with blanks or path separators ("/").

Example

Breaking up an input into path and block names

```
Program code                                Comment
DEF INT PFADIDX, PROGIDX
DEF STRING[26] INPUT
DEF INT LISTIDX
INPUT = "/_N_MPF_DIR/_N_EXECUTE_MPF"
LISTIDX = MINDEX (INPUT, "M,N,O,P")         ; The value returned in LISTIDX is 3; because
+ 1                                         "N" is the first character in the parameter IN-
                                            PUT from the selection list starting from the
                                            beginning.
PFADIDX = INDEX (INPUT, "/") +1             ; Therefore the following applies: PFADIDX = 1
PROGIDX = RINDEX (INPUT, "/") +1            ; Therefore the following applies: PROGIDX = 12
                                            ; The SUBSTR function introduced in the next
                                            section can be used to break-up
                                            variable INPUT into the components "path" and
                                            "module":
VARIABLE = SUBSTR (INPUT,                   ; Then returns "_N_MPF_DIR"
PFADIDX, PROGIDX-PFADIDX-1)
VARIABLE = SUBSTR (INPUT, PROGIDX)          ; Then returns "_N_EXECUTE_MPF"
```

2.9.7 Selection of a substring (SUBSTR)

Arbitrary parts within a string can be read with the SUBSTRING function.

Syntax

```
<STRING_RES>=SUBSTR(<string>,<index>,<length>)
<STRING_RES>=SUBSTR(<string>,<index>)
```

Meaning

SUBSTR:	This function returns a substring from **<string>**, starting with **<index>** with the specified **<length>**. If the parameter **<length>** is not specified, the function returns a substring starting with **<index>** until the end of the string.
<index>:	Start position of the substring within the string. If the start position is after the end of the string, an empty string (" ") is returned. First character of the string: Index = 0 Range of values: 0 ... (string length - 1)
<length>:	Length of the substring. If too long a length is specified, only the substring up to the end of the string is returned. Range of values: 1 ... (string length - 1)

Example

Program code	**Comment**
DEF STRING[29] RES	
; 1	
; 01234567890123456789	
RES = SUBSTR("QUITTUNG: 10 to 99", 10, 2)	; RES == "10"
RES = SUBSTR("QUITTUNG: 10 to 99", 10)	; RES == "10 to 99"

2.9.8 Reading and writing of individual characters

Individual characters can be read and written within a string.

The following supplementary conditions must be observed:

- Only possible with user-defined variables, not with system variables
- Individual characters of a string are only transferred "call by value" for subprogram calls

Syntax

```
<Character>=<string>[<index>]
<Character>=<string_array>[<array_index>,<index>]
<String>[<index>]=<character>
<String_array>[<array_index>,<index>]=<character>
```

Meaning

<string>:	Any string
<character>:	Variable of type CHAR
<index>:	Position of the character within the string. First character of the string: Index = 0 Range of values: 0 ... (string length - 1)

Examples

Example 1: Variable message

```
Program code                                Comment
;                          0123456789
DEF STRING [50] MESSAGE = "Axis n has reached position"
MESSAGE [6] = "X"
MSG (MESSAGE)                               ; "Axis X has reached position"
```

Example 2: Evaluating a system variable

```
Program code                                Comment
DEF STRING[50] STRG                         ; Buffer for system variable
...
STRG = $P_MMCA                              ; Load system variable
IF STRG[0] == "E" GOTO ...                  ; Evaluating the system variable
```

Example 3: Parameter transfer "call by value" and "call by reference"

```
Program code                                Comment
;                        0123456
DEF STRING[50] STRG = "Axis X"
DEF CHAR CHR
...
EXTERN UP_VAL(ACHSE)                        ; Definition of subprogram with "call
                                            by value" parameters
EXTERN UP_REF(VAR ACHSE)                    ; Definition of subprogram with "call
                                            by reference" parameters
...
UP_VAL(STRG[6])                             ; Parameter transfer "by value"
...
CHR = STRG[6]                               ; Buffer
UP_REF(CHR)                                 ; Parameter transfer "by reference"
```

2.9.9 Formatting a string (SPRINT)

Using the pre-defined SPRINT function, character strings can be formatted and e.g. prepared for output on external devices (also see "Process DataShare - Output to an external device/ file (EXTOPEN, WRITE, EXTCLOSE): (Page 570)").

Syntax

```
"<Result_string>"=SPRINT("<Format_string>",<value_1>,<value_2>,...,
<value_n>)
```

Meaning

SPRINT:	Identifier for a pre-defined function that supplies a value, type STRING.
"<Format_String>":	Character string that contains fixed and variable elements. The variable elements are defined using the format control character % and a subsequent format description.
< value_1>,< value_2>,…,< value_n>:	Value in the form of a constant or NC variables, which is inserted at the location where the nth format control character % is located, corresponding to the format description in the <format_string>.
"<result_string>":	Formatted character string (maximum 400 bytes)

Format descriptions available

%B:	Conversion into the "TRUE" string, if the value to be converted:
	• Is not equal to 0.
	• Is not an empty string (for string values).
	Conversion into the "FALSE" string, if the value to be converted:
	• Is equal to 0.
	• Is an empty string.
	Example:
	`N10 DEF BOOL BOOL_VAR=1`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF BOOL_VAR:%B", BOOL_VAR)`
	Result: The character string "CONTENT OF BOOL_VAR:TRUE" is written to the RESULT string variable.

%C:	Conversion into an ASCII character.
	Example:
	`N10 DEF CHAR CHAR_VAR="X"`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF CHAR_VAR:%C",CHAR_VAR)`
	Result: The character string "CONTENT OF CHAR_VAR:X is written to the string variable RESULT.

%D:	Conversion into a string with an integer value (INTEGER). **Example:** N10 DEF INT INT_VAR=123 N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF INT_VAR:%D",INT_VAR) Result: The character string "CONTENT OF INT_VAR:123" is written to the string variable RESULT.
%<m>D:	Conversion into a string with an integer value (INTEGER). The string has a minimum length of <m> characters. The missing locations are filled with spaces, left-justified. **Example:** N10 DEF INT INT_VAR=-123 N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF INT_VAR:%6D",INT_VAR) Result: The character string "CONTENT OF INT_VAR:xx-123" is written to string variable RESULT ("x" in the example represents spaces).
%F:	Conversion into a string with a decimal number with 6 decimal places. Where relevant, the decimal places are rounded-off or filled with 0. **Example:** N10 DEF REAL REAL_VAR=-1.2341234EX+03 N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%F",REAL_VAR) Result: The string variable RESULT is written with the character string "CONTENT OF REAL_VAR: -1234.123400".
%<m>F:	Conversion into a string with a decimal number with 6 decimal places and a total length of at least <m> characters. Where relevant, the decimal places are rounded-off or filled with 0. Missing characters are filled up to the total length <m> using spaces, left-justified. **Example:** N10 DEF REAL REAL_VAR=-1.23412345678EX+03 N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%15F",REAL_VAR) Result: The string variable RESULT is written with the character string "CONTENT OF REAL_VAR: xxx-1234.123457" (where "x" is a placeholder for space).
%.<n>F:	Conversion into a string with a decimal number with <n> decimal places. Where relevant, the decimal places are rounded-off or filled with 0. **Example:** N10 DEF REAL REAL_VAR=-1.2345678EX+03 N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%.3F",REAL_VAR) Result: The character string "CONTENT OF REAL_VAR:-1234.568" is written to the string variable RESULT.
%<m>.<n>F:	Conversion into a string with a decimal number with <n> decimal places and a total length of at least <m> characters. Where relevant, the decimal places are rounded-off or filled with 0. Missing characters are filled up to the total length <m> using spaces, left-justified. **Example:** N10 DEF REAL REAL_VAR=-1.2341234567890EX+03 N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%10.2F",REAL_VAR) Result: The character string "CONTENT OF REAL_VAR:xx-1234.12" is written to the string variable RESULT ("x" in the example represents spaces).

%E:	Conversion into a string with a decimal number in the exponential representation. The mantissa is saved, normalized with one pre-decimal place and 6 decimal places. Where relevant, the decimal places are rounded-off or filled with 0. The exponent starts with the keyword "EX". It is followed by the sign ("+" or "-") and a two or three-digit number.
	Example:
	`N10 DEF REAL REAL_VAR=-1234.567890`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%E",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:-1.234568EX+03" is written to the string variable RESULT.

%<m>E:	Conversion into a string with a decimal number in the exponential representation and a total length of at least <m> characters. The missing characters are filled with spaces, left-justified. The mantissa is saved, normalized with one pre-decimal place and 6 decimal places. Where relevant, the decimal places are rounded-off or filled with 0. The exponent starts with the keyword "EX". It is followed by the sign ("+" or "-") and a two or three-digit number.
	Example:
	`N10 DEF REAL REAL_VAR=-1234.5`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%20E",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:xxxxxx-1.234500EX+03" is written to the string variable RESULT ("x" in the example represents spaces).

%.<n>E:	Conversion into a string with a decimal number in the exponential representation. The mantissa is saved, normalized with one pre-decimal place and <n> decimal places. Where relevant, the decimal places are rounded-off or filled with 0. The exponent starts with the keyword "EX". It is followed by the sign ("+" or "-") and a two or three-digit number.
	Example:
	`N10 DEF REAL REAL_VAR=-1234.5678`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%.2E",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:-1.23EX+03" is written to the string variable RESULT.

%<m>.<n>E:	Conversion into a string with a decimal number in the exponential representation and a total length of at least <m> characters. The missing characters are filled with spaces, left-justified. The mantissa is saved, normalized with one pre-decimal place and <n> decimal places. Where relevant, the decimal places are rounded-off or filled with 0. The exponent starts with the keyword "EX". It is followed by the sign ("+" or "-") and a two or three-digit number.
	Example:
	`N10 DEF REAL REAL_VAR=-1234.5678`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%12.2E", REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:xx-1.23EX+03" is written to the string variable RESULT ("x" in the example represents spaces).

%G:	Conversion into a string with a decimal number – depending on the value range – in a decimal or exponential representation: If the absolute value to be represented is less than 1.0EX-04 or greater than/equal to 1.0EX+06, then the exponential notation is selected, otherwise the decimal notation. A maximum of six significant places are displayed or if required, rounded-off.
	Example with decimal notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX-04`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:0.000123457" is written to the string variable RESULT.
	Example with exponential notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX+06`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:1.23457EX+06" is written to the string variable RESULT.

%<m>G:	Conversion into a string with a decimal number – depending on the value range – in a decimal or exponential notation (like %G). The string has a total length of at least <m> characters. The missing characters are filled with spaces, left-justified.
	Example with decimal notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX-04`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%15G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:xxxx0.000123457" is written to the string variable RESULT ("x" in the example represents spaces).
	Example with exponential notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX+06`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%15G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:xxx1.23457EX+06" is written to the string variable RESULT ("x" in the example represents spaces).

%.<n>G:	Conversion into a string with a decimal number – depending on the value range – in a decimal or exponential representation. A maximum of <n> significant places are displayed or if required, rounded-off. If the absolute value to be represented is less than 1.0EX-04 or greater than/equal to 1.0EX(+<n>), then the exponential notation is selected, otherwise the decimal notation.
	Example with decimal notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX-04`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%.3G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:0.000123" is written to the string variable RESULT.
	Example with exponential notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX+03`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT = SPRINT("CONTENT OF REAL_VAR:%.3G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:1.23EX+03" is written to the string variable RESULT.

%<m>.<n>G:	Conversion into a string with a decimal number – depending on the value range – in a decimal or exponential notation (like %.<n>G). The string has a total length of at least <m> characters. The missing characters are filled with spaces, left-justified.
	Example with decimal notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX-04`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%12.4G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:xxx0.0001235" is written to the string variable RESULT ("x" in the example represents spaces).
	Example with exponential notation:
	`N10 DEF REAL REAL_VAR=1.234567890123456EX+04`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF REAL_VAR:%12.4G",REAL_VAR)`
	Result: The character string "CONTENT OF REAL_VAR:xx1.235EX+06" is written to the string variable RESULT ("x" in the example represents spaces).

%.<n>P:	Converting a REAL value into an INTEGER value taking into account <n> decimal places. The INTEGER value is output as a 32-bit binary number. If the value to be converted cannot be represented with 32 bits, then processing is interrupted with an alarm.
	As a byte sequence generated using the format statement %.<n>P can also contain binary zeroes, then the total string that is generated in this way no longer corresponds to the conventions of the NC data type STRING. As a consequence, it can neither be saved in a variable, type STRING, nor be further processed using the string commands of the NC language. The only possible use is to transfer the parameter to the WRITE command with output at an appropriate external device (see the following example).
	As soon as the <Format_String> contains a format description, type %P then the complete string, with the exception of the binary number generated with %.<n>P, is output corresponding to the MD10750 $MN_SPRINT_FORMAT_P_CODE in the ASCII character code, ISO (DIN6024) or EIA (RS244). If a character that cannot be converted is programmed, then processing is interrupted with an alarm.
	Example:
	`N10 DEF REAL REAL_VAR=123.45`
	`N20 DEF INT ERROR`
	`N30 DEF STRING[20] EXT_DEVICE="/ext/dev/1"`
	`...`
	`N100 EXTOPEN(ERROR,EXT_DEVICE)`
	`N110 IF ERROR <> 0`
	`...  ; error handling`
	`N200 WRITE(ERROR,EXT_DEVICE,SPRINT("INTEGER BINARY CODED:%.3P",REAL_VAR)`
	`N210 IF ERROR <> 0`
	`… ; error handling`
	Result: The string "INTEGER BINARY CODED: 'H0001E23A'" is transferred to the output device /ext/dev/1. The hexadecimal value 0x0001E23A corresponds to the decimal value 123450.

%<m>.<n>P:	Conversion of a REAL value corresponding to the setting in machine data MD10751 $MN_SPRINT_FORMAT_P_DECIMAL into a string with:
	• An integer of <m> + <n> places **or**
	• A decimal number with a maximum of <m> pre-decimal places and precisely <n> decimal places.
	Just the same as for the format description %.<n>P, the complete string is saved in the character code defined by MD10750 $MN_SPRINT_FORMAT_P_CODE.
	Conversion for MD10751 = 0:
	The REAL value is converted into a string with an integer number of <m> + <n> places. If required, decimal places are rounded-off to <n> places or filled with 0. The missing pre-decimal places are filled with spaces. The minus sign is attached, left-justified; a space is entered instead of the plus sign.
	Example:
	`N10 DEF REAL REAL_VAR=-123.45`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("PUNCHED TAPE FORMAT:%5.3P",REAL_VAR)`
	Result: The character string "PUNCHED TAPE FORMAT:-xx123450" is written to the string variable RESULT ("x" in the example represents spaces).
	Conversion for MD10751 = 1:
	The REAL value is converted into a string with a decimal number with a maximum of <m> pre-decimal places and precisely <n> decimal places. Where necessary, the pre-decimal places are cut-off and the decimal places are rounded-off or filled with 0. If <n> is equal to 0, then the decimal point is also omitted.
	Example:
	`N10 DEF REAL REAL_VAR1=-123.45`
	`N20 DEF REAL REAL_VAR2=123.45`
	`N30 DEF STRING[80] RESULT`
	`N40 RESULT=SPRINT("PUNCHED TAPE FORMAT:%5.3P VAR2:%2.0P",`
	`REAL_VAR1,REAL_VAR2)`
	Result: The character string "PUNCHED TAPE FORMAT:-123.450 VAR2:23" is written to the string variable RESULT.

%S:	Inserting a string.
	Example:
	`N10 DEF STRING[16] STRING_VAR="ABCDEFG"`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF STRING_VAR:%S",STRING_VAR)`
	Result: The character string "CONTENT OF STRING_VAR:ABCDEFG" is written to the string variable RESULT.

%<m>S:	Inserting a string with a minimum of <m> characters. The missing places are filled with spaces.
	Example:
	`N10 DEF STRING[16] STRING_VAR="ABCDEFG"`
	`N20 DEF STRING[80] RESULT`
	`N30 RESULT=SPRINT("CONTENT OF STRING_VAR:%10S",STRING_VAR)`
	Result: The character string "CONTENT OF STRING_VAR:xxxABCDEFG" is written to the string variable RESULT ("x" in the example represents spaces).

%.<n>S:	Inserting <n> characters of a string (starting with the first character). **Example:** N10 DEF STRING[16] STRING_VAR="ABCDEFG" N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF STRING_VAR:%.3S",STRING_VAR) Result: The character string "CONTENT OF STRING_VAR:ABC" is written to the string variable RESULT.
%<m>.<n>S:	Inserting <n> characters of a string (starting with the first character). The total length of the generated string has at least <m> characters. The missing places are filled with spaces. **Example:** N10 DEF STRING[16] STRING_VAR="ABCDEFG" N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("CONTENT OF STRING_VAR:%10.5S", STRING_VAR) Result: The character string "CONTENT OF STRING_VAR:xxxxxABCDE" is written to the string variable RESULT ("x" in the example represents spaces).
%X:	Converting an INTEGER value into a string with the hexadecimal notation. **Example:** N10 DEF INT INT_VAR='HA5B8' N20 DEF STRING[80] RESULT N30 RESULT=SPRINT("INTEGER HEXADECIMAL:%X",INT_VAR) Result: The character string "INTEGER HEXADECIMAL:A5B8" is written to the string variable RESULT.

Note

A property of the NC language, where a distinction is not made between uppercase and lowercase letters for identifiers and keywords, also applies to the format descriptions. As a consequence, you can program using either lowercase or uppercase letters without any functional difference.

Combination options

The following table provides information as to which NC data types can be combined with which format description. The rules regarding implicit data type conversion apply (see "Data types (Page 56)").

	NC data types						
	BOOL	CHAR	INT	REAL	STRING	AXIS	FRAME
%B	+	+	+	+	+	-	-
%C	-	+	-	-	+	-	-
%D	+	+	+	+	-	-	-
%F	-	-	+	+	-	-	-
%E	-	-	+	+	-	-	-
%G	-	-	+	+	-	-	-
%S	-	+	-	-	+	-	-
%X	+	+	+	-	-	-	-
%P	-	-	+	+	-	-	-

Note

The table indicates that the NC data types AXIS and FRAME cannot be directly used in the SPRINT function. However it is possible:

- To convert the AXIS data type into a string using the AXSTRING function – which can then be processed with SPRINT.
- To read the individual values of the FRAME data type per frame component access. As a consequence, a REAL data type is obtained, which can be processed with SPRINT.

2.10 Program jumps and branches

2.10.1 Return jump to the start of the program (GOTOS)

The GOTOS command can be used to jump back to the beginning of a main or subprogram in order to repeat the program.

Machine data can be used to set that for every return jump is made to the program start:

- The program runtime is set to "0".
- Workpiece counting is incremented by the value "1".

Syntax

```
GOTOS
```

Meaning

GOTOS:	Jump statement where the destination is the beginning of the program.	
	The execution is controlled via the NC/PLC interface signal: DB21, to DBX384.0 (control program branching)	
	Value:	Meaning:
	0	No return jump to the beginning of the program. Program execution is resumed with the next part program block after GOTOS.
	1	Return jump to the beginning of the program. The part program is repeated.

Supplementary conditions

- GOTOS internally initiates a STOPRE (pre-processing stop).
- For a subprogram with data definitions (LUD variables) with the GOTOS, a jump is made to the first program block after the definition section, i.e. data definitions are not executed again. This is the reason that the defined variables retain the value reached in the GOTOS block and are not reset to the standard values programmed in the definition section.
- The GOTOS command is not available in synchronized actions and technology cycles.

Example

Program code	Comment
N10 ...	; Start of the program.
...	
N90 GOTOS	; Jump to beginning of the program.
...	

2.10.2 Program jumps to jump markers (GOTOB, GOTOF, GOTO, GOTOC)

Jump labels can be set in a program, which can be jumped to from another location within the same program using the commands GOTOF, GOTOB, GOTO, or GOTOC. Program execution is resumed with the statement that immediately follows the jump label. This means that branches can be realized within the program.

In addition to jump labels, main and sub-block numbers are possible as jump designation.

If a jump condition (IF ...) is formulated before the jump statement, the program jump is only executed if the jump condition is fulfilled.

Syntax

```
GOTOB <jump destination>
IF <jump condition> == TRUE GOTOB <jump destination>

GOTOF <jump destination>
IF <jump condition> == TRUE GOTOF <jump destination>

GOTO <jump destination>
IF <jump condition> == TRUE GOTO <jump destination>

GOTOC <jump destination>
IF <jump condition> == TRUE GOTOC <jump destination>
```

Meaning

GOTOB:	Jump statement with jump destination toward the beginning of the program.
GOTOF:	Jump statement with jump destination toward the end of the program.
GOTO:	Jump statement with jump destination search. The search is first made in the direction of the end of the program, then in the direction of the beginning of the program.
GOTOC:	Same effect as for GOTO with the difference that Alarm 14080 "Jump designation not found" is suppressed. This means that program execution is not interrupted in the case that the jump destination search is unsuccessful – but is continued with the program line following the GOTOC command.

<jump destination>:	Jump destination parameter Possible data include:	
	<jump label>:	Jump destination is the jump label set in the program with a user-defined name:<jump label>:
	<block number>:	Jump destination is main block or sub-block number (e.g.: 200, N300)
	STRING type variable:	Variable jump destination. The variable stands for a jump label or a block number.
IF:	Keyword to formulate the jump condition. The jump condition permits all comparison and logical operations (result: TRUE or FALSE). The program jump is executed if the result of this operation is TRUE.	

Note

Jump labels

Jump labels are always located at the beginning of a block. If a program number exists, the jump label is located immediately after the block number.

The following rules apply when naming jump labels:

- Number of characters:
  - Minimum 2
  - Maximum 32
- Permissible characters are:
  - Letters
  - Numbers
  - Underscores
- The first two characters must be letters or underscores.
- The name of the jump label is followed by a colon (":").

Supplementary conditions

- The jump destination can only be a block with jump label or block number that is located **within** the program.
- A jump statement without jump condition must be programmed in a separate block. This restriction does not apply to jump statements with jump conditions. In this case, several jump statements can be formulated in a block.
- For programs with jump statements without jump conditions, the end of the program M2/ M30 does not necessarily be at the end of the program.

Examples

Example 1: Jumps to jump labels

```
Program code                    Comment
N10 …
```

Program code	Comment
N20 GOTOF Label_1	; Jump toward end of program to
	; jump label "Label_1".
N30 …	
N40 Label_0: R1=R2+R3	; Jump label "Label_0" set.
N50 …	
N60 Label_1:	; Jump label "Label_1" set.
N70 …	
N80 GOTOB Label_0	; Jump toward beginning of program
	; to the jump label "Label_0."
N90 …	

Example 2: Indirect jump to the block number

Program code	Comment
IF <condition> == TRUE	
R10=100	; Assign jump destination
ELSE	
R10=110	; Assign jump destination
ENDIF	
; Jump toward end of program to the block whose block number is located in R10	
N10 GOTOF "N"<<R10	
...	
N90 ...	
N100 ...	; Jump destination
N110 ...	
...	

Example 3: Jump to variable jump destination

Program code	Comment
DEF STRING[20] DESTINATION	
IF <condition> == TRUE	
DESTINATION = "Label1"	; Assign jump destination
ELSE	
DESTINATION = "Label2"	; Assign jump destination
ENDIF	
; Jump toward end of program to the variable jump destination "Content of DESTINATION."	
GOTOF DESTINATION	
Label1: T="Drill1"	; Jump destination 1
...	
Label2: T="Drill2"	; Jump destination 2
...	

Example 4: Jump with jump condition

```
Program code                                                    Comment
N40 R1=30 R2=60 R3=10 R4=11 R5=50 R6=20                         ; Assignment of the initial values
N41 LA1: G0 X=R2*COS(R1)+R5 Y=R2*SIN(R1)+R6                     ; Jump label LA1
N42 R1=R1+R3 R4=R4-1
; IF jump condition == TRUE
; THEN jump toward beginning of program to the jump label LA1
N43 IF R4>0 GOTOB LA1
N44 M30                                                         ; End of program
```

2.10.3 Program branch (CASE ... OF ... DEFAULT ...)

The CASE function provides the possibility of checking the actual value (type: INT) of a variable or an arithmetic function and, depending on the result, to jump to different positions in the program.

Syntax

```
CASE(<expression>) OF <constant_1> GOTOF <jump target_1>
<constant_2> GOTOF <jump target_2> ... DEFAULT GOTOF <jump target_n>
```

Meaning

CASE:	Jump statement	
<expression>:	Variable or arithmetic function	
OF:	Keyword to formulate conditional program branches.	
<constant_1>:	First specified constant value for the variable or arithmetic function	
	Type:	INT
<constant_2>:	Second specified constant value for the variable or arithmetic function	
	Type:	INT
DEFAULT:	For the cases where the variable or arithmetic function does not assume any of the specified constant values, the DEFAULT statement can be used to determine the jump target. **Note:** If the DEFAULT statement is not programmed, then in these cases, the block following the CASE statement is the jump target.	
GOTOF:	Jump statement with jump target towards the end of the program. Instead of GOTOF all other GOTO commands can be programmed (refer to the subject "Program jumps to jump markers").	

<jump target_1>:	A branch is made to this jump target if the value of the variable or arithmetic function corresponds to the first specific constant. The jump target can be specified as follows:	
	<jump marker>:	Jump target is the jump marker (label) set in the program with a user-defined name: <jump marker>:
	<block number>:	Jump target is main block or sub-block number (e.g.: 200, N300)
	STRING type variable:	Variable jump target. The variable stands for a jump marker or a block number.
<jump target_2>:	A branch is made to this jump target if the value of the variable or arithmetic function corresponds to the second specified constant.	
<jump target_n>:	A branch is made to this jump target if the value of the variable does not assume the specified constant value.	

Example

Program code

```
...
N20 DEF INT VAR1 VAR2 VAR3
N30 CASE(VAR1+VAR2-VAR3) OF 7 GOTOF Label_1 9 GOTOF La-
bel_2 DEFAULT GOTOF Label_3
N40 Label_1: G0 X1 Y1
N50 Label_2: G0 X2 Y2
N60 Label_3: G0 X3 Y3
...
```

The CASE statement from N30 defines the following program branch possibilities:

1. If the value of the arithmetic function VAR1+VAR2-VAR3 = 7, then jump to the block with the jump marker definition "Label_1" (→ N40).
2. If the value of the arithmetic function VAR1+VAR2-VAR3 = 9, then jump to the block with the jump marker definition "Label_2" (→ N50).
3. If the value of the arithmetic function VAR1+VAR2-VAR3 is neither 7 nor 9, then jump to the block with the jump marker definition "Label_3" (→ N60).

2.11 Repeat program section (REPEAT, REPEATB, ENDLABEL, P)

Program section repetition allows you to repeat existing program sections within a program in any order.

The program lines or program sections to be repeated are identified by jump markers (labels).

Note

Jump markers (labels)

Jump markers are always located at the beginning of a block. If a program number exists, the jump marker is located immediately after the block number.

The following rules apply when naming jump markers:

- Number of characters:
  - Minimum 2
  - Maximum 32
- Permissible characters are:
  - Letters
  - Numbers
  - Underscores
- The first two characters must be letters or underscores.
- The name of the jump marker is followed by a colon (":").

Syntax

1. Repeat individual program line:

```
<jump marker>: ...
...
REPEATB <jump marker> P=<n>
...
```

2. Repeat program section between jump marker and REPEAT statement:

```
<jump marker>: ...
...
REPEAT <jump marker> P=<n>
...
```

3. Repeat section between two jump markers:

```
<start jump marker>: ...
...
<end jump marker>: ...
...
REPEAT <start jump marker> <end jump marker> P=<n>
...
```

Note

It is not possible to nest the REPEAT statement with the two jump markers in parentheses. If the <start jump marker> appears before the REPEAT statement and the <end jump marker> is not reached before the REPEAT statement, the section between the <start jump marker> and the REPEAT statement will be repeated.

4. Repeat section between jump marker and ENDLABEL:

```
<jump marker>: ...
...
ENDLABEL: ...
...
REPEAT <jump marker> P=<n>
...
```

Note

It is not possible to nest the REPEAT statement with the <jump marker> and the ENDLABEL in parentheses. If the <jump marker> appears before the REPEAT statement and the ENDLABEL is not reached before the REPEAT statement, the section between the <jump marker> and the REPEAT statement will be repeated.

Meaning

REPEATB:	Command for repeating a program line
REPEAT:	Command for repeating a program section
<jump marker>:	The <jump marker> identifies: • The program line to be repeated (in the case of REPEATB) or • The start of the program section to be repeated (in the case of REPEAT)
	The program line identified by the <jump marker> can appear before or after the REPEAT/REPEATB statement. The search initially commences toward the start of the program. If the jump marker is not found in this direction, the search continues working toward the end of the program. **Exception:** If the program section between the jump marker and the REPEAT statement needs to be repeated (see 2. under Syntax), the program line identified by the <jump marker> has to appear **before** the REPEAT statement, since in this case the search runs **only** toward the beginning of the program. If the line with the <jump marker> contains further operations, these are executed again on each repetition.
ENDLABEL:	Keyword marking the end of a program section to be repeated. If the line with the ENDLABEL contains further operations, these are executed again on each repetition. ENDLABEL can be used more than once in the program.

P:	Address for specifying the number of repetitions	
<n>:	Number of program section repetitions	
	Type:	INT
	The program section to be repeated is repeated <n> times. After the last repetition, the program is resumed at the line following the REPEAT/ REPEATB line. **Note:** In the absence of a number being specified for P=<n>, the program section is repeated just once.	

Examples

Example 1: Repeat individual program line

```
Program code                          Comment
N10 POSITION1: X10 Y20
N20 POSITION2: CYCLE(0,,9,8)          ;Position cycle
N30 ...
N40 REPEATB POSITION1 P=5             ; Execute BLOCK N10 five times.
N50 REPEATB POSITION2                 ; Execute block N20 once.
N60 ...
N70 M30
```

Example 2: Repeat program section between jump marker and REPEAT statement:

```
Program code                  Comment
N5 R10=15
N10 Begin: R10=R10+1          ;Width
N20 Z=10-R10
N30 G1 X=R10 F200
N40 Y=R10
N50 X=-R10
N60 Y=-R10
N70 Z=10+R10
N80 REPEAT BEGIN P=4          ; Execute section from N10 to N70 four times.
N90 Z10
N100 M30
```

Example 3: Repeat section between two jump markers

```
Program code                      Comment
N5 R10=15
N10 Begin: R10=R10+1              ;Width
N20 Z=10-R10
N30 G1 X=R10 F200
N40 Y=R10
N50 X=-R10
```

```
Program code                    Comment
N60 Y=-R10
N70 END: Z=10
N80 Z10
N90 CYCLE(10,20,30)
N100 REPEAT BEGIN END P=3       ; Execute section from N10 to N70 three times.
N110 Z10
N120 M30
```

Example 4: Repeat section between jump marker and ENDLABEL

```
Program code                    Comment
N10 G1 F300 Z-10
N20 BEGIN1:
N30 X10
N40 Y10
N50 BEGIN2:
N60 X20
N70 Y30
N80 ENDLABEL: Z10
N90 X0 Y0 Z0
N100 Z-10
N110 BEGIN3: X20
N120 Y30
N130 REPEAT BEGIN3 P=3      ; Execute section from N110 to N120 three times.
N140 REPEAT BEGIN2 P=2      ; Execute section from N50 to N80 twice.
N150 M100
N160 REPEAT BEGIN1 P=2      ; Execute section from N20 to N80 twice.
N170 Z10
N180 X0 Y0
N190 M30
```

Example 5: Milling, machine drill position with different technologies

```
Program code                Comment
N10 CENTER DRILL()          ; Load centering drill.
N20 POS_1:                  ;Drilling positions 1
N30 X1 Y1
N40 X2
N50 Y2
N60 X3 Y3
N70 ENDLABEL:
N80 POS_2:                  ;Drilling positions 2
N90 X10 Y5
N100 X9 Y-5
N110 X3 Y3
```

```
Program code                Comment
N120 ENDLABEL:
N130 DRILL()                ; Change drill and drilling cycle.
N140 THREAD(6)              ; Load tap M6 and threading cycle.
N150 REPEAT POS_1           ; Repeat program section once from POS_1 up to ENDLABEL.
N160 DRILL()                ; Change drill and drilling cycle.
N170 THREAD(8)              ; Load tap M8 and threading cycle.
N180 REPEAT POS_2           ; Repeat program section once from POS_2 up to ENDLABEL.
N190 M30
```

Further information

- Program section repetitions can be nested. Each call uses a subprogram level.
- If `M17` or `RET` is programmed during processing of a program section repetition, the repetition is canceled. The program is resumed at the block following the `REPEAT` line.
- In the actual program display, the program section repetition is displayed as a separate subprogram level.
- If the level is canceled during the program section repetition, the program resumes at the point after the program section repetition call.
  Example:

```
Program code                    Comments
N5 R10=15
N10 BEGIN: R10=R10+1            ;Width
N20 Z=10-R10
N30 G1 X=R10 F200
N40 Y=R10                       ; Interrupt level
N50 X=-R10
N60 Y=-R10
N70 END: Z10
N80 Z10
N90 CYCLE(10,20,30)
N100 REPEAT BEGIN END P=3
N120 Z10                        ; Resume program execution.
N130 M30
```

- Check structures and program section repetitions can be used in combination. There should be no overlap between the two, however. A program section repetition should appear within a check structure branch or a check structure should appear within a program section repetition.
- If jumps and program section repetitions are mixed, the blocks are executed purely sequentially. For example, if a jump is performed from a program section repetition, processing continues until the programmed end of the program section is found.
  Example:

```
Program code
N10 G1 F300 Z-10
N20 BEGIN1:
N30 X=10
N40 Y=10
N50 GOTOF BEGIN2
N60 ENDLABEL:
N70 BEGIN2:
N80 X20
N90 Y30
N100 ENDLABEL: Z10
N110 X0 Y0 Z0
N120 Z-10
N130 REPEAT BEGIN1 P=2
N140 Z10
N150 X0 Y0
N160 M30
```

Note

The REPEAT statement should appear after the traversing block.

2.12 Check structures

The control processes the NC blocks as standard in the programmed sequence.

This sequence can be variable by programming alternative program blocks and program loops. These check structures are programmed using the key words IF, ELSE, ENDIF, LOOP, FOR, WHILE and REPEAT.

NOTICE

Programming error

Check structures may only be inserted in the statement section of a program. Definitions in the program header may not be executed conditionally or repeatedly.

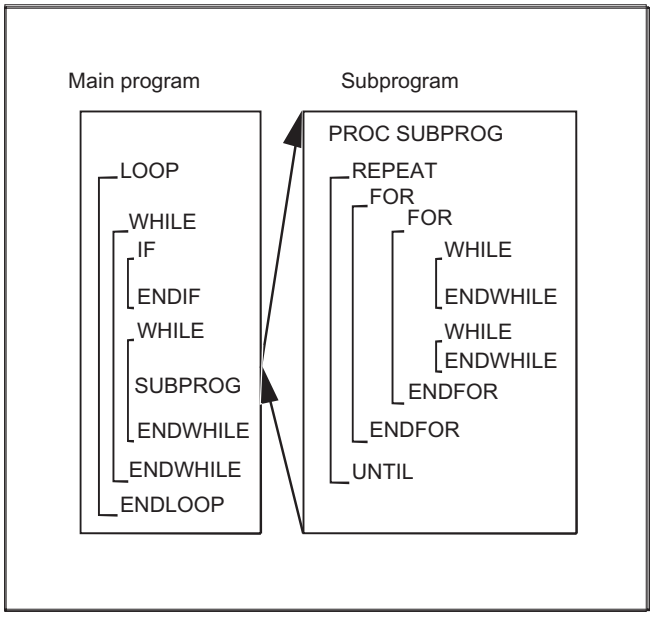
It is not permissible to superimpose macros on keywords for check structures or on jump targets. No such check is made when the macro is defined.

Effectiveness

The check structure cannot be used program-wide.

Nesting depth

A nesting depth of up to 16 check structures can be set up on each subprogram level.



Runtime response

In interpreter mode (active as standard), it is possible to shorten program processing times more effectively by using program branches than can be obtained with check structures.

There is no difference between program branches and check structures in precompiled cycles.

Current block display for program loops

If only selected blocks are executed within a program loop, the last main run block **before** the program loop is shown in the current block display.

So that the processed selected blocks are also visible in the current block display, e.g. for diagnostic purposes, the decoding single block SBL2 must be activated.

References

Function Manual, Basic Functions, Section: Mode group, channel, program operation, reset response (K1) > Single block > Decoding single block SBL2 with implicit preprocessing stop

Supplementary conditions

- Blocks with check structure elements cannot be suppressed.
- Jumper markers (labels) are not permitted in blocks with check structure elements.
- Check structures are processed interpretively. When a loop end is detected, a search is made for the loop beginning, allowing for the check structures found in the process. For this reason, the block structure of a program is not checked completely in interpreter mode.

- It is not generally advisable to use a mixture of check structures and program branches.
- A check can be made to ensure that check structures are nested correctly when cycles are preprocessed.

2.12.1 Conditional statement and branch (IF, ELSE, ENDIF)

Conditional statement: IF - program block - ENDIF

With a conditional statement, the program block between `IF` and `ENDIF` is only executed when the condition is satisfied.

Branch: IF - program block_1 - ELSE - program block_2 - ENDIF

With a branch, one of two program blocks is always executed.

If the condition is satisfied, program block_1 between `IF` and `ELSE` is executed.

If the condition is **not** satisfied, program block_2 between `ELSE` and `ENDIF` is executed.

Syntax

Conditional statement

```
IF <condition>
Program block                          ; Execution when: <condition> == TRUE
ENDIF
```

Branch

```
IF <condition>
   Program block_1                     ; Execution when: <condition> == TRUE
ELSE
   Program block_2                     ; Execution when: <condition> == FALSE
ENDIF
```

Meaning

`IF`:	Introduces the conditional statement or branch.
`ELSE`:	Introduces the alternative program block.
`ENDIF`:	Marks the end of the conditional statement or branch.
`<condition>`:	Logical expression that is evaluated as TRUE or FALSE.

Example: Tool change subprogram

Program code	**Comment**
`PROC L6`	`Tool change routine`
`N500 DEF INT TNR_AKTUELL`	`Variable for active T number`

Program code	Comment
N510 DEF INT TNR_VORWAHL	Variable for preselected T number
	Determine current tool
N520 STOPRE	
N530 IF $P_ISTEST	In the program test mode ...
N540 TNR_AKTUELL = $P_TOOLNO	... The "current" tool is read from the program context.
N550 ELSE	Otherwise ...
N560 TNR_AKTUELL = $TC_MPP6[9998,1]	... The tool of the spindle is read-out.
N570 ENDIF	
N580 GETSELT(TNR_VORWAHL)	Read the T number of the pre-selected tool in the spindle.
N590 IF TNR_AKTUELL <> TNR_VORWAHL	If the pre-selected tool is still not the current tool, then ...
N600 G0 G40 G60 G90 SUPA X450 Y300 Z300 D0	... Approach tool change position ...
N610 M206	... and execute a tool change.
N620 ENDIF	
N630 M17	

2.12.2 Continuous program loop (LOOP, ENDLOOP)

Endless loops are used in endless programs. At the end of the loop, there is always a branch back to the beginning.

Syntax

```
LOOP
...
ENDLOOP
```

Meaning

LOOP:	Initiates the endless loop.
ENDLOOP:	Marks the end of the loop and results in a return jump to the beginning of the loop.

Example

Program code
...
LOOP
MSG ("no tool cutting edge active")

```
Program code
M0
STOPRE
ENDLOOP
...
```

2.12.3 Count loop (FOR ... TO ..., ENDFOR)

The count loop is used if an operation must be repeated with a fixed number of runs.

Syntax

```
FOR <variable> = <initial value> TO <end value>
...
ENDFOR
```

Meaning

FOR:	Initiates the count loop.	
ENDFOR:	Marks the end of the loop and results in a return jump to the beginning of the loop, as long as the end value of the count has still not been reached.	
<variable>:	Count variable, which is incremented from the initial to the end value and is increased by the value "1" at each run.	
	Type	INT or REAL **Note:** The REAL type is used if R parameters are programmed for a count loop, for example. If the count variable is of the REAL type, its value is rounded to an integer.
<initial value>:	Initial value of the count Condition: The start value must be lower than the end value.	
<full-scale value>:	End value of the count	

Examples

Example 1: INTEGER variable or R parameter as count variable

INTEGER variable as count variable:

```
Program code                          Comment
DEF INT iVARIABLE1
R10=R12-R20*R1 R11=6
FOR iVARIABLE1 = R10 TO R11           ; Count variable = INTEGER variable
 R20=R21*R22+R33
ENDFOR
```

Program code	Comment
M30	

R parameter as count variable:

Program code	Comment
R11=6	
FOR R10=R12-R20*R1 TO R11	; Count variable = R parameter (real variable)
R20=R21*R22+R33	
ENDFOR	
M30	

Example 2: Production of a fixed quantity of parts

Program code	Comment
DEF INT WKPCCOUNT	; Defines type INT variable with the name "WKPCCOUNT".
FOR WKPCCOUNT = 0 TO 100	; Initiates the count loop. The "WKPCCOUNT" variable increments from the initial value "0" to the end value "100".
G01 …	
ENDFOR	; End of count loop
M30	

2.12.4 Program loop with condition at start of loop (WHILE, ENDWHILE)

For a WHILE loop, the condition is at the beginning of the loop. The WHILE loop is executed as long as the condition is fulfilled.

Syntax

```
WHILE <condition>
...
ENDWHILE
```

Meaning

WHILE:	Initiates the program loop.
ENDWHILE:	Marks the end of the loop and results in a return jump to the beginning of the loop.
<condition>:	The condition must be fulfilled so that the WHILE loop is executed.

Example

Program code	Comment
...	
WHILE $AA_IW[DRILL_AXIS] > -10	; Call the WHILE loop under the following condition: The actual WCS setpoint for the drilling axis must be greater than -10.
G1 G91 F250 AX[DRILL_AXIS] = -1	
ENDWHILE	
...	

2.12.5 Program loop with condition at the end of the loop (REPEAT, UNTIL)

For a REPEAT loop, the condition is at the end of the loop. The REPEAT loop is executed once and repeated continuously until the condition is fulfilled.

Syntax

```
REPEAT
...
UNTIL <significance>
```

Meaning

REPEAT:	Initiates the program loop.
UNTIL:	Marks the end of the loop and results in a return jump to the beginning of the loop.
<condition>:	The condition that must be fulfilled so that the REPEAT loop is no longer executed.

Example

Program code	Comment
...	
REPEAT	; Call the REPEAT loop.
...	
UNTIL ...	; Check whether the condition is fulfilled.
...	

2.12.6 Program example with nested check structures

```
Program code                                    Comment
LOOP
IF NOT $P_SEARCH                                ; IF no block search
      G1 G90 X0 Z10 F1000
      WHILE $AA_IM[X] <= 100                    ; WHILE (setpoint X axis <= 100)
         G1 G91 X10 F500                        ; Drilling pattern
         Z-5 F100
         Z5
      ENDWHILE
   ELSE                                         ; ELSE block search
      MSG("No drilling during block search")
   ENDIF                                        ; ENDIF
   $A_OUT[1] = 1                                ; Next drilling plate
   G4 F2
ENDLOOP
M30
```

2.13 Program coordination (INIT, START, WAITM, WAITMC, WAITE, SETM, CLEARM)

In principle, a channel of the NC can process the program started in it independently of other channels in its mode group (MG). If, however, several programs in several channels of the MG are involved in machining a workpiece, the program sequences in the different channels must be coordinated.

Note

Mode group

Program coordination is only possible between channels that belong **to the same mode group** (MG).

Syntax

```
INIT(<channel no. >,<program>,<acknowledgement mode>)
START(<channel no.>,<channel no.>, ... )
WAITM(<marker no.>,<channel no.>,<channel no.>, ...)
WAITMC(<marker no.>,<channel no.>,<channel no.>, ...)
WAITE(<channel no.>,<channel no.>,...)
SETM(<marker no.>,<marker no.>,...)
CLEARM(<marker no.>,<marker no.>,...)
```

Note

A program coordination command must stand alone in its own block.

Meaning

INIT:	Predefined procedure for selecting the NC program that is to be processed in the specified channel			
	<channel no.>:	Channel number		
		Type:	INT	
	<Program>:	Optional path specification (absolute or relative) + program name		
		Type:	STRING	
		Rules regarding path data, see "Addressing program memory files (Page 205)".		
	<acknowledgement mode>:	Parameters of the type CHAR (**optional**)		
		Values:	"N"	Without acknowledgement Program execution is continued when the command has been transmitted. The sender is not informed if the command cannot be executed successfully.
			"S"	Synchronous acknowledgement The program execution is paused until the receiving component acknowledges the command. If the acknowledgement is positive, the next command is executed. If the acknowledgement is negative, an error message is output.
		Note: Default setting: "S" (synchronous acknowledgement)		
START:	Predefined procedure for starting the program selected in the respective channel			
	<channel no.>,...:	Enumeration of the channel numbers		
		Type:	INT	
WAITM:	Predefined procedure to wait for a marker to be reached in the specified channels The specified marker is set by WAITM in the same channel. The previous block is terminated with exact stop. The marker is deleted after synchronization. Ten markers can be set per channel simultaneously.			
	<marker no.>:	Number of the marker		
		Type:	INT	
	<channel no.>,...:	Channel number,... (the number of own channel does not have to be specified)		
		Type:	INT	

WAITE:	Predefined procedure to wait for the end of program in one or more other channels		
	<channel no.>,...:	Enumeration of the channel numbers	
		Type:	INT
WAITMC:	Predefined procedure to wait for a marker to be reached in the specified channels In contrast to WAITM, exact stop is only initiated if the other channels have not yet reached the marker. Parameters as for WAITM.		
SETM:	Predefined procedure to set one or more markers for the channel coordination Processing in the same channel is not affected by this. SETM remains valid after reset and NC start.		
	<marker no.>,...:	List of the marker numbers	
CLEARM:	Predefined procedure to delete one or more markers for channel coordination The processing in own channel is not affected by this All markers in the channel can be deleted with CLEARM(). CLEARM(0) deletes the marker "0". CLEARM remains valid after reset and NC start.		
	<marker no.>,...:	List of the marker numbers	

Note

Channel number

Channel names must be converted to numbers by means of variables.

⚠ CAUTION

Channel number

Protect the number assignments so that they are not changed unintentionally.

Note

Channel name

Instead of channel numbers, the channel names (identifiers or keywords) defined via $MC_CHAN_NAME can also be programmed (type: STRING).

⚠ CAUTION

Channel name

The names must not already exist in the NC with a different meaning, e.g. as key words, commands, axis names.

Note

Variables, which all channels can access (NCK-specific global variables), can be used for data exchange between programs. Otherwise separate programs must be written for each channel.

Examples

START channel name parameterized in the machine data item

Parameter assignment:

```
$MC_CHAN_NAME[ 0 ] = "CHAN_1"  ; Name of channel 1
$MC_CHAN_NAME[ 1 ] = "CHAN_2"  ; Name of channel 2
```

The names "CHAN_1" and "CHAN_2" parameterized in the machine data item MD20000 $MC_CHAN_NAME represent the channel numbers 1 and 2 in the internal control system.

Program code	Comment
START(CHAN_1)	; Start of channel 1
START(CHAN_2)	; Start of channel 2

START using local "channel names" and user variables

Definition:

- Channel 1: Local program name "MACHINE"
- Channel 2: Local program name "LOADER"

Program code	Comment
DEF INT MACHINE = 1	; Definition of user variable for channel 1
DEF INT LOADER = 2	; Definition of user variable for channel 2
...	
START(MACHINE)	; Start of channel 1
START(LOADER)	; Start of channel 2

START using local "channel names", user variables and parameterized channel names

Program code	Comment
DEF INT chanNo1	; Definition of user variable for channel 1
DEF INT chanNo2	; Definition of user variable for channel 2
chanNo1 = CHAN_1	; Assignment of parameterized channel name channel 1
chanNo2 = CHAN_2	; Assignment of parameterized channel name channel 2
...	
START(chanNo1)	; Start of channel 1
START(chanNo2)	; Start of channel 2

INIT command with absolute path specification

Selection of program /_N_MPF_DIR/_N_ABSPAN1_MPF in channel 2.

Program code
INIT(2,"/_N_WKS_DIR/_N_WELLE1_WPD/_N_ABSPAN1_MPF")

INIT command with program name

Selection of the program with the name "MYPROG". The control looks for the program using the search path.

```
Program code
INIT(2,"MYPROG")
```

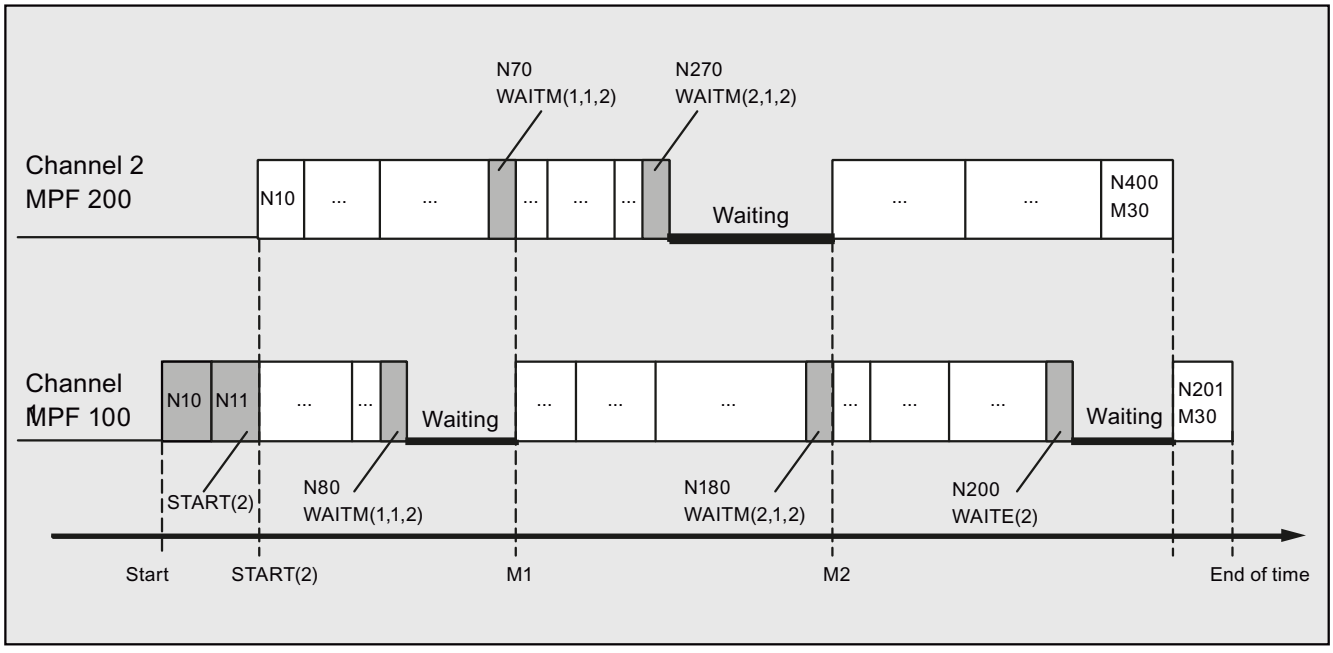
Program coordination with WAITM

Channel 1: The program /_N_MPF_DIR/_N_MPF100_MPF has already been selected and started.

Program code	Comment
	; Program MPF100
N10 INIT(2,"MPF200","N")	; Selection of program MPF200, channel 2
N11 START(2)	; Start of channel 2
...	
N80 WAITM(1,1,2)	; Wait for WAIT marker 1 in channels 1 and 2
N81 ...	; Channel 1, N81 and channel 2, N71 are
	; started synchronously
...	
N180 WAITM(2,1,2)	; Wait for WAIT marker 2 in channels 1 and 2
N181 ...	; Channel 1, N181 and channel 2, N271 are
	; started synchronously
...	
N200 WAITE(2)	; Wait for end of program in channel 2
N201 ...	; N201 is only started after the end of program
	; MPF200 in channel 2
N201 M30	; End of program channel 1

Channel 2: In channel 1, the program MPF200_MPF is selected and started for channel 2 using blocks N10 and N20.

Program code	Comment
;$PATH=/_N_MPF_DIR	; Program MPF200
...	
N70 WAITM(1,1,2)	Wait for WAIT marker 1 in channels 1 and 2
N71 ...	; Channel 1, N81 and channel 2, N71 are
	; started synchronously
...	
N270 WAITM(2,1,2)	Wait for WAIT marker 2 in channels 1 and 2
N271 ...	; Channel 1, N181 and channel 2, N271 are
	; started synchronously
...	
N400 M30	End of program channel 2



Supplementary conditions

Non-synchronous start of processing of subsequent blocks after WAIT markers

In the case of channel coordination by means of WAIT markers, a non-synchronous start to processing of subsequent blocks can come about. This behavior occurs if an action is triggered in one of the channels to be synchronized immediately before reaching the common WAIT marker, the consequence of which is an implicit repositioning (REPOSA) in this deletion of distance-to-go.

Assumption: Current axis allocation in channels 1 and 2

- Channel 1: Axes X1 and U
- Channel 2: Axis X2

Table 2-2 Time sequence in channels 1 and 2

Channel 1	Channel 2	Description
. . .	. . .	Any processing in channels 1 and 2
`N100 WAITM(20,1,2)`		Channel 1: reaches the `WAIT` marker and waits for synchronization with channel 2
Start of the GETD(U) processing: • *Axis replacement* • *Delete distance-to-go* • *REPOSA* *End*	`N200 GETD(U)`	Channel 2: requests axis U from channel 1 Channel 1: processing of `GET(U)` in the background
	`N210 WAITM(20,1,2)`	Channel 2: reaches the `WAIT` marker. ⇒ The synchronization of channels 1 and 2 is then complete
	`N220 G0 X2=100`	Channel 2: start of processing of N220
`N110 G0 X1=100`		Channel 1: staggered start of processing of N110

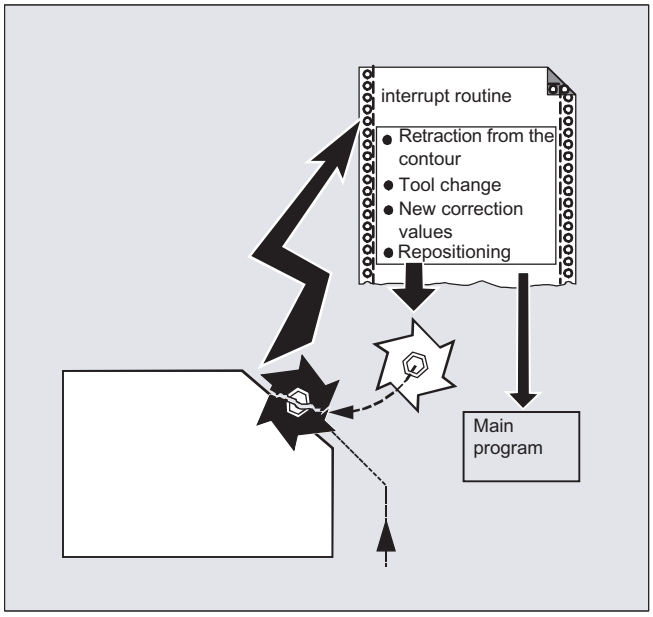
2.14 Interrupt routine (ASUB)

2.14.1 Function of an interrupt routine

Note

The terms "asynchronous subprogram (ASUB)" and "interrupt routine" are used interchangeably in the description below to refer to the same functionality.

A typical example should clarify the function of an interrupt routine:



The tool breaks during machining. This triggers a signal that stops the current machining process and simultaneously starts a subprogram – the so-called interrupt routine. The interrupt routine contains all the statements which are to be executed in this case.

When the interrupt routine execution has finished and the machine is ready to continue operation, the control jumps back to the main program and continues machining at the point of interruption – depending on the `REPOS` command (see " Repositioning at contour (Page 448) ").

⚠ CAUTION

Risk of collision

If a `REPOS` command has not been programmed in the subprogram, then the control goes to the end point of the block that follows the interrupted block.

References

Function Manual, Basic Functions; Mode Group, Channel, Program Operation, Reset Response (K1), Section: "Asynchronous subprograms (ASUBs), interrupt routines"

2.14.2 Creating an interrupt routine

Create interrupt routine as subprogram

The interrupt routine is identified as a subprogram in the definition.

Example:

```
Program code          Comment
PROC LIFT_Z           ; Program name "ABHEB_Z"
N10 ...               ; The NC blocks then follow:
...
N50 M17               ; Finally, end the program and return to the main program.
```

Save modal G functions (SAVE)

The interrupt routine can be designated by defining with SAVE.

The SAVE attribute means that the active modal G functions are saved before calling the interrupt routine and are reactivated after the end of the interrupt routine (see "Subprograms with SAVE mechanism (SAVE) (Page 156)").

This means that it is possible to resume processing at the interruption point after the interrupt routine has been completed.

Example:

```
Program code
PROC LIFT_Z SAVE
N10 ...
...
N50 M17
```

Assign additional interrupt routines (SETINT)

SETINT statements can be programmed within the interrupt routine (see "Assign and start interrupt routine (SETINT)" (Page 120)) therefore activating additional interrupt routines. They are triggered via the input.

References

You will find more information on how to create subprograms in Section "Subprograms, Macros".

2.14.3 Assign and start interrupt routine (SETINT, PRIO, BLSYNC)

The control has several fast inputs (inputs 1 ... 8), which initiate an interrupt (1 ... 8). Each interrupt can be assigned a priority and an interrupt routine using the `SETINT` command. If the interrupt is initiated by setting the fast input, then processing in the channel is interrupted and the interrupt routine started.

Interrupt priority

If, in a part program, several inputs are assigned interrupts, then the interrupts must be assigned different priorities.

An interrupt can be assigned a priority value from 1 ... 128. Priority value 1 corresponds to the highest priority and 128 the lowest.

Syntax

```
SETINT(<n>) <NAME>
SETINT(<n>) PRIO=<value> <NAME>
SETINT(<n>) PRIO=<value> <NAME> BLSYNC
SETINT(<n>) PRIO=<value> <NAME> LIFTFAST
```

Meaning

`SETINT(<n>):`	Input <n> is assigned the interrupt routine <Name>. The assigned interrupt routine is started as soon as input <n> == 1 is detected. **Note:** If an already programmed input <n> is assigned another interrupt routine, then the previous assignment is no longer effective.	
`<n>`:	Input number	
	Type:	INT
	Range of values:	1 ... 8
`PRIO= :`	Priority of the interrupt (optional)	
`<value>`:	Priority value (optional)	
	Type:	INT
	Range of values:	1 ... 128 (1 ⇒ highest priority)
`<NAME>`:	Name of the interrupt routine (subprogram)	
`BLSYNC`:	`BLSYNC` ensures that after initiating the interrupt, the system first waits until the actual block has been completed. Only then is the interrupt routine executed. (optional)	
`LIFTFAST`:	`LIFTFAST` ensures that after initiating the interrupt, initially a fast retraction is realized (see Chapter "Fast retraction from the contour (SETINT LIFTFAST, ALF) (Page 123)"). Only then is the interrupt routine executed. (optional)	

Supplementary conditions

Interrupt rules

1. For every interrupt that cannot be immediately executed, or is presently already being processed, an additional interrupt request is saved. All other interrupt requests for this interrupt are lost.
2. If an interrupt is currently being processed and an additional interrupt with higher priority initiated, then this interrupts the lower-priority interrupt. The lower priority interrupt is continued after the higher priority interrupt has been completed. If, while the higher priority interrupt is being processed, additional requests are received for the lower-priority interrupt, then one request is saved. All others are lost.
3. If an interrupt is currently being processed and an additional interrupt with higher priority initiated, then this interrupts the lower-priority interrupt. The higher priority interrupt is processed. If a higher priority interrupt is initiated, the actual interrupt is interrupted and the higher priority interrupt processed. A maximum of six active interrupt levels are possible. One interrupt level presently being processed and five waiting interrupt levels. For each active interrupt level, a maximum of one additional interrupt request is saved. All other interrupt requests are lost. Interrupt requests are also lost if these are requested for additional interrupt levels (interrupt level ≥ 7).

Examples

Example 1: Assign interrupt routines and define the priority

Program code	Comment
...	
N20 SETINT(3) PRIO=1 ABHEB_Z	; IF input 3 == 1 THEN start interrupt routine "ABHEB_Z"
N30 SETINT(2) PRIO=2 ABHEB_X	; IF input 2 == 1 THEN start interrupt routine "ABHEB_X".
...	

The interrupt routines are executed in the sequence of the priority values if the inputs become available simultaneously (are energized simultaneously): First "ABHEB_Z", then "ABHEB_X".

Example 2: Newly assign an interrupt routine

Program code	Comment
...	
N20 SETINT(3) PRIO=2 ABHEB_Z	; IF input 3 == 1 THEN start interrupt routine **"ABHEB_Z"**
...	
N80 SETINT(3) PRIO=1 ABHEB_X	; IF input 3 == 1 THEN start interrupt routine **"ABHEB_X"**
...	

2.14.4 Deactivating/reactivating the assignment of an interrupt routine (DISABLE, ENABLE)

A SETINT statement can be deactivated with DISABLE and reactivated with ENABLE without losing the input → interrupt routine assignment.

Syntax

```
DISABLE(<n>)
ENABLE(<n>)
```

Meaning

DISABLE(<n>):	Command: **Deactivating** the interrupt routine assignment of input <n>	
ENABLE(<n>):	Command: **Reactivating** the interrupt routine assignment of input <n>	
<n>:	Parameter: Number of the interrupt signal	
	Type:	INT
	Range of values:	1 ... 32

Example

Program code	**Comment**
N20 SETINT(3) PRIO=1 ABHEB_Z	; If input 3 switches, then interrupt
	; routine "ABHEB_Z" should start.
...	
N90 DISABLE(3)	; The SETINT statement from N20 is deactivated.
...	
N130 ENABLE(3)	; The SETINT statement from N20 is reactivated.
...	

2.14.5 Delete assignment of interrupt routine (CLRINT)

An interrupt signal assignment defined with SETINT for an NC program (ASUP) can be deleted with CLRINT.

Syntax

```
CLRINT(<n>)
```

Meaning

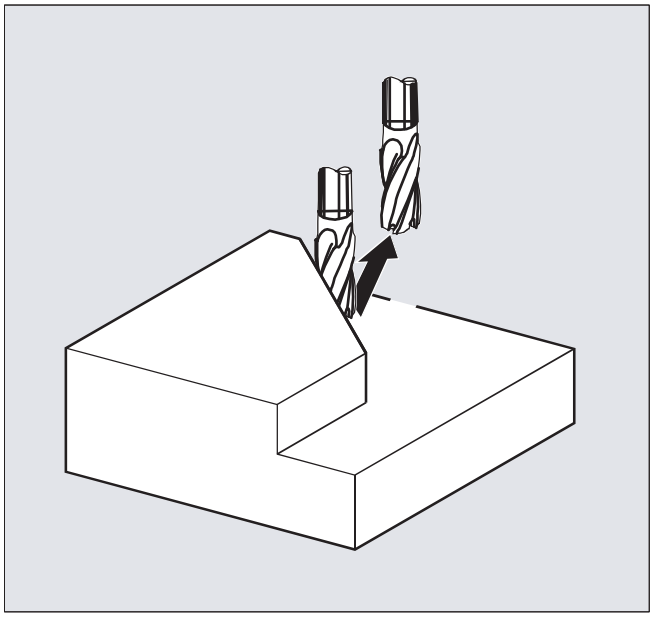
CLRINT(<n>):	Command: Delete assignment of the interrupt signal <n> to the NC program (ASUP) defined with SETINT <n>	
<n>:	Parameter: Number of the interrupt signal	
	Type:	INT
	Range of values:	1 ... 32

Example

Program code	Comment
N20 SETINT(3) PRIO=2 ABHEB_Z	
...	
N50 CLRINT(3)	; The assignment between input "3" and interrupt routine "ABHEB_Z" is deleted.

2.14.6 Fast retraction from the contour (SETINT LIFTFAST, ALF)

For a SETINT statement with LIFTFAST, when the input is switched, the tool is moved away from the workpiece contour using fast retraction.



The further sequence is then dependent on whether the SETINT statement includes an interrupt routine in addition to LIFTFAST:

With interrupt routine: **After** the fast retraction, the interrupt routine is executed.
Without interrupt routine: Machining is stopped after fast retraction and an alarm is output.

Syntax

```
SETINT(<n>) PRIO=1 LIFTFAST
SETINT(<n>) PRIO=1 <NAME> LIFTFAST
```

Meaning

SETINT(<n>):	Command: Assign input <n> to an interrupt routine. The assigned interrupt routine starts when input <n> switches.	
<n>:	Parameter: Input number	
	Type:	INT
	Range of values:	1 ... 8
PRIO= :	Defining the priority	
<value>:	Priority value	
	Range of values:	1 ... 128
	Priority 1 corresponds to the highest priority.	
<NAME>:	Name of the subprogram (interrupt routine) that is to be executed.	
LIFTFAST:	Command: Fast retraction from the contour	
ALF=… :	Command: Programmable traverse direction (in motion block) Regarding the possibilities of programming with ALF, refer to the subject "Traversing direction for fast retraction from the contour (Page 125)".	

Supplementary conditions

Behavior for active frame with mirroring

When determining the retraction direction, a check is performed to see whether a frame with mirror is active. In this case, for the retraction direction, right and left are interchanged referred to the tangential direction. The direction components in tool direction are not mirrored. This behavior is activated with the MD setting:

MD21202 $MC_LIFTFAST_WITH_MIRROR = TRUE

Example

A broken tool should be automatically replaced by a daughter tool. Machining is then continued with the new tool.

Main program:

Main program	**Comment**
N10 SETINT(1) PRIO=1 W_WECHS LIFTFAST	; When input 1 is switched, the tool is immediately retracted from the contour with fast retraction (code no. 7 for tool radius compensation G41). Then interrupt routine "W_WECHS" is executed.
N20 G0 Z100 G17 T1 ALF=7 D1	
N30 G0 X-5 Y-22 Z2 M3 S300	
N40 Z-7	

Main program	Comment
N50 G41 G1 X16 Y16 F200	
N60 Y35	
N70 X53 Y65	
N90 X71.5 Y16	
N100 X16	
N110 G40 G0 Z100 M30	

Subprogram:

Subprogram	Comment
PROC W_CHANGE SAVE	; Subprogram where the actual operating state is saved
N10 G0 Z100 M5	;Tool changing position, spindle stop
N20 T11 M6 D1 G41	;Change tool
N30 REPOSL RMBBL M3	; Reposition at the contour and return jump into the main program (this is programmed in a block)

2.14.7 Traversing direction for fast retraction from the contour

Retraction movement

The following G code defines the retraction movement plane:

- LFTXT
  The retraction movement plane is defined by the path tangent and the tool direction (default setting).
- LFWP
  The plane of the retraction movement is the active working plane selected with G codes G17, G18 or G19. The direction of the retraction movement is not dependent on the path tangent. This allows a fast retraction to be programmed parallel to the axis.
- LFPOS
  Retraction of the axis declared using POLFMASK/POLFMLIN to the absolute axis position programmed with POLF.
  ALF has no influence on the retraction direction for several axes and for several axes in a linear system.
  References:
  Programming Manual, Fundamentals, Section: "Rapid retraction during thread cutting"

Programmable traversing direction (ALF=…)

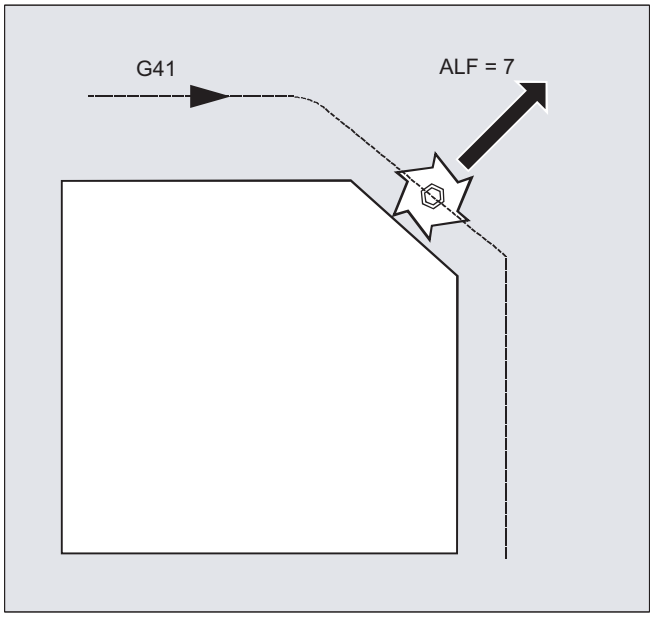
The direction is programmed in discrete steps of 45 degrees with ALF in the plane of the retraction movement.

The possible traversing directions are stored in special code numbers on the control and can be called up using these numbers.

Example:

```
Program code
N10 SETINT(2) PRIO=1 ABHEB_Z LIFTFAST
ALF=7
```

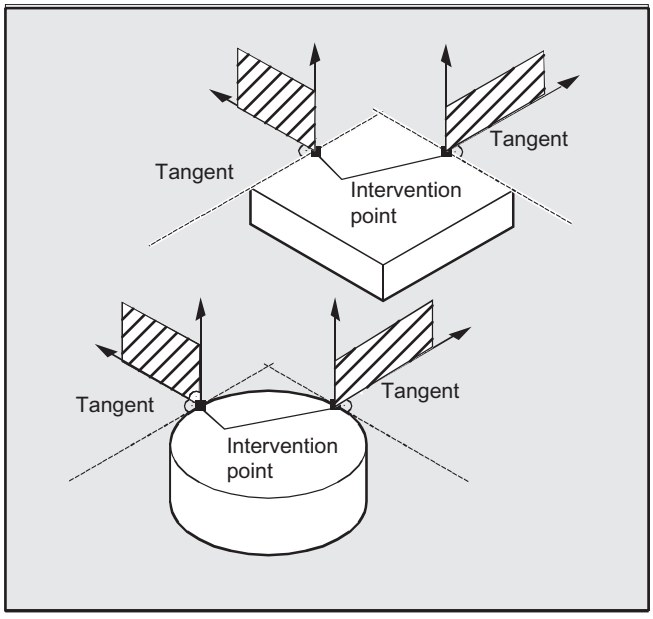
With `G41` activated (machining direction to the left of the contour) the tool vertically moves away from the contour.



Reference plane for defining the traversing direction for LFTXT

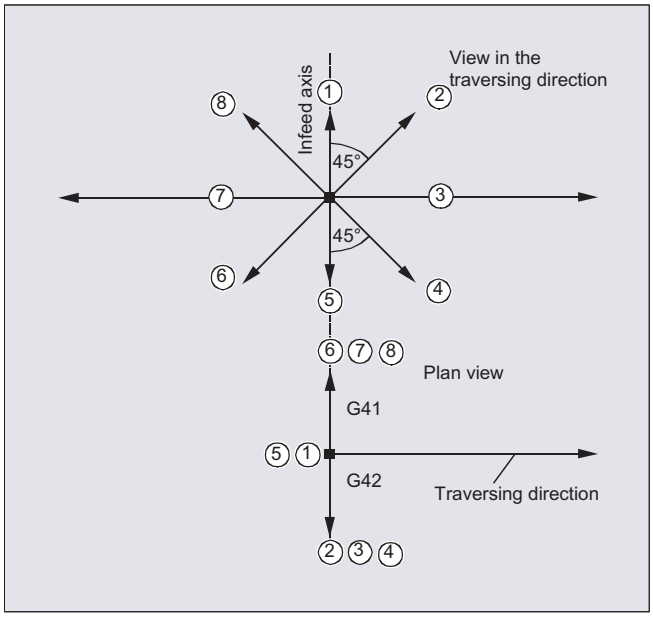
At the point of application of the tool to the programmed contour, the tool is clamped at a plane which is used as a reference for specifying the retraction movement with the corresponding code number.

The reference plane is derived from the longitudinal tool axis (infeed direction) and a vector positioned perpendicular to this axis and perpendicular to the tangent at the point of application of the tool.



Code numbers with traversing direction for LFTXT

Starting from the reference plane, you will find the code numbers with traversing directions in the following diagram.



The retraction in the tool direction is defined for `ALF=1`.

The "fast retraction" function is deactivated with `ALF=0`.

⚠ CAUTION

Risk of collision

When the tool radius compensation is activated, then:

- For `G41` codes 2, 3, 4
- For `G42` codes 6, 7, 8

should not be used, as in these cases, the tool would move to the contour and would collide with the workpiece.

Code numbers with traversing directions for LFWP

With `LFWP`, the direction in the working plane is derived from the following assignment:

- `G17`: X/Y plane
  `ALF=1`: Retraction in the X direction
  `ALF=3`: Retraction in the Y direction
- `G18`: Z/X plane
  `ALF=1`: Retraction in the Z direction
  `ALF=3`: Retraction in the X direction
- `G19`: Y/Z plane
  `ALF=1`: Retraction in the Y direction
  `ALF=3`: Retraction in the Z direction

2.14.8 Motion sequence for interrupt routines

Interrupt routine without LIFTFAST

Axis motion is braked along the path down to standstill (zero speed). The interrupt routine then starts.

The standstill position is saved as interrupt position and is approached at the end of the interrupt routine for `REPOS` with `RMIBL`.

Interrupt routine with LIFTFAST

Axis motion is braked along the path. The `LIFTFAST` motion is simultaneously executed as superimposed motion. If the path motion and `LIFTFAST` motion have come to a standstill (zero speed), the interrupt routine is started.

The position on the contour is saved as interrupt position where the `LIFTFAST` motion is started and therefore the path was left.

The interrupt routine with LIFTFAST and ALF=0 behaves in precisely the same way as the interrupt routine without LIFTFAST.

Note

The absolute value through which the geometry axes move when quickly retracting from the contour can be set using machine data.

2.15 Axis replacement, spindle replacement (RELEASE, GET, GETD)

One or more axes or spindles can only ever be interpolated in one channel. If an axis has to alternate between two different channels (e.g. pallet changer) it must first be enabled in the current channel and then transferred to the other channel. Axis replacement is effective between channels.

Axis replacement extensions

An axis/spindle can be replaced either with a preprocessing stop and synchronization between preprocessing and main run, or without a preprocessing stop. Axis replacement is also possible via:

- Axis container rotation AXCTSWE or AXCTWED using implicit GET/GETD
- Frame with rotation if this process links the axis with other axes.
- Synchronized actions, see Motion-synchronous actions, "Axis replacement RELEASE, GET".

Machine manufacturer

Please refer to the machine manufacturer's instructions. For the purpose of axis replacement, one axis must be defined uniquely in all channels in the configurable machine data and the axis replacement characteristics can also be set using machine data.

Syntax

```
RELEASE (axis name, axis name, ...) or RELEASE (S1)
GET (axis name, axis name, ...) or GET (S2)
GETD(axis name, axis name, ...) or GETD(S3)
```

With GETD (GET Directly), an axis is fetched directly from another channel. This means that no suitable RELEASE must be programmed for this GETD in another channel. It also means that other channel communication has to be established (e.g. wait markers).

Meaning

RELEASE (axis name, axis name, etc.):	Release the axis (axes)
GET (axis name, axis name, etc.):	Accept the axis (axes)
GETD (axis name, axis name, etc.):	Directly accept the axis (axes)
Axis name:	Axis assignment in the system: AX1, AX2, ... or specify machine axis name

`RELEASE(S1):`	Release spindles S1, S2, ...
`GET(S2):`	Accept spindles S1, S2, ...
`GETD(S3):`	Direct acceptance of spindles S1, S2, ...

GET request without preprocessing stop

If, following a GET request **without** preprocessing stop, the axis is enabled again with `RELEASE(axis)` or `WAITP(axis)`, a subsequent `GET` will induce a `GET` **with** preprocessing stop.

⚠ CAUTION

Axis assignment changed

An axis or spindle accepted with GET remains assigned to this channel even after a key or program RESET.

When a program is restarted the replaced axes or spindles must be reassigned in the program if the axis is required in its original channel.

It is assigned to the channel defined in the machine data on POWER ON.

Examples

Example 1: Axis exchange between two channels

Of the six axes, the following are used for machining in channel 1: 1., 2., 3. 1st, 2nd, 3rd and 4th axis.
The 5th and 6th axes in channel 2 are used for the workpiece change.

Axis 2 should be exchanged between two channels and after POWER ON can be assigned to channel 1.

Program "MAIN" in channel 1:

Program code	**Comment**
`INIT (2,"TRANSFER2")`	`; Select program TRANSFER2 in channel 2.`
`N… START (2)`	`; Start the program in channel 2.`
`N… GET (AX2)`	`; Accept axis AX2.`
`...`	
`N… RELEASE (AX2)`	`; Release axis AX2.`
`N… WAITM (1,1,2)`	`; Wait for WAIT marker in channel 1 and 2 for synchro-nizing in both channels.`
`...`	`; Rest of program after axis replacement.`
`N… M30`	

Program "TRANSFER2" in channel 2:

Programming	**Comment**
`N… RELEASE (AX2)`	

Programming	Comment
N160 WAITM(1,1,2)	; Wait for WAIT marker in channel 1 and 2 for synchronizing in both channels.
N150 GET(AX2)	; Accept axis AX2.
...	; Rest of program after axis replacement.
N… M30	

Example 2: Axis exchange without synchronization

If the axis does not have to be synchronized no preprocessing stop is generated by GET.

Programming	Comment
N01 G0 X0	
N02 RELEASE(AX5)	
N03 G64 X10	
N04 X20	
N05 GET(AX5)	; If synchronization is not required, then this is not a block that can be executed.
N06 G01 F5000	; Block that cannot be executed.
N07 X20	; Block that cannot be executed, because X position as in N04.
N08 X30	; First block that can be executed after N05.
...	

Example 3: Activating an axis exchange without a preprocessing stop

Requirement: Axis replacement without a preprocessing stop must be configured via machine data.

Programming	Comment
N010 M4 S100	
N011 G4 F2	
N020 M5	
N021 SPOS=0	
N022 POS[B]=1	
N023 WAITP(B)	; Axis B becomes the neutral axis.
N030 X1 F10	
N031 X100 F500	
N032 X200	
N040 M3 S500	; Axis does not trigger a preprocessing stop / REORG
N041 G4 F2	
N050 M5	
N099 M30	

If the spindle or axis B is traversed, e.g. to 180 degrees and then back to 1 degree immediately after block N023 as the **PLC axis**, this axis will revert to its neutral status and will not trigger a preprocessing stop in block `N40`.

Further information

Requirements for axis replacement

- The axis must be defined in all channels that use the axis in the machine data.
- It is necessary to define to which channel the axis will be assigned after POWER ON in the **axis**-specific machine data.

Description

Release axis: RELEASE

When enabling the axis please note:

1. The axis must not be involved in a transformation.
2. All the axes involved in an axis link (tangential control) must be enabled.
3. A concurrent positioning axis cannot be replaced in this situation.
4. All the following axes of a gantry master axis are transferred with the master.
5. With coupled axes (coupled motion, master value coupling, electronic gear) only the leading axis of the group can be enabled.

Accept axis: GET

The actual axis replacement is performed with this command. The channel for which the command is programmed takes full responsibility for the axis.

Effects of GET:

Axis replacement with synchronization:

An axis always has to be synchronized if it has been assigned to another channel or the PLC in the meantime and has not been resynchronized with "WAITP", G74 or delete distance-to-go before GET.

- A preprocessing stop follows (as for STOPRE).
- Execution is interrupted until the replacement has been completed.

Automatic "GET"

If an axis is in principle available in a channel but is not currently defined as a "channel axis", GET is executed automatically. If the axis/axes is/are already synchronized no preprocessing stop is generated.

Varying the axis replacement behavior

The transfer point of axes can be set as follows using machine data:

- Automatic axis replacement between two channels then also takes place when the axis has been brought to a neutral state by WAITP (response as before)
- When requesting an axis container rotation, all axes of the axis container which can be assigned to the executing channel are brought into the channel using implicit GET or GETD. A subsequent axle replacement is only permitted again once the axis container rotation has been completed.
- When an intermediate block is inserted in the main run, a check will be made to determine whether or not reorganization is required. Reorganization is only necessary if the axis states of this block do **not** match the current axis states.
- Instead of a GET block with preprocessing stop and synchronization between preprocessing and main run, axes can be replaced without a preprocessing stop. In this case, an intermediate block is simply generated with the GET request. In the main run, when this block is executed, the system checks whether the states of the axes in the block match the current axis states.

For more information about how axis or spindle replacement works, see
Function Manual, Extended Functions, Mode Groups, Channels, Axis Replacement (K5).

2.16 Transfer axis to another channel (AXTOCHAN)

The `AXTOCHAN` language command can be used to request an axis in order to move it to a different channel. The axis can be moved to the corresponding channel both from the NC part program and from a synchronized action.

Syntax

```
AXTOCHAN(axis name,channel number[,axis name,channel number[,...]])
```

Meaning

Element	Description
`AXTOCHAN`:	Request axis for a specific channel
`Axis name`:	Axis assignment in the system: X, Y, … or entry of machine axis names concerned. The executing channel does not have to be the same channel or even the channel currently in possession of the interpolation right for the axis.
`Channel number`:	Name of the channel to which the axis is to be assigned

Note

Competing positioning axis and PLC controlled axis exclusively

A PLC axis cannot replace the channel as a competing positioning axis. An axis controlled exclusively by the PLC cannot be assigned to the NC program.

References:
Function Manual, Extended Functions; Positioning Axes (P2)

Example

AXTOCHAN in the NC program

Axes X and Y have been declared in the first and second channels. Currently, channel 1 has the interpolation right and the following program is started in that channel.

Program code	Comment
N110 AXTOCHAN(Y,2)	;Move Y axis to the second channel
N111 M0	
N120 AXTOCHAN(Y,1)	; Retrieve Y axis (neutral).
N121 M0	
N130 AXTOCHAN(Y,2,X,2)	;Move Y axis and X axis to the second channel (axes are neutral).
N131 M0	
N140 AXTOCHAN(Y,2)	; Move Y axis to the second channel (NC program).
N141 M0	

Further information

AXTOCHAN in the NC program

A GET is only executed in the event of the axis being requested for the NC program in the same channel (this means that the system waits for the state to actually change). If the axis is requested for another channel or is to become the neutral axis in the same channel, the request is sent accordingly.

AXTOCHAN from a synchronized action

In the event of an axis being requested for the same channel, AXTOCHAN from a synchronized action is mapped to a GET from a synchronized action. In this case, the axis becomes the neutral axis on the first request for the same channel. On the second request, the axis is assigned to the NC program in the same way as the GET request in the NC program. For more information about GET requests from a synchronized action, see "Motion-synchronous actions".

2.17 Activate machine data (NEWCONF)

The NEWCONF command sets all machine data of the "NEW_CONFIG" effectiveness level active. The function can also be activated in the HMI user interface by pressing the "MD data effective" softkey.

When the "NEWCONF" function is executed there is an implicit preprocessing stop; in other words, path movement is interrupted.

Syntax

```
NEWCONF
```

Meaning

NEWCONF:	Command for setting all machine data of the "NEW_CONFIG" effectiveness level active

Cross-channel execution of NEWCONF from the part program

If changes are made to axial machine data from the part program and then activated with NEWCONF, NEWCONF will only activate the machine data containing changes affecting the part program channel.

Note

In order to ensure that all changes are applied, the NEWCONF command must be executed in every channel in which the axes or functions affected by the changes to the machine data is being calculated.

No axial machine data is effective for NEWCONF.

An axial RESET must be performed for axes controlled by the PLC.

Example

Milling: Machine drill position with different technologies

```
Program code                          Comment
N10 $MA_CONTOUR_TOL[AX]=1.0           ; Change machine data.
N20 NEWCONF                           ; Activate machine data.
...
```

2.18 Write file (WRITE)

The WRITE command writes sets/data from the NC program at the end of a file (log file) in the passive file system or to external program memory. This can also be the program that is presently being executed.

Note

If no such file exists in the program memory, one will be created and can be written to using the WRITE command.

Requirement

The currently set protection level must be equal to or greater than the WRITE right of the file. If this is not the case, access is denied with an error message (return value of error variable = 13).

Syntax

```
DEF INT <error>
...
WRITE(<error>,"<file name>"/"<ExtG>","<set/data>")
```

Meaning

WRITE:	Command for appending a block or data to the end of the specified file.		
<error>:	**Parameter 1:** Variable for returning the error value		
	Type:	INT	
	Value:	0	No error
		1	Path not permitted
		2	Path not found
		3	File not found
		4	Incorrect file type
		10	File is full
		11	The file is in use
		12	No resources available
		13	No access rights
		14	Missing or unsuccessful EXTOPEN for the output device
		15	Error when writing to an external device
		16	Invalid external path has been programmed
<file name>:	**Parameter 2:** The name of the file in which the specified block or specified data is to be added.		
	Type:	STRING	
	The absolute path can be specified before the actual file name. If a path is not specified, the file is searched for in the current directory (= directory of selected program). Rules regarding path data, see "Addressing program memory files (Page 205)".		
<ExtG>:	If the data is to be output to an external device/file using the "Process DataShare" function, then the symbolic identifiers for the external device/file to be opened must be specified instead of the file name.		
	Type:	STRING	
	For further information, see "Process DataShare - Output to an external device/file (EXTOPEN, WRITE, EXTCLOSE): (Page 570)". **Note:** The identifier must be identical to the identifier specified in the EXTOPEN command.		
<block/data>:	**Parameter 3:** The block or data to be added to the specified file.		
	Type:	STRING	

Note

When writing to the passive file system or to an external program memory, the `WRITE` command implicitly inserts an "LF" character (LINE FEED = new line) at the end of the output string.

This behavior does not apply for output to an external device/file using the "Process DataShare" function. If an "LF" is also to be output, then this must be explicitly specified in the output string.

→ also refer to example 3: Implicit/explicit "LF"!

Supplementary conditions

- **Maximum file size (→ machine manufacturer)**
  The maximum possible file size of log files in the passive file system is set with the machine data:
  MD11420 $MN_LEN_PROTOCOL_FILE
  The maximum file length is applicable for all files created using the `WRITE` command in the passive file system. If it is exceeded, an error message is output and the block or data is not saved. If there is sufficient free memory, a new file can be created.

Examples

Example 1: WRITE command into the passive file system without absolute path data

Program code	Comment
`N10 DEF INT ERROR`	`; Definition of error variables.`
`N20 WRITE(ERROR,"PROT","LOG FROM 7.2.97")`	`; Write the text "LOG FROM 7.2.97" to file _N_PROT_MPF.`
`N30 IF ERROR`	`;Error evaluation.`
`N40 MSG ("Error with WRITE command:" <<ERROR)`	
`N50 M0`	
`N60 ENDIF`	
`...`	

Example 2: WRITE command into the passive file system with absolute path data

Program code
`...`
`WRITE(ERROR,"/_N_WKS_DIR/_N_PROT_WPD/_N_PROT_MPF","LOG FROM 7.2.97")`
`...`

Example 3: Implicit/explicit "LF"

a) Write to the passive file system with implicitly generated "LF"

Program code

```
...
N110 DEF INT ERROR
N120 WRITE(ERROR,"/_N_MPF_DIR/_N_MYPROTFILE_MPF","MY_STRING")
N130 WRITE(ERROR,"/_N_MPF_DIR/_N_MYPROTFILE_MPF","MY_STRING")
N140 M30
```

Output result:

MY_STRING

MY_STRING

b) Write to an external file without implicitly generated "LF"

Program code

```
...
N200 DEF STRING[30] DEV_1
N210 DEF INT ERROR
N220 DEV_1="LOCAL_DRIVE/myprotfile.mpf"
N230 EXTOPEN(ERROR,DEV_1)
N240 WRITE(ERROR,DEV_1,"MY_STRING")
N250 WRITE(ERROR,DEV_1,"MY_STRING")
N260 EXTCLOSE(ERROR,DEV_1)
N270 M30
```

Output result:

MY_STRINGMY_STRING

c) Write to an external file with explicitly generated "LF"

The following must be programmed in order to achieve the same result as under a:

Program code

```
...
N200 DEF STRING[30] DEV_1
N210 DEF INT ERROR
N220 DEV_1="LOCAL_DRIVE/myprotfile.mpf"
N230 EXTOPEN(ERROR,DEV_1)
N240 WRITE(ERROR,DEV_1,"MY_STRING'H0A'")
N250 WRITE(ERROR,DEV_1,"MY_STRING'H0A'")
N260 EXTCLOSE(ERROR,DEV_1)
N270 M30
```

Output result:

MY_STRING

MY_STRING

2.19 Delete file (DELETE)

The `DELETE` command deletes all files, irrespective of whether these were created using the `WRITE` command or not. Files that were created using a higher access authorization can also be deleted with `DELETE`.

Syntax

```
DEF INT <error>
DELETE(<error>,"<file name>")
```

Meaning

`DELETE`:	Command for deleting the specified file.		
`<error>`:	Variable for returning the error value.		
	Type.	INT	
	Value:	0	No error
		1	Path not allowed
		2	Path not found
		3	File not found
		4	Incorrect file type
		11	The file is in use
		12	No resources available
		20	Other error
`<file name>`:	Name of the file to be deleted		
	Type:	STRING	
	The absolute path can be specified before the actual file name. If a path is not specified, the file is searched for in the current directory (= directory of selected program). Rules regarding path data, see "Addressing program memory files (Page 205)".		

Example

Program code	**Comment**
`N10 DEF INT ERROR`	`; Definition of error variables.`
`N15 STOPRE`	`; Preprocessing stop.`
`N20 DELETE(ERROR,"/_N_SPF_DIR/_N_TEST1_SPF")`	`; Deletes file TEST1 in the sub-program directory.`
`N30 IF ERROR`	`; Error evaluation.`
`N40 MSG("error for DELETE command:" <<ERROR)`	
`N50 M0`	

Program code	Comment
N60 ENDIF	

2.20 Read lines in the file (READ)

The READ command reads one or several lines in the specified file and stores the information read in a STRING type array. In this array, each read line occupies an array element.

Requirement

The currently set protection level must be equal to or greater than the READ right of the file. If this is not the case, access is denied with an error message (return value of error variable = 13).

Syntax

```
DEF INT <error>
DEF STRING[<string length>] <result>[<n>,<m>]
READ(<error>,"<file name>",<start line>,<number of lines>,<result>)
```

Meaning

READ:	Command for reading lines from the specified file and storing these lines in a variable array.		
<error>:	Variable for returning the error value (call-by-reference parameter)		
	Type.	INT	
	Value:	0	No error
		1	Path not allowed
		2	Path not found
		3	File not found
		4	Incorrect file type
		11	The file is in use
		13	Insufficient access rights
		21	Line does not exist (<start line> or <number of lines> parameter exceeds the number of lines in the specified file).
		22	Field length of the result variable (<result>) is too small.
		23	Line range too large (<number of lines> parameter selected so large that the read would go beyond the end of the file).

<file name>:	Name of the file to be read (call-by-value parameter)		
	Type:	STRING	
	The absolute path can be specified before the actual file name. If a path is not specified, the file is searched for in the current directory (= directory of selected program). Rules regarding path data, see "Addressing program memory files (Page 205)".		
<start line>:	Start line of the file section to be read (call-by-value parameter)		
	Type:	INT	
	Value:	0	Reads the number of lines specified with the <number of lines> parameter before the end of the file.
		1 to n	Number of the first line to be read.
<number of lines>:	Number of lines to be read (call-by-value parameter)		
	Type:	INT	
<result>:	Result variable (call-by-reference parameter) Variable array in which the read text is stored.		
	Type:	STRING (max. length: 255)	
	If fewer lines are specified in the <number of lines> parameter than the array size [<n>,<m>] of the result variable, the remaining array elements will not be modified. Termination of a line by means of the control characters "LF" (Line Feed) or "CR LF" (Carriage Return Line Feed) is **not** stored in the result variable. Read lines are cropped if the line is longer than the defined string length. An error message is not output.		

Note

Binary files cannot be read in. The "incorrect data type" error is output (return value of the error variable = 4). The following types of file are not readable: _BIN, _EXE, _OBJ, _LIB, _BOT, _TRC, _ACC, _CYC, _NCK.

Example

Program code	Comment
N10 DEF INT ERROR	; Definition of error variables.
N20 DEF STRING[255] RESULT[5]	; Definition of result variables.
N30 READ(ERROR,"/_N_CST_DIR/_N_TESTFILE_MPF", 1,5,RESULT)	;File name with domain and file identifier and path name.
N40 IF ERROR <>0	;Error evaluation.
N50 MSG("ERROR"<<ERROR<<"ON READ COMMAND")	
N60 M0	
N70 ENDIF	
...	

2.21 Check for presence of file (ISFILE)

The `ISFILE` command checks whether a file exists in the program memory.

Syntax

```
<result>=ISFILE("<file name>")
```

Meaning

`ISFILE`:	Command to check the availability of a file		
`<file name>`:	Name of the file whose availability is to be checked.		
	Type:	STRING	
	The absolute path can be specified before the actual file name. If a path is not specified, the file is searched for in the current directory (= directory of selected program). Rules regarding path data, see "Addressing program memory files (Page 205)".		
`<result>`:	Result variable to which the result of the check is assigned.		
	Type.	BOOL	
	Value:	TRUE	File exists
		FALSE	File does not exist

Example

Program code	**Comment**
`N10 DEF BOOL RESULT`	`; Definition of result variables.`
`N20 RESULT=ISFILE("TESTFILE")`	
`N30 IF(RESULT==FALSE)`	
`N40 MSG("FILE DOES NOT EXIST")`	
`N50 M0`	
`N60 ENDIF`	
`...`	

or:

Program code	**Comment**
`N10 DEF BOOL RESULT`	`; Definition of result variables.`
`N20 RESULT=ISFILE("TESTFILE")`	
`N30 IF(NOT ISFILE("TESTFILE"))`	
`N40 MSG("FILE DOES NOT EXIST")`	
`N50 M0`	
`N60 ENDIF`	
`...`	

2.22 Read out file information (FILEDATE, FILETIME, FILESIZE, FILESTAT, FILEINFO)

The FILEDATE, FILETIME, FILESIZE, FILESTAT, and FILEINFO commands read out specific file information such as date/time of the last write access, current file size, file status or all of this information.

Requirement

The currently set protection level must be equal to or greater than the show right of the superordinate directory. If this is not the case, access is denied with an error message (return value of error variable = 13).

Syntax

```
FILE....(<error>,"<file name>",<result>)
```

Meaning

FILEDATE:	Returns the **date** of the last write access to a file		
FILETIME:	Returns the **time** of the last write access to a file		
FILESIZE:	Returns the **current size** of a file		
FILESTAT:	Returns a file with regard to the following **rights** for the **status**: • Read (r: read) • Write (w: write) • Execute (x: execute) • Show (s: show) • Delete (d: delete) **Note:** These protection levels are specific properties of the passive file system. When accessing an external program memory, FILESTAT therefore supplies default access rights (77777).		
FILEINFO:	Returns the **sum of the information** for a file that can be read out via FILEDATE, FILETIME, FILESIZE and FILESTAT		
<error>:	Variable for returning the error value (call-by-reference parameter)		
	Type.	VAR INT	
	Value:	0	No error
		1	Path not allowed
		2	Path not found
		3	File not found
		4	Incorrect file type
		13	Insufficient access rights
		22	String length of the result variable (<result>) is too small.

<file name>:	Name of the file from which the file information is to be read out			
	Type:	CHAR[160]		
	The absolute path can be specified before the actual file name. If a path is not specified, the file is searched for in the current directory (= directory of selected program). Rules regarding path data, see "Addressing program memory files (Page 205)".			
<result>:	Result variable (call-by-reference parameter) Variable in which the requested file information is stored.			
	Type:	VAR CHAR[8]	at	FILEDATE Format: "dd.mm.yy"
		VAR CHAR[8]	at	FILETIME Format: "hh:mm.ss"
		VAR INT	at	FILESIZE The file size is output in bytes.
		VAR CHAR[5]	at	FILESTAT Format: "rwxsd" (r: **r**ead, w: **w**rite, x: e**x**ecute, s: **s**how, d: **d**elete)
		VAR CHAR[32]	at	FILEINFO Format: "rwxsd nnnnnnnn dd.mm.yy hh:mm:ss"

Example

Program code	**Comment**
N10 DEF INT ERROR	; Definition of error variables.
N20 STRING[32] RESULT	; Definition of result variables.
N30 FILEINFO(ERROR,"/_N_MPF_DIR/_N_TESTFILE_MPF",RESULT)	; File name with domain, file identifier and path specification.
N40 IF ERROR <>0	; Error evaluation
N50 MSG("ERROR"<<ERROR<<"ON FILEINFO COMMAND")	
N60 M0	
N70 ENDIF	
...	

The example could return the following result in the RESULT result variable, for example:

"77777 12345678 26.05.00 13:51:30"

2.23 Roundup (ROUNDUP)

Input values, type REAL (fractions with decimal point) can be rounded up to the next higher integer number using the ROUNDUP" function.

Syntax

```
ROUNDUP(<value>)
```

Meaning

`ROUNDUP`:	Command to roundup an input value
`<value>`:	Input value, type REAL

Note

Input value, type INTEGER (an integer number) is returned unchanged.

Examples

Example 1: Various input values and their rounding up results

Example	Rounding up result
`ROUNDUP(3.1)`	4.0
`ROUNDUP(3.6)`	4.0
`ROUNDUP(-3.1)`	-3.0
`ROUNDUP(-3.6)`	-3.0
`ROUNDUP(3.0)`	3.0
`ROUNDUP(3)`	3.0

Example 2: ROUNDUP in the NC program

Program code

```
N10 X=ROUNDUP(3.5) Y=ROUNDUP(R2+2)
N15 R2=ROUNDUP($AA_IM[Y])
N20 WHEN X=100 DO Y=ROUNDUP($AA_IM[X])
...
```

2.24 Subprogram technique

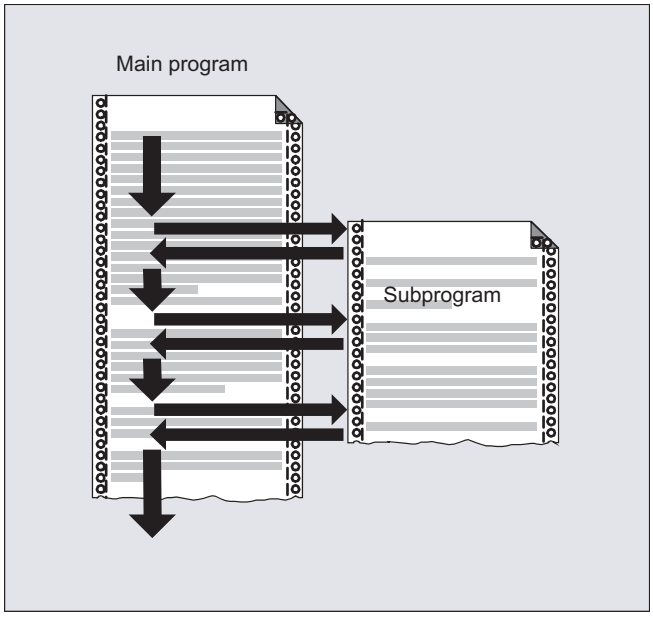
2.24.1 General information

2.24.1.1 Subprogram

The term "subprogram" has its origins during the time when part programs were split strictly into main and subprograms. Main programs were the part programs selected for processing on the control and then launched. Subprograms were the part programs called from within the main program.

This strict division no longer exists with today's SINUMERIK NC language. In principle, each part program can be selected as a main program and launched or called from another part program as a subprogram.

Accordingly, the subprogram can then be used to refer to a part program called from within another part program.



Application

As in all high-level programming languages, in the NC language, subprograms swaps out program sections used more than once to independent, self-contained programs.

Subprograms offer the following advantages:

- Increase the transparency and readability of programs
- Increase quality by reusing tested program parts
- Offer the possibility of creating specific machining libraries
- Save memory space

2.24.1.2 Subprogram names

Naming rules

The subprogram name can be chosen freely providing the following rules are observed:

- Permissible characters:
  - Letters: A ... Z, a ... z
  - Numbers: 0 ... 9
  - Underscore: _
- The first two characters **must** be two letters or an underscore followed by a letter.
- Maximum length: 24 characters

Uppercase/lowercase letters

The SINUMERIK NC language does **not** distinguish between uppercase and lowercase letters.

Control-internal extensions

The program name assigned when the subprogram is created is expanded within the control with the addition of a prefix and a suffix.

- Prefix: `_N_`
- Suffix: `_SPF`

Using the program name

When using the program name, e.g. in the context of a subprogram call, all combinations of prefix, program name, and suffix are possible.

Example:

The subprogram with the program name `SUB_PROG` can be started using the following identifiers:

1. `SUB_PROG`
2. `_N_SUB_PROG`
3. `SUB_PROG_SPF`
4. `_N_SUB_PROG_SPF`

Main programs and subprograms with the same name

If a main program (.MPF) and a subprogram (.SPF) exist with the same program name, the appropriate file extension for the unique identification must be specified when the program name in the NC program is used. Otherwise the program found first in the search path with the specified name is used.

2.24.1.3 Nesting of subprograms

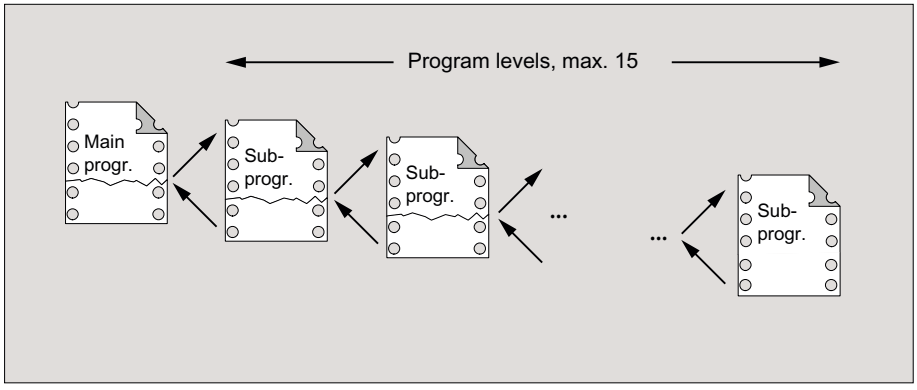
A main program can call subprograms which in turn call more subprograms. As such, the sequences of the programs are nested within each other. Each program runs on a dedicated program level.

Nesting depth

The NC language currently provides 16 program levels. The main program always runs at the uppermost program level, 0. A subprogram always runs at the next lowest program level following the call. Program level 1 is, therefore, the first subprogram level.

Division of program levels:

- Program level 0: Main program level
- Program level 1 to 15: Subprogram level 1 to 15



Interrupt routines (ASUB)

If a subprogram is called in the context of an interrupt routine, this will not be executed at the program level currently active in the channel (n) but at the next lowest program level (n+1). So that this remains possible even at the lowest program level, 2 additional program levels (16 and 17) are available in conjunction with interrupt routines.

If more than 2 program levels are required, this has to be taken into account explicitly in the structuring of the part program executed in the channel. In other words, only a maximum of as many program levels may be used in order to leave sufficient program levels available for interrupt processing.

If interrupt processing needs 4 program levels for example, the part program must be structured so that it uses a maximum of up to program level 13. In the event of an interrupt, the 4 program levels it requires (14 to 17) will be available to it.

Siemens cycles

Siemens cycles need 3 program levels. Therefore, a Siemens cycle must be called at the latest in:

- Part program processing: program level 12
- interrupt routine: program level 14

2.24.1.4 Search path

When a subprogram without path details is called, the control system searches the available program memory using a predefined search sequence (see "Search path for subprogram call (Page 209)").

2.24.1.5 Formal and actual parameters

Formal and actual parameters occur in conjunction with the definition and calling of subprograms with parameter transfer.

Formal parameter

When a subprogram is defined, the parameters to be transferred to it (known as the formal parameters) have to be defined with type and parameter name.

The formal parameters define, therefore, the interface of the subprogram.

Example:

```
Program code                          Comment
PROC CONTOUR (REAL X, REAL Y)         ; Formal parameters: X and Y, both REAL type
N20 X1=X Y1=Y                         ; Traversing of axis X1 to position X and axis
                                      Y1 to position Y
...
N100 RET
```

Actual parameters

When a subprogram is called, absolute values or variables (known as actual parameters) have to be transferred to it.

As such, the actual parameters assign up-to-date values to the interface of the subprogram when the latter is called.

Example:

```
Program code                          Comment
N10 DEF REAL WIDTH                    ; Variable definition
N20 WIDTH=20.0                        ; Variable assignment
N30 CONTOUR(5.5, WIDTH)               ; Subprogram call with actual parameters: 5.5
                                      and WIDTH
...
N100 M30
```

2.24.1.6 Parameter transfer

Definition of a subprogram with parameter transfer

A subprogram with parameter transfer is defined using the `PROC` keyword and a complete list of all the parameters expected by the subprogram.

Incomplete parameter transfer

When the subprogram is called, not all the parameters defined in the subprogram interface have to be transferred explicitly. If a parameter is omitted, the default value "0" is transferred for it.

So that the parameter sequence can be uniquely identified, however, the commas used as parameter separators always have to be included. The last parameter is an exception. If it is omitted from the call, the last comma can also be left out.

Example:

Subprogram:

Program code	Comment
`PROC SUB_PROG (REAL X, REAL Y, REAL Z)`	`; Formal parameters: X, Y, and Z`
`...`	
`N100 RET`	

Main program:

Program code	Comment
`PROC MAIN_PROG`	
`...`	
`N30 SUB_PROG(1.0,2.0,3.0)`	`; Subprogram call with complete parameter transfer: X=1.0, Y=2.0, Z=3.0`
…	
`N100 M30`	

Examples for the subprogram call in `N30` with incomplete parameter transfer:

```
N30 SUB_PROG( ,2.0,3.0)          ; X=0.0, Y=2.0, Z=3.0
N30 SUB_PROG(1.0, ,3.0)          ; X=1.0, Y=0.0, Z=3.0
N30 SUB_PROG(1.0,2.0)            ; X=1.0, Y=2.0, Z=0.0
N30 SUB_PROG( , ,3.0)            ; X=0.0, Y=0.0, Z=3.0
N30 SUB_PROG( , , )              ; X=0.0, Y=0.0, Z=0.0
```

NOTICE

Call-by-reference parameter transfer

Parameters transferred using call-by-reference must not be left out of the subprogram call.

NOTICE
AXIS data type
AXIS data type parameters must not be left out of the subprogram call.

Checking the transfer parameters

System variable $P_SUBPAR [n] where n = 1, 2, etc., can be used to check whether a parameter has been transferred explicitly or left out in the subprogram. The index n refers to the sequence of the formal parameters. Index n = 1 refers to the first formal parameter, index n = 2 to the second formal parameter, and so on.

The following program excerpt shows an example of how a check can be performed based on the first formal parameter:

```
Programming                                  Comment
PROC SUB_PROG (REAL X, REAL Y, REAL Z)       ; Formal parameters: X, Y, and Z
N20 IF $P_SUBPAR[1]==TRUE                    ; Check of the first formal parameter
                                               X.
...                                          ; These actions are taken if the formal
                                               parameter X has been transferred ex-
                                               plicitly.
N40 ELSE
...                                          ; These actions are taken if the formal
                                               parameter X has not been transferred.
N60 ENDIF
...                                          ; General actions
N100 RET
```

2.24.2 Definition of a subprogram

2.24.2.1 Subprogram without parameter transfer

When defining subprograms without parameter transfer, the definition line at the beginning of the program can be omitted.

Syntax

```
[PROC <program name>]
...
```

Meaning

PROC:	Definition operation at the beginning of a program
<program name>:	Name of the program

Example

Example 1: Subprogram with PROC operation

```
Program code                    Comment
PROC SUB_PROG                   ; Definition line
N10 G01 G90 G64 F1000
N20 X10 Y20
...
N100 RET                        ; Subprogram return
```

Example 2: Subprogram without PROC operation

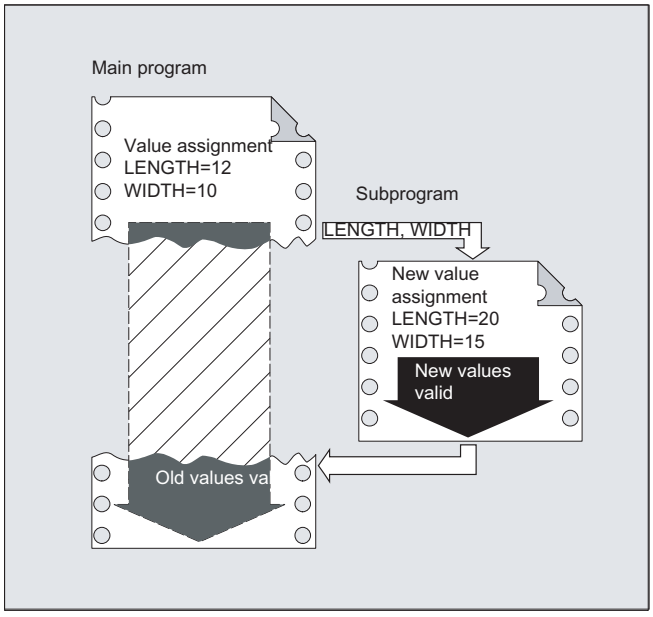
```
Program code                    Comment
N10 G01 G90 G64 F1000
N20 X10 Y20
...
N100 RET                        ; Subprogram return
```

2.24.2.2 Subprogram with call-by-value parameter transfer (PROC)

A subprogram with call-by-reference parameter transfer is defined using the PROC keyword followed by the name of the program and a complete list of all the parameters with their type and name. The definition operation must appear in the first program line.

Call-by-value

The calling program transfers only the value of a variable to the subprogram on a call-by-reference parameter transfer. Thus the subprogram is not given direct access to the variable. In this way, only the value visible in the subprogram is modified when the parameter value is changed. The value of the variables defined in the calling program remains unchanged. Call-by-reference parameter transfer therefore does not affect the calling program.



Syntax

```
PROC <program name> (<parameter type> <parameter
name>=<init_value>, ...)
```

Note

Up to 127 parameters can be transferred.

Meaning

PROC:	Definition operation at the beginning of a program
<program name>:	Name of the program
<parameter type>:	Data type of the parameter (e.g. REAL, INT, BOOL)
<parameter name>:	Name of the parameter
<init_value>:	Optional value for the initialization of the parameter (**optional**) If no parameter is specified when calling the subprogram, the parameter is assigned the initialization value.

Example

Definition of a subprogram SUB_PROG with three parameters of type REAL with default values:

Program code

```
PROC SUB_PROG(REAL LENGTH=10.0, REAL WIDTH=20.0, REAL
HIGHT=30.0)
...
```

Program code

```
N100 RET
```

Different call variants:

Program code

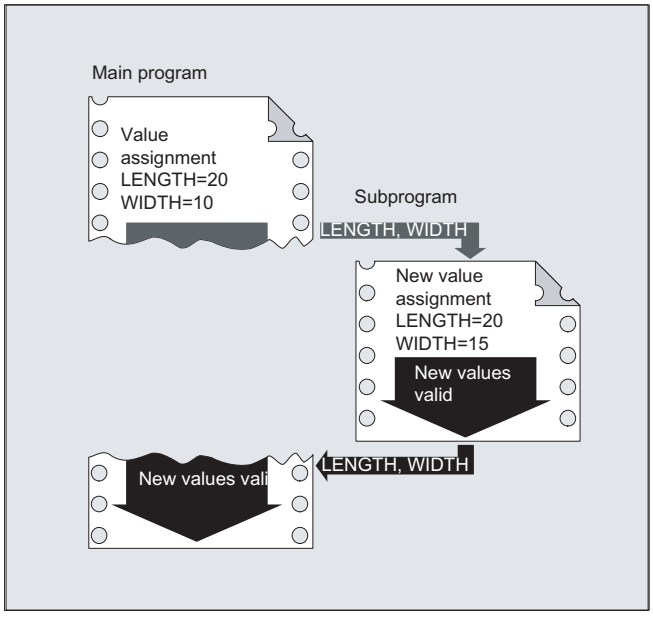
```
PROC MAIN_PROG
  REAL PAR_1 = 100
  REAL PAR_2 = 200
  REAL PAR_3 = 300
  ; Call variants
  SUB_PROG
  SUB_PROG(PAR_1, PAR_2, PAR_3)
  SUB_PROG(PAR_1)
  SUB_PROG(PAR_1, , PAR_3)
  SUB_PROG( , , PAR_3)
N100 RET
```

2.24.2.3 Subprogram with call-by-reference parameter transfer (PROC, VAR)

A subprogram with call-by-reference parameter transfer is defined using the `PROC` keyword followed by the name of the program and a complete list of all the parameters with the `VAR` keyword, type, and name. The definition operation must appear in the first program line. As parameters, references to arrays can also be transferred.

Call-by-reference

The calling program transfers not the value of a variable to the subprogram on a call-by-reference parameter transfer, but a reference (pointer) to the variable. This gives the subprogram direct access to the variable. In this way, not only the value visible in the subprogram is modified when a parameter value is changed, but also the value of the variables defined in the calling program. Call-by-reference parameter transfer therefore affects the calling program, even after the subprogram has ended.



Note

The call-by-reference parameter transfer is then only necessary if the transferred variable was defined locally in the calling program (LUD). Channel-global or NC-global variables do not have to be transferred, since these cannot be accessed directly from within the subprogram.

Syntax

```
PROC <program name> (VAR <parameter type> <parameter name>, etc.)
PROC <program name> (VAR <array type> <array name>, [<m>,<n>,<o>],
etc.)
```

Note

Up to 127 parameters can be transferred.

Meaning

PROC:	Definition operation at the beginning of a program
VAR:	Keyword for parameter transfer via reference
<program name>:	Name of the program
<parameter type>:	Data type of the parameter (e.g. REAL, INT, BOOL)
<parameter name>:	Name of the parameter
<array type>:	Data type of the array elements (e.g. REAL, INT, BOOL)
<array name>:	Name of the array

[<m>,<n>,<o>]:	Array size Currently, up to 3-dimensional arrays are possible:	
	<m>:	Array size for 1st dimension
	<n>:	Array size for 2nd dimension
	<o>:	Array size for 3rd dimension

Note

- The program name specified after the PROC keyword must match the program name assigned on the user interface.
- With arrays of an undefined array length, subprograms can process arrays of variable length as formal parameter. When defining a two-dimensional array as a formal parameter, for example, the length of the 1st dimension is not specified. However, the comma must be written.
  Example: PROC <program name> (VAR REAL ARRAY[,5])

Example

Definition of a subprogram with two parameters as reference to REAL type:

Program code	Comment
PROC SUB_PROG(VAR REAL LENGTH, VAR REAL WIDTH)	; Parameter 1: Reference to type: REAL, name: LENGTH Parameter 2: Reference to type: REAL, name: WIDTH
...	
N100 RET	

2.24.2.4 Save modal G functions (SAVE)

The SAVE attribute means that before the subprogram call, active modal G functions are saved and are reactivated after the end of the subprogram.

NOTICE

Interrupt continuous-path mode

If, for active continuous-path mode, a subprogram is called with the SAVE attribute, the continuous-path mode is interrupted at the end of the subprogram (return jump).

Syntax

```
PROC <subprogram name> SAVE
```

Meaning

SAVE:	Saves the modal G functions before the subprogram call and restores after the end of the subprogram.

Example

The modal G function G91 is effective in the CONTOUR subprogram (incremental dimension). The modal G function G90 is effective in the main program (absolute dimension). G90 is again effective in the main program after the end of the subprogram due to the subprogram definition with SAVE.

Subprogram definition:

Program code	Comment
PROC CONTOUR (REAL VALUE1) SAVE	; Subprogram definition with the SAVE parameter
N10 G91 ...	; Modal G function G91: Incremental dimension
N100 M17	; End of subprogram

Main program:

Program code	Comment
N10 G0 X... Y... G90	; Modal G function G90: Absolute dimensions
N20 ...	
...	
N50 CONTOUR (12.4)	;Subprogram call
N60 X... Y...	; Modal G function G90 reactivated using SAVE

Supplementary conditions

Frames

The behavior of frames regarding subprograms with the SAVE attribute depends on the frame time and can be set using machine data.

References

Function Manual, Basic Functions; Axes, Coordinate Systems, Frames (K2),
Section: "Subprogram return with SAVE"

2.24.2.5 Suppress single block execution (SBLOF, SBLON)

Single-block suppression for the complete program

Programs designated with SBLOF are completely executed just like a block when single-block execution is active, i.e. single-block execution is suppressed for the complete program.

SBLOF is in the PROC line and is valid up to the end of the subprogram or until it is interrupted. At the return command, the decision is made whether to stop at the end of the subprogram:

Return jump with M17:	Stop at the end of the subprogram
Return jump with RET:	No stop at end of subprogram

Single-block suppression within the program

SBLOF alone must remain in the block. Single block is deactivated after this block until:

- The next SBLON
  or
- The end of the active subprogram level

Syntax

Single-block suppression for the complete program:
```
PROC ... SBLOF
```

Single-block suppression within the program:

```
SBLOF
...
SBLON
```

Meaning

PROC:	First operation in a program
SBLOF:	Command to deactivate single-block execution SBLOF can be written in a PROC block or alone in the block.
SBLON:	Command to activate single-block execution SBLON must be in a separate block.

Supplementary conditions

- **Single-block suppression and block display**
  The current block display can be suppressed in cycles/subprograms using DISPLOF. If DISPLOF is programmed together withSBLOF, then the cycle/subprogram call continues to be displayed on single-block stops within the cycle/subprogram.
- **Single-block suppression in the system ASUB or user ASUB**
  If the single-block stop in the system or user ASUB is suppressed using the settings in machine data MD10702 $MN_IGNORE_SINGLEBLOCK_MASK (bit0 = 1 or bit1 = 1), then the single-block stop can be reactivated by programming SBLON in the ASUB.
  If the single-block stop in the user ASUB is suppressed using the setting in machine data MD20117 $MC_IGNORE_SINGLEBLOCK_ASUP, then the single-block stop **cannot** be reactivated by programming SBLON in the ASUB.
- **Special features of single-block suppression for various single-block execution types**
  When single-block execution SBL2 is active (stop after each part program block) there is **no** execution stop in the SBLON block if bit 12 is set to "1" in the MD10702 $MN_IGNORE_SINGLEBLOCK_MASK (prevent single-block stop).
  When single-block execution SBL3 is active (stop after every part program block - also in the cycle), the SBLOF command is suppressed.

Examples

Example 1: Single-block suppression within a program

Program code	Comment
N10 G1 X100 F1000	
N20 SBLOF	; Deactivate single block.
N30 Y20	
N40 M100	
N50 R10=90	
N60 SBLON	; Reactivate single block.
N70 M110	
N80 ...	

The area between N20 and N60 is executed as one step in single-block mode.

Example 2: A cycle is to act like a command for a user

Main program:

Program code
N10 G1 X10 G90 F200
N20 X-4 Y6
N30 CYCLE1
N40 G1 X0
N50 M30

Cycle CYCLE1:

```
Program code                                    Comment
N100 PROC CYCLE1 DISPLOF SBLOF                  ; Suppress single block
N110 R10=3*SIN(R20)+5
N120 IF (R11 <= 0)
N130    SETAL(61000)
N140 ENDIF
N150 G1 G91 Z=R10 F=R11
N160 M17
```

CYCLE1 is processed for active single-block execution, i.e. the Start key must be pressed once to process CYCLE1.

Example 3: An ASUB, which is started by the PLC in order to activate a modified zero offset and tool offsets, is to be executed invisibly.

```
Program code
N100 PROC ZO SBLOF DISPLOF
N110 CASE $P_UIFRNUM OF        0 GOTOF _G500
                               1 GOTOF _G54
                               2 GOTOF _G55
                               3 GOTOF _G56
                               4 GOTOF _G57
                               DEFAULT GOTOF END
N120 _G54: G54 D=$P_TOOL T=$P_TOOLNO
N130 RET
N140 _G54: G55 D=$P_TOOL T=$P_TOOLNO
N150 RET
N160 _G56: G56 D=$P_TOOL T=$P_TOOLNO
N170 RET
N180 _G57: G57 D=$P_TOOL T=$P_TOOLNO
N190 RET
N200 END: D=$P_TOOL T=$P_TOOLNO
N210 RET
```

Example 4: Is not stopped with MD10702 Bit 12 = 1

Initial situation:

- Single-block execution is active.
- MD10702 $MN_IGNORE_SINGLEBLOCK_MASK Bit12 = 1

Main program:

Program code	Comment
N10 G0 X0	; Stop in this part program line.
N20 X10	; Stop in this part program line.
N30 CYCLE	; Traversing block generated by the cycle.
N50 G90 X20	; Stop in this part program line.
M30	

Cycle CYCLE:

Program code	Comment
PROC CYCLE SBLOF	; Suppress single-block stop.
N100 R0 = 1	
N110 SBLON	; Execution is not stopped in the part program line due to the fact that MD10702 bit12=1.
N120 X1	; Execution is stopped in this part program line.
N140 SBLOF	
N150 R0 = 2	
RET	

Example 5: Single-block suppression for program nesting

Initial situation:

Single-block execution is active.

Program nesting:

Program code	Comment
N10 X0 F1000	; Execution is stopped in this block.
N20 UP1(0)	
PROC UP1(INT _NR) SBLOF	; Suppress single-block stop.
N100 X10	
N110 UP2(0)	
PROC UP2(INT _NR)	
N200 X20	
N210 SBLON	; Activate single-block stop.
N220 X22	; Execution is stopped in this block.
N230 UP3(0)	
PROC UP3(INT _NR)	
N300 SBLOF	; Suppress single-block stop.
N305 X30	
N310 SBLON	; Activate single-block stop.
N320 X32	; Execution is stopped in this block.

Program code	Comment
N330 SBLOF	; Suppress single-block stop.
N340 X34	
N350 M17	; SBLOF is active.
N240 X24	; Execution is stopped in this block. SBLON is active.
N250 M17	; Execution is stopped in this block. SBLON is active.
N120 X12	
N130 M17	; Execution is stopped in this return jump block. SBLOF of the PROC statement is active.
N30 X0	; Execution is stopped in this block.
N40 M30	; Execution is stopped in this block.

Further information

Single-block disable for unsynchronized subprograms

In order to execute an ASUB in one step, a PROC statement must be programmed in the ASUB with SBLOF. This also applies to the function "Editable system ASUB" (MD11610 $MN_ASUP_EDITABLE).

Example of an editable system ASUB:

Program code	Comments
N10 PROC ASUB1 SBLOF DISPLOF	
N20 IF $AC_ASUP==’H200’	
N30 RET	; No REPOS for mode change.
N40 ELSE	
N50 REPOSA	; REPOS in all other cases.
N60 ENDIF	

Program control in single-block mode

With the single-block execution function, the user can execute a part program block-by-block. The following setting types exist:

- SBL1: IPO single block with stop after each machine function block.
- SBL2: Single block with stop after each block.
- SBL3: Stop in the cycle (the SBLOF command is suppressed by selecting SBL3).

Single-block suppression for program nesting

If SBLOF was programmed in the PROC statement in a subprogram, then execution is stopped at the subprogram return jump with M17. That prevents the next block in the calling program from already running. If SBLOF, without SBLOF is programmed in the PROC statement in a subprogram, single-block suppression is activated, execution is only stopped after the next machine function block of the calling program. If that is not wanted, SBLON must be programmed in the subprogram before the return (M17). Execution does not stop for a return jump to a higher-level program with RET.

2.24.2.6 Suppress current block display (DISPLOF, DISPLON, ACTBLOCNO)

The current program block is displayed as standard in the block display. The display of the current block can be suppressed in cycles and subprograms using the DISPLOF command. Instead of the current block, the call of the cycle or the subprogram is displayed. The DISPLON command revokes suppression of the block display.

DISPLOF and DISPLON are programmed in the program line with the PROC operation and are effective for the entire subprogram and implicitly for all subprograms called from it which do not contain a DISPLON or DISPLOF command. This is true for all ASUBs.

Syntax

```
PROC … DISPLOF
PROC … DISPLOF ACTBLOCNO
PROC … DISPLON
```

Meaning

DISPLOF:	Command to suppress the current block display.	
	Location:	At the end of the program line with the PROC operation
	Effective:	Up to the return jump from the subprogram or end of program.
	Note: If further subprograms are called from the subprogram using the DISPLOF command, then the current block display is also suppressed in these subprograms unless DISPLON is explicitly programmed in them.	
DISPLON:	Command for revoking suppression of the display of the current block	
	Location:	At the end of the program line with the PROC operation
	Effective:	Up to the return jump from the subprogram or end of program.
	Note: If further subprograms are called from the subprogram using the DISPLON command, then the current block will also be displayed in these subprograms unless DISPLOF is explicitly programmed in them.	
ACTBLOCNO:	DISPLOF together with the ACTBLOCNO attribute means that in the case of an alarm, the number of the actual block is output in which the alarm occurred. This also applies if only DISPLOF is programmed in a lower program level. On the other hand, for DISPLOF without ACTBLOCNO, the block number of the cycle or subprogram call from the last program level not designated with DISPLOF is displayed.	

Examples

Example 1: Suppress current block display in the cycle

Program code	Comment
PROC CYCLE (AXIS TOMOV, REAL POSITION) SAVE DISPLOF	; Suppress current block display Instead, the cycle call should be displayed, e.g.: CYCLE(X,100.0)
DEF REAL DIFF	;Cycle contents
G01 ...	

Program code	Comment
...	
RET	; Subprogram return jump. The block following the cycle call is displayed in the block display.

Example 2: Block display for alarm output

Subprogram SUBPROG1 (with ACTBLOCNO):

Program code	Comment
PROC SUBPROG1 DISPLOF ACTBLOCNO	
N8000 R10 = R33 + R44	
...	
N9040 R10 = 66 X100	; Trigger alarm 12080
...	
N10000 M17	

Subprogram SUBPROG2 (without ACTBLOCNO):

Program code	Comment
PROC SUBPROG2 DISPLOF	
N5000 R10 = R33 + R44	
...	
N6040 R10 = 66 X100	; Trigger alarm 12080
...	
N7000 M17	

Main program:

Program code	Comment
N1000 G0 X0 Y0 Z0	
N1010 ...	
...	
N2050 SUBPROG1	; Alarm output = "12080 channel K1 block N9040 syntax error for text R10="
N2060 ...	
N2350 SUBPROG2	; Alarm output = "12080 channel K1 block N2350 syntax error for text R10="
...	
N3000 M30	

Example 3: Revoke suppression of the current block display

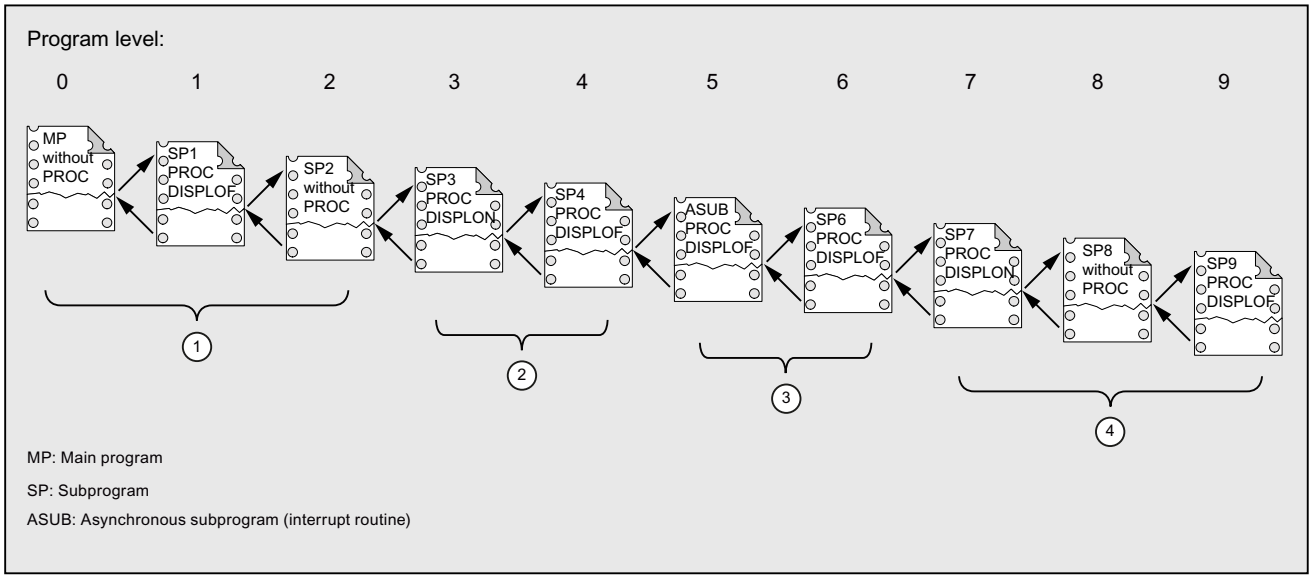
Subprogram SUB1 with suppression:

Program code	Comment
PROC SUB1 DISPLOF	; Suppress current block display in SUB1 subprogram. Instead, the block is to be displayed with the SUB1 call.
...	
N300 SUB2	; Call subprogram SUB2.
...	
N500 M17	

Subprogram SUB2 without suppression:

Program code	Comment
PROC SUB2 DISPLON	; Revoke suppression of the current block display in subprogram SUB2.
...	
N200 M17	; Return to subprogram SUB1. Suppression of the current block display is restored in SUB1.

Example 4: Display response for different DISPLON/DISPLOF combinations



① The part program lines from program level 0 are displayed in the current block display.

② The part program lines from program level 3 are displayed in the current block display.

③ The part program lines from program level 3 are displayed in the current block display.

④ The part program lines from program level 7/8 are displayed in the current block display.

2.24.2.7 Identifying subprograms with preparation (PREPRO)

All files can be identified with the PREPRO keyword at the end of the PROC operation line during power up.

Note

This type of program preparation depends on the relevant set machine data. Please follow the manufacturer's instructions.

References:
Function Manual, Special Functions, Preprocessing (V2)

Syntax

```
PROC … PREPRO
```

Meaning

PREPRO:	Keyword for identifying all files (of the NC programs stored in the cycle directories) prepared during power up

Read subprogram with preparation and subprogram call

The cycle directories are processed in the same order both for subprograms preprocessed with parameters during power up and during subprogram call.

1. _N_CUS_DIR user cycles
2. _N_CMA_DIR manufacturer cycles
3. _N_CST_DIR standard cycles

In the case of NC programs sharing the same name but having different characteristics, the first PROC operation found is activated and the other PROC operation is overlooked without an alarm message.

2.24.2.8 Subprogram return M17

The return command M17 (or the part program end command M30) appears at the end of a subprogram. It prompts the return to the calling program at the part program block following the subprogram call.

Note

M17 and M30 are treated as equivalents in the NC language.

Syntax

```
PROC <program name>
```

```
...
M17/M30
```

Supplementary conditions

Effect of the subprogram return on continuous-path mode

If M17 (or M30) appears on its own in the part program block, active continuous-path mode in the channel will be interrupted.

To avoid continuous-path mode being interrupted, M17 (or M30) has to be included in the last traversing block. Furthermore, the following machine data must be set to "0":

MD20800 $MC_SPF_END_TO_VDI = 0 (no M30/M17 output to the NC/PLC interface)

Example

1. Subprogram with M17 in a separate block

Program code	Comment
N10 G64 F2000 G91 X10 Y10	
N20 X10 Z10	
N30 M17	; Return jump with interruption of continuous-path mode.

2. Subprogram with M17 in the last traversing block

Program code	Comment
N10 G64 F2000 G91 X10 Y10	
N20 X10 Z10 M17	; Return jump without interruption of continuous-path mode.

2.24.2.9 RET subprogram return

The RET command can also be used in the subprogram as a substitute for the M17 return jump command. RET must be programmed in a separate part program block. Like M17, RET prompts the return to the calling program at the part program block following the subprogram call.

Note

Parameters can be programmed to change the return jump behavior of RET (see "Parameterizable subprogram return jump (RET ...) (Page 168)").

Application

The RET operation should then be used if a G64 continuous-path mode (G641 to G645) is not to be interrupted by the return jump.

Requirement

The RET command can only be used in subprograms, which were not defined with the SAVE attribute.

Syntax

```
PROC <program name>
...
RET
```

Example

Main program:

Program code	Comment
PROC MAIN_PROGRAM	; Start of the program
...	
N50 SUB_PROG	; Subprogram call: SUB_PROG
N60 ...	
...	
N100 M30	; End of program

Subprogram:

Program code	Comment
PROC SUB_PROG	
...	
N100 RET	; Return jump to block N60 in the main program.

2.24.2.10 Parameterizable subprogram return jump (RET ...)

Generally, a return jump is made from a subprogram into the calling program using the RET command. Processing is then continued with the program line following the subprogram call. The following options are available if program processing is to be continued at another location:

- Resuming program execution after calling the stock removal cycles in the ISO dialect mode (after describing the contour).
- Return to main program from any subprogram level (even after ASUB) for error handling.
- Return jump through several program levels for special applications in compile cycles and in the ISO dialect mode.

To achieve this, the RET command should be programmed with additional parameters.

Search direction

When specifying parameter <target block>, a return jump is first made to the block after the calling block. A search is then made for the target in the direction of the **end** of the program

into which a return jump was made. A search is made toward the start of the program if the search was not successful.

Syntax

```
RET("<target block>")
RET("<target block>",<block after target block>)
RET("<target block>",<block after target block> <number of return
jump levels>)
RET("<target block>", ,<number of return jump levels>)
RET("<target block>",<block after target block>,<number of return
jump levels>,
<return jump to the beginning of the program>)
RET( , ,<number of return jump levels>,<return jump to the beginning
of the program>)
```

Meaning

RET:	End of subprogram		
<target block>:	Declares as jump target the block where program execution should be resumed. If parameter < number of return jump levels> is not programmed, then the jump target is in the program from which the current subprogram was called. Possible data include:		
	<block number>	Number of the target block. The search for the block number is realized in the program to which a return jump was made initially in the direction toward the end of the program.	
	<jump marker>	Jump marker, which must be available in the program into which a return jump is made. The search for the jump marker is realized in the program to which a return jump was made initially in the direction toward the end of the program.	
	<character string>	Character string that must be available in the program into which a return jump is made (e.g. program or variable name). The search for the character string is realized in the program to which a return jump was made initially in the direction toward the end of the program. The following rules apply when programming the character string: • **Blank at the end** (contrary to the jump marker, that is designated using ":" at the end). • **Before the character string** only one block number and/or a jump marker may be set, **no program commands**.	
<block after target block>:	The parameter specifies as to whether program processing should be continued in the block specified under parameter <target block> or in the following block.		
	Type:	INT	
	Value:	0	The return jump is made to the block specified in parameter <target block>.
		> 0	The return jump is made to the next block specified in parameter <target block>.

<number of return jump levels>:	The parameter specifies the number of program levels that should be jumped through (return jumps) to search there for the target block and continue processing the program.		
	Type:	INT	
	Value:	1	The program is resumed at the "current program level - 1" (like RET without parameter).
		2	The program is resumed at the "current program level - 2", i.e. one level is skipped.
		3	The program is resumed at the "current program level - 3", i.e. two levels are skipped.
		...	
	Range of values:	1 ... 15	
<return jump to the beginning of the program>:	The parameter specifies whether, for a return jump into the main program, the program should be continued at the start of the program in the active **ISO dialect mode**.		
	Type:	BOOL	
	Value:	1	If the return jump is made into the main program and an **ISO dialect mode** is active there, then the program branches to the beginning of the program.

Note

For a subprogram return jump with a character string to specify the target block search, initially, a search is always made for a jump marker in the calling program.

If a jump target is to be uniquely defined using a character string, it is not permissible that the character string matches the name of a jump marker, as otherwise the subprogram return jump would always be made to the jump marker and not to the character string (refer to example 2).

Supplementary conditions

When making a return jump through several program levels, the SAVE statements of the individual program levels are evaluated.

If, for a return jump over several program levels, a modal subprogram is active and if in one of the skipped programs the deselection command MCALL is programmed for the modal subprogram, then the modal subprogram remains active.

NOTICE

Programming error

For a return jump through several program levels it is the user's responsibility to ensure that processing is continued with the necessary modal settings. This can be achieved, e.g. by programming an appropriate main block.

Examples

Example 1: Resuming in the main program after ASUB execution

Programming	Comment
N10010 CALL "UP1"	;Program level 0 (main program)
N11000 PROC UP1	; Program level 1
N11010 CALL "UP2"	
N12000 PROC UP2	; Program level 2
...	
N19000 PROC ASUB	; Program level 3 (ASUB execution)
...	
N19100 RET("N10900", ,$P_STACK)	; Subprogram return jump into the main program
	; $P_STACK: actual program level
N10900	; Target block in the main program
N10910 MCALL	; Deactivate the modal subprogram call
N10920 G0 G60 G40 M5	; Initialize additional modal settings

Example 2: Character string (string>) to specify the target block search

Main program:

Program code	Comment
PROC MAIN_PROGRAM	
N1000 DEF INT iVar1=1, iVar2=4	
N1010 ...	
N1200 subProg1	; Calls subprogram "subProg1"
N1210 M2 S1000 X10 F1000	
N1220	
N1400 subProg2	; Calls subprogram "subProg2"
N1410 M3 S500 Y20	
N1420 ..	
N1500 lab1: iVar1=R10*44	
N1510 F500 X5	
N1520 ...	
N1550 subprog1: G1 X30	; "subProg1" is defined here as jump marker.
N1560 ...	
N1600 subProg3	; Calls subprogram "subProg3"
N1610 ...	
N1900 M30	

Subprogram subProg1:

Program code	Comment
PROC subProg1	

Program code	Comment
N2000 R10=R20+100	
N2010 ...	
N2200 RET("subProg2")	; Return jump into the main program at block N1400

Subprogram subProg2:

Program code	Comment
PROC subProg2	
N2000 R10=R20+100	
N2010 ...	
N2200 RET("iVar1")	; Return jump into the main program at block N1500

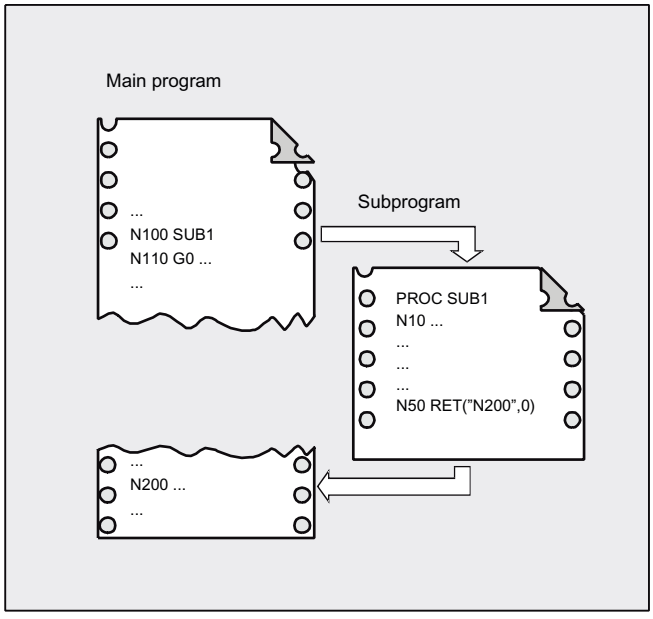
Subprogram subProg3:

Program code	Comment
PROC subProg3	
N2000 R10=R20+100	
N2010 ...	
N2200 RET("subProg1")	; Return jump into the main program at block N1550

Additional information

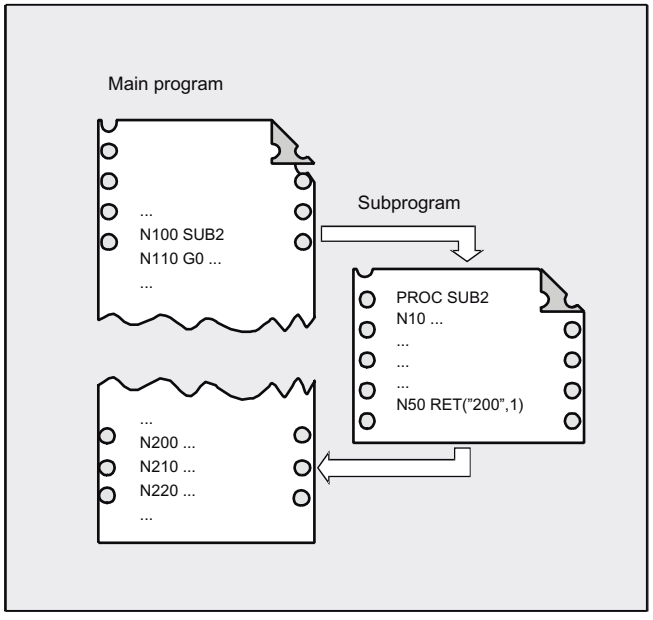
The following diagrams show the different effects of return jump parameters

1. <target block> = "N200", <block after target block> = 0



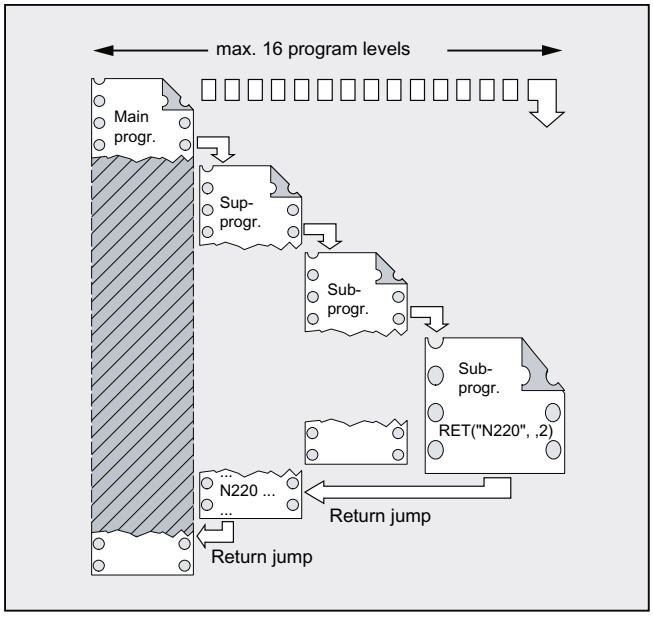
After the `RET` command, program execution is continued with block `N200` in the main program.

2. <target block> = "N200", <block after target block> = 1



After the `RET` command, program execution is continued with the block (`N210`) that follows block `N200` in the main program.

3. <target block> = "N220", <number of return jump levels> = 2



After the `RET` command, two program levels are jumped through and program execution is continued with block `N220`.

2.24.2.11 Parameterizable subprogram return jump (RETB ...)

Generally, a return jump is made from a subprogram into the calling program using the `RETB` command. Processing is then continued with the program line following the subprogram call. The following options are available if program processing is to be continued at another location:

- Resume program execution after calling the stock removal cycles in the ISO dialect mode (after describing the contour).
- Return to main program from any subprogram level (even after ASUB) for error handling.
- Return jump through several program levels for special applications in compile cycles and in the ISO dialect mode.

To achieve this, the `RETB` command should be programmed with additional parameters.

Search direction

When specifying parameter `<target block>`, a return jump is first made to the block after the calling block. A search is then made for the target in the direction of the **beginning** of the program into which a return jump was made. A search is made toward the end of the program if the search was not successful.

Syntax

```
RETB("<target block>")
RETB("<target block>",<block after target block>)
RETB("<target block>",<block after target block> <number of return
jump levels>)
RETB("<target block>", ,<number of return jump levels>)
```

```
RETB("<target block>",<block after target block>,<number of return
jump levels>,
<return jump to the beginning of the program>)
RETB( , ,<number of return jump levels>,<return jump to the beginning
of the program>)
```

Meaning

RETB:	End of subprogram		
<target block>:	Declares as jump target the block where program execution should be resumed. If parameter < number of return jump levels> is not programmed, then the jump target is in the program from which the current subprogram was called. Possible data include:		
	<block number>	Number of the target block. The search for the block number is realized in the program to which a return jump was made initially in the direction toward the beginning of the program.	
	<jump marker>	Jump marker, which must be available in the program into which a return jump is made. The search for the jump marker is realized in the program to which a return jump was made initially in the direction toward the beginning of the program.	
	<character string>	Character string that must be available in the program into which a return jump is made (e.g. program or variable name). The search for the character string is realized in the program to which a return jump was made initially in the direction toward the beginning of the program. The following rules apply when programming the character string: • **Blank at the end** (contrary to the jump marker, that is designated using ":" at the end). • **Before the character string** only one block number and/or a jump marker may be set, **no program commands**.	
<block after target block>:	The parameter specifies as to whether program processing should be continued in the block specified under parameter <target block> or in the following block.		
	Type:	INT	
	Value:	0	The return jump is made to the block specified in parameter <target block>.
		> 0	The return jump is made to the next block specified in parameter <target block>.

<number of return jump levels>:	The parameter specifies the number of program levels that should be jumped through (return jumps) to search there for the target block and continue processing the program.		
	Type:	INT	
	Value:	1	The program is resumed at the "current program level - 1" (like RET without parameter).
		2	The program is resumed at the "current program level - 2", i.e. one level is skipped.
		3	The program is resumed at the "current program level - 3", i.e. two levels are skipped.
		...	
	Range of values:	1 ... 15	
<return jump to the beginning of the program>:	The parameter specifies whether, for a return jump into the main program, the program should be continued at the start of the program in the active **ISO dialect mode**.		
	Type:	BOOL	
	Value:	1	If the return jump is made into the main program and an **ISO dialect mode** is active there, then the program branches to the beginning of the program.

Note

For a subprogram return jump with a character string to specify the target block search, initially, a search is always made for a jump marker in the calling program.

If a jump target is to be uniquely defined using a character string, it is not permissible that the character string matches the name of a jump marker, as otherwise the subprogram return jump would always be made to the jump marker and not to the character string (refer to example 2).

Supplementary conditions

When making a return jump through several program levels, the SAVE statements of the individual program levels are evaluated.

If, for a return jump over several program levels, a modal subprogram is active and if in one of the skipped programs the deselection command MCALL is programmed for the modal subprogram, then the modal subprogram remains active.

NOTICE

Programming error

For a return jump through several program levels it is the user's responsibility to ensure that processing is continued with the necessary modal settings. This can be achieved, e.g. by programming an appropriate main block.

Example

Program code	Comment
EXAMPLE.MPF	
…	
N3000 START_CYC(param1, param2, …)	
N3010 TECH_CYC1(param1, param2, …)	
N3020 TECH_CYC2(param1, param2, …)	
N3030 TECH_CYC3(param1, param2, …)	
N3040 END_CYC(param1, param2, …)	
N3040 **END_CYC**(param1, param2, …)	
N3050 …	
N4500 START_CYC(param11, param12, …)	
N4510 …	
N4590 END_CYC(param11, param12, ..)	
N5000 …	
…	
N6000 M30	

Program code	Comment
PROC END_CYC(…)	; Call in the main program, line N3040
N10000 …	
N15000 if status == 1	
N15010 RETB("START_CYC")	; Return jump to the calling program EXAMPLE.MPF
	; Search for character string "START_CYC"
	; Search direction: backward in the program start direction
	; Program processing is continued with line N3000
N15020 endif	
N15030 if status == 0	
N15040 RET	; Return jump to the calling program EXAMPLE.MPF
	; Program processing is continued with line N3050
N15050 endif	
N16000 RET("START_CYC")	; Return jump to the calling program EXAMPLE.MPF
	; Search for character string "START_CYC"
	; Search direction: forward in the program end direction
	; Program processing is continued with line N4500
N17060 RETB	; Return jump to the calling program EXAMPLE.MPF
	; Program processing is continued with line N3050
	; RETB without parameter is identical to RET

2.24.3 Subprogram call

2.24.3.1 Subprogram call without parameter transfer

A subprogram is called either with address L and subprogram number or by specifying the program name.

A main program can also be called as a subprogram. The end of program `M2` or `M30` set in the main program is evaluated as `M17` in this case (end of program with return to the calling program).

Note

Accordingly, a subprogram can also be started as a main program.

Search strategy of the control:

Are there any *_MPF?

Are there any *_SPF?

This means, if the name of the subprogram to be called is identical to the name of the main program, the main program that issued the call is called again. This is generally an undesirable effect and must be avoided by assigning unique names to subprograms and main programs.

Note

Subprograms not requiring parameter transfer can also be called from an initialization file.

Syntax

```
L<number>/<program name>
```

Note

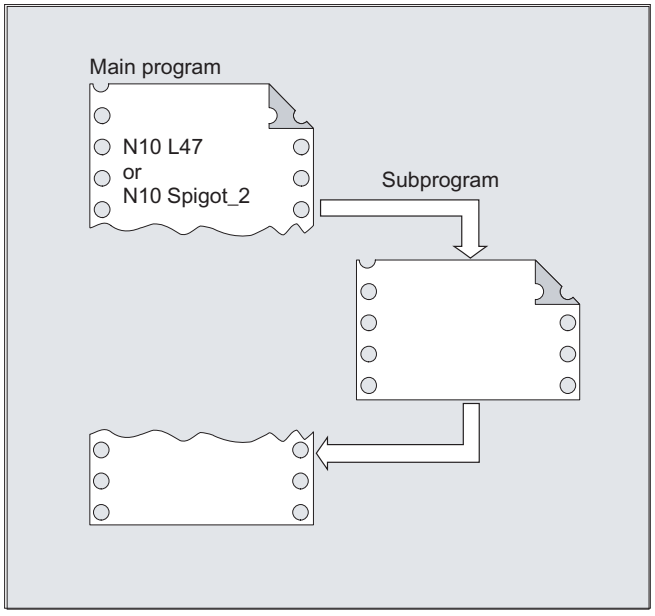
The subprogram call must always be programmed in a separate NC block.

Meaning

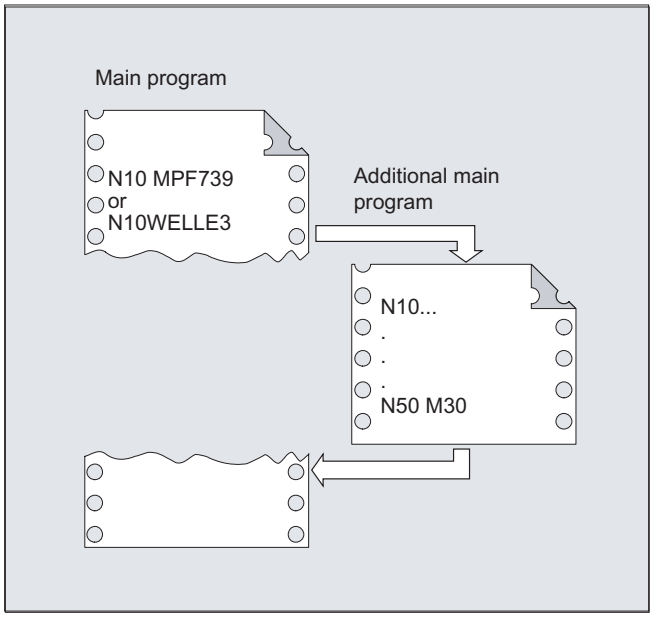
`L`:	Address for the subprogram call	
`<number>`:	Name of the subprogram	
	Type:	INT
	Value:	Maximum 7 decimal places **Notice:** Leading zeros are significant in names (⇒ L123, L0123 and L00123 are three different subprograms).
`<program name>`:	Name of the subprogram (or main program)	

Examples

Example 1: Subprogram call without parameter transfer



Example 2: Calling a main program as a subprogram



2.24.3.2 Subprogram call with parameter transfer (EXTERN)

For a subprogram call with parameter transfer, variables or values can be transferred directly (but not `VAR` parameters).

Subprograms with parameter transfer must be declared with EXTERNAL in the main program before they are called in the main program (e.g. at the beginning of the program). The name of the subprogram and the variable types are thereby specified in the sequence in which they are transferred.

NOTICE
Risk of confusion Both the variable types and the sequence of the transfer must match the definitions declared under PROC in the subprogram. The parameter names can be different in the main program and the subprogram.

Syntax

```
EXTERNAL <program name>(<type_Par1>,<type_Par2>,<type_Par3>)
...
<program name>(<value_Par1>,<value_Par2>,<value_Par3>)
```

Note

The subprogram call must always be programmed in a separate NC block.

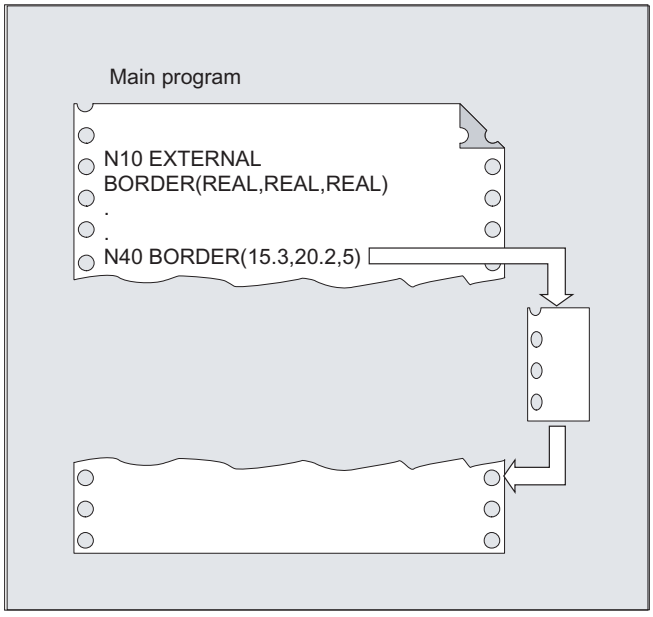
Meaning

<program name>:	Name of subprogram
EXTERNAL:	Keyword to declare a subprogram with parameter transfer. **Note:** You only have to specify EXTERNAL if the subprogram is in the workpiece or in the global subprogram directory. Cycles do not have to be declared as EXTERNAL.
<type_par1>,<type_par2>,<type_par3>:	Variable types of the parameters to be transferred in the sequence of the transfer
<value_par1>,<value_par2>,<value_par3>:	Variable values for the parameters to be transferred

Examples

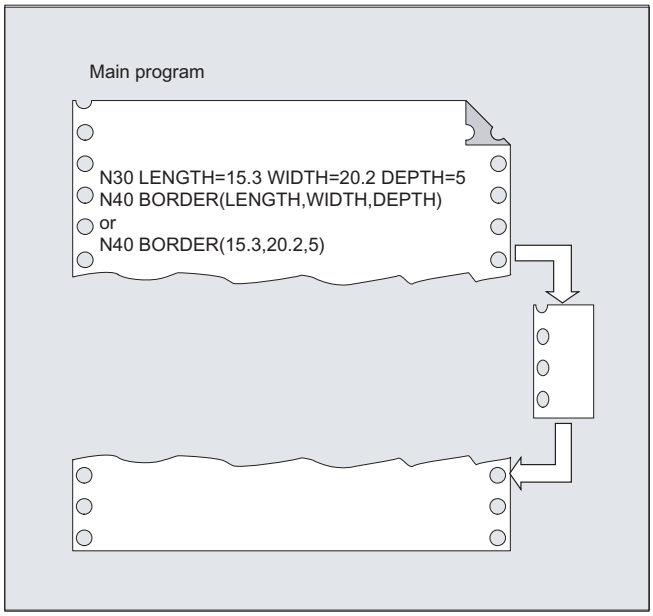
Example 1: Subprogram call preceded by declaration

Program code	Comment
N10 EXTERNAL BORDERS(REAL,REAL,REAL)	; Specify the subprogram.
...	
N40 BORDER(15.3,20.2,5)	; Call the subprogram with parameter transfer.



Example 2: Subprogram call without declaration

Program code	Comment
N10 DEF REAL LENGTH, WIDTH, DEPTH	
N20 …	
N30 LENGTH=15.3 WIDTH=20.2 DEPTH=5	
N40 BORDER(LENGTH,WIDTH,DEPTH)	; or: N40 BORDER(15.3,20.2,5)



2.24.3.3 Number of program repetitions (P)

If a subprogram is to be executed several times in succession, the desired number of program repetitions can be entered at address `P` in the block with the subprogram call.

> **⚠ CAUTION**
>
> **Subprogram call with program repetition and parameter transfer**
>
> Parameters are transferred only when the program is called, i.e., on the first run. The parameters remain unchanged for the remaining repetitions. If you want to change the parameters during program repetitions, you must make the appropriate provision in the subprogram.

Syntax

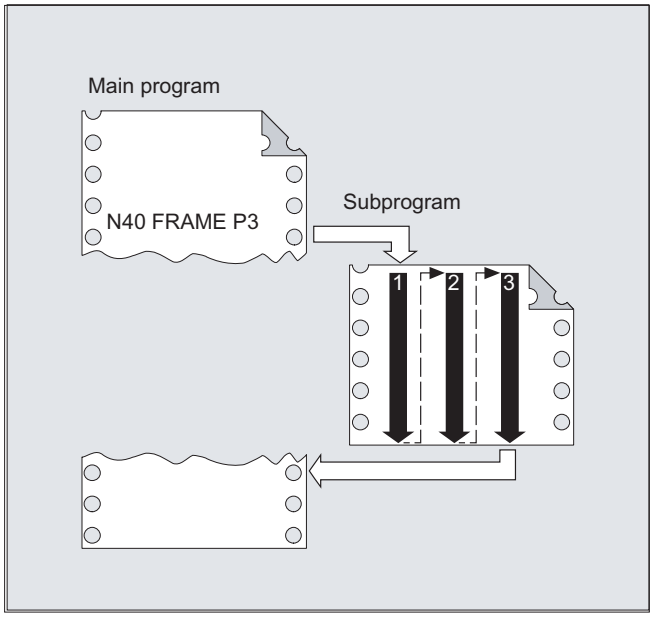
```
<program name> P<value>
```

Meaning

`<program name>`:	Subprogram call	
`P`:	Address to program program repetitions	
`<value>`:	Number of program repetitions	
	Type:	INT
	Range of values:	1 … 9999 (unsigned)

Example

```
Program code          Comment
...
N40 FRAME P3          ; The BORDER subprogram is to be executed three times one after
                      the other.
...
```



2.24.3.4 Modal subprogram call (MCALL)

The specified subprogram is not immediately called as a result of the modal subprogram call `MCALL(<program name>)`. Instead, the call is performed as of this time in the part program after each traversing block with path motion. Also across program levels.

Note

When a program is being executed only the last modal subprogram call `MCALL(<program name>)` is effective (this is always the case). The current modal subprogram call replaces the one that has been active up until then.

If parameters are transferred to the subprogram, the parameters are only transferred with call `MCALL(<program name>(Par1, Par2, ...))`.

NOTICE

Modal subprogram calls without path motion

In the following situations the modal subprogram is also called without programming path motion:

- Programming addresses `S` or `F` if `G0` or `G1` is active.
- If `G0` or `G1` were programmed alone in the block or with additional G codes.

Syntax

```
MCALL <program name>
...
MCALL
```

Meaning

MCALL <program name>:	Activate the "Modal subprogram call" function
<program name>:	Name of subprogram
MCALL:	The "Modal subprogram call" function is deactivated with MCALL without specification of a program name.

Supplementary conditions

ASUB

If the part program processing is interrupted by an ASUB (see Chapter "Interrupt routine (ASUB) (Page 118)"), then no modal subprogram calls are executed in this ASUB.

If an ASUB is started in the "Reset" channel state, then it behaves just like a normal part program with regard to the modal subprogram calls.

Tool change cycle

If the "Modal subprogram call" function is deselected during the tool change cycle, note that the tool change cycle is called implicitly, even after a block search, via the search ASUB, or manually via overstore. In this situation, the "Modal subprogram call" function must not be deselected because otherwise the search result is falsified. It is therefore recommended that the deselection of the "Modal subprogram call" function in the tool change cycle is programmed as follows:

```
Program code                Comment
...
  IF $AC_ASUP == 0          ; Call is not performed via search ASUB or overstore.
    MCALL                   ; Deactivate the "Modal subprogram call" function.
  ENDIF
...
```

Examples

Example 1

```
Program code                Comment
N10 G0 X0 Y0
N20 MCALL L70               ; Activate the modal subprogram call for L70.
N30 X10 Y10                 ; X10 Y10 is approached, and then L70 is called.
N40 X20 Y20                 ; X20 Y20 is approached, and then L70 is called.
...
N100 MCALL                  ; Deactivate the "Modal subprogram call" function.
N110 X0 Y0                  ; X0 Y0 is approached, L70 is not called.
```

Example 2

Program code
N10 G0 X0 Y0
N20 MCALL L70
N30 L80

In this example, the following NC blocks with programmed path axes are in subprogram L80. L70 is called by L80.

2.24.3.5 Indirect subprogram call (CALL)

Depending on the prevailing conditions at a particular point in the program, different subprograms can be called. The name of the subprogram is stored in a variable of the STRING type. The subprogram call is realized with `CALL` and the variable name.

Note

The indirect subprogram call is only possible for subprograms without parameter transfer. For a direct subprogram call, save the name in a STRING constant.

Syntax

```
CALL <program name>
```

Meaning

`CALL`:	Command for the indirect subprogram call.	
`<program name>`:	Name of the subprogram (variable or constant)	
	Type:	STRING

Example

Direct call with STRING constant:

Program code	Comment
…	
CALL "/_N_WKS_DIR/_N_SUBPROG_WPD/_N_PART1_SPF"	; Direct call to subprogram PART1 with CALL.
…	

Indirect call via variable:

Program code	Comment
…	
DEF STRING[100] PROGNAME	: Define variable.

Program code	Comment
PROGNAME="/_N_WKS_DIR/_N_SUBPROG_WPD/_N_PART1_SPF"	; Assign subprogram PART1 to the PROGNAME variable.
CALL PROGNAME	; Indirect call to subprogram PART1 via CALL and the PROGNAME variable.
…	

2.24.3.6 Indirect subprogram call with specification of the calling program part (CALL BLOCK ... TO ...)

CALL and the keyword combination BLOCK ... TO is used to call a subprogram indirectly and execute the program section designated by the start and end labels.

Syntax

```
CALL <program name> BLOCK <start label> TO <end label>
CALL BLOCK <start label> TO <end label>
```

Meaning

CALL:	Command for the indirect subprogram call.	
<program name>:	Name of the subprogram (variable or constant) that contains the program section to be executed (**specification optional**).	
	Type:	STRING
	Note: If a <program name> has not been programmed, the program section designated by <start label> and <end label> is searched for in the current program and executed.	
BLOCK ... TO ... :	Keyword combination for indirect program section execution	
<start label>:	Variable that refers to the start of the program section to be executed.	
	Type:	STRING
<end label>:	Variable that refers to the end of the program section to be executed.	
	Type:	STRING

Example

Main program:

Program code	Comment
...	
DEF STRING[20] STARTLABEL, ENDLABEL	; Variable definition for the start and end labels.
STARTLABEL="LABEL_1"	
ENDLABEL="LABEL_2"	
...	

Program code	Comment
CALL "CONTUR_1" BLOCK STARTLABEL TO ENDLABEL	; Indirect subprogram call and identifier associated with the calling program section.
...	

Subprogram:

Program code	Comment
PROC CONTUR_1 ...	
LABEL_1	; Start label: Start of program section execution.
N1000 G1 ...	
...	
LABEL_2	; End label: End of program section execution.
...	

2.24.3.7 Indirect call of a program programmed in ISO language (ISOCALL)

A program programmed in an ISO language can be called using the indirect program call `ISOCALL`. The ISO mode set in the machine data is then activated. The original execution mode becomes effective again at the end of the program. If no ISO mode is set in the machine data, the subprogram is called in Siemens mode.

For further information about the ISO mode, see
References:
ISO Dialects Functional Description

Syntax

```
ISOCALL <program_name>
```

Meaning

`ISOCALL`:	Keyword for an indirect subprogram call with which the ISO mode set in the machine data is activated.
`<program name>`:	Name of the program programmed in an ISO language (variable or constant, type STRING)

Example: Calling a contour with cycle programming from ISO mode

Program code	Comment
0122_SPF	; Contour description in ISO mode
N1010 G1 X10 Z20	
N1020 X30 R5	
N1030 Z50 C10	
N1040 X50	
N1050 M99	
N0010 DEF STRING[5] PROGNAME = "0122"	; Siemens part program (cycle)

Program code	Comment
...	
N2000 R11 = $AA_IW[X]	
N2010 ISOCALL PROGNAME	
N2020 R10 = R10+1	; Execute program 0122.spf in ISO mode
...	
N2400 M30	

2.24.3.8 Call subprogram with path specification and parameters (PCALL)

With PCALL, you can call subprograms with the absolute path and parameter transfer.

Syntax

```
PCALL <path/program name>(<parameter 1>,…,<parameter n>)
```

Meaning

PCALL:	Keyword for subprogram call with absolute path name
<path/program name>:	Absolute path data including subprogram names. Rules regarding path data, see "Addressing program memory files (Page 205)". If no absolute path name is specified, PCALL behaves like a standard subprogram call with a program identifier. The program name is specified without prefix and without file identifier. If the program name is to be programmed with prefix and file identifier, then it must be explicitly declared with prefix and file identifier using the EXTERN command.
<parameter 1>, ...:	Actual parameters in accordance with the PROC operation of the subprogram.

Example

Program code
PCALL/_N_WKS_DIR/_N_SHAFT_WPD/SHAFT(parameter1,parameter2,…)

2.24.3.9 Extend search path for subprogram calls (CALLPATH)

The search path for subprogram calls can be extended using the CALLPATH command. This means that also subprograms can be called from a non-selected workpiece directory without having to specify the complete, absolute path name of the subprogram.

Another application option is possible in the EES mode "EES without GDIR", if another directory is used on an external program memory to save global subroutines. In this case, using CALLPATH the search path can be extended by this subprogram directory.

The search path extension is made before the entry for user cycles (_N_CUS_DIR).

The search path extension is deselected again as a result of the following events:

- `CALLPATH` with blanks
- `CALLPATH` without parameter
- End of part program
- Reset

Syntax

```
CALLPATH("<path name>")
```

Meaning

`CALLPATH`:	Keyword for the programmable search path extension. Is programmed in a separate part program line.
`<path name>`:	Constant or variable, STRING type. Contains the absolute path name of the directory by which the search path should be extended. Rules regarding path data, see "Addressing program memory files (Page 205)".

Example

The search path should be extended by a certain workpiece directory:

Program code

```
...
CALLPATH ("/_N_WKS_DIR/_N_MYWPD_WPD")
...
```

This means that the following search path is set (position 5. is new):

1. Actual directory/*name*
2. Actual directory/*name_SPF*
3. Actual directory/*name_MPF*
4. //NC:/_N_SPF_DIR / *name_SPF*
5. **/_N_WKS_DIR/_N_MYWPD_WPD/name_SPF**
6. /N_CUS_DIR/*name_SPF*
7. /_N_CMA_DIR/*name_SPF*
8. /_N_CST_DIR/*name_SPF*

Supplementary conditions

- CALLPATH checks whether the programmed path name actually exists. In the case of an error, part program execution is interrupted with correction block alarm 14009.
- CALLPATH can also be programmed in INI files. It is only effective for the time it takes to process the INI file (WPD-INI file or initialization program for NC active data, e.g. frames in the 1st channel _N_CH1_UFR_INI). The search path is again reset.

2.24.3.10 Execute external subprogram (840D sl) (EXTCALL)

A part program can be loaded from an external memory and executed with the EXTCALL command.

The following are available as external memory:

- Local drive
- Network drive
- USB drive

Note

Only the USB interfaces on the operator panel front or the TCU can be used as interface for the processing of an external program on a USB drive.

NOTICE

Tool/workpiece damage when using a USB flash drive

It is recommended that a USB flash drive is not used to execute an external subprogram. A communication interruption to the USB flash drive when executing the subprogram if the flash drive has contact problems, drops out, is interrupted because it has been accidentally knocked or has been inadvertently withdrawn stops the machining immediately. The tool and/or workpiece could be damaged.

Default setting of the external program path

The path for the external program directory can be preset with the setting data:

SD42700 $SC_EXT_PROG_PATH

Together with the program path and identifier specified with the EXTCALL call, this forms the entire path for the subprogram to be called.

Note

Parameter

When calling an external program, none of these parameters can be transferred to it.

Syntax

```
EXTCALL("<path/><program name>")
```

Meaning

EXTCALL:	Command for calling an external subprogram.	
"<path/><program name>":	Constant/variable of type STRING	
	<path>:	Absolute or relative path data (**optional**)
	<program name>:	The program name is specified without prefix "_N_". The file extension ("MPF", "SPF") can be attached to program names using the "_" or "." character (**optional**). Example: "SHAFT" "SHAFT_SPF" "SHAFT.SPF"

Path specification: Short designations

The following short designations can be used to specify the path:

- Local drive: "**LOCAL_DRIVE:**"
- CF card: "**CF_CARD:**"
- USB drive (operator panel front): "**USB:**"

The short designations "CF_CARD:" and "LOCAL_DRIVE:" can be alternatively used.

Example

Execute from local drive

The "MAIN.MPF" main program is stored in NC memory and is selected for execution.

Subprogram "SP_1"

The external subprogram "SP_1.SPF" or "SP_1.MPF" is on the local drive in the directory "/user/sinumerik/data/prog/WKS.DIR/WST1.WPD".

The path for the external program directory is set with:

SD42700 $SC_EXT_PROG_PATH = LOCAL_DRIVE:WKS.DIR/WST1.WPD

Note

Specification of the path for the call of the external subprogram:

- Without the default setting: "LOCAL_DRIVE:WKS.DIR/WST1.WPD/SP_1"
- With the default setting: "SP_1"

Subprogram "SP_2"

The external subprogram "SP_2.SPF " or "SP_2.MPF " is in the WKS.DIR /WST1.WPD directory of the USB drive. The default path setting to the external program directory is used

for the path of subprogram "SP_1" and is also not rewritten in the main program. Therefore, the complete path needs to be specified when subprogram "SP_2" is called.

Main program "MAIN"

```
Program code
N010 PROC MAIN
N020 ...
N030 EXTCALL("SP_1")
N030 EXTCALL("USB:WKS.DIR/WST1.WPD/SP_2")
N050 ...
N060 M30
```

Additional information

EXTCALL call with absolute path name

If the subprogram exists under the specified path, it is executed with the EXTCALL call. If the subprogram does not exist under the specified path, the program execution is aborted with the EXTCALL call.

EXTCALL call with relative path name / without path name

In the event of an EXTCALL call with a relative path name or without a path name, the available program memories are searched as follows:

1. If a path name is preset in SD42700 $SC_EXT_PROG_PATH, the data specified in the EXTCALL call (program name or with relative path name) is searched for first, starting from this path. The absolute path is obtained from linking the following characters:
   - Default path specification in SD42700 $SC_EXT_PROG_PATH
   - Separator "/"
   - Path specification and subprogram name in the EXTCALL command
2. If the subprogram was not found under 1., the directories of the user memory are searched.

The search ends when the subprogram is found for the first time. If the subprogram is not found, the program execution is aborted with the EXTCALL call.

Adjustable reload memory (FIFO buffer)

A reload memory is required for the execution of an external subprogram. The size of the reload memory is preset with 30KB and can only be changed by the machine manufacturer (using MD18360 MM_EXT_PROG_BUFFER_SIZE).

Note

Subprograms with jump commands

For external subprograms that contain jump commands (GOTOF, GOTOB, CASE, FOR, LOOP, WHILE, REPEAT, IF, ELSE, ENDIF etc.) the jump destinations must lie within the post loading memory.

Note

ShopMill/ShopTurn programs

The contour descriptions added at the file end mean the ShopMill and ShopTurn programs must be stored completely in the read-only memory.

A separate reload memory is required for external subprograms executed in parallel.

Reset / end of program / POWER ON

Reset and POWER ON cause external subprogram calls to be interrupted and the associated load memory to be deleted.

A program selected for "Execution from external source" remains selected for "Execution from external source" after a reset / end of program. The behavior does not differ from internally selected programs, assuming that the external program memory is still available.

References

Further information on "Execution from external source" can be found in:

Function Manual, Basic Functions, Mode Group, Channel, Program Operation, Reset Behavior (K1)

2.24.3.11 Execute external subprogram (828D) (EXTCALL)

A part program can be loaded from an external memory and executed with the `EXTCALL` command.

The following are available as external memory:

- User CF card
- Network drive
- USB drive

Note

As the interface for the execution of an external program located on a USB drive, only the USB interface of the operator panel front (PPU) may be used.

NOTICE
Tool/workpiece damage when using a USB flash drive
It is recommended that a USB flash drive is not used to execute an external subprogram. A communication interruption to the USB flash drive when executing the subprogram if the flash drive has contact problems, drops out, is interrupted because it has been accidentally knocked or has been inadvertently withdrawn stops the machining immediately. The tool and/or workpiece could be damaged.

Default setting of the external program path

The path for the external program directory can be preset with the setting data:

SD42700 $SC_EXT_PROG_PATH

Together with the program path and identifier specified with the EXTCALL call, this forms the entire path for the subprogram to be called.

Note

Parameter

When calling an external program, none of these parameters can be transferred to it.

Syntax

```
EXTCALL("<path/><program name>")
```

Meaning

EXTCALL:	Command for calling an external subprogram.	
"<path/><program name>":	Constant/variable of type STRING	
	<path>:	Absolute or relative path data (**optional**)
	<program name>:	The program name is specified without prefix "_N_". The file extension ("MPF", "SPF") can be attached to program names using the "_" or "." character (**optional**). Example: "SHAFT" "SHAFT_SPF" "SHAFT.SPF"

Path specification: Short designations

The following short designations can be used to specify the path:

- User CF card: "**CF_CARD:**"
- USB drive (operator panel front): "**USB:**"

Example

The "MAIN.MPF" main program is stored in NC memory and is selected for execution.

Subprogram "SP_1"

The external subprogram "SP_1.SPF " or "SP_1.MPF " is on the user CF card in the "/WKS.DIR /WST1.WPD" directory.

The path for the external program directory is set with:

SD42700 $SC_EXT_PROG_PATH = CF_CARD:WKS.DIR/WST1.WPD

Note

Specification of the path for the call of the external subprogram:

- Without the default setting: "CF_CARD:WKS.DIR/WST1.WPD/SP_1"
- With the default setting: "SP_1"

Subprogram "SP_2"

The external subprogram "SP_2.SPF " or "SP_2.MPF " is in the WKS.DIR /WST1.WPD directory of the USB drive. The default path setting to the external program directory is used for the path of subprogram "SP_1" and is also not rewritten in the main program. Therefore, the complete path needs to be specified when subprogram "SP_2" is called.

Main program "MAIN"

```
Program code
N010 PROC MAIN
N020 ...
N030 EXTCALL("SP_1")
N030 EXTCALL("USB:WKS.DIR/WST1.WPD/SP_2")
N050 ...
N060 M30
```

Additional information

EXTCALL call with absolute path name

If the subprogram exists under the specified path, it is executed with the EXTCALL call. If the subprogram does not exist under the specified path, the program execution is aborted with the EXTCALL call.

EXTCALL call with relative path name / without path name

In the event of an EXTCALL call with a relative path name or without a path name, the available program memories are searched as follows:

1. If a path name is preset in SD42700 $SC_EXT_PROG_PATH, the data specified in the EXTCALL call (program name or with relative path name) is searched for first, starting from this path. The absolute path is obtained from linking the following characters:
   - Default path specification in SD42700 $SC_EXT_PROG_PATH
   - Separator "/"
   - Path specification and subprogram name in the EXTCALL command
2. If the subprogram was not found under 1., the directories of the user memory are searched.

The search ends when the subprogram is found for the first time. If the subprogram is not found, the program execution is aborted with the EXTCALL call.

Adjustable reload memory (FIFO buffer)

A reload memory is required for the execution of an external subprogram. The size of the reload memory is preset (see MD18360 MM_EXT_PROG_BUFFER_SIZE).

Note

Subprograms with jump commands

For external subprograms that contain jump commands (`GOTOF`, `GOTOB`, `CASE`, `FOR`, `LOOP`, `WHILE`, `REPEAT`, `IF`, `ELSE`, `ENDIF` etc.) the jump destinations must lie within the post loading memory.

Note

ShopMill/ShopTurn programs

The contour descriptions added at the file end mean the ShopMill and ShopTurn programs must be stored completely in the read-only memory.

A separate reload memory is required for external subprograms executed in parallel.

Reset / end of program / POWER ON

Reset and POWER ON cause external subprogram calls to be interrupted and the associated load memory to be deleted.

A program selected for "Execution from external source" remains selected for "Execution from external source" after a reset / end of program. The behavior does not differ from internally selected programs, assuming that the external program memory is still available.

References

Further information on "Execution from external source" can be found in:

Function Manual, Basic Functions, Mode Group, Channel, Program Operation, Reset Behavior (K1)

2.25 Macro technique (DEFINE ... AS)

NOTICE
Complicated programming Macros can significantly alter the control's programming language. Macro technology may only be used with great care.

A macro is a sequence of individual statements which have together been assigned a name of their own. G, M and H functions or L subprogram names can also be used as macros. When a macro is called during a program run, the statements programmed under the program name are executed one after the other.

Application

Statement sequences that are repeated are only programmed once as macro in a dedicated macro block (macro file) or once at the beginning of the program. The macro can then be called in any main program or subprogram and executed.

Activation

In order to use the macros of a macro file in the NC program, the macro file must be downloaded into the NC.

Syntax

Macro definition:
```
DEFINE <Macro name> AS <statement 1> <statement 2> ...
```

Call in the NC program:
```
<macro name>
```

Meaning

`DEFINE ... AS` :	Keyword combination to define a macro
`<macro name>`:	Macro name Only identifiers are permissible as macro names. The macro is called from the NC program with the macro name.
`<statement>`:	Programming statement that should be included in the macro.

Rules when defining a macro

- Any identifier, G, M, H functions and L program names can be defined in a macro.
- Macros can also be defined in the NC program.
- G function macros can only be defined in the macro module globally for the entire control (modal).
- H and L functions can be programmed with two digits.
- M and G functions can be programmed with three digits.

Note

Keywords and reserved names may not be redefined with macros.

Supplementary conditions

Nesting of macros is not possible.

Examples

Example 1: Macro definition at the beginning of the program

Program code	Comment
DEFINE LINE AS G1 G94 F300	; Macro definition
...	
N70 LINE X10 Y20	; Macro call
...	

Example 2: Macro definitions in a macro file

Program code	Comment
DEFINE M6 AS L6	; A subprogram is called at tool change to handle the necessary data transfer. The actual M function is output in the subprogram (e.g. M106).
DEFINE G81 AS DRILL(81)	; Emulation of the DIN G function.
DEFINE G33 AS M333 G333	; During thread cutting, synchronization is requested with the PLC. The original G function G33 was renamed to G333 by machine data so that the programming is identical for the user.

Example 3: External macro file

The macro file must be downloaded into the NC after reading the external macro file into the control. Only then can macros be used in the NC program.

Program code	Comment
%_N_UMAC_DEF	
;$PATH=/_N_DEF_DIR	; Customer-specific macros
DEFINE PI AS 3.14	
DEFINE TC1 AS M3 S1000	
DEFINE M13 AS M3 M7	; Spindle clockwise, coolant on
DEFINE M14 AS M4 M7	; Spindle counter-clockwise, coolant on
DEFINE M15 AS M5 M9	; Spindle stop, coolant off
DEFINE M6 AS L6	; Call tool change program
DEFINE G80 AS MCALL	; Deselect drilling cycle
M30	

File and Program Management 3

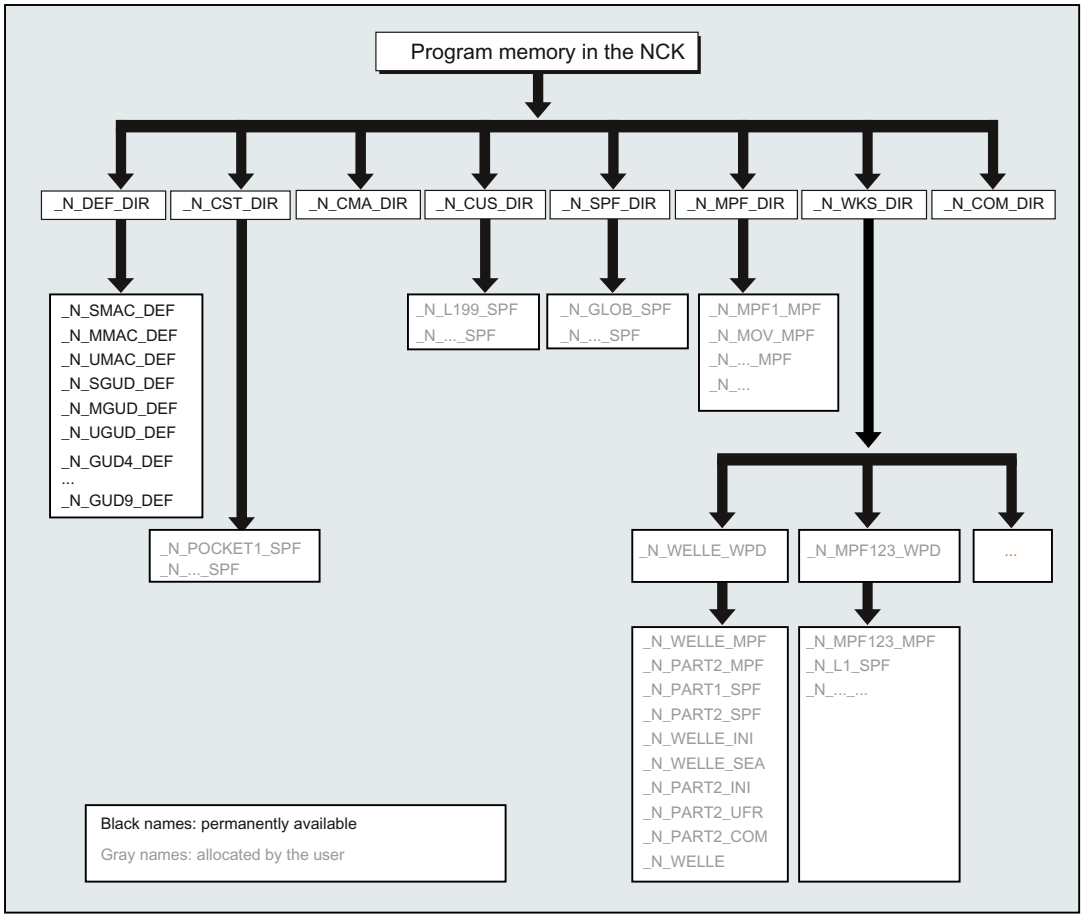
3.1 Program memory

3.1.1 Program memory in the NCK

Files and programs (e.g. main programs and subprograms, macro definitions) are saved in the non-volatile program memory (→ passive file system).

References:
Function Manual, Extended Functions; Memory Configuration (S7)

A number of file types are also stored here temporarily; these can be transferred to the work memory as required (e.g. for initialization purposes when machining a specific workpiece).



Standard directories

The following standard directories are available:

Directory	Content
_N_DEF_DIR	Data modules and macro modules
_N_CST_DIR	Standard cycles
_N_CMA_DIR	Manufacturer cycles
_N_CUS_DIR	User cycles
_N_WKS_DIR	Workpieces
_N_SPF_DIR	Global subprograms
_N_MPF_DIR	Main programs
_N_COM_DIR	Comments

File types

The following file types can be stored in the main memory:

File type	Description
<name>_MPF	Main program
<name>_SPF	Subprogram
<name>_TEA	Machine data
<name>_SEA	Setting data
<name>_TOA	Tool offsets
<name>_UFR	Zero offsets/frames
<name>_INI	Initialization files
<name>_GUD	Global user data
<name>_RPA	R-parameters
<name>_COM	Comment
<name>_DEF	Definitions for global user data and macros

Workpiece main directory (_N_WKS_DIR)

The workpiece main directory exists in the standard setup of the program memory under the name _N_WKS_DIR. The workpiece main directory contains all the workpiece directories for the workpieces that you have programmed.

Workpiece directories (..._WPD)

A workpiece directory contains all files required for machining a workpiece. These can be main programs, subprograms, any initialization programs and comment files.

The first time a part program is started, initialization programs are executed once, depending on the selected program (in accordance with machine data MD11280 $MN_WPD_INI_MODE).

Example:

The workpiece directory _N_SHAFT_WPD, created for SHAFT workpiece contains the following files:

File	Description
_N_SHAFT_MPF	Main program
_N_PART2_MPF	Main program
_N_PART1_SPF	Subprogram
_N_PART2_SPF	Subprogram
_N_SHAFT_INI	General initialization program for the data of the workpiece
_N_SHAFT_SEA	Setting data initialization program
_N_PART2_INI	General initialization program for the data for the Part 2 program
_N_PART2_UFR	Initialization program for the frame data for the Part 2 program
_N_SHAFT_COM	Comment file

Data can also be stored in the workpiece directory which is not directly required by the NCK for the machining. In addition to ASCII files, this can be binary files, such as images in JPG format or descriptions in PDF format. In order that these can be interpreted as binary files by the NCK, the file extensions must be known in the NCK (setting during commissioning via MD17000 $MN_ EXTENSIONS_OF_BIN_FILES; the following file extensions are preset in the basic setting: JPG, GIF, PNG, BMP, PDF, ICO, HTM).

Select workpiece for machining

A workpiece directory can be selected for execution in a channel. If a main program with the **same name** or only a single main program (_MPF) is stored in this directory, this is automatically selected for execution.

References:
Operating Manual

3.1.2 External program memory

In addition to the passive file system in the NCK, external program memories can also be available at the machine (e.g. on the local drive or on a network drive).

Using the functions "Execute from external" or "EES (Execution from External Storage)" part programs can be **directly** executed from external program memories.

Reference:
Function Manual Basic Functions; K1: Mode Group, Channel, Program Operation, Reset Response

Global part program memory (GDIR)

When declaring the drives, one of the drives can be designated the global part program memory (GDIR).

References:
Operating Manual: section: "Manage programs" > "Setting up drives"

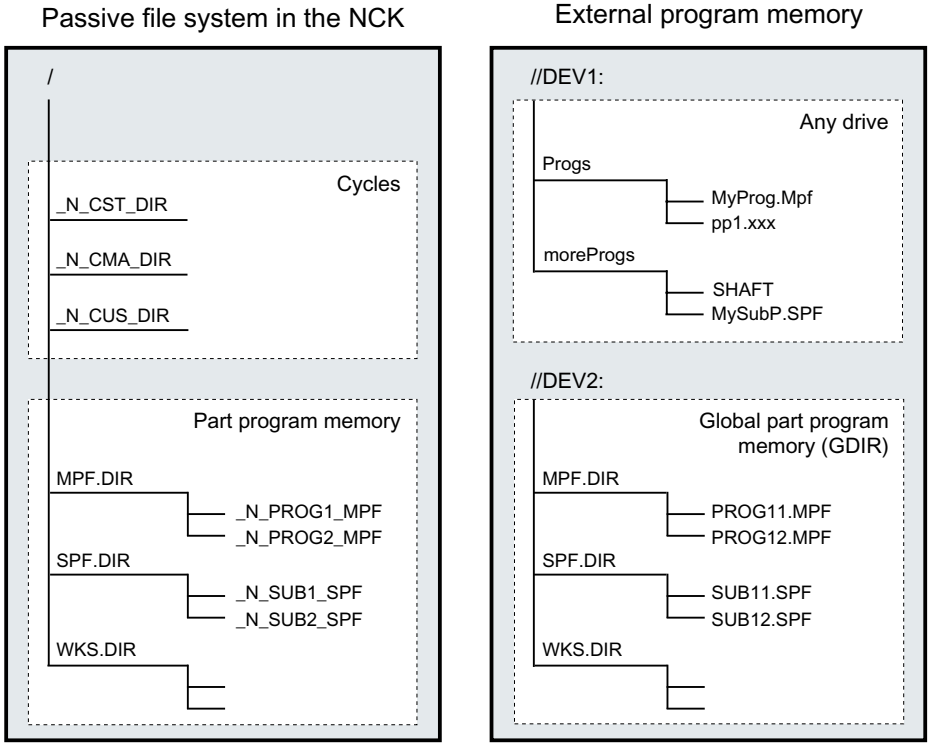
The system automatically creates the MPF.DIR, SPF.DIR and WKS.DIR directories on the drive. These three directories form the GDIR.

The GDIR only plays a role for the EES function. Depending on the drive configuration, the GDIR replaces or extends the NC part program memory. The creation of a GDIR is, however, not essential for EES operation.

The directories and files of the GDIR can be addressed in the part program in the same way as in the passive file system. This permits a compatible transfer of an NC program with path details from the passive file system to the GDIR. The directory SPF.DIR of the GDIR is contained in the search path for subprograms.

Program organization

The program organization on external program memories is shown in the following diagram:



Case-insensitive file systems

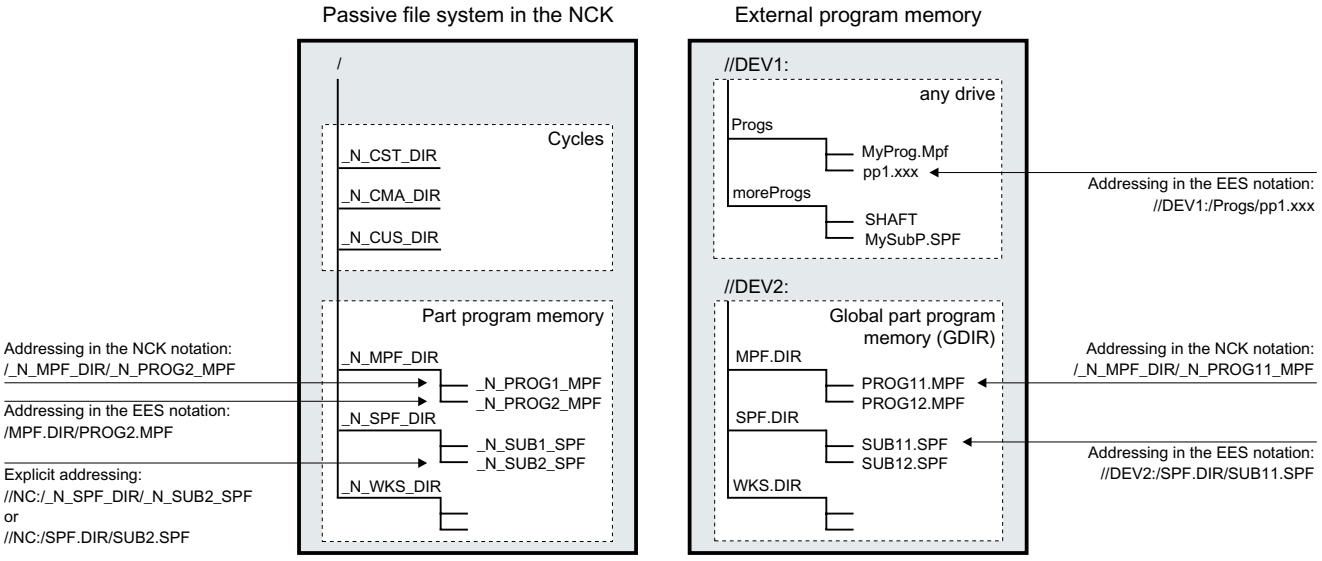
Note

To avoid problems with case-sensitivity for the file addressing (see "Addressing program memory files (Page 205)"), **case-insensitive** file systems should be used as external program memory.

3.1.3 Addressing program memory files

A file in the program memory, which is addressed with a file handling command (e.g. WRITE, DELETE, READ, ISFILE, FILEDATE, FILETIME, FILESIZE, FILESTAT, FILEINFO), is referenced with an absolute path plus file names or only with the file names. In the second case, the path of the selected program is used as file path.

Addressing in the NCK/EES notation



Addressing files of the passive file system

Files of the passive file system are generally addressed in the **NCK notation** (directory and file names begin with the domain identifier "_N_", "_" is the separator for the file identifier) without specifying the drive name. An addressing in **EES notation** (without domain identification "_N_", separator for the directory/file extension is ".") is, however, also permitted.

Example:

- NCK notation: "/_N_SPF_DIR/_N_SUB1_SPF"
- EES notation: "/SPF.DIR/SUB1.SPF"

Note

The addressing schemes for files of the passive file system in EES notation are converted internally into NCK notation in accordance with the following rules:

- Directory and file names are extended with the domain identification "_N_".
- If the fourth-last character in the directory or file name is a period ("."), it will be converted into an underscore ("_").

Using the predefined drive name "//NC:" the passive file system can also be explicitly addressed.

Example:

- NCK notation: "//NC:/_N_SPF_DIR/_N_SUB1_SPF"
- EES notation: "//NC:/SPF.DIR/SUB1.SPF"

Addressing files of an external program memory

Files of an external program memory not recorded as GDIR must be addressed in EES notation. The drive name (e.g. "//DEV1:") must be specified at the beginning of the addressing path. All symbolic device names configured in /user/sinumerik/hmi/cfg/logdrive.ini are permissible.

Example:

- EES notation: "//DEV1:/MyProgDir/pp1.xxx"
- NCK notation: Not permissible

Addressing files of the global part program memory (GDIR)

When addressing files of the GDIR, in addition to specifying the path in the EES notation, it is also permissible to specify the path in the NCK notation.

Example:

- EES notation: "//DEV2:/MPF.DIR/PROG11.MPF"
- NCK notation: "/_N_MPF_DIR/_N_PROG11_MPF"

Note

The addressing schemes for files of the GDIR in NCK notation are converted internally into EES notation in accordance with the following rules:

- The domain identification "_N_" in directory and file names is removed.
- If the fourth-last character in the directory or file name is an underscore ("_"), it will be converted into a period (".").

Rules for the path specification

A complete path specification consists of drive name, directory path and file name.

Drive name

The following rules govern the specification of the drive name:

- All symbolic device names configured in /user/sinumerik/hmi/cfg/logdrive.ini are permissible.
- The character "//" is at the beginning, followed by at least one letter or one digit.
- The following characters can be any combination of letters, digits, "_" and spaces.
- The name is ended with a letter or a digit, followed by a ":".
- Other special characters are not permitted.

Note

The drive name "//NC:" is predefined for the passive file system.

Examples:

- External program memory:
  - //Drive1:
  - //Drive_1:
  - //Drive 1:
  - //A B:
  - //1 B C 2:

Directory path

The following rules govern the specification of the directory path:

- A "/" is located at the start and end of the directory path and as separator for the individual path sections.

  Note

  A double slash ("//") within the directory path is **not** permitted!

- Directory names:
  - Directory names must begin with a letter or a digit. Only for addressing in NCK notation do directory names begin with the domain identification "_N_".
  - The following characters can be any combination of letters, digits and "_".

    Note

    Spaces in directory names are also permitted for external program memories. This is not true, however, when the external program memory is created as global part program memory (GDIR).

  - Other special characters are not permitted.
- Directory extensions:
  - Directory extensions must consist of three letters/digits.
  - They are separated with "_" (NCK notation) or "." (EES notation) for the directory name.

  Note

  The passive file system has only the directory extensions _DIR and _WPD.

Examples:

- Passive file system or GDIR:
  - NCK notation: _N_WKS_DIR/_N_MYNCPROGS_WPD/...
  - EES notation: WKS.DIR/MYPROGS.WPD/...
- External program memory:
  - /abc
  - /ab_c.def
  - /ab c1.def
  - /a b c .d11
  - /abc.def/ghi.klm

File name

The following rules apply to the file names:

- Only for addressing in NCK notation do file names begin with the domain identification "_N_".
- The next two characters must be two letters or an underscore followed by a letter.
- The following characters can be any combination of letters, digits and "_".
- File extension:
  - The file extension must consist of three letters/digits.

    Note

    Permitted file extensions in the passive file system, see "Program memory in the NCK (Page 201)".

  - It is separated with "_" (NCK notation) or "." (EES notation) for the file name.

Examples:

- Passive file system or GDIR:
  - NCK notation: _N_SUB1_SPF
  - EES notation: SUB1.SPF
- External program memory:
  - Part 1
  - _Part1
  - Teil_1.spf
  - Teil1.mpf

DIN subprogram name

The following rules apply to DIN subprogram names:

- The first character must be the letter "L".
- The following characters are digits (at least one).
- File extension:
  - The file extension must consist of three letters.
  - It is separated with "_" (NCK notation) or "." (EES notation) for the file name.

Examples:

- L123
- L1_SPF (NCK notation) or L1.SPF (EES notation)

Maximum path length

Maximum 128 bytes are available for specifying the drive name and the directory path; the maximum length of the file name is 31 bytes. The maximum length of the complete path is 159 bytes.

3.1.4 Search path for subprogram call

For subprogram calls without path data, the absolute path is determined by processing a fixed search path.

A search is then made in the program memory in the following sequence:

		Directory	Description
1		current directory / *name*	The current directory is the directory in which the program is selected. This can be: • A workpiece directory or the standard directory _N_MPF_DIR in the NCK part program memory or global part program memory or • Any directory of an external program memory
2		current directory / *name_SPF*	
3		current directory / *name_MPF*	
4	a	//NC:/_N_SPF_DIR / *name_SPF*	Subprogram directory in the NCK part program memory
	b	//DEV2:/_N_SPF_DIR / name_SPF [1)]	Subprogram directory in the global part program memory **Note:** This search step is not executed if a global part program memory has not been created, or the program is selected in the NCK part program memory.

	Directory	Description
5	Search path extension programmed with CALLPATH (see "Extend search path for subprogram calls (CALLPATH) (Page 189)"). **Note:** This search step is not executed if CALLPATH has not been programmed.	
6	/_N_CUS_DIR / *name_SPF*	User cycle directory
7	/_N_CMA_DIR / *name_SPF*	Manufacturer cycle directory
8	/_N_CST_DIR / *name_SPF*	Standard cycle directory

1) //DEV2:" for example represents the drive on which the global part program memory has been created.

The following rules apply for the search:

- The search path is run through for each individual subprogram call, this means that it is irrelevant where the higher-level program is located.
- Depending on the directory, different file types are taken into account.
- A search is made in a directory, and not in lower-level, i.e. nested directories.

3.1.5 Interrogating the path and file name

The following system variables, which can be read in the part program, are available to interrogate the path and file name of an NC program:

System variable	Type	Meaning
$P_STACK	INT	Supplies the program level in which the current NC program is executed.
$P_PATH[<n>]	STRING	Supplies the path of the NC program, which is processed at the program level selected using field index <n>. Examples: $P_PATH[0] supplies the path for the main program, e.g. "/_N_WKS_DIR/_N_WELLE_WPD/". $P_PATH[$P_STACK - 1] supplies the path of the calling program. If the path refers to an NC program, which is saved in the passive file system of the NCK or in the global part program memory (GDIR), then the path is supplied in the NCK notation. If the path refers to an NC program, which is executed by an external program memory other than the global part program memory then $P_PATH supplies the path in the EES notation.
$P_PROG[<n>]	STRING	Supplies the name of the NC program, which is processed at the program level selected using field index <n>. If the NC program is saved in the passive file system of the NCK or in the global part program memory, then the program name is supplied in the NCK notation. If the NC program is executed by an external drive other than the global part program memory, then $P_PROG supplies the name in the EES notation.

System variable	Type	Meaning	
$P_PROGPATH	STRING	Supplies the path of the NC program that is presently being processed. Calling $P_PROGPATH is identical to $P_PATH[$P_STACK].	
$P_IS_EES_PATH[<n>]	BOOL	Interrogates whether the path supplied by $P_PATH[<n>] or the program name supplied by $P_PROG[<n>] corresponds to the NCK notation or the EES notation.	
		= FALSE	$P_PATH[<n>] and $P_PROG[<n>] supply the NCK notation, which means that each identifier has the prefix "_N_". The separator for the file identifier is "_". Examples: • Path in the NCK notation: "/_N_WKS_DIR/_N_MYWPD_WPD/" • Program name in the NCK notation:"_N_MYPROG_MPF" A path in the NCK notation can refer to the passive file system in the NCK as well as also the global part program memory.
		= TRUE	$P_PATH[<n>] and $P_PROG[<n>] supply the EES notation, which means that the identifiers do not have the prefix "_N_". The separator for the file identifier is ".". Examples: • Path in the EES notation: "//DEV1:/WKS.DIR/MYWPD.WPD/" • Program name in the EES notation: "MYPROG.MPF"

<n>: Index <n> defines the program level, from which the path information should be read (value range: 0 ... 17)

Note

In the EES mode, outside the global part program memory (GDIR), system variables $P_PROG, $P_PATH and $P_PROGPATH path names in the EES notation. For the EES mode, user programs that evaluate and process these path names must be extended so that they can also process pathnames in the EES notation.

3.2 Working memory (CHANDATA, COMPLETE, INITIAL)

Function

The working memory contains the current system and user data with which the control is operated (active file system), e.g.:

- Active machine data
- Tool offset data

- Zero offsets
- ...

Initialization programs

These are programs with which the working memory data is initialized. The following file types can be used for this:

File type	Description
name_TEA	Machine data
name_SEA	Setting data
name_TOA	Tool offsets
name_UFR	Zero offsets/frames
name_INI	Initialization files
name_GUD	Global user data
name_RPA	R-parameters

Data areas

The data can be organized in different areas in which they are to apply. For example, a control can have several channels or, as is commonly the case, several axes at its disposal.

There are:

Identifier	Data areas
NCK	NCK-specific data
CH<n>	Channel-specific data (<n> specifies the channel name)
AX<n>	Axis-specific data (<n> specifies the number of the machine axis)
TO	Tool data
COMPLETE	All data

Create initialization program at an external PC

The data area identifier and the data type identifier can be used to determine the areas which are to be treated as a unit when the data is saved:

_N_AX5_TEA_INI	Machine data for axis 5
_N_CH2_UFR_INI	Frames of channel 2
_N_COMPLETE_TEA_INI	All machine data

When the control is started up initially, a set of data is automatically loaded to ensure proper operation of the control.

Procedure for multi-channel controls (CHANDATA)

CHANDATA(<channel number>) for several channels is only permissible in the file _N_INITIAL_INI. This is the commissioning file with which all data of the control is initialized.

```
%_N_INITIAL_INI
CHANDATA(1)
                                        ; Machine axis assignment, channel 1:
$MC_AXCONF_MACHAX_USED[0]=1
$MC_AXCONF_MACHAX_USED[1]=2
$MC_AXCONF_MACHAX_USED[2]=3
CHANDATA(2)
                                        ; Machine axis assignment, channel 2:
$MC_AXCONF_MACHAX_USED[0]=4
$MC_AXCONF_MACHAX_USED[1]=5
CHANDATA(1)
                                        ; Axial machine data:
                                        ; Exact stop window coarse:
$MA_STOP_LIMIT_COARSE[AX1]=0.2          ; Axis 1
$MA_STOP_LIMIT_COARSE[AX2]=0.2          ; Axis 2
                                        ; Exact stop window fine:
$MA_STOP_LIMIT_FINE[AX1]=0.01           ; Axis 1
$MA_STOP_LIMIT_FINE[AX1]=0.01           ; Axis 2
```

NOTICE

CHANDATA statement

In the part program, the CHANDATA statement may only be set for that channel in which the NC program is executed. This means the statement can be used to protect NC programs so that they are not executed in the wrong channel.

Program processing is aborted if an error occurs.

Note

INI files in job lists do not contain any CHANDATA statements.

Save initialization program (COMPLETE, INITIAL)

The files of the working memory can be saved on an external PC and then read in again from there.

- The files are saved with COMPLETE.
- INITIAL is used to create an INI file (_N_INITIAL_INI) over all areas.

Read-in initialization program

NOTICE
Data loss If the file is read-in with the name "INITIAL_INI", then all data that is not supplied in the file is initialized using standard data. Only machine data is an exception. This means that **setting data, tool data, ZO, GUD values, ...** are supplied with standard data (normally "ZERO").

For example, the file COMPLETE_TEA_INI is suitable for reading-in individual machine data. The control only expects machine data in this file. This is the reason that the other data areas remain unaffected in this case.

Loading initialization programs

The INI programs can also be selected and called as part programs if they only use data of one channel. This means that it is also possible to initialize program-controlled data.

Protection zones

4

4.1 Defining the protection zones (CPROTDEF, NPROTDEF)

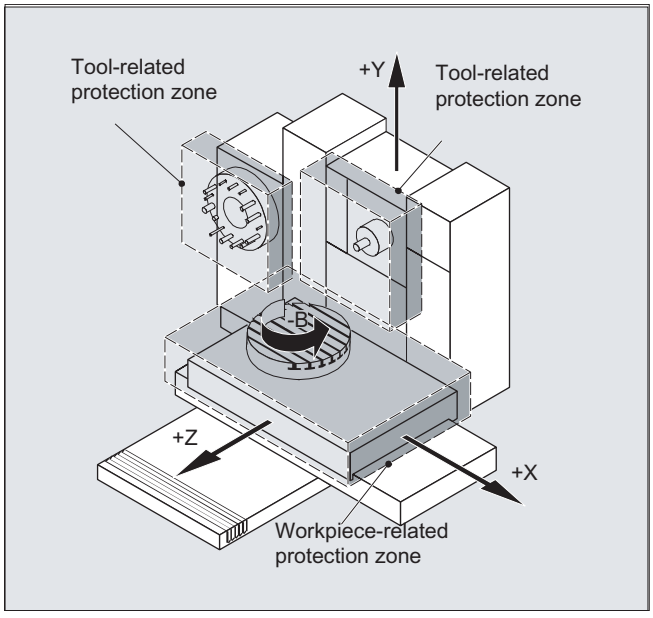
You can use protection zones to protect various elements on the machine, their components and the workpiece against incorrect movements.

Tool-related protection zones:

For parts that belong to the tool (e.g. tool, toolholder).

Workpiece-oriented protection zones:

For parts that belong to the workpiece (e.g. parts of the workpiece, clamping table, clamping shoe, spindle chuck, tailstock).



Syntax

```
DEF INT NOT_USED
G17/G18/G19
CPROTDEF/NPROTDEF(<n>,<t>,<applim>,<applus>,<appminus>)
G0/G1/... X/Y/Z...
...
EXECUTE(NOT_USED)
```

Meaning

DEF INT NOT_USED:	Define local variable of the INTEGER data type	
G17/G18/G19:	The required plane is selected before CPROTDEF or NPROTDEF with G17/G18/G19 and must not be altered before EXECUTE. The applicate must not be programmed between CPROTDEF or NPROTDEF and EXECUTE.	
CPROTDEF:	Define channel-specific protection zones	
NPROTDEF:	Define machine-specific protection zones	
G0/G1/... X/Y/Z... ...:	The contour of the protection zones is specified with up to 11 traversing movements in the selected plane. The first traversing movement is the movement to the contour. The valid protection zone is the zone left of the contour. **Note:** The travel motions programmed between CPROTDEF or NPROTDEF and EXECUTE are not executed, but merely define the protection zone.	
EXECUTE:	End definition	
<n>:	Number of defined protection zone	
<t>:	Type of protection zone	
	TRUE:	**Tool**-related protection zone
	FALSE:	**Workpiece**-related protection zone
<applim>:	Type of limitation in the 3rd dimension	
	0:	No limitation
	1:	Limit in plus direction
	2:	Limit in minus direction
	3:	Limit in positive and negative direction
<applus>:	Value of the limit in the positive direction in the 3rd dimension	
<appminus>:	Value of the limit in the negative direction in the 3rd dimension	
NOT_USED:	Error variable has no effect for protection zones with EXECUTE	

Supplementary conditions

During definition of the protection zones:

- No cutter or tool nose radius compensation must be active.
- No transformation must be active.
- No frame must be active.

Neither must reference point approach (G74), fixed point approach (G75), block preprocessing stop nor program end be programmed.

Further information

Definition of protection zones

Definition of the protection zones includes the following:

- `CPROTDEF` for channel-specific protection zones
- `NPROTDEF` for machine-specific protection zones
- Contour description for protection zone
- Termination of the definition with `EXECUTE`

You can specify a relative offset for the reference point of the protection zone when the protection zone is activated in the NC part program.

Reference point for contour description

The workpiece-oriented protection zones are defined in the basic coordinate system.

The tool-related protection zones are defined with reference to the tool carrier reference point F.

Permissible contour elements

The following are permissible to define the contour of the protection zone:

- `G0`, `G1` for straight contour elements
- `G2` for clockwise circle segments (only for workpiece-oriented protection zones)
- `G3` for circular segments in the counter-clockwise direction

Note

If a full circle describes the protection zone, it must be divided into two half circles. The sequence `G2`, `G3` or `G3`, `G2` is not permitted. A short `G1` block must be inserted, if necessary.

The last point in the contour description must coincide with the first.

External protection zones

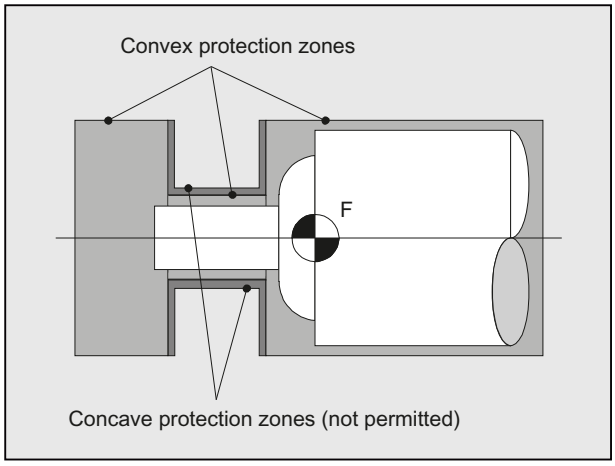
External protection zones (only possible for workpiece-related protection zones) must be defined in the clockwise direction.

Protection zones symmetrical around the center of rotation

For protection zones symmetrical around the axis or rotation (e.g. spindle chuck), you must describe the complete contour and not only up to the center of rotation!

Tool-related protection zones

Tool-related protection zones must always be convex. If a concave protected zone is desired, this should be subdivided into several convex protection zones.



4.2 Activating/deactivating protection zones (CPROT, NPROT)

Activate or preactivate previously defined protection zones for collision monitoring or deactivate active protection zones.

The maximum number of protection zones, which can be active simultaneously on the same channel, is defined in machine data.

If no tool-related protection zone is active, the tool path is checked against the workpiece-related protection zones.

Note

If no workpiece-related protection zone is active, protection zone monitoring does not take place.

Syntax

```
CPROT(<n>,<state>,<xMov>,<yMov>,<zMov>)
NPROT(<n>,<state>,<xMov>,<yMov>,<zMov>)
```

Meaning

CPROT:	Call channel-specific protection zone	
NPROT:	Call machine-specific protection zone	
<n>:	Number of the protection zone	
<state>:	Status parameter	
	0:	Deactivate protection zone
	1:	Preactivate protection zone
	2:	Activate protection zone
	3:	Preactivate protection zone with conditional stop
<xMov>,<yMov>,<zMov>:	Move defined protection zone on the geometry axes	

Supplementary conditions

Protection zone monitoring for active tool radius compensation

For active tool radius compensation, a functioning protection zone monitoring is only possible if the plane of the tool radius compensation is identical to the plane of the protection zone definitions.

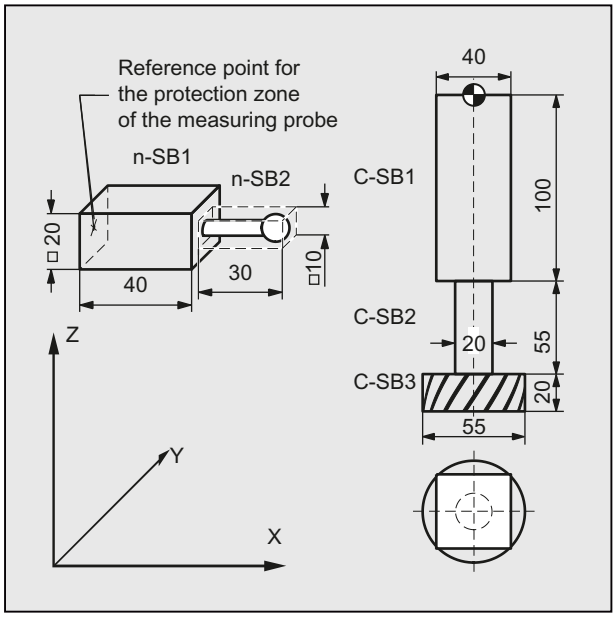
Example

Possible collision of a milling cutter with the measuring probe is to be monitored on a milling machine. The position of the measuring probe is to be defined by an offset when the function is activated. The following protection zones are defined for this:

- A machine-specific and a workpiece-related protection zone for both the measuring probe holder (n-SB1) and the measuring probe itself (n-SB2).
- A channel-specific and a tool-related protection zone for the milling cutter holder (c-SB1), the cutter shank (c-SB2) and the milling cutter itself (c-SB3).

The orientation of all protection zones is in the Z direction.

The position of the reference point of the measuring probe on activation of the function must be X = -120, Y = 60 and Z = 80.



Program code	Comment
DEF INT PROTECTB	; Definition of a Help variable
Definition of protection zone G17	; Set orientation

Program code	Comment
NPROTDEF(1,FALSE,3,10,-10)G01 X0 Y-10	; Protection zone n-SB1
X40	
Y10	
X0	
Y-10	
EXECUTE(PROTECTB)	
NPROTDEF(2,FALSE,3,5,-5)	; Protection zone n-SB2
G01 X40 Y-5	
X70	
Y5	
X40	
Y-5	
EXECUTE(PROTECTB)	
CPROTDEF(1,TRUE,3,0,-100)	; Protection zone c-SB1
G01 X-20 Y-20	
X20	
Y20	
X-20	
Y-20	
EXECUTE(PROTECTB)	
CPROTDEF(2,TRUE,3,-100,-150)	; Protection zone c-SB2
G01 X0 Y-10	
G03 X0 Y10 J10	
X0 Y-10 J-10	
EXECUTE(PROTECTB)	
CPROTDEF(3,TRUE,3,-150,-170)	; Protection zone c-SB3
G01 X0 Y-27.5	
G03 X0 Y27,5 J27.5	
X0 Y27.5 J-27.5	
EXECUTE(PROTECTB)	
Activation of protection zones:	
NPROT(1,2,-120,60,80)	; Activate protection zone n-SB1 with offset
NPROT(2.2,-120,60,80)	; Activate protection zone n-SB2 with offset
CPROT(1,2,0,0,0)	; Activate protection zone c-SB1 with offset
CPROT(2,2,0,0,0)	; Activate protection zone c-SB2 with offset
CPROT(3,2,0,0,0)	; Activate protection zone c-SB3 with offset

Further information

Activation status (<state>)

- **<state>=2**
  A protection zone is generally activated in the part program with status = 2.
  The status is always channel-specific even for machine-related protection zones.
- **<state>=1**
  If a PLC user program provides a protection zone to be effectively set by a PLC user program, the required preactivation is implemented with status = 1.
- **<state>=3**
  In the case of preactivation with conditional stop, the system never stops in front of a preactivated protection zone which has been violated. The stop only occurs if the protection zone has been activated. This facilitates uninterrupted processing if the protection zones have only been activated in special cases. N.B. The system may move into the protection zone as a result of the braking ramp if the protection zone is only activated immediately prior to motion.
  Preactivation with conditional stop is realized with status = 3.
- **<state>=0**
  The protection zones are deactivated and therefore disabled with Status = 0. No shift is required

Movement of protection zones for (pre)activating

The offset can take place in 1, 2, or 3 dimensions. The offset refers to:

- The machine zero in workpiece-specific protection zones.
- The tool carrier reference point F in tool-specific protection zones.

Status after booting

Protection zones can be activated straight after booting and subsequent reference point approach. The system variable $SN_PA_ACTIV_IMMED[<n>] or $SC_PA_ACTIV_IMMED[<n>] must be set to TRUE for this purpose. They are always activated with Status = 2 and have no offset.

Multiple activation of protection zones

A protection zone can be active simultaneously in several channels (e.g. tailstock where there are two opposite sides). The protection zones are only monitored if all geometry axes have been referenced.

The following applies:

- The protection zone cannot be activated simultaneously with different offsets in a single channel.
- Machine-related protection zones must have the same orientation on both channels.

4.3 Checking for protection zone violation, working area limitation and software limit switches (CALCPOSI)

As of the start position, the `CALCPOSI()` function checks whether active limits have been violated along the traversing distance in the workpiece coordinate system (WCS) with regard to the **geometry axes**.

If the distance cannot be fully traversed because of limits, a positive, decimal-coded status value and the maximum possible traversing distance are returned.

Syntax

```
<Status> = CALCPOSI(VAR <Start>, VAR <Dist>, VAR <Limit>, VAR
<MaxDist>, <System>, <TestLim>)
```

Meaning

`CALCPOSI`:	Test for limit violations with regard to the geometry axes.	
	Preprocessing stop:	No
	Alone in the block:	Yes
`<Status>`: (Part 1)	Function return value. Negative values indicate error states.	
	Data type:	INT
	Range of values:	-8 ≤ x ≤ 100000
	Values	**Meaning**
	0	The distance can be traversed completely.
	Error statuses	
	-1	At least one component is negative in <Limit>.
	-2	Error in a transformation calculation. Example: The traversing distance passes through a singularity so that the axis positions cannot be defined.
	-3	The specified traversing distance <Dist> and the maximum possible traversing distance <MaxDist> are linearly dependent. **Note** Can only occur in conjunction with <TestLim>, bit 4 == 1.
	-4	The projection of the traversing direction contained in <Dist> on to the limitation surface is the zero vector, or the traversing direction is perpendicular to the violated limitation surface. **Note** Can only occur in conjunction with <TestLim>, bit 5 == 1.
	-5	In <TestLim>, bit 4 == 1 AND bit 5 == 1
	-6	At least one machine axis that has to be considered for checking the traversing limits has not been homed.
	-7	Collision avoidance function: Invalid definition of the kinematic chain or the protection areas.
	-8	Collision avoidance function: This command cannot be executed because of insufficient memory.

<Status>: (Part 2)	**Units digit** **Note** If several limits are violated simultaneously, the limit with the greatest restriction on the specified traversing distance is signaled.	
	1	Software limit switches are limiting the traversing distance
	2	Working area limits are limiting the traversing distance
	3	Protection areas are limiting the traversing distance
	4	Collision avoidance function: Protection areas are limiting the traversing distance
	Tens digit	
	1x	The initial value violates the limit
	2x	The specified straight line violates the limit. This value is also returned if the end point does not violate any limit itself, but the path from the starting point to the end point would cause a limit value to be violated (e.g. by passing through a protection area, curved software limit switches in the WCS for non-linear transformations, e.g. transmit).
<Status>: (Part 3)	**Hundreds digit**	
	1xx	AND units digit == 1 or 2: The positive limit value has been violated.
		AND units digit == 3 [1]: An NC-specific protection area has been violated.
	2xx	AND units digit == 1 or 2: The negative limit value has been violated.
		AND units digit == 3 [1]: A channel-specific protection area has been violated.
<Status>: (Part 4)	**Thousands digit**	
	1xxx	AND units digit == 1 or 2: Factor with which the axis number is multiplied that violates the limit. Numbering of the axes begins with 1. Reference: • Software limit switch: Machine axes • Working area limitation: Geometry axes AND units digit == 3 [1]: Factor with which the number of the violated protection area is multiplied.
<Status>: (Part 5)	**Hundred thousands digit**	
	0xxxxx	Hundred thousands digit == 0: <dist> remains unchanged
	1xxxxx	A direction vector is returned in <Dist> which defines the further motion direction on the limitation surface. Can only occur with the following supplementary conditions: • Software limit switch or working area limit violated (not in the starting point) • A transformation is **not** active • <TestID>, bit 4 or bit 5 == 1
	1) If several protection areas are violated, the protection area with the greatest restriction on the specified traversing distance is signaled.	

<Start>:	Reference to a vector with the start positions: • <Start> [0]: Abscissa • <Start> [1]: Ordinate • <Start> [2]: Applicate	
	Parameter type:	Input
	Data type:	VAR REAL [3]
	Range of values:	-max. REAL value ≤ x[n] ≤ +max. REAL value
<Dist>:	Reference to a vector with the incremental traversing distance: • <Dist> [0]: Abscissa • <Dist> [1]: Ordinate • <Dist> [2]: Applicate	
	For set hundred thousands digit in <Status>:	
	<Dist> contains a unit vector **v** as output value, which defines the further traversing direction in the WCS.	
	Case 1: Formation of vector **v** for <TestID>, bit 4 == 1 The input vectors <Dist> and <MaxDist> span the motion plane. This plane is cut by the violated limitation surface. The intersecting line of the two planes defines the direction of vector **v**. The orientation (sign) is selected so that the angle between the input vector <MaxDist> and **v** is not greater than 90 degrees.	
	Case 2: Formation of vector **v** for <TestID>, bit 5 == 1 Vector **v** is the unit vector in the projection direction of the traversing vector contained in <Dist> on the limitation surface. If the projection of the traversing vector on the limitation surface is the zero vector, an error is returned.	
	Parameter type:	Input/output
	Data type:	VAR REAL [3]
	Range of values:	-max. REAL value ≤ x[n] ≤ +max. REAL value

<Limit>:	Reference to an array of length 5. • <Limit> [0 - 2]: Minimum clearance of the geometry axes, abscissa, ordinate, applicate The first three elements include the minimum clearances of the geometry axis, which must be maintained with respect to the monitored limits. Regarding the working area limitation they are always used – and regarding the software limit switch they are used when no transformation is active, or a transformation is active in which a clear assignment of the geometry axes to the linear machine axes is possible, e.g. 5-axis transformations. • <Limit> [3]: Contains the minimum clearance for linear machine axes which, for example, cannot be assigned a geometry axis because of a non-linear transformation. This value is also used as limit value for the monitoring of the conventional protection areas and the collision avoidance protection areas. • <Limit> [4]: Contains the minimum clearance for rotary machine axes which, for example, cannot be assigned a geometry axis because of a non-linear transformation. **Note** This value is only active for the monitoring of the software limit switches for special transformations.	
	Parameter type:	Input
	Data type:	VAR REAL [5]
	Range of values:	-max. REAL value ≤ x[n] ≤ +max. REAL value
<MaxDist>:	Reference to a vector with the incremental traversing distance in which the specified minimum clearance of an axis limit is not violated by any of the relevant machine axes: • <Dist> [0]: Abscissa • <Dist> [1]: Ordinate • <Dist> [2]: Applicate If the traversing distance is not restricted, the contents of this return parameter are the same as the contents of <Dist>.	
	For <TestID>, Bit 4 == 1: <Dist> and <MaxDist> <MaxDist> and <Dist> must contain vectors as input values that span a motion plane. The two vectors must be mutually linearly dependent. The absolute value of <MaxDist> is arbitrary. For the calculation of the motion direction, see the description for <Dist>.	
	Parameter type:	Output
	Data type:	VAR REAL [3]
	Range of values:	-max. REAL value ≤ x[n] ≤ +max. REAL value

<System>:	Measuring system (inch/metric) for position and distance specifications (**optional**)		
	Data type:		BOOL
	Default value:		FALSE
	Value		**Meaning**
	FALSE		Measuring system corresponding to the currently active G function from G group 13 (G70, G71, G700, G710). **Note** If G70 is active and the basic system is metric (or G71 is active and the basic system is inch), the system variables $AA_IW and $AA_MW are provided in the basic system and, if used, must be converted for CALCPOSI().
	TRUE		Measuring system corresponding to the set basic system: MD52806 $MN_ISO_SCALING_SYSTEM
<TestLim>:	Bit-coded selection of the limits to be monitored (**optional**)		
	Data type:		INT
	Default value:		Bit 0, 1, 2, 3 == 1 (15)
	Bit	**Decimal**	**Meaning**
	0	1	Software limit switch
	1	2	Working area limitation
	2	4	Activated conventional protection areas
	3	8	Preactivated conventional protection areas
	4	16	With violated software limit switches or working area limits in <Dist>, return the traversing direction as in **Case 1**.
	5	32	With violated software limit switches or working area limits in <Dist>, return the traversing direction as in **Case 2**.
	6	64	Activated collision avoidance protection zones
	7	128	Preactivated collision avoidance protection zones
	8	256	Pairs of activated and preactivated collision avoidance protection zones

References

Function Manual, Basic Functions, (A3) Axis Monitoring, Protection Zones, Chapter "Protection zones"

Special motion commands 5

5.1 Approaching coded positions (CAC, CIC, CDC, CACP, CACN)

You can traverse linear and rotary axes via position numbers to fixed axis positions saved in machine data tables using the following commands. This type of programming is called "approach coded positions".

Syntax

```
CAC(<n>)
CIC(<n>)
CACP(<n>)
CACN(<n>)
```

Meaning

`CAC(<n>)`:	Approach coded position from position number n
`CIC(<n>)`:	Starting from the actual position number, approach the coded position n position locations before (+n) or back (–n)
`CDC(<n>)`:	Approach the position from position number n along the shortest path (only for rotary axes)
`CACP(<n>)`:	Approach coded position from position number n in the positive direction (only for rotary axes)
`CACN(<n>)`:	Approach coded position from position number n in the negative direction (only for rotary axes)
`<n>`:	Position number within the machine data table Range of values: 0, 1, … (max. number of table locations - 1)

Example: Approach coded positions of a positioning axis

Programming code	**Comment**
`N10 FA[B]=300`	`; Feedrate for positioning axis B`
`N20 POS[B]=CAC(10)`	`; Approach coded position from position number 10`
`N30 POS[B]=CIC(-4)`	`; Approach coded position from "current position number" - 4`

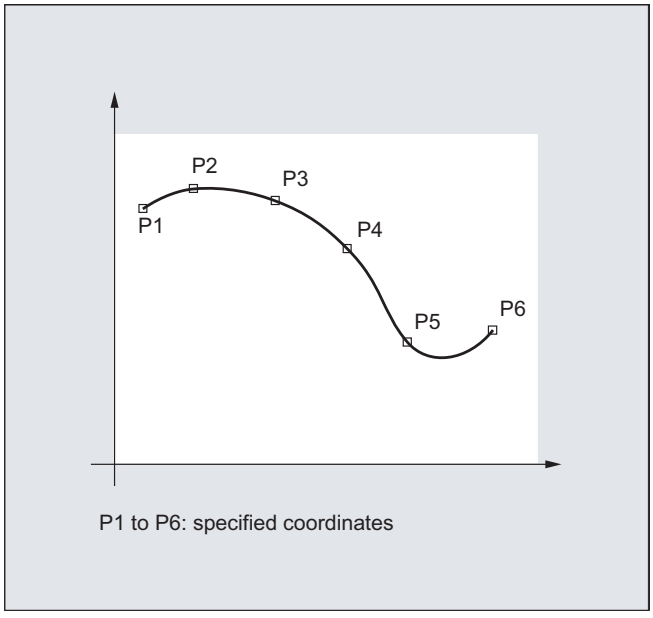
References

- Function Manual Expanded Functions; Indexing Axes (T1)
- Function Manual, Synchronized Actions

5.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)

Randomly curved workpiece contours cannot be precisely defined in an analytic form. This is the reason why these type of contours are approximated using a limited number of points along curves, e.g. when digitizing surfaces. The points along the curve must be connected to define a contour in order to generate the digitized surface of a workpiece. Spline interpolation permits this.

A spline defines a curve which is formed from polynomials of 2nd or 3rd degree. The characteristics of the points along the curve of a spline can be defined **depending on the spline type being used**.



For SINUMERIK solution line, the following spline types are available:

- A spline
- B spline
- C spline

Syntax

General:

```
ASPLINE X... Y... Z... A... B... C...
BSPLINE X... Y... Z... A... B... C...
CSPLINE X... Y... Z... A... B... C...
```

For a B spline, the following can be additionally programmed:

```
PW=<n>
SD=2
PL=<value>
```

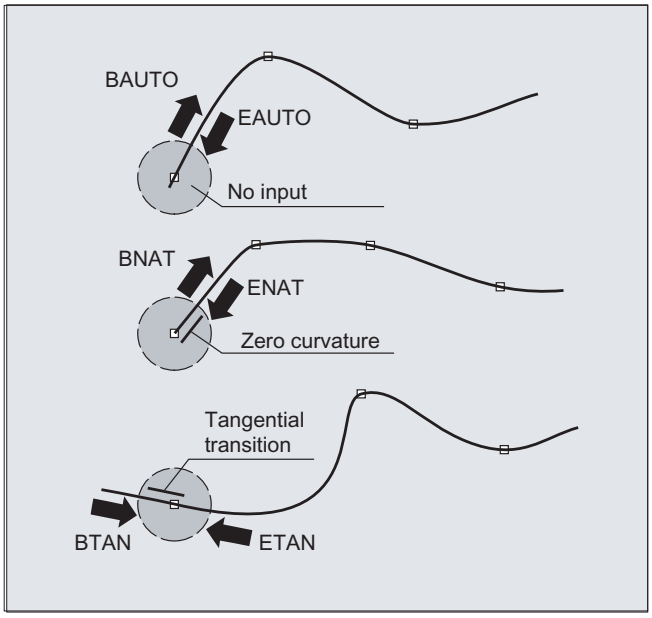
For A and C splines, the following can be additionally programmed:

```
BAUTO / BNAT / BTAN
```

EAUTO / ENAT / ETAN

Meaning

Spline interpolation type:			
ASPLINE:	Command to activate A spline interpolation		
BSPLINE:	Command to activate B spline interpolation		
CSPLINE:	Command to activate C spline interpolation		
	The ASPLINE, BSPLINE and CSPLINE commands are modally effective and belong to the group of motion commands.		
Points along a curve and check points:			
X... Y... Z... A... B... C...	Positions in Cartesian coordinates		
Point weight (only B spline):			
PW:	Using the PW command, a so-called "point weight" can be programmed for every point along the curve.		
<n>:	"Point weight"		
	Range of values:	0 ≤ n ≤ 3	
	Increment:	0.0001	
	Effect:	n > 1	The checkpoint attracts the curve more significantly.
		n < 1	The checkpoint attracts the curve less significantly.
Spline degree (only B spline):			
SD:	A third degree polygon is used as standard, However, a second degree polygon can also be used by programming SD=2.		
Distance between nodes (only B spline):			
PL :	The distances between nodes are suitably calculated internally. The control can also machine pre-defined node clearances that are specified in the so-called parameter-interval-length using the PL command.		
<value>:	Parameter interval length		
	Range of values:	As for path dimension	
Transitional behavior at the start of the spline curve (only A or C spline):			
BAUTO:	No specifications for the transitional behavior. The start is determined by the position of the first point.		
BNAT:	Zero curvature		
BTAN:	Tangential transition to the previous block (delete position)		
Transitional behavior at the end of the spline curve (only A or C spline):			
EAUTO:	No specifications for the transitional behavior. The end is determined by the position of the last point.		
ENAT:	Zero curvature		

ETAN:	Tangential transition to the previous block (delete position)
	

Note

The programmable transitional behavior has no influence on the B spline. The B spline is always tangential to the check polygon at its start and end points.

Supplementary conditions

- Tool radius compensation may be used.
- Collision monitoring is carried out in the projection in the plane.

Examples

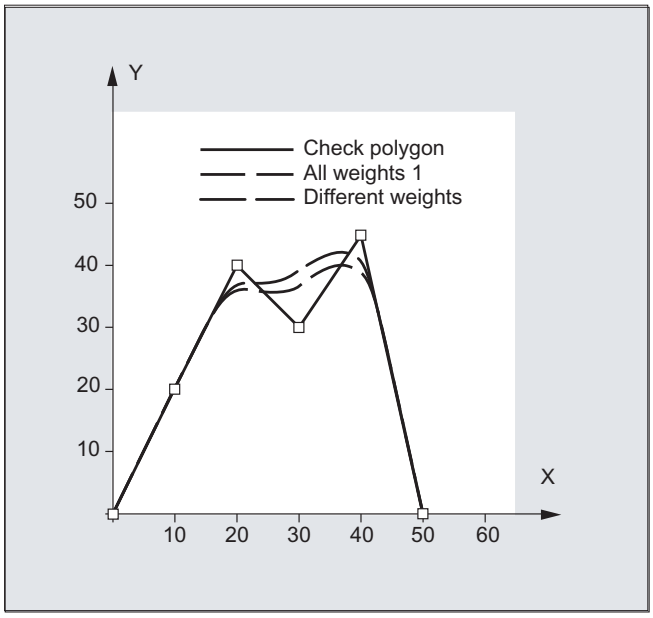
Example 1: B spline

```
Program code 1 (all weights 1)
N10 G1 X0 Y0 F300 G64
N20 BSPLINE
N30 X10 Y20
N40 X20 Y40
N50 X30 Y30
N60 X40 Y45
N70 X50 Y0
```

Program code 2 (different weights)

```
N10 G1 X0 Y0 F300 G64
N20 BSPLINE
N30 X10 Y20 PW=2
N40 X20 Y40
N50 X30 Y30 PW=0.5
N60 X40 Y45
N70 X50 Y0
```

Program code 3 (check polygon)	Comment
N10 G1 X0 Y0 F300 G64	
N20	; n.a.
N30 X10 Y20	
N40 X20 Y40	
N50 X30 Y30	
N60 X40 Y45	
N70 X50 Y0	



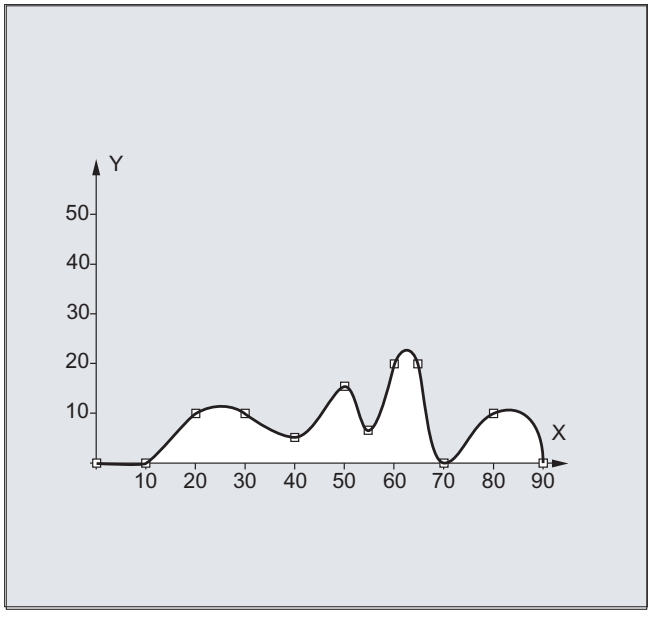
Example 2: C spline, zero curvature at the start and at the end

Program code

```
N10 G1 X0 Y0 F300
N15 X10
N20 BNAT ENAT
N30 CSPLINE X20 Y10
```

Program code

```
N40 X30
N50 X40 Y5
N60 X50 Y15
N70 X55 Y7
N80 X60 Y20
N90 X65 Y20
N100 X70 Y0
N110 X80 Y10
N120 X90 Y0
N130 M30
```



Example 3: Spline interpolation (A spline) and coordinate transformation (ROT)

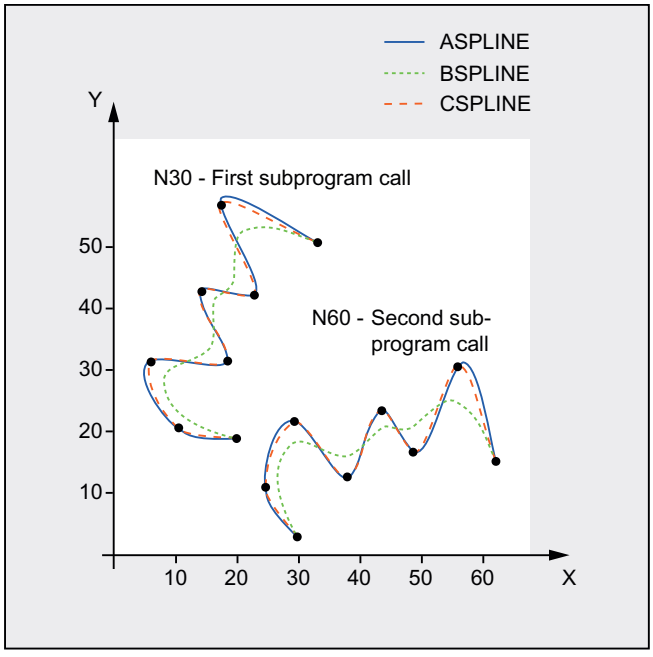
Main program:

Program code	Comment
N10 G00 X20 Y18 F300 G64	; Approach starting point.
N20 ASPLINE	; Activate interpolation type A spline.
N30 CONTOUR	; First subprogram call.
N40 ROT Z-45	; Coordinate transformation: Rotation of the WCS through -45° around the Z axis.
N50 G00 X20 Y18	; Approach contour starting point.
N60 CONTOUR	; Second subprogram call.
N70 M30	; End of program

Subprogram "contour" (includes the coordinates of the points along the curve):

```
Program code
N10 X20 Y18
N20 X10 Y21
N30 X6 Y31
N40 X18 Y31
N50 X13 Y43
N60 X22 Y42
N70 X16 Y58
N80 X33 Y51
N90 M1
```

In addition to the spline curve, resulting from the example program (ASPLINE), the following diagram also contains the spline curves that would have been obtained when activating either B or C spline interpolation (BSPLINE, CSPLINE):



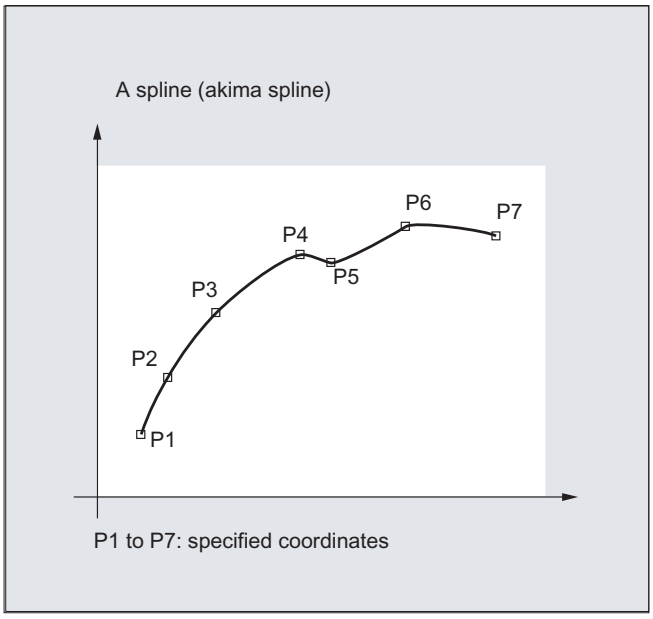
Further information

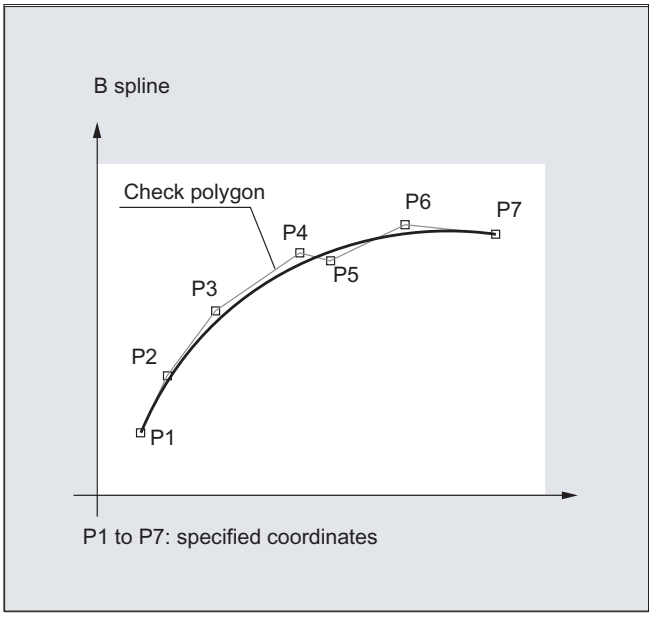
Advantages of spline interpolation

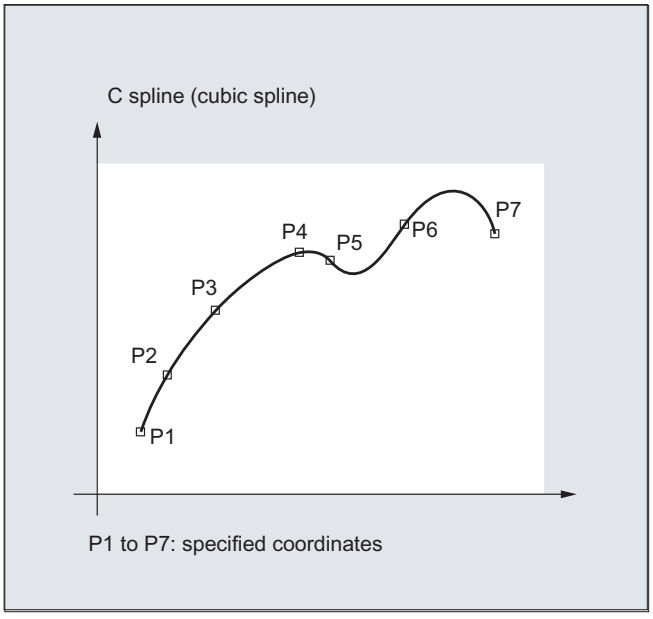
With spline interpolation, the following advantages can be obtained contrary to using straight line blocks `G01`:

- The number of part program blocks required to describe the contour are reduced
- Soft, curve characteristics that reduce the stress on the mechanical system at transitions between part program blocks.

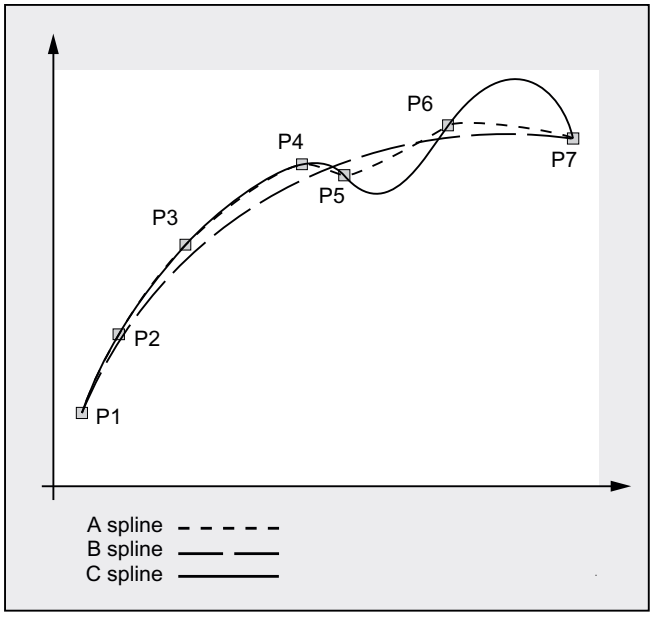
Properties and use of the various spline types

Spline type	Properties and use
A spline	 **Properties:** • Passes exactly through the specified intermediate points along the curve. • The curve characteristic is tangential, but does not have continuous curvature. • Produces hardly any undesirable oscillations. • The area of influence of changes to intermediate points along the curve is local. This means that a change to an intermediate point along the curve only affects up to max. 6 adjacent intermediate points. **Application:** The A spline is especially suitable for interpolating curves with large changes in the gradient (e.g. staircase-type curves and characteristics).

Spline type	Properties and use
B spline	 **Properties:** • Does not run through the specified intermediate points along the curve, but only close to them. The intermediate points to not attract the curve. The curve characteristic can be additionally influenced by weighting the intermediate points using a factor. • The curve characteristic is tangential with continuous curvature. • Does not generate any undesirable oscillations. • The area of influence of changes to intermediate points along the curve is local. This means that a change to an intermediate point along the curve only affects up to max. 6 adjacent intermediate points. **Application:** Their B spline is primarily intended as interface to CAD systems.

Spline type	Properties and use
C spline	 **Properties:** • Passes exactly through the specified intermediate points along the curve. • The curve characteristic is tangential with continuous curvature. • Frequently generates undesirable oscillations, especially at positions where the gradient changes significantly. • The area of influence of changes to the intermediate points is global. This means that if an intermediate point is changed then this influences the complete curved characteristic. **Application:** The C spline can be well used when the intermediate points lie on a curve defined analytically (circle, parabola, hyperbola).

Comparison of three spline types with identical interpolation points



Minimum number of spline blocks

The G codes `ASPLINE`, `BSPLINE` and `CSPLINE` link block end points with splines. For this purpose, a series of blocks (end points) must be simultaneously calculated. The buffer size for calculations is ten blocks as standard. Not every piece of block information is a spline end point. However, the control needs a certain number of spline end-point blocks for every ten blocks:

Spline type	Minimum number of spline blocks
A spline:	At least **4** blocks out of every 10 must be spline blocks. These do not include comment blocks or parameter calculations.
B spline:	At least **6** blocks out of every 10 must be spline blocks. These do not include comment blocks or parameter calculations.
C spline:	The required minimum number of spline blocks is the result of the following sum: Value of MD20160 $MC_CUBIC_SPLINE_BLOCKS + 1 The number of points to calculate the spline segment is entered in MD20160. The default setting is 8. As standard, at least **9** blocks out of every 10 must be spline blocks.

Note

An alarm is output if the tolerated value is undershot and likewise when one of the axes involved in the spline is programmed as a positioning axis.

Combine short spline blocks

Spline interpolation can result in short spline blocks, which reduce the path velocity unnecessarily. The "Combine short spline blocks" function allows you to combine these blocks such that the resulting block length is sufficient and does not reduce the path velocity.

The function is activated via the channel-specific machine data:

MD20488 $MC_SPLINE_MODE (setting for spline interpolation).

References:
Function Manual, Basic Functions; Continuous-Path Mode, Exact Stop, Look Ahead (B1), Chapter: Combine short spline blocks

5.3 Spline group (SPLINEPATH)

The axes to be interpolated in the spline group are selected using the `SPLINEPATH` command. Up to eight path axes can be involved in a spline interpolation grouping.

Note

If `SPLINEPATH` is not explicitly programmed, then the first three axes of the channel are traversed as spline group.

Syntax

The spline group is defined in a separate block:

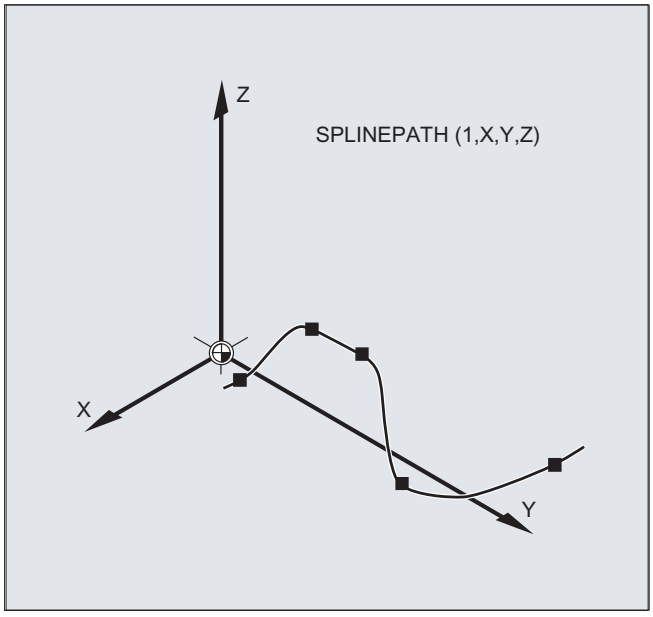
```
SPLINEPATH(n,X,Y,Z,…)
```

Meaning

`SPLINEPATH`:	Command to define a spline group
`n`:	=1 (fixed value)
`X,Y,Z,…` :	Identifier of the path axes to be interpolated in the spline group

Example: Spline group with three path axes

```
Program code                                          Comment
N10 G1 X10 Y20 Z30 A40 B50 F350
N11 SPLINEPATH(1,X,Y,Z)                               ; Spline group
N13 CSPLINE BAUTO EAUTO X20 Y30 Z40 A50 B60           ; C spline
N14 X30 Y40 Z50 A60 B70                               ; Intermediate points
…
N100 G1 X… Y…                                         ; Deselect spline interpola-
                                                      tion
```



5.4 Activating/deactivating NC block compression (COMPON, COMPCURV, COMPCAD, COMPSURF, COMPOF)

The compression functions are activated/deactivated using G commands of the G group 30. The commands are modal.

Syntax

```
COMPON / COMPCURV / COMPCAD / COMPSURF
...
COMPOF
```

Meaning

COMPON:	Activating the compressor function COMPON
COMPCURV:	Activating the compressor function COMPCURV
COMPCAD:	Activating the compressor function COMPCAD
COMPSURF:	Activating the compressor function COMPSURF
COMPOF :	Deactivating the currently active compressor function

Note

The rounding function G642 and jerk limitation SOFT further improve the surface quality. These commands must be written at the beginning of the program.

Example: COMPCAD

Program code	Comment
N10 G00 X30 Y6 Z40	
N20 G1 F10000 G642	; Activation: Rounding function G642
N30 SOFT	; Activation: Jerk limitation SOFT
N40 **COMPCAD**	; Activation: Compressor function COMPCAD
N50 STOPFIFO	
N24050 Z32.499	; 1st traversing block
N24051 X41.365 Z32.500	; 2nd traversing block
...	
N99999 X... Z...	; last traversing block
COMPOF	; compressor function off.
...	

5.5 Polynomial interpolation (POLY, POLYPATH, PO, PL)

It actually involves a polynomial interpolation (`POLY`) and not a spline interpolation type. Its main purpose is to act as an interface for programming externally generated spline curves where the spline sections can be programmed directly.

This mode of interpolation relieves the NC of the task of calculating polynomial coefficients. It can be optimally applied in cases where the coefficients are supplied directly by a CAD system or post processor.

Syntax

3rd degree polynomial:
```
POLY PO[X]=(xe,a2,a3) PO[Y]=(ye,b2,b3) PO[Z]=(ze,c2,c3) PL=n
```

5th degree polynomial and new polynomial syntax:
```
POLY X=PO(xe,a2,a3,a4,a5) Y=PO(ye,b2,b3,b4,b5) Z=PO(ze,c2,c3,c4,c5)
PL=n
POLYPATH("AXES","VECT")
```

Note

The sum of the polynomial coefficients and axes programmed in an NC block must not exceed the maximum permitted number of axes per block.

Meaning

`POLY` :	Activation of polynomial interpolation with a block containing POLY.
`POLYPATH` :	Polynomial interpolation can be selected for both AXIS or VECT axis groups
`PO[axis identifier/variable]` :	End points and polynomial coefficients

X, Y, Z:	Axis identifier
xe, ye, ze :	Specification of end position for the particular axis; value range as for path dimension
a2, a3, a4, a5:	The coefficients a_2, a_3, a_4, and a_5 are written with their value; value range as for path dimension. The last coefficient in each case can be omitted if it equals zero.
PL:	Length of the parameter interval where polynomials are defined (definition range of the function f(p)). The interval always starts at 0, p can assume values from 0 to PL. Theoretical value range for PL: 0.0001 … 99 999.9999 **Note:** The PL value applies to the block in which it is located. If no PL is programmed, then PL=1 is applied.

Activating/deactivating polynomial interpolation

The polynomial interpolation is activated in the part program using the POLX G command.

The POLY G command together with G0, G1, G2, G3, ASPLINE, BSPLINE and CSPLINE belong to the 1st group.

Axes, which are only programmed with name and end point (e.g. X10), are linearly moved. If all axes of an NC block are programmed in this way, the control behaves the same as for G1.

The polynomial interpolation is implicitly deactivated again by programming another command of the 1st G group G0, G1).

Polynomial coefficient

The PO value (PO[]=) or ...=PO(...) specifies all polynomial coefficients for an axis. Several values are specified, separated by commas corresponding the degree of the polynomial. Different degrees of polynomials are possible for various axes within one block.

POLYPATH subprogram

Using POLYPATH(...), the polynomial interpolation can be selectively released for certain axis groups:

Only path axes and supplementary axes:	POLYPATH("AXES")
Only orientation axes: (when moving with orientation transformation)	POLYPATH("VECT")

The axes that are not released are linearly moved.

Polynomial interpolation is enabled as standard for both axis groups.

Polynomial interpolation is deactivated for all axes by programming without the POLYPATH( ) parameter.

Example

Program code	Comment
N10 G1 X… Y… Z… F600	
N11 POLY PO[X]=(1,2.5,0.7) PO[Y]=(0.3,1,3.2) PL=1.5	; Polynomial interpolation on
N12 PO[X]=(0,2.5,1.7) PO[Y]=(2.3,1.7) PL=3	
...	
N20 M8 H126 …	
N25 X70 PO[Y]=(9.3,1,7.67) PL=5	; Mixed data for the axes
N27 PO[X]=(10,2.5) PO[Y]=(2.3)	; No PL programmed; PL=1 applies
N30 G1 X… Y… Z.	; Polynomial interpolation off
…	

Example: New polynomial syntax

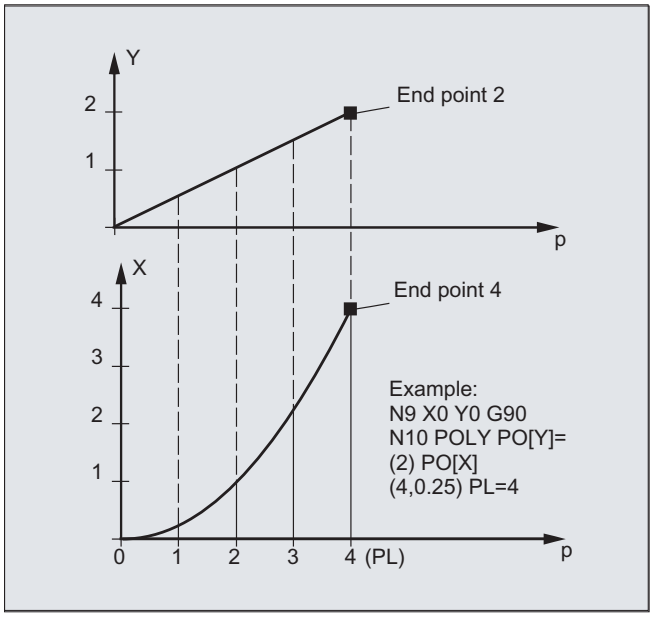
Polynomial syntax that is still valid	New polynomial syntax
PO[axis identifier]=(.. , ..)	Axis identifier=PO(.. , ..)
PO[PHI]=(.. , ..)	PHI=PO(.. , ..)
PO[PSI]=(.. , ..)	PSI=PO(.. , ..)
PO[THT]=(.. , ..)	THT=PO(.. , ..)
PO[]=(.. , ..)	PO(.. , ..)
PO[variable]=IC(.. , ..)	variable=PO IC(.. , ..)

Example: Curve in the X/Y plane.

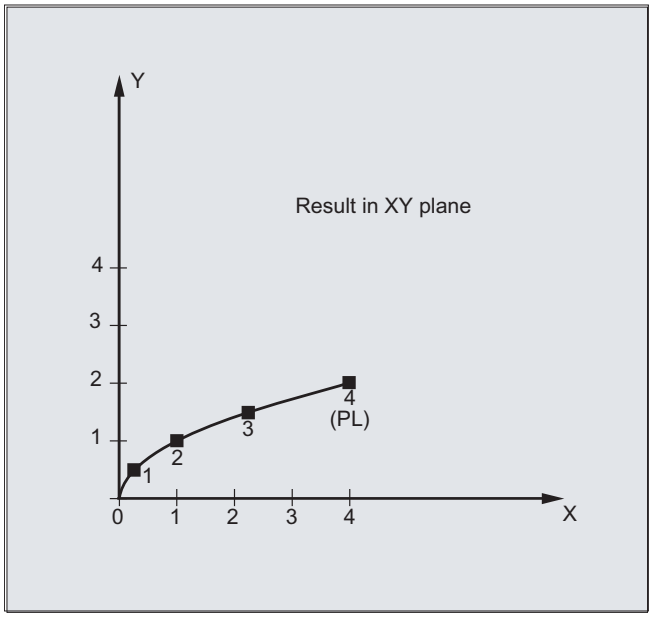
Programming

Program code
N9 X0 Y0 G90 F100
N10 POLY PO[Y]=(2) PO[X]=(4,0.25) PL=4

Shape of the curves X(p) and Y(p)



Shape of the curve in the XY plane



Description

The equation to express the polynomial function is generally as follows:

$f(p)= a_0 + a_1p + a_2p^2 +. . . + a_np^n$

with: a_n: Constant coefficients
p: Parameter

In the control, polynomials up to a maximum of the 5th degree can be programmed:

$$f(p)= a_0 + a_1p + a_2p^2 + a_3p^3 + a_4p^4 + a_5p^5$$

By assigning concrete values to these coefficients, it is possible to generate various curve shapes such as line, parabola and power functions.

A straight line is generated with $a_2 = a_3 = a_4 = a_5 = 0$:

$$f(p) = a_0 + a_1p$$

The following still applies:

$a_{0:}$ Axis position at the end of the preceding block
p = PL
$a_1 = (x_E - a_0 - a_2*p^2 - a_3*p^3)/p$

It is possible to program polynomials **without** the polynomial interpolation having been activated using the G command `POLY`. In this case, the programmed polynomials are not interpolated, but instead, all of the programmed end points of the axis are linearly approached (`G1`). The programmed polynomials are only moved as such after explicitly activating polynomial interpolation in the part program (`POLY`).

Special feature: Denominator polynomial

Command `PO[]=(…)` can be used to program a common denominator polynomial for the geometry axes (without specifying an axis name), i.e. the motion of the geometry axes is then interpolated as the quotient of two polynomials.

With this programming option, it is possible to represent shapes such as conics (circle, ellipse, parabola, hyperbola) exactly.

Example:

Program code	Comment
`POLY G90 X10 Y0 F100`	`; Geometry axes traverse linearly to position X10 Y0.`
`PO[X]=(0,-10) PO[Y]=(10) PO[]=(2,1)`	`; Geometry axes traverse along the quadrant to X0 Y10.`

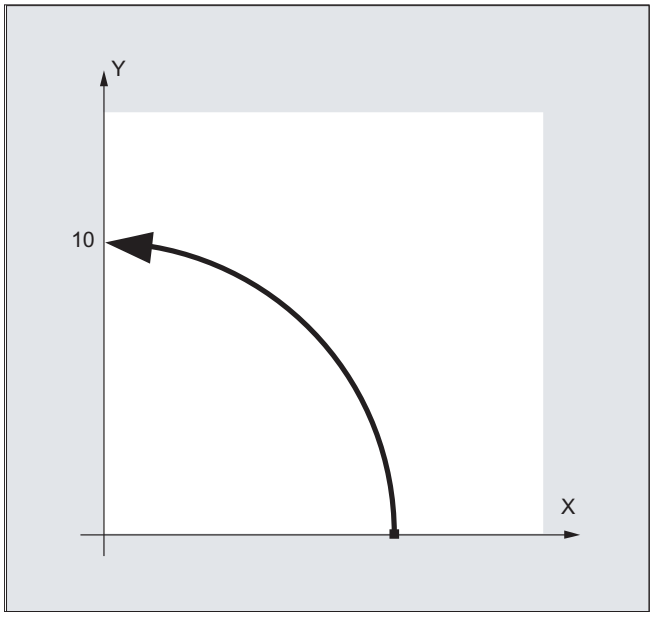
The constant coefficient (a_0) of the denominator polynomial is always assumed to be 1. The programmed end point is independent of `G90` / `G91`.

X(p) and Y(p) are calculated as follows from the programmed values:

$X(p) = (10 - 10 * p^2) / (1 + p^2)$
$Y(p) = 20 * p / (1 + p^2)$
with $0 \le p \le 1$

As a result of the programmed start points, end points, coefficient a_2 and PL=1, the intermediate results are as follows:

Numerator (X) = 10 + 0 * p - 10 * p^2
Numerator (Y) = 0 + 20 * p + 0 * p^2
Denominator = 1 + p^2



If polynomial interpolation is active and a denominator polynomial is programmed with zeros within the interval `[0,PL]`, this is rejected and an alarm is output. Denominator polynomials have no effect on the motion of special axes.

Note

Tool radius compensation can be activated with `G41`, `G42` in conjunction with polynomial interpolation and can be applied in the same way as in linear or circular interpolation modes.

5.6 Settable path reference (SPATH, UPATH)

For polynomial interpolation (`POLY`, `ASPLINE`, `BSPLINE`, `CSPLINE`, `COMPON`, `COMPCURV`), the positions of the path axes i are specified as polynomials $p_i(U)$. The curve parameter U moves from 0 to 1 within an NC block.

`FGROUP` selects the axes (FGROUP axes) to which the path feedrate `F` applies. An interpolation with constant speed on the path S of the FGROUP axes means during the polynomial interpolation normally a non-constant change of the curve parameter U. Consequently, two possibilities are available for selecting the axes not contained in `FGROUP` on how they should follow the FGROUP axes:

- Synchronous to path S (`SPATH`)
- Synchronous to curve parameter U (`UPATH`)

Syntax

```
SPATH
UPATH
```

Meaning

SPATH:	The axes not contained in FGROUP are traversed with reference to path S
UPATH:	The axes not contained in FGROUP are traversed with reference to curve parameter U

Note

UPATH and SPATH also define the interrelationship of the F word polynomial (FPOLY, FCUB, FLIN) with path motion.

Supplementary conditions

SPATH and UPATH have no meaning for:

- Linear interpolation (G1)
- Circuit interpolation (G2, G3)
- Thread blocks (G33, G34, G35, G33x, G63)
- All path axes are contained in FGROUP

Example

The following example shows the difference between both types of motion control.

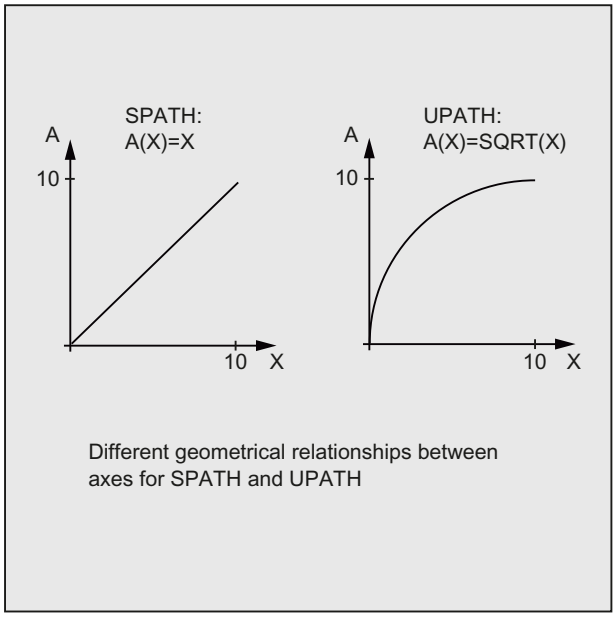
Program code

```
N10 FGROUP(X,Y,Z)
N15 G1 X0 A0 F1000 SPATH                ; SPATH
N20 POLY PO[X]=(10,10) A10
```

Program code

```
N10 FGROUP(X,Y,Z)
N15 G1 X0 A0 F1000 UPATH                ; UPATH
N20 POLY PO[X]=(10,10) A10
```

In both program sections, the path S of the FGROUP axes in N20 is dependent on the square of curve parameter U. Therefore, different position arise for synchronized axis A along path X, according to whether SPATH or UPATH is active.



Different geometrical relationships between axes for SPATH and UPATH

Further information

Control behavior for reset and machine/option data

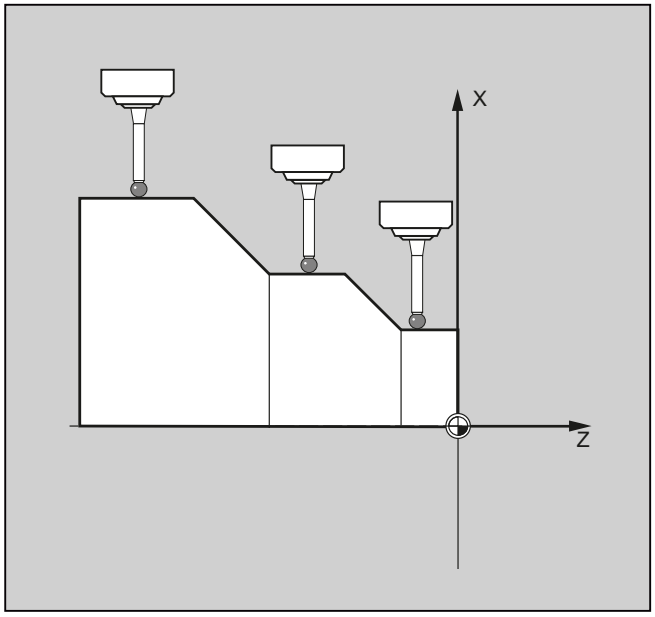
The G code, defined with MD20150 $MC_GCODE_RESET_VALUES[44] (45th G function) is effective after a reset.

The initial state for the type of smoothing is defined with MD20150 $MC_GCODE_RESET_VALUES[9] (10th G code group).

The axis-specific machine data MD33100 $MA_COMPRESS_POS_TOL[<n>] has an extended significance: It contains the tolerances for the compressor function and for smoothing with G642.

5.7 Measuring with touch-trigger probe (MEAS, MEAW)

The "Measure with touch-trigger probe" is used to approach actual positions on the workpiece. On the probe's switching edge, the positions for all axes programmed in the measurement block are measured and written to the appropriate memory cell for each axis.



The following two fixed addresses are available for programming the function:

- `MEAS`
  `MEAS` deletes the distance-to-go between the actual and setpoint positions.
- `MEAW`
  `MEAW` is used in the case of measuring tasks where the programmed position always needs to be approached.

`MEAS` and `MEAW` are non-modal; they are programmed together with motion operations. The feedrate and interpolation type (G0, G1, etc.) as well as the number of axes must be adapted for the respective measuring task.

Syntax

```
MEAS=<TE> G... X... Y... Z...
MEAW=<TE> G... X... Y... Z...
```

Meaning

`MEAS`:	Command: Measurement **with** delete distance-to-go	
	Effective:	Non-modal
`MEAW`:	Command: Measurement **without** delete distance-to-go	
	Effective:	Non-modal

<TE>:	Trigger event to initiate measurement		
	Type:	INT	
	Range of values:	-2, -1, 1, 2	
	Meaning:		
	(+)1	Rising edge of probe 1 (measuring input 1)	
	-1	Falling edge of probe 1 (measuring input 1)	
	(+)2	Rising edge of probe 2 (measuring input 2)	
	-2	Falling edge of probe 2 (measuring input 2)	
	Note: There are up to a maximum of 2 probes (dependent on configuration level).		
G...:	Type of interpolation, e.g. G0, G1, G2 or G3		
X... Y... Z...:	End points in Cartesian coordinates		

Example

Program code	Comment
N10 MEAS=1 G1 F1000 X100 Y730 Z40	; Measurement block with probe at first measuring input and linear interpolation. A preprocessing stop is automatically generated.
...	

Further information

Measuring task status

If an evaluation of whether or not the probe has been triggered is required in the program, status variable $AC_MEA[<n>] (<n> = number of the measuring probe) can be checked:

Value	Meaning
0	Measuring task not completed.
1	Measuring task completed successfully (the probe has been triggered).

Note

If the program is deflected in the program, the variable is set to 1. At the start of a measurement block, the variable is automatically set to the initial state of the probe.

Reading measured values

The positions of all traversing path and positioning axes of the block are acquired (maximum number of axes depending on the control configuration). In the case of `MEAS`, the motion is decelerated in a defined way following the triggering of the probe.

Note

If a geometry axis is programmed in a measuring block, the measured values are stored for all current geometry axes.

If an axis participating in a transformation is programmed in a measurement block, the measured values for all axes participating in this transformation are recorded.

Reading measurement results

The measurement results for the axes measured with probes can be read via the following system variables:

- $AA_MM[<axis>]
  Measurement results in the machine coordinate system
- $AA_MW[<axis>]
  Measurement results in the workpiece coordinate system

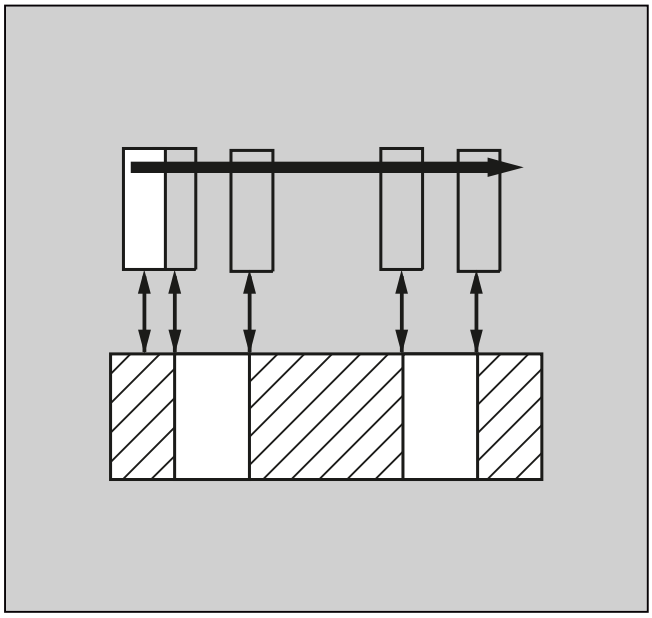
No internal preprocessing stop is generated when these variables are read.

Note

A preprocessing stop must be programmed with `STOPRE` at the appropriate position in the program. The system will otherwise read false values.

5.8 Axial measurement (MEASA, MEAWA, MEAC) (option)

Several probes and several measuring systems can be used for the axial measuring.



The keywords MEASA, MEAWE and MEAC are available for programming the function.

With MEASA or MEAWA for the programmed axis, up to four measured values are acquired for each measurement and are then saved in system variables in accordance with the trigger event.

Measuring operations can be executed with MEAC. In this case, the measurement results are stored in FIFO variables.

Syntax

```
MEASA[<axis>]=(<mode>,<TE1>,...,<TE4>)
MEAWA[<axis>]=(<mode>,<TE1>,...,<TE4>)
MEAC[<axis>]=(<mode>,<measurement memory>,<TE1>,...,<TE4>)
```

Note

MEASA and MEAWA are non-modal; they can be programmed together in one block. However, if MEASA/MEAWA is programmed together with MEAS/MEAW in the same block, an error message is output.

Meaning

MEASA:	Keyword: Axial measurement with delete distance-to-go	
	Effective:	Non-modal
MEAWA:	Keyword: Axial measurement without delete distance-to-go	
	Effective:	Non-modal

MEAC:	Keyword: Continuous axial measurement without delete distance-to-go	
	Effective:	Non-modal
<axis>:	Name of channel axis used for measurement	
<mode>:	Two-digit number indicating the operating mode (measuring mode and measuring system)	
	Units decade (measuring mode):	
	0	Cancel measuring task.
	1	Up to four different trigger events that can be activated at the same time.
	2	Up to four different trigger events that can be activated in succession.
	3	Up to four different trigger events that can be activated in succession, but with no monitoring of trigger event 1 at the start (alarms 21700/21703 are suppressed). **Note:** This mode is not supported by MEAC.
	Tens decade (measuring system):	
	0 (or no data)	active measuring system
	1	Measuring system 1
	2	Measuring system 2
	3	Both measuring systems
<TE>:	Trigger event to initiate measurement	
	Type:	INT
	Range of values:	-2, -1, 1, 2
	Meaning:	
	(+)1	Rising edge of probe 1
	-1	Falling edge of probe 1
	(+)2	Rising edge of probe 2
	-2	Falling edge of probe 2
<measurement memory>:	Number of FIFO (circulating storage)	

Examples

Example 1: Axial measurement with delete distance-to-go in mode 1 (evaluation in chronological sequence)

a) With one measuring system

```
Program code                                  Comment
...
N100 MEASA[X]=(1,1,-1) G01 X100 F100          ; Measuring in mode 1 with active measur-
                                              ing system. Wait for measuring signal
                                              with rising/falling edge from probe 1 for
                                              travel path to X=100.
N110 STOPRE                                   ;Preprocessing stop
N120 IF $AC_MEA[1]==FALSE GOTOF END           ; Check that the measurement was success-
                                              ful.
```

Program code	Comment
N130 R10=$AA_MM1[X]	; Save measured value acquired at the first programmed trigger event (rising edge).
N140 R11=$AA_MM2[X]	; Save measured value acquired at the second programmed trigger event (falling edge).
N150 END:	

b) With two measuring systems

Program code	Comment
...	
N200 MEASA[X]=(31,1,-1) G01 X100 F100	; Measuring in mode 1 with both measuring systems. Wait for measuring signal with rising/falling edge from probe 1 for travel path to X=100.
N210 STOPRE	;Preprocessing stop
N220 IF $AC_MEA[1]==FALSE GOTOF END	; Check that the measurement was successful.
N230 R10=$AA_MM1[X]	; Save measured value of measuring system 1 at rising edge.
N240 R11=$AA_MM2[X]	; Save measured value of measuring system 2 at rising edge.
N250 R12=$AA_MM3[X]	; Save measured value of measuring system 1 at falling edge.
N260 R13=$AA_MM4[X]	; Save measured value of measuring system 2 at falling edge.
N270 END:	

Example 2: Axial measurement with delete distance-to-go in mode 2 (evaluation in programmed sequence)

Program code	Comment
...	
N100 MEASA[X]=(2,1,-1,2,-2) G01 X100 F100	; Measuring in mode 2 with active measuring system. Wait for measuring signal in the sequence rising edge probe 1, falling edge probe 1, rising edge probe 2, falling edge probe 2 while traversing path to X=100.
N110 STOPRE	;Preprocessing stop
N120 IF $AC_MEA[1]==FALSE GOTOF PROBE2	; Check that the measurement with probe 1 is successful.
N130 R10=$AA_MM1[X]	; Save measured value acquired at the first programmed trigger event (rising edge of probe 1).

Program code	Comment
N140 R11=$AA_MM2[X]	; Save measured value acquired at the second programmed trigger event (rising edge of probe 1).
N150 PROBE2:	
N160 IF $AC_MEA[2]==FALSE GOTOF END	; Check that the measurement with probe 2 is successful.
N170 R12=$AA_MM3[X]	; Save measured value acquired at the third programmed trigger event (rising edge of probe 2).
N180 R13=$AA_MM4[X]	; Save measured value acquired at the fourth programmed trigger event (rising edge of probe 2).
N190 END:	

Example 3: Continuous axial measurement in mode 1 (evaluation in chronological sequence)

a) Measurement of up to 100 measured values

Program code	Comment
...	
N110 DEF REAL MEASVALUE[100]	
N120 DEF INT loop=0	
N130 MEAC[X]=(1,1,-1) G01 X1000 F100	; Measuring in mode with active measuring system, save measured values; under $AC_FIFO1, wait for measuring system with falling edge from probe 1 travel path to X=1000.
N135 STOPRE	
N140 MEAC[X]=(0)	; Terminate measurement when axis position is reached.
N150 R1=$AC_FIFO1[4]	; Save number of accumulated measured values in parameter R1.
N160 FOR loop=0 TO R1-1	
N170 MEASURED VALUE[loop]=$AC_FIFO1[0]	; Read-out measured values from $AC_FIFO1 and save.
N180 ENDFOR	

b) Measurement with delete distance-to-go after 10 measured values

Program code	Comment
...	
N10 WHEN $AC_FIFO1[4]>=10 DO MEAC[x]=(0) DELDTG(x)	; Delete distance-to-go.
N20 MEAC[x]=(1,1,1,-1) G01 X100 F500	
N30 MEAC [X]=(0)	
N40 R1 = $AC_FIFO1[4]	; Number of measured values.
...	

c) Measurement of a falling/rising tooth flank with two probes

```
Program code                                    Comment
...
N110 DEF REAL MEASVALUE[16]
N120 DEF INT loop=0
N130 MEAC[X]=(1,1,-1,2) G01 X100 F100           ; Measurement in mode 1 with active meas-
                                                uring system, save measured values under
                                                $AC_FIFO1, wait for measuring signal in
                                                the sequence falling edge of probe 1, ris-
                                                ing edge of probe 2 while traversing the
                                                path to X=100.
N140 STOPRE                                     ;Preprocessing stop
N150 MEAC[X]=(0)                                ; Terminate measurement when axis posi-
                                                tion is reached.
N160 R1=$AC_FIFO1[4]                            ; Save number of accumulated measured val-
                                                ues in parameter R1.
N170 FOR loop=0 TO R1-1
N180 MEASURED VALUE[loop]=$AC_FIFO1[0]          ; Read-out measured values from $AC_FIFO1
                                                and save.
N190 ENDFOR
```

Additional information

Measurement job

A measuring task can be programmed in the part program or from a synchronized action (see Chapter "Synchronized actions (Page 521)"). Please note that only one measuring job can be active at any given time for each axis.

Note

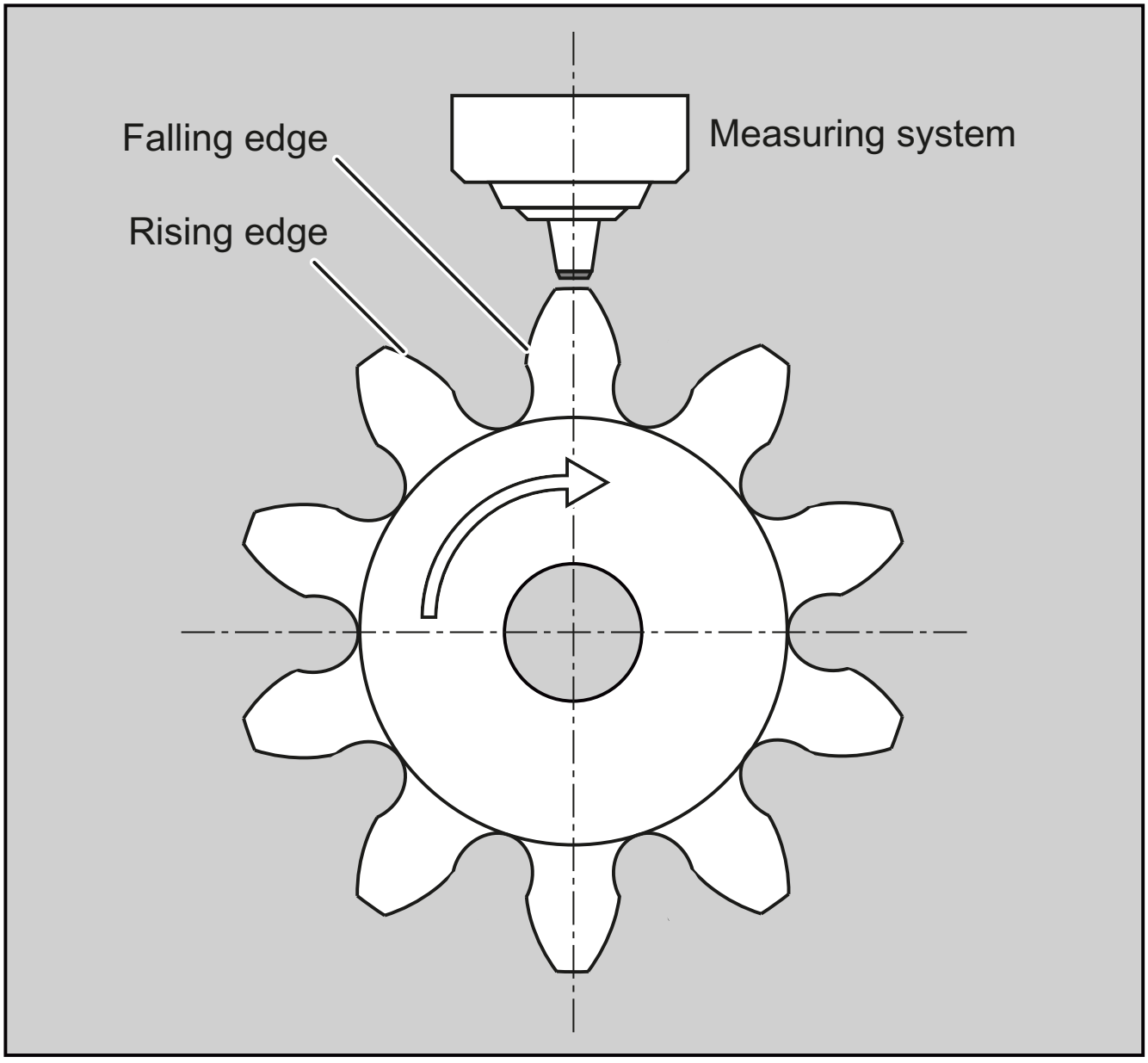
The feed must be adjusted to suit the measuring task in hand.

In the case of MEASA and MEAWA, the correctness of results can be only guaranteed for feedrates at which no more than 1 trigger event of the same type and no more than 4 trigger events of different types occur in each position control cycle.

In the case of continuous measurement with MEAC, the ratio between interpolator clock cycle and position control cycle must not exceed 1:8.

Trigger event

A trigger event comprises the number of the probe and the trigger criterion (rising or falling edge) of the measuring signal.



Up to 4 trigger events of the addressed probe can be processed for each measurement; in other words, up to 2 probes with 2 measuring signal edges each. The processing sequence and the maximum number of trigger events depend on the selected mode.

Note

The following applies for measuring mode 1: The same trigger event may only be programmed once in one measuring task.

With `MEAC`, the number of measured values per position control cycle can be increased to eight rising edges and to eight falling edge for each probe. This enables higher feedrates and speeds to be implemented.

References:
Function Manual, Extended Functions; Measurements (M5), Section: Axial measurement

Operating mode

The first digit (tens decade) of the operating mode selects the required measuring system. If only one measuring system is installed, but a second programmed, the installed system is automatically selected.

The second digit (units decade) selects the required measuring mode. The measuring process is thus adapted to the options supported by the relevant control:

- **Mode 1**
  Trigger events are evaluated in the chronological sequence in which they occur. When this mode is selected, only one trigger event can be programmed for six-axis modules. If more than one trigger event is specified, the mode selection is switched automatically to mode 2 (without message).
- **Mode 2**
  Trigger events are evaluated in the programmed sequence.
- **Mode 3**
  Trigger events are evaluated in the programmed sequence but there is no monitoring of trigger event 1 at START.

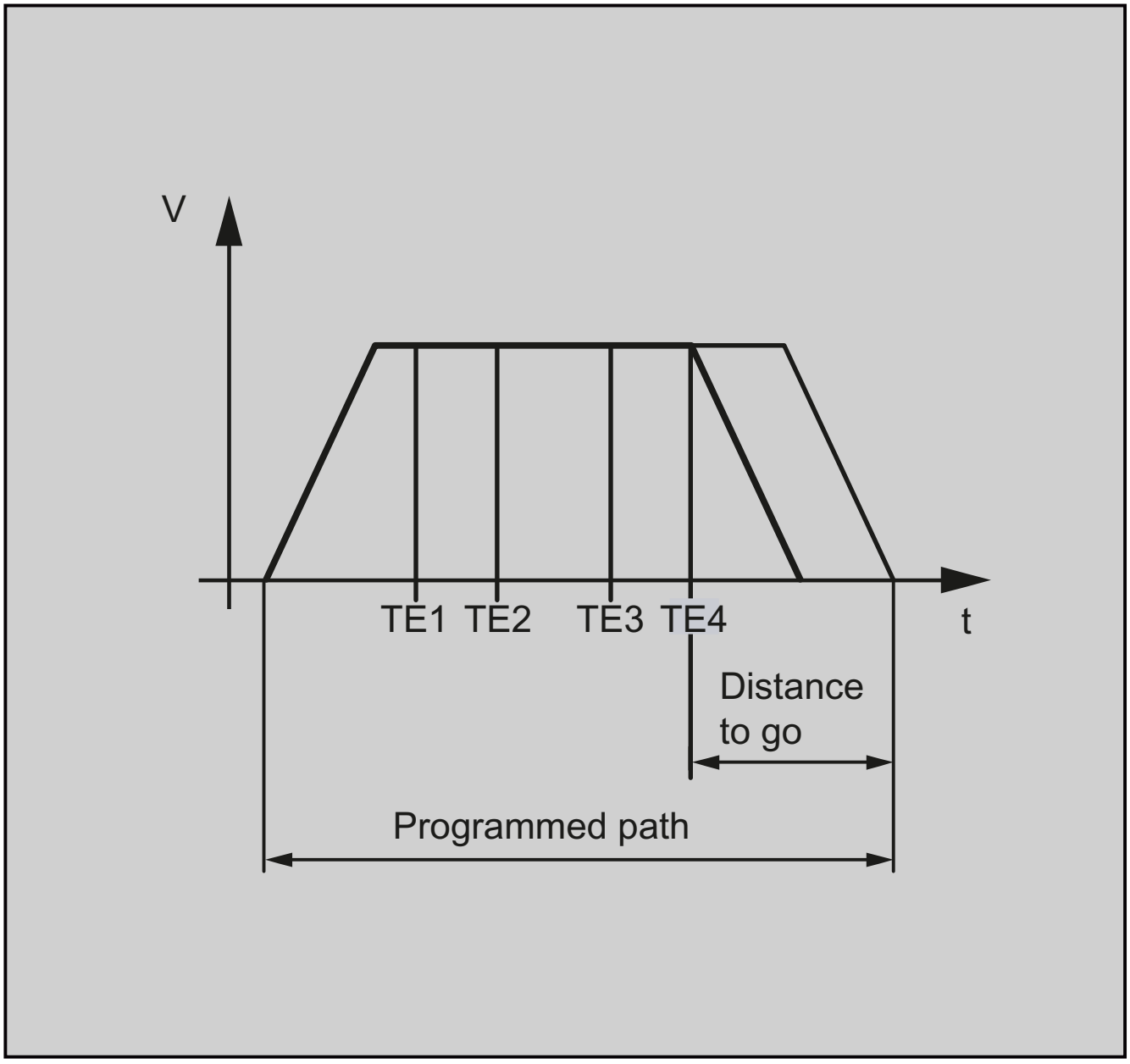
Note

No more than two trigger events can be programmed if two measuring systems are in use.

Measurement with and without delete distance-to-go

When command `MEASA` is programmed, the distance-to-go is not deleted until all required measured values have been recorded.

The `MEAWA` function is used in the case of special measuring tasks where a programmed position always needs to be approached.



Note

MEASA cannot be programmed in synchronized actions. As an alternative, MEAWA plus delete distance-to-go can be programmed as a synchronized action.

If the measuring task with MEAWA is started from synchronized actions, the measured values will only be available in the machine coordinate system.

Measurement results for MEASA, MEAWA

The results of measurements are available under the following system variables:

- In the machine coordinate system:

$AA_MM1[<axis>]	Measured value of programmed measuring system on trigger event 1
...	...
$AA_MM4[<axis>]	Measured value of programmed measuring system on trigger event 4

- In the workpiece coordinate system:

$AA_WM1[<axis>]	Measured value of programmed measuring system on trigger event 1
...	...
$AA_WM4[<axis>]	Measured value of programmed measuring system on trigger event 4

Note

No internal preprocessing stop is generated when these variables are read. A preprocessing stop must be programmed with STOPRE at the appropriate position in the program. False values will otherwise be read in.

Geometry axes/Transformations

If axial measurement is to be started for a geometry axis, the same measuring job must be programmed explicitly for all remaining geometry axes. The same applies to axes involved in a transformation.

Examples:

```
N10 MEASA[Z]=(1,1) MEASA[Y]=(1,1) MEASA[X]=(1,1) G0 Z100
```

or

```
N10 MEASA[Z]=(1,1) POS[Z]=100
```

Measurement job with two measuring systems

If a measuring job is executed by two measuring systems, each of the two possible trigger events of both measuring systems of the relevant axis is acquired. The assignment of the reserved variables is therefore preset:

$AA_MM1[<axis>]	or	$AA_MW1[<axis>]	Measured value from measuring system 1 on trigger event 1
$AA_MM2[<axis>]	or	$AA_MW2[<axis>]	Measured value from measuring system 2 on trigger event 1
$AA_MM3[<axis>]	or	$AA_MW3[<axis>]	Measured value from measuring system 1 on trigger event 2
$AA_MM4[<axis>]	or	$AA_MW4[<axis>]	Measured value from measuring system 2 on trigger event 2

System variables

The probe status is available in the following system variables:

$A_PROBE[<n>]

Value	Meaning
1	Probe deflected
0	Probe not deflected

The probe limitation is available in the following system variables:

$A_PROBE_LIMITED[<n>]

Value	Meaning
1	Probe limitation active
0	Probe limitation inactive

<n> = probe

Reference:
List Manual, System Variables

Measuring job status for MEASA, MEAWA

If an evaluation is required in the program, the measuring task status can be queried via $AC_MEA[<n>], where <n> = number of the probe. Once all the trigger events of probe <n> that are programmed in a block have occurred, this variable returns a value of "1". Otherwise, its value is 0.

Note

If measurement is started from synchronized actions, $AC_MEA is no longer updated. In this case, the new PLC interface signal DB31, … DBX62.3 or the equivalent variable $AA_MEAACT[<axis>] must be queried.

Meaning:

$AA_MEAACT==1: Measurement active

$AA_MEAACT==0: Measurement not active

Continuous measurement (MEAC)

The measured values for `MEAC` are available in the machine coordinate system and stored in the programmed FIFO[n] memory (circular buffer). If two probes are configured for the measurement, the measured values of the second probe are stored separately in the FIFO[n +1] memory configured especially for this purpose (defined in machine data).

The FIFO memory is a circular buffer in which measured values are written to $AC_FIFO variables according to the circular principle, see Section "Synchronized actions (Page 521)".

Note

FIFO contents can be read only once from the circulating storage. If this measured data is to be used several times, it must be buffered in the user data.

If the number of measured values for the FIFO memory exceeds the maximum value defined in machine data, the measurement is automatically terminated.

An endless measuring process can be implemented by reading out measured values cyclically. In this case, data must be read out at the same frequency as new measured values are read in.

References:

- Function Manual, Synchronized Actions; Detailed Description, Section: Parameters ($AC_FIFO)
- Function Manual, Extended Functions; Measurements (M5), Section: Axial measurement

Protection against programming errors

The following programming errors are detected and indicated appropriately:

- MEASA/MEAWA programmed with MEAS/MEAW in the same block
  Example:
  ```
  N01 MEAS=1 MEASA[X]=(1,1) G01 F100 POS[X]=100
  ```
- MEASA/MEAWA with number of parameters <2 or >5
  Example:
  ```
  N01 MEAWA[X]=(1) G01 F100 POS[X]=100
  ```
- MEASA/MEAWA with trigger event not equal to 1/ -1/ 2/ -2
  Example:
  ```
  N01 MEASA[B]=(1,1,3) B100
  ```
- MEASA/MEAWA with invalid mode
  Example:
  ```
  N01 MEAWA[B]=(4,1) B100
  ```
- MEASA/MEAWA with trigger event programmed twice
  Example:
  ```
  N01 MEASA[B]=(1,1,-1,2,-1) B100
  ```
- MEASA/MEAWA and missing geometry axis
  Example:
  ```
  N01 MEASA[X]=(1,1) MEASA[Y]=(1,1) G01 X50 Y50 Z50 F100 ;GEO axis X/Y/Z
  ```
- Inconsistent measuring task with geometry axes
  Example:
  ```
  N01 MEASA[X]=(1,1) MEASA[Y]=(1,1) MEASA[Z]=(1,1,2) G01 X50 Y50 Z50 F100
  ```

5.9 Special functions for OEM users (OMA1 ... OMA5, OEMIPO1, OEMIPO2, G810 ... G829)

OEM addresses

The meaning of OEM addresses is determined by the OEM user. Their functionality is incorporated by means of compile cycles. Five OEM addresses are reserved (OMA1 ... OMA5). The address identifiers are settable. OEM addresses can be programmed in any block.

Reserved G function calls

The following G function calls are reserved for OEM users:

- OEMIPO1, OEMIPO2 (from G function group 1)
- G810 ... G819 (G function group 31)
- G820 ... G829 (G function group 32)

Their functionality is incorporated by means of compile cycles.

Functions and subprograms

OEM users can also set up predefined functions and subprograms with parameter transfer.

Note

Workpiece simulation

Up to SW 4.4, no compile cycles were supported, as of SW 4.4, only selected compile cycles (CC) are supported for the workpiece simulation.

Language commands in the part program of compile cycles that are not supported (OMA1 ... OMA5, OEMIPO1/2, G810 ... G829, own procedures and functions) - therefore result in an alarm message and cancellation of the simulation without any individual handling.

Solution: Individually handle the missing CC-specific language elements in the part program ($P_SIM query).
Example:

```
N1 G01 X200 F500
IF (1==$P_SIM)
N5 X300 ;not active for CC simulation
ELSE
N5 X300 OMA1=10
ENDIF
```

5.10 Feedrate reduction with corner deceleration (FENDNORM, G62, G621)

With automatic corner deceleration the feed rate is reduced according to a bell-shaped curve before reaching the corner. It is also possible to parameterize the extent of the tool behavior relevant to machining via setting data. These are:

- Start and end of feed rate reduction
- Override with which the feed rate is reduced
- Detection of a relevant corner

Relevant corners are those whose inside angle is less than the corner parameterized in the setting data.

Default value `FENDNORM` deactivates the function of the automatic corner override.

References:
/FBFA/ "Function Description ISO Dialects"

Syntax

```
FENDNORM
G62 G41
G621
```

Meaning

`FENDNORM`:	Automatic corner deceleration OFF
`G62`:	Corner deceleration at inside corners when tool radius offset is active
`G621`:	Corner deceleration at all corners when tool radius offset is active

G62 only applies to inside corners with

- active tool radius offset `G41`, `G42` and
- active continuous-path mode `G64`, `G641`

The corner is approached at a reduced feed rate resulting from:

F * (override for feed rate reduction) * feed rate override

The maximum possible feed rate reduction is attained at the precise point where the tool is to change directions at the corner, with reference to the center path.

G621 applies analogously with G62 at each corner of the axes defined by `FGROUP`.

5.11 Programmable end of motion criteria (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)

Similar to the block change criterion for path interpolation (G601, G602, and G603) it is also possible to program the end-of-motion criterion for single-axis interpolation in a part program or in synchronized actions for command/PLC axes.

The end-of-motion criterion set will affect how quickly or slowly part program blocks and technology cycle blocks with single-axis movements are completed. The same applies for PLC via FC15/16/18.

Syntax

```
FINEA[<axis>]
COARSEA[<axis>]
IPOENDA[<axis>]
IPOBRKA(<axis>[,<instant in time>])
ADISPOSA[<axis>]=(<mode>,<window size>)
```

Meaning

FINEA:	End-of-motion criterion: "Exact stop fine"		
	Effective:	Modal	
COARSEA:	End-of-motion criterion: "Exact stop coarse"		
	Effective:	Modal	
IPOENDA:	End-of-motion criterion: "Interpolator stop"		
	Effective:	Modal	
IPOBRKA:	Block change criterion: Braking ramp		
	Effective:	Modal	
ADISPOSA:	Tolerance window for end-of-motion criterion		
	Effective:	Modal	
<axis>:	Channel axis name (X, Y, ...)		
<instant in time>:	Time of the block change, referred to the braking ramp as a %: • 100% = start of the braking ramp • 0% = end of the braking ramp, the same significance as IPOENDA		
	Type:	REAL	
<mode>:	Reference of the tolerance window		
	Range of values:	0	Tolerance window not active
		1	Tolerance window with respect to set position
		2	Tolerance window with respect to actual position
	Type:	INT	
<window size>:	Size of the tolerance window		
	Type:	REAL	

Examples

Example 1: End-of-motion criterion: "Interpolator stop"

```
Program code
; traverse positioning axis X to 100, velocity 200 m/
min, acceleration 90%,
; end-of-motion criterion: Interpolator stop
N110 G01 POS[X]=100 FA[X]=200 ACC[X]=90 IPOENDA[X]

; Synchronized action:
; ALWAYS IF: Input 1 is set
; THEN traverse positioning axis X to 50, velocity 200
m/min, acceleration 140%,
;      end-of-motion criterion: Interpolator stop
N120 EVERY $A_IN[1] DO POS[X]=50 FA[X]=200 ACC[X]=140
IPOENDA[X]
```

Example 2: Block change criterion: "Braking ramp"

Program code	Comment
	; Default setting is effective
N40 POS[X]=100	; Positioning motion from X to position 100.
	Block change criterion: Exact stop fine
N20 IPOBRKA(X,100)	; Block change criterion: "Braking ramp",
	100% = start of the braking ramp
N30 POS[X]=200	; The block is changed as soon as the X axis starts to brake
N40 POS[X]=250	; X axis no longer brakes at position 200, but rather continues to traverse to position 250.
	As soon as the axis starts to brake, the block changes.
N50 POS[X]=0	; Axis X brakes and returns to position 0.
	The block change takes place at position 0 and "exact stop fine"
N60 X10 F100	; Axis X traverses as path axis to position 10

Further information

System variable for end-of-motion criterion

The effective end-of-motion criterion can be read using the system variable $AA_MOTEND.

References: /LIS2sl/ List Manual, Book 2

Block-change criterion: "Braking ramp" (IPOBRKA)

If, when activating the block change criterion "brake ramp", a value is programmed for the optional block change instant in time, then this becomes effective for the next positioning motion and is written into the setting data synchronized to the main run. If no value is specified for the block change instant in time, then the actual value of the setting data is effective.

SD43600 $SA_IPOBRAKE_BLOCK_EXCHANGE

`IPOBRKA` is deactivated for the corresponding access when an axis end-of-motion criterion (`FINEA`, `COARSEA` , `IPOENDA`) is next programmed for the axis.

Additional block-change criterion: "Tolerance window" (ADISPOSA)

Using `ADISPOSA`, a tolerance window around the end of block (either as actual or setpoint position) can be defined as additional block change criterion. Then, two conditions must be fulfilled for the block change:

- Block-change criterion: "Braking ramp"
- Block-change criterion: "Tolerance window"

References

For further information about the block change criterion for positioning axes, see:

- Function Manual, Extended Functions; Positioning Axes (P2)
- Programming Manual, Fundamentals; Chapter "Feedrate control".

Coordinate transformations (frames) 6

6.1 Coordinate transformation via frame variables

In addition to the commands such as ROT, AROT and SCALE described in the Fundamentals Programming Manual, "Coordinate transformations (frames)" section, the workpiece coordinate system (WCS) can also be transformed by the frame variables $P_...FR (data storage frames) and $P_...FRAME (active frames).

The following diagram provides an overview of structuring frame variables:

- Data management frames
- Active frames
- Active total frame: Chain of all active frames
- NCU global frames
- Channel-specific frames

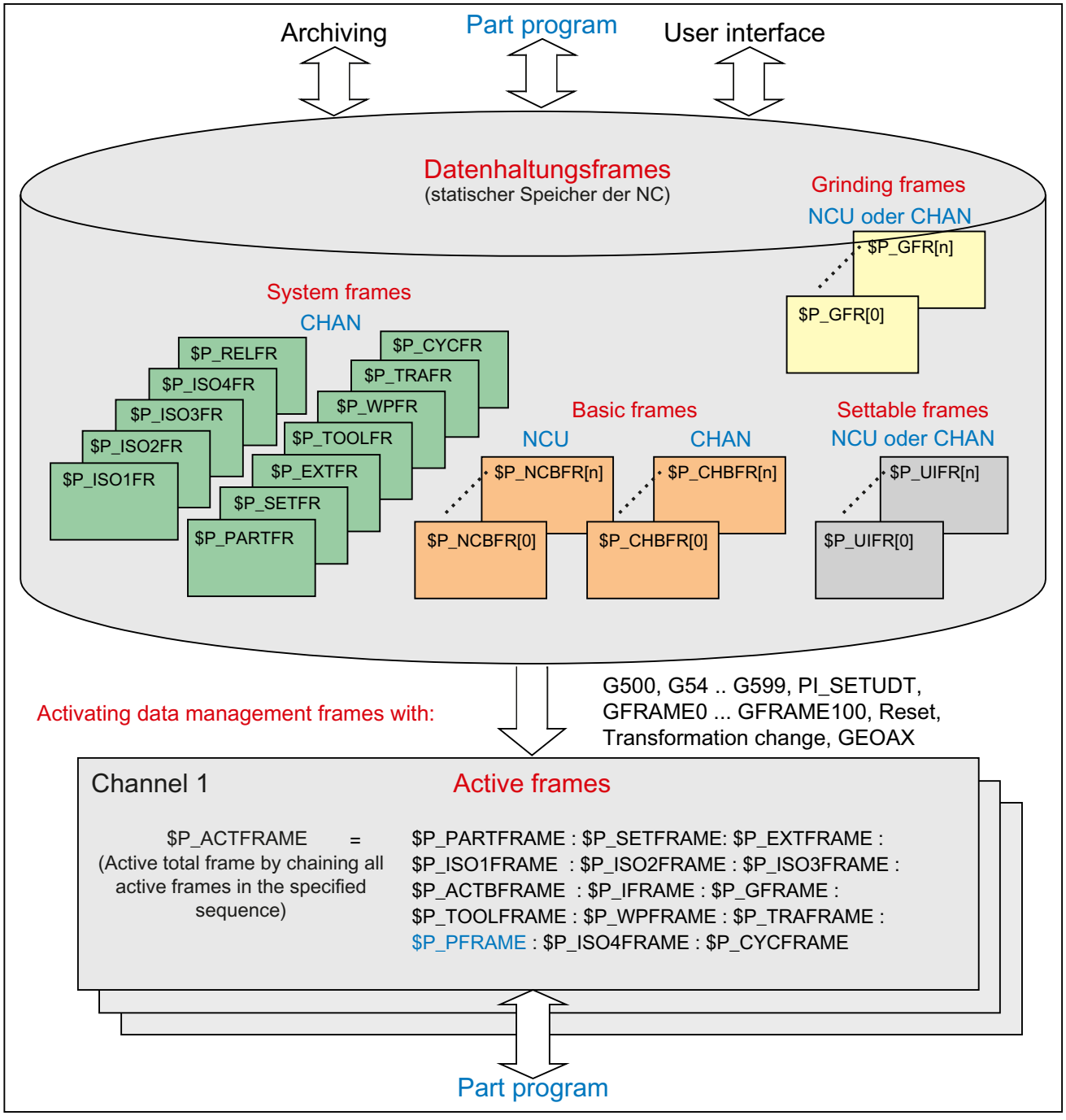


Figure 6-1 Overview of the frame variables

6.1.1 Predefined frame variable ($P_CHBFRAME, $P_IFRAME, $P_PFRAME, $P_ACTFRAME)

Active: channel-specific base frames $P_CHBFRAME[<n>] ($P_BFRAME)

Note

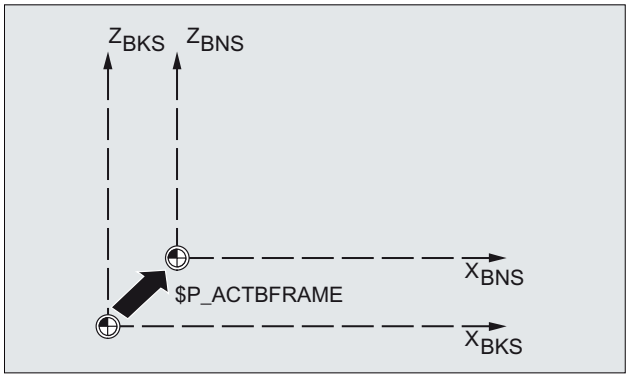
The current base frame `$P_BFRAME` and the data storage base frame `$P_UBFR` are retained for compatibility reasons.

- `$P_BFRAME` □ `$P_CHBFRAME[0]`
- `$P_UBFR` □ `$P_CHBFR[0]`.

The frame variables $P_CHBFRAME[<n>] define the reference between the basic coordinate system (BCS) and the basic origin system (BOS).

If the current channel-specific base frame $P_CHBFRAME[`<n>`] should be active immediately in the NC program, the following possibilities are available.

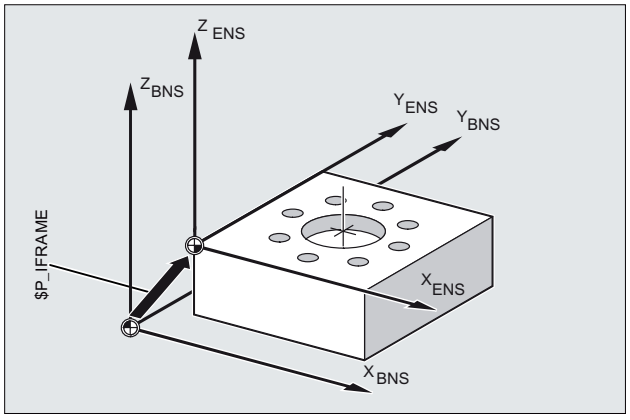
- Commands:
  - `G500` (deactivate all settable frames, the base frames remain active)
  - `G54` to `G599` (settable zero offsets)
- Assignment of a channel-specific base frames of the data storage to a current channel-specific base frame:
  `$P_CHBFRAME[<n>] = $P_CHBFR[<m>]`



Active: Channel-specific settable frame $P_IFRAME

The frame variable $P_IFRAME defines the reference between the basic origin system (BOS) and the settable zero system (SZS).

- `$P_IFRAME` corresponds to `$P_UIFR[$P_IFRNUM]`
- After `G54` is programmed, for example, `$P_IFRAME` contains the translation, rotation, scaling and mirroring defined by `G54`.

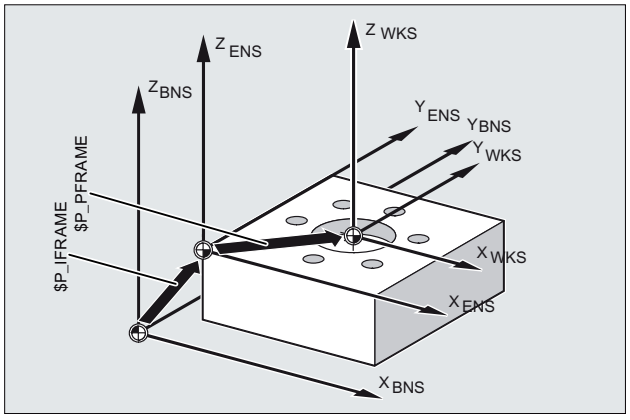


Active: Channel-specific programmable frame $P_PFRAME

The $P_PFRAME frame variable defines the reference between the settable zero system (SZS) and the workpiece coordinate system (WCS).

`$P_PFRAME` contains the resulting frame, that results

- **From the programming** of `TRANS/ATRANS`, `ROT/AROT`, `SCALE/ASCALE`, `MIRROR/AMIRROR` or
- **From the assignment** of `CTRANS`, `CROT`, `CMIRROR`, `CSCALE` to the programmed FRAME



Active: Total frame $P_ACTFRAME

The total frame active in the channel results from the chaining of all frames acting in the channel.

```
$P_ACTFRAME =    $P_PARTFRAME : $P_SETFRAME : $P_EXTFRAME :
                 $P_ISO1FRAME : $P_ISO2FRAME : $P_ISO3FRAME :
                 $P_ACTBFRAME : $P_IFRAME : $P_GFRAME :
                 $P_TOOLFRAME : $P_WPFRAME : $P_TRAFRAME :
                 $P_PFRAME    : $P_ISO4FRAME : $P_CYCFRAME
```

`$P_ACTFRAME` describes the currently valid workpiece coordinate system.

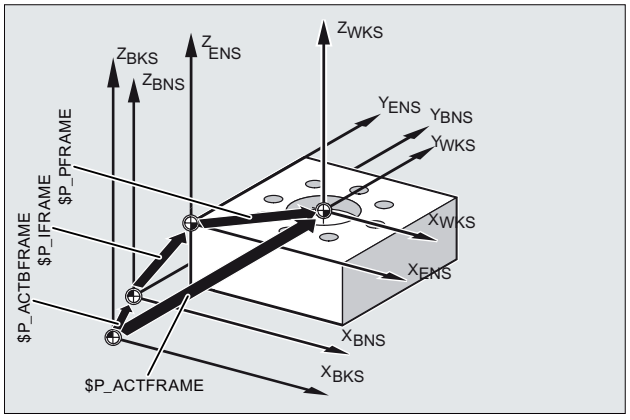
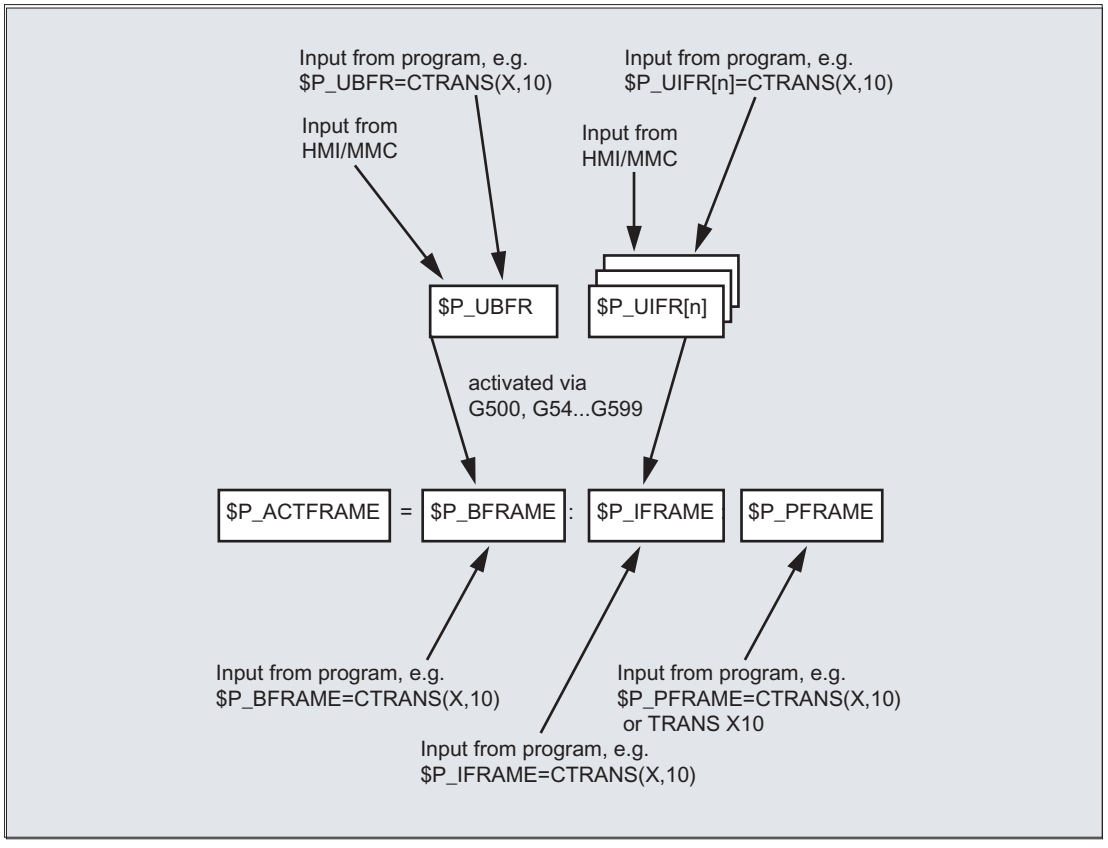


Figure 6-2 Frame variable $P_ACTFRAME

If one of the following frames `$P_BFRAME` / `$P_CHBFRAME[<n>]`, `$P_IFRAME` or `$P_PFRAME` is changed, the current total frame `$P_ACTFRAME` is recalculated.



Basic frame and settable frame are effective after Reset if MD 20110 RESET_MODE_MASK is set as follows:

Bit0=1, bit14=1 --> `$P_UBFR` (basic frame) acts

Bit0=1, bit5=1 --> `$P_UIFR [$P_UIFRNUM]`(settable frame) acts

Data storage: channel-specific base frames $P_CHBFR[<n>]

The frame variables $P_CHBFR[<n>] read/write the base frames in the data storage. The data storage frame is not immediately active in the channel when written. The written frame is activated with:

- Channel reset and MD20110 $MC_RESET_MODE_MASK, Bit0 == 1 and Bit14 == 1
- Command `G500`, `G54` ... `G57`, `G505` ... `G599` (activation/deactivation of base frames with subsequent recalculation of the current total frames)

Data storage: Channel-specific settable frames $P_UIFR[<n>]

The frame variables $P_UIFR[<n>] read/write the settable base frames in the data storage. The frame is not immediately active in the channel when written. The written frame in the channel is calculated with:

- `G500` command (deactivate all settable frames or zero offsets)
- `G54` ... `G57`, `G505` ... `G599` command (activate a settable frame or zero offset)

Active settable frame	Data storage frame	(corresponds to command)
`$P_IFRAME` =	`$P_UIFR[0]`	`G500`
	`$P_UIFR[1]`	`G54`
	`$P_UIFR[2]`	`G55`
	`$P_UIFR[3]`	`G56`
	`$P_UIFR[4]`	`G57`
	`$P_UIFR[5]`	`G505`
	`$P_UIFR[6]`	`G506`
	...	...
	`$P_UIFR[99]`	`G599`

6.2 Value assignments to frames

6.2.1 Assigning direct values (axis value, angle, scale)

You can directly assign values to frames or frame variables in the NC program.

Syntax

Syntax

```
$P_PFRAME = CTRANS(X, <offset value>, Y, <offset value>, Z, <offset value>, ...)
$P_PFRAME = ROT(X, <angle>, Y, <angle>, Z, <angle>, ...)
$P_UIFR[..] = CROT(X, <angle>, Y, <angle>, Z, <angle>,  ...
```

```
$P_PFRAME = CSCALE(X, <scale>, Y, <scale>, Z, <scale>, ...)
$P_PFRAME = CMIRROR(X, Y, Z)
```

The syntax for `$P_CHBFRAME[<n>]` is identical to `$P_PFRAME`.

Meaning

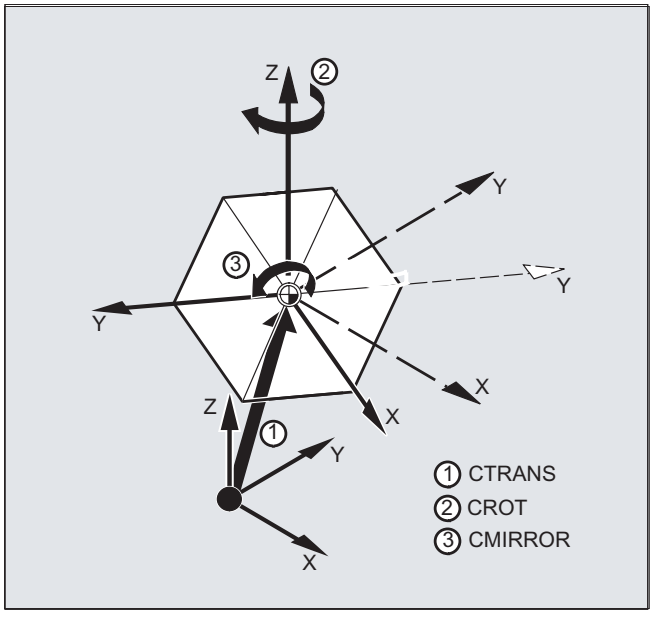
`CTRANS`:	Translation of specified axes
`CROT`:	Rotation around specified axes
`CSCALE`:	Scale change on specified axes
`CMIRROR`:	Direction reversal on specified axis
`X, Y, Z`:	Offset value in the direction of the specified geometry axis
`<offset value>`:	Offset value
`<angle>`:	The angle with the rotation
`<scale>`:	Scale value

Examples

Value assignments to frame components of the current programmable frame

Value assignment to the translation, rotation and mirror frame components of the current programmable frame:

```
$P_PFRAME = CTRANS(X,10,Y,20,Z,5) : CROT(Z,45) : CMIRROR(Y)
```



Writing the rotation components of a frame

Assignment of values to all three axes of the rotation component of the settable data storage frame `$P_UIFR` with `CROT` :

```
$P_UIFR[5] = CROT(X, 0, Y, 0, Z, 0)
```

Alternatively, the direct assignment of the individual values to the associated axis of the rotation component of the data storage frame:

```
$P_UIFR[5, Y, RT]=0
$P_UIFR[5, X, RT]=0
$P_UIFR[5, Z, RT]=0
```

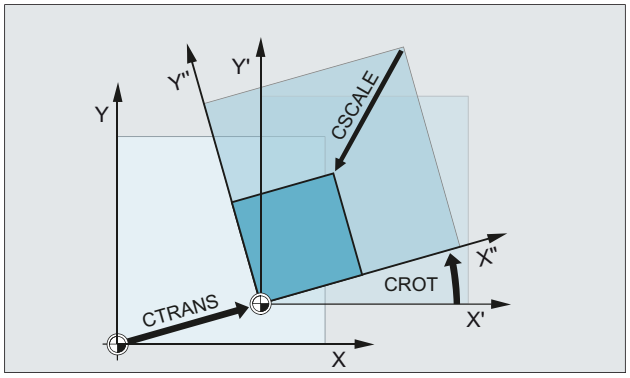
Description

The chaining operator : combines several operations on a frame with each other. The operations are processed successively from left to right.

Example

Chained operations on $P_PFRAME with offset, rotation and scaling:

```
$P_PFRAME = CTRANS(...) : CROT(...) : CSCALE...
```



6.2.2 Reading and changing frame components (TR, FI, RT, SC, MI)

This feature allows you to access **individual** data of a frame, e.g. a specific offset value or angle of rotation. You can modify these values or assign them to another variable.

Syntax

`R10=$P_UIFR[$P_UIFNUM,X,RT]`	Assign the angle of rotation RT around the X axis from the currently valid settable zero offset $P_UIFR-NUM to the variable R10.
`R12=$P_UIFR[25,Z,TR]`	Assign the offset value TR in Z from the data set of set frame no. 25 to the variable R12.
`R15=$P_PFRAME[Y,TR]`	Assign the offset value TR in Y of the current programmable frame to the variable R15.
`$P_PFRAME[X,TR] = 25`	Modify the offset value TR in X of the current programmable frame. X25 applies immediately.

Meaning

$P_UIFRNUM:	This command automatically establishes the reference to the currently valid settable zero offset.
P_UIFR[n,…,…]:	Specify the frame number n to access the settable frame no. n.
	Specify the component to be read or modified:
TR:	TR Translation
FI:	FI Translation Fine
RT:	RT Rotation
SC:	SC Scale scale modification
MI:	MI Mirroring
X, Y, Z:	The corresponding axis X, Y, Z is also specified (see examples).

Value range for RT rotation

Rotation around 1st geometry axis: -180° to +180°
Rotation around 2nd geometry axis: -90° to +90°
Rotation around 3rd geometry axis: -180° to +180°

Description

Calling frame

By specifying the system variable $P_UIFRNUM you can access the current zero offset set with $P_UIFR or G54, G55, ...
($P_UIFRNUM contains the number of the currently set frame).

All other stored settable $P_UIFR frames are called up by specifying the appropriate number $P_UIFR[n].

For predefined frame variables and user-defined frames, specify the name, e.g. $P_IFRAME.

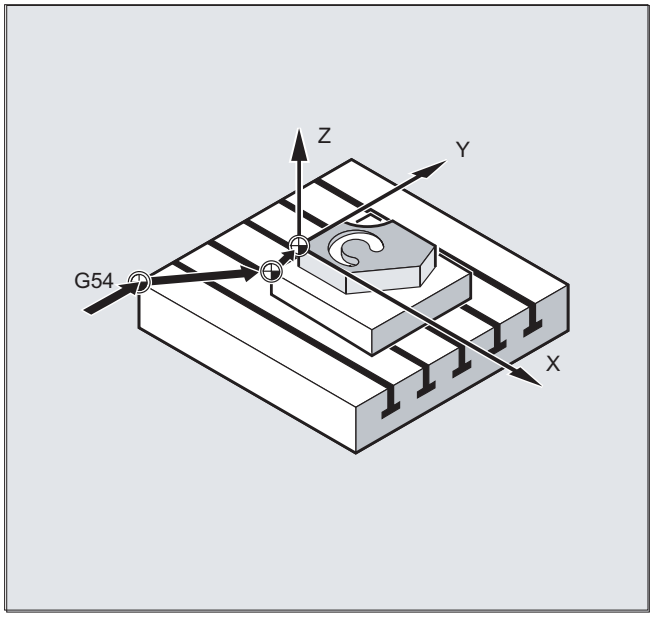
Calling data

The axis name and the frame component of the value you want to access or modify are written in square brackets, e.g. [X, RT] or [Z, MI].

6.2.3 Calculating with frames

A frame can be assigned to another frame or frames can be chained to each other in the NC program.

Frame chainings are suitable for the description of several workpieces, arranged on a pallet, which are to be machined in the same process.



The frame components can only contain intermediate values for the description of pallet tasks. These are chained to generate various workpiece zeroes.

Examples

Assignments

Program code	**Comment**
`DEF FRAME SETTING_1`	`; Definition of a local frame variable`
`SETTING_1 = CTRANS(X,10)`	`; Assignment of the function result to the frame variable`
`$P_PFRAME = SETTING_1`	`; Assignment of the frame variable to the current frame`
`DEF FRAME SETTING_4`	`; Definition of a local frame variable`
`SETTING_4 = $P_PFRAME`	`; Buffer the current frame in the frame variable`
`...`	
`$P_PFRAME = SETTING_4`	`; Fetch the current frame from the frame variable`

Chainings

The operator : chains frames with each other in the programmed sequence. The frame components, such as offsets and rotations, are executed successively additive.

Program code	Comment
$P_IFRAME = $P_UIFR[15] : $P_UIFR[16]	; Assignment of the result frame from the chaining of the ; two settable data storage frames on the active ; settable total frame. ; Application example: ; $P_UIFR[15]: Offset ; $P_UIFR[16]: Rotation
$P_UIFR[3] = $P_UIFR[4] : $P_UIFR[5]	; Assignment of the result frame from the chaining of the ; two settable data storage frames on a ; different settable data storage frame

6.2.4 Definition of frame variables (DEF FRAME)

In addition to the predefined frame variables, user frame variables can also be defined. The user-defined frame variables are user variables of type FRAME. The name of the frame can be assigned freely in accordance with the rules for user variables.

The CTRANS, CROT, CSCALE and CMIRROR functions assign values to user-defined frame variables.

Syntax

```
DEF FRAME <name>
```

Meaning

DEF FRAME:	Define user variable of the type FRAME.
<name>:	Name of the frame variable

Example

Definition of a "PALETTE" frame variable and the assignment of offset and rotation values:

Program code	Comment
DEF FRAME PALETTE	; Define PALETTE frame variable
PALETTE = CTRANS(...) : CROT(...)	; Assignment of the result frame of the chaining for ; offset and rotation on the PALETTE frame variable

6.3 Coarse and fine offsets (CTRANS, CFINE)

Fine offset

A fine offset `CFINE(...)` can be applied to the following frames:

- Settable frames: $P_UIFR or $P_IFRAME
- Basic frames: $P_NCBFR[<n>], $P_CHBFR[<n>], $P_CHBFRAMES[<n>] or $P_ACTBFRAME
- Programmable frame: $P_PFRAME

The fine offset of a frame is programmed with the `CFINE(...)` command.

Coarse offset

A coarse offset `CTRANS(...)` can be applied to all frames.

Total offset

The total offset results from the addition of the coarse and the fine offset.

Machine data

Enable of the fine offset

The fine offset is enabled with the machine data:

MD18600 $MN_MM_FRAME_FINE_TRANS = 1

Syntax

Fine offset

- Complete frame
  - `<frame> = CFINE(<K_1>,<value>)`
  - `<frame> = CFINE(<K_1>,<value>, <K_2>, <value>)`
  - `<frame> = CFINE(<K_1>,<value>, <K_2>, <value>, <K_3>, <value>)`
- Frame component
  - `<frame>[<n>, <K_1>, FI] = <value>`

Coarse offset

- Complete frame
  - `<frame> = CTRANS(<K_1>,<value>)`
  - `<frame> = CTRANS(<K_1>,<value>, <K_2,<value>)`
  - `<frame> = CTRANS(<K_1>,<value>, <K_2,<value>, <K_3,<value>)`
- Frame component
  - `<frame>[<n>,<K_1>,TR] = <value>`

In particular for the programmable frame $P_PFRAME:

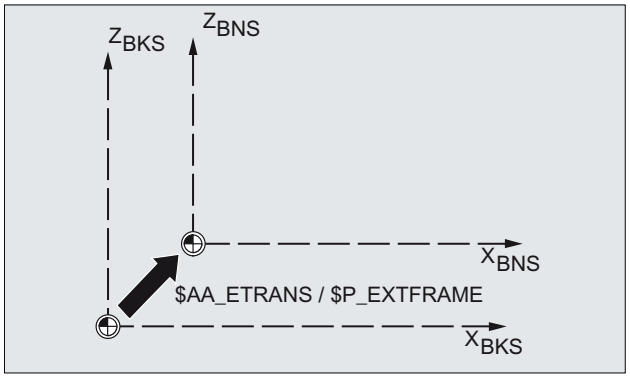
- TRANS <K_1> <value>
- TRANS <K_1> <value> <K_2> <value>
- TRANS <K_1> <value> <K_2> <value> <K_3> <value>

Meaning

<Frame>:	Frame, e.g. settable frame of the data storage $P_UIFR[<n>]
CFINE:	Fine offset, additive offset.
CTRANS:	Coarse offset, absolute offset.
TRANS:	Only programmable frame: Coarse offset, absolute offset.
<K_n>:	Coordinate axes X, Y, Z
<value>:	Offset value

6.4 External zero offset ($AA_ETRANS)

The external zero offset is a linear offset between the base coordinate system (BCS) and the basic origin system (BOS).



The external zero offset with $AA_ETRANS acts in two ways depending on the machine data parameterization:

1. After activation by the NC/PLC interface signal, the system variable $AA_ETRANS acts directly as offset value
2. After activation by the NC/PLC interface signal, the value of the system variable $AA_ETRANS is transferred to the active system frames $P:EXTFRAME and the data storage frame $P_EXTFR. The active total frame $P_ACTFRAME is then recalculated.

Machine data

In conjunction with the system variable $AA_ETRANS, a differentiation is made between two procedures selected with the following machine data:

MD28082 $MC_MM_SYSTEM_FRAME_MASK,Bit1 = <value>

<value>	Meaning
0	Function: $AA_ETRANS[<axis>] written directly by PLC, HMI or NC program. Enable for retraction of the zero offset for $AA_ETRANS[<axis>] in the next possible traversing block: DB31, ... DBX3.0
1	Function: Activation of the active system frame $P:EXTFRAME and the data storage frame $P_EXTFR Enable for retraction of the zero offset for $AA_ETRANS[<axis>] by: DB31, ... DBX3.0. The following is performed in the channel: • Stop all traversal movements in the channel (other than command and PLC axes) • Preprocessing stop with subsequent reorganization (STOPRE) • Coarse offset of active frame $P_EXTFRAME[<axis>] = $AA_ETRANS[<axis>] • Coarse offset of data storage frame $P_EXTFR[<axis>] = $AA_ETRANS[<axis>] • Recalculation of the active total frame $P_ACTFRAME • Retraction of the offset in the programmed axes. • Continuation of the interrupted traversing motion or of the NC program

Programming

- Syntax
  `$AA_ETRANS[<axis>] = <value>`
- Meaning

`$AA_ETRANS`:	System variable for buffering the external zero offset
`<axis>`:	Channel axis
`<value>`:	Offset value

NC/PLC interface signal

DB31, ... DBX3.0 = 0 → 1 ⇒ $P_EXTFRAME[<axis>] = $P_EXTFR[<axis>] = $AA_ETRANS[<axis>]

6.5 Set actual value with loss of the referencing status (PRESETON)

The `PRESETON()` procedure sets for one or more axes a new actual value in the machine coordinate system (MCS). This corresponds to a zero offset of the MCS of the axis. This does not cause the axis to be traversed.

`PRESETON` initiates a preprocessing stop with synchronization. The actual position is assigned to the axis only at standstill.

If the axis for `PRESETON` is not assigned to the channel, the further procedure depends on the axis-specific configuring of the axis replacement behavior:

MD30552 $MA_AUTO_GET_TYPE

Referencing status

By setting a new actual value in the machine coordinate system, the referencing status of the machine axis is reset:

DB31, ... DBX60.4/.5 = 0 (referenced / synchronized measuring system 1/2)

For this reason it is recommended that PRESETON only be used for axes that do not require a reference point.

To restore the original machine coordinate system, the measuring system of the machine axis must be referenced again, e.g. through active referencing from the part program (G74).

⚠ CAUTION

Loss of the referencing status

The setting of a new actual value in the machine coordinate system with PRESETON resets the referencing status of the machine axis to "not referenced / synchronized".

Programming

Syntax

```
PRESETON(<axis_1>, <value_1> [, <axis_2>, <value_2>, ... <axis_8>,
<value_8>])
```

Meaning

PRESETON:	Set actual value with loss of the referencing status	
	Preprocessing stop:	yes
	Alone in the block:	yes
<axis_x>:	Machine axis name	
	Type:	AXIS
	Range of values:	Machine axis names defined in the channel
<value_x>:	New actual value of the machine axis in the machine coordinate system (MCS) The input is made in the currently valid measuring system (inch/metric) An active diameter programming (DIAMON) is considered	
	Type:	REAL

References

PRESETONS in NC programs

A detailed description of PRESETON in NC programs is contained in:

Function Manual Basic Functions, Chapter "K2: Axes, coordinate systems, frames" > "Coordinate systems" > "Machine coordinate system (MCS)" > "Set actual value with loss of the referencing status (PRESETON)"

PRESETONS in synchronous actions

A detailed description of PRESETON in synchronous actions is contained in:

Function Manual, Synchronized Actions; Section: "Detailed description" > "Actions in synchronous actions" > "Set actual value with loss of the referencing status (PRESETON)"

6.6 Set actual value without loss of the referencing status (PRESETONS)

The PRESETONS() procedure sets for one or more axes a new actual value in the machine coordinate system (MCS). This corresponds to a zero offset of the MCS of the axis. This does not cause the axis to be traversed.

PRESETONS initiates a preprocessing stop with synchronization. The actual position is assigned to the axis only at standstill.

If the axis for PRESETONS is not assigned to the channel, the further procedure depends on the axis-specific configuring of the axis replacement behavior:

MD30552 $MA_AUTO_GET_TYPE

Referencing status

The setting of a new actual value in the machine coordinate system (MCS) with PRESETONS does **not** change the referencing status of the machine axis.

Requirements

- **Encoder type**
  PRESETONS is possible only for the following encoder types of the active measuring system:
  - MD30240 $MA_ENC_TYPE[<measuring system>] = 0 (simulated encoder)
  - MD30240 $MA_ENC_TYPE[<measuring system>] = 1 (raw signal encoder)
- **Referencing mode**
  PRESETONS is possible only for the following referencing modes of the active measuring system:
  - MD34200 $MA_ENC_REFP_MODE[<measuring system>] = 0 (no reference point approach possible)
  - MD34200 $MA_ENC_REFP_MODE[<measuring system>] = 1 (referencing for incremental, rotary or linear measuring systems: Zero pulse on the encoder track)

Programming

Syntax

```
PRESETONS(<axis_1>, <value_1> [, <axis_2>, <value_2>, ... <axis_8>,
<value_8>])
```

Meaning

PRESETONS:	Set actual value without loss of the referencing status	
	Preprocessing stop:	yes
	Alone in the block:	yes
<axis_x>:	Machine axis name	
	Type:	AXIS
	Range of values:	Machine axis names defined in the channel

<value_x>:	New current actual value of the machine axis in the machine coordinate system (MCS) The input is made in the active measuring system (inch/metric) An active diameter programming (DIAMON) is considered	
	Type:	REAL

References

PRESETONS in NC programs

A detailed description of PRESETONS in NC programs is contained in:

Function Manual Basic Functions, Chapter "K2: Axes, coordinate systems, frames" > "Coordinate systems" > "Machine coordinate system (MCS)" > "Set actual value without loss of the referencing status (PRESETONS)"

PRESETONS in synchronous actions

A detailed description of PRESETONS in synchronous actions is contained in:

Function Manual, Synchronized Actions; Section: "Detailed description" > "Actions in synchronous actions" > "Set actual value without loss of the referencing status (PRESETONS)"

6.7 Frame calculation from three measuring points in space (MEAFRAME)

The MEAFRAME function is used to support measuring cycles. It calculates the frame from three ideal points and the corresponding measured points.

When a workpiece is positioned for machining, its position relative to the Cartesian machine coordinate system is generally both offset and rotated in relation to its ideal position. For exact machining or measuring either a costly physical adjustment of the part is required or the motions defined in the part program must be changed.

A frame can be defined by sampling three points in space whose ideal positions are known. A touch-trigger probe or optical sensor is used for sampling that touches special holes precisely fixed on the supporting plate or probe balls.

Syntax

```
MEAFRAME(<ideal points>,<measuring points>,<quality>)
```

Meaning

MEAFRAME:	Function call
<ideal points>:	2-dim. REAL array containing the three coordinates of the ideal points
<measuring points>:	2-dim. REAL array containing the three coordinates of the measured points

<quality>:	Variable with which information on the quality of the FRAME calculation is returned		
	Type:	VAR REAL	
	Value:	-1	The ideal points are almost on a straight line: The frame could not be calculated. The returned FRAME variable contains a neutral frame.
		-2	The measuring points are almost on a straight line: The frame could not be calculated. The returned FRAME variable contains a neutral frame.
		-4	The calculation of the rotation matrix failed for a different reason.
		≥ 0.0	Sum of distortions (distances between the points), that are required to transform the measured triangle into a triangle that is congruent to the ideal triangle.

Note

Quality of the measurement

In order to map the measured coordinates onto the ideal coordinates using a rotation and a translation, the triangle formed by the measured points must be congruent to the ideal triangle. This is achieved by means of a compensation algorithm that minimizes the sum of squared deviations needed to reshape the measured triangle into the ideal triangle.

Since the effective distortion can be used to judge the quality of the measurement, MEAFRAME returns it as an additional variable.

Note

The frame created by MEAFRAME can be transformed by the ADDFRAME function into another frame in the frame chain (see example "Chaining with ADDFRAME").

Examples

Example 1:

Part program 1:

```
Program code
...
DEF FRAME CORR_FRAME
```

Setting measuring points:

```
Program code                                          Comment
DEF REAL IDEAL_POINT[3,3]=
SET(10.0,0.0,0.0,0.0,10.0,0.0,0.0,0.0,10.0)
DEF REAL MEAS_POINT[3,3]=                             ; For test.
SET(10.1,0.2,-0.2,-0.2,10.2,0.1,-0.2,0.2,9.8)
```

Program code	Comment
DEF REAL FIT_QUALITY=0	
DEF REAL ROT_FRAME_LIMIT=5	; Permits max. five degree rotation of the part position.
DEF REAL FIT_QUALITY_LIMIT=3	; Permits max. three mm offset between the ideal and the measured triangle.
DEF REAL SHOW_MCS_POS1[3]	
DEF REAL SHOW_MCS_POS2[3]	
DEF REAL SHOW_MCS_POS3[3]	

Program code	Comment
N100 G01 G90 F5000	
N110 X0 Y0 Z0	
N200 CORR_FRAME=MEAFRAME(IDEAL_POINT,MEAS_POINT,FIT_QUALITY)	
N230 IF FIT_QUALITY < 0	
SETAL(65000)	
GOTOF NO_FRAME	
ENDIF	
N240 IF FIT_QUALITY > FIT_QUALITY_LIMIT	
SETAL(65010)	
GOTOF NO_FRAME	
ENDIF	
N250 IF CORR_FRAME[X,RT] > ROT_FRAME_LIMIT	; Limiting the 1st RPY angle.
SETAL(65020)	
GOTOF NO_FRAME	
ENDIF	
N260 IF CORR_FRAME[Y,RT] > ROT_FRAME_LIMIT	; Limiting the 2nd RPY angle.
SETAL(65021)	
GOTOF NO_FRAME	
ENDIF	
N270 IF CORR_FRAME[Z,RT] > ROT_FRAME_LIMIT	; Limiting the 3rd RPY angle.
SETAL(65022)	
GOTOF NO_FRAME	
ENDIF	
N300 $P_IFRAME=CORR_FRAME	; Activate sample frame with settable frame.
	; Check frame by positioning the geometry axes to the ideal point.
N400 X=IDEAL_POINT[0,0] Y=IDEAL_POINT[0,1] Z=IDEAL_POINT[0,2]	
N410 SHOW_MCS_POS1[0]=$AA_IM[X]	
N420 SHOW_MCS_POS1[1]=$AA_IM[Y]	
N430 SHOW_MCS_POS1[2]=$AA_IM[Z]	

```
Program code                                          Comment
N500 X=IDEAL_POINT[1,0] Y=IDEAL_POINT[1,1] Z=IDEAL_POINT[1,2]
N510 SHOW_MCS_POS2[0]=$AA_IM[X]
N520 SHOW_MCS_POS2[1]=$AA_IM[Y]
N530 SHOW_MCS_POS2[2]=$AA_IM[Z]
N600 X=IDEAL_POINT[2,0] Y=IDEAL_POINT[2,1] Z=IDEAL_POINT[2,2]
N610 SHOW_MCS_POS3[0]=$AA_IM[X]
N620 SHOW_MCS_POS3[1]=$AA_IM[Y]
N630 SHOW_MCS_POS3[2]=$AA_IM[Z]
N700 G500                                             ; Deactivate settable frame as
                                                      with zero frame (no value entered,
                                                      pre-assigned).
No_FRAME                                              ; Deactivate settable frame, as
                                                      pre-assigned with zero frame (no
                                                      value entered).
M0
M30
```

Example 2: Chaining of frames

Chaining of MEAFRAME for offsets

The MEAFRAME function returns an offset frame. If this offset frame is chained to the settable frame $P_UIFR[1] that was active during the call of the function (e.g. G54), a settable frame is provided for further conversions for the traversing or machining.

Chaining with ADDFRAME

If you want this offset frame in the frame chain to apply at a different position or if other frames are active before the settable frame, the ADDFRAME function can be used for chaining into one of the channel basic frames or a system frame.

The following must not be active in the frames:

- Mirroring with MIRROR
- Scaling with SCALE

The input parameters for the setpoints and actual values are the workpiece coordinates. These coordinates must always be specified metrically or in inches (G71/G70) and radius-related (DIAMOF) in the basic system of the control.

References:
For further information on ADDFRAME, see:
Function Manual, Basic Functions; K2: Axis Types, Coordinate Systems, Frames

6.8 NCU global frames

Only one set of NCU global frames is used for all channels on each NCU. NCU global frames can be read and written from all channels. The NCU global frames are activated in the respective channel.

Channel axes and machine axes with offsets can be scaled and mirrored by means of global frames.

Geometrical relationships and frame chains

With global frames there is no geometrical relationship between the axes. It is therefore not possible to perform rotations or program geometry axis identifiers.

- Rotations cannot be used on global frames. The programming of a rotation is denied with alarm: "18310 Channel %1 Block %2 Frame: rotation not allowed" is displayed.
- It is possible to chain global frames and channel-specific frames. The resulting frame contains all frame components including the rotations for all axes. The assignment of a frame with rotation components to a global frame is denied with alarm "Frame: rotation not allowed".

NCU global frames

NCU global basic frames $P_NCBFR[n]

Up to eight NCU global basic frames can be configured:

Channel-specific basic frames can also be available.

Global frames can be read and written from all channels of an NCU. When writing global frames, the user must ensure channel coordination. This can be implemented, for example, through wait markers (WAITMC).

Machine manufacturer

The number of global basic frames is configured via the machine data.

References:
Function Manual, Basic Functions; Axes, Coordinate Systems, Frames (K2)

NCU global settable frames $P_UIFR[n]

All settable frames G500, G54...G599 can be configured NCU-globally or channel-specifically.

Machine manufacturer

All settable frames can be reconfigured as global frames with the aid of machine data MD18601 $MN_MM_NUM_GLOBAL_USER_FRAMES.

Channel axis identifiers and machine axis identifiers can be used as axis identifiers in frame program commands. Programming the geometry identifiers is rejected with an alarm.

6.8.1 Channel-specific frames ($P_CHBFR, $P_UBFR)

Settable frames or basic frames can be read and written via the part program and via the OPI by the operator and by the PLC.

The fine offset can also be used for global frames. Suppression of global frames also takes place, as is the case with channel-specific frames, via G53, G153, SUPA and G500.

Machine manufacturer

The number of basic frames can be configured in the channel via the machine data MD28081 $MC_MM_NUM_BASE_FRAMES. The standard configuration is designed for at least one basic frame per channel. A maximum of eight basic frames are supported per channel. In addition to the eight basic frames, there can also be eight NCU global basic frames in the channel.

Channel-specific frames

$P_CHBFR[n]

System variable $P_CHBFR[n] can be used to read and write the basic frames. When a basic frame is written, the chained total basic frame is not activated until the execution of a G500, G54...G599 statement. The variable is used primarily for storing write operations to the basic frame on HMI or PLC. These frame variables are saved by the data backup.

First basic frame in the channel

The basic frame with array index 0 is not activated simultaneously when writing to the predefined $P_UBFR variable, but rather activation only takes place on execution of a G500, G54...G599 statement. The variable can also be read and written in the program.

$P_UBFR

$P_UBFR is identical to $P_CHBFR[0]. One basic frame always exists in the channel by default, so that the system variable is compatible with older versions. If there is no channel-specific basic frame, an alarm is issued at read/write: "Frame: statement not permissible".

6.8.2 Frames active in the channel

Frames active in the channel are entered from the part program via the relevant system variables of these frames. This also includes system frames. The current system frame can be read and written in the part program via these system variables.

Frames currently active in the channel

Overview

Current system frames	For:
$P_PARTFRAME	TCARR and PAROT
$P_SETFRAME	Preset actual value memory and scratching
$P_EXTFRAME	External zero offset
$P_NCBFRAME[n]	Current NCU global basic frames
$P_CHBFRAME[n]	Current channel basic frames
$P_BFRAME	Current first basic frame in the channel
$P_ACTBFRAME	Complete basic frame
$P_CHBFRMASK and $P_NCBFRMASK	Complete basic frame
$P_IFRAME	Current settable frame

Current system frames	For:
$P_TOOLFRAME	TOROT and TOFRAME
$P_WPFRAME	Workpiece reference points
$P_TRAFRAME	Transformations
$P_PFRAME	Current programmable frame
Current system frame	For:
$P_CYCFRAME	Cycles
P_ACTFRAME	Current total frame
FRAME chaining	Current frame is made up of the complete basic frame

$P_NCBFRAME [n] Current NCU global basic frames

System variable $P_NCBFRAME[n] can be used to read and write the current global basic frame field elements. The resulting total basic frame is calculated by means of the write process in the channel.

The modified frame is activated only in the channel in which the frame was programmed. If the frame is to be modified for all channels of an NCU, $P_NCBFR[n] and $P_NCBFRAME[n] must be written simultaneously. The other channels must then activate the frame, e.g. with G54. Whenever a basic frame is written, the complete basic frame is calculated again.

$P_CHBFRAME[n] Current channel basic frames

System variable $P_CHBFRAME[n] can be used to read and write the current channel basic frame field elements. The resulting complete basic frame is calculated by means of the write process in the channel. Whenever a basic frame is written, the complete basic frame is calculated again.

$P_BFRAME Current first basic frame in the channel

The predefined frame variable $P_BFRAME can be used to read and write the current basic frame with the array index 0, which is valid in the channel, in the part program. The written basic frame is immediately included in the calculation.

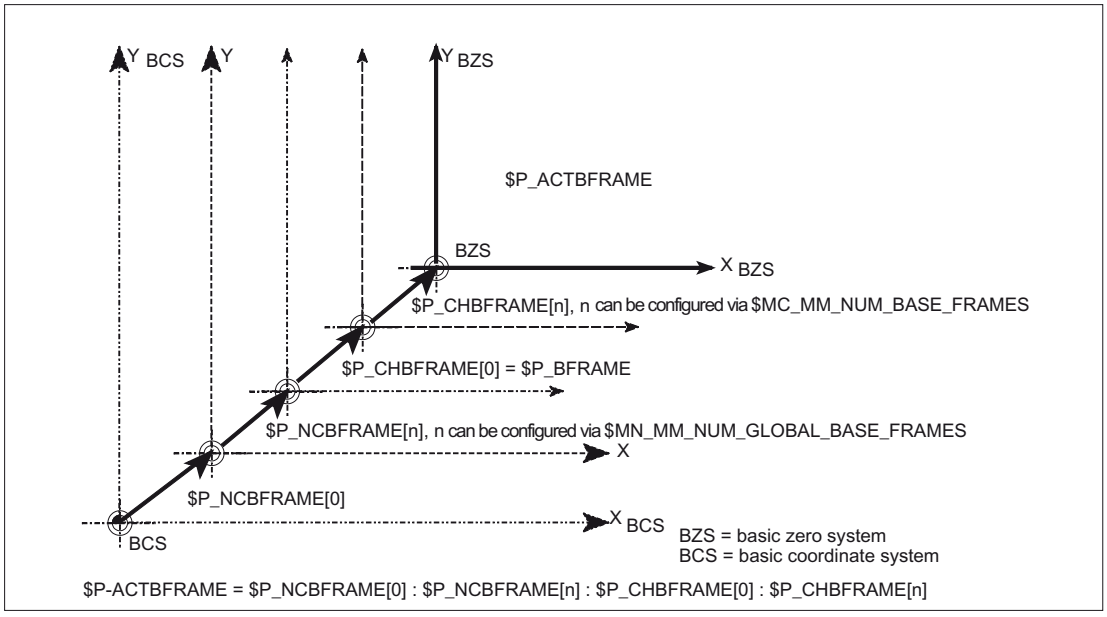
$P_BFRAME is identical to $P_CHBFRAME[0]. The system variable always has a valid default value. If there is no channel-specific basic frame, an alarm is issued at read/write: "Frame: statement not permissible".

$P_ACTBFRAME Complete basic frame

The $P_ACTFRAME variable determines the chained complete basic frame. The variable is read-only.

$P_ACTFRAME corresponds to:

$P_NCBFRAME[0] : ... : $P_NCBFRAME[n] : $P_CHBFRAME[0] : ... : $P_CHBFRAME[n].



$P_CHBFRMASK and $P_NCBFRMASK Complete basic frame

The user can select which basic frames are to be included in the calculation of the "Complete" basic frame via the system variables $P_CHBFRMASK and $P_NCBFRMASK. The variables can only be programmed in the program and read via the OPI. The value of the variable is interpreted as a bit mask and specifies which basic frame field element of $P_ACTFRAME is to be included in the calculation.

$P_CHBFRMASK can be used to specify which channel-specific basic frames and $P_NCBFRMASK can be used to specify which NCU global basic frames are to be included in the calculation.

The complete basic frame and the complete frame are recalculated with the programming of the variables. After a reset and in the basic setting, the values of $P_CHBFRMASK and $P_NCBFRMASK are as follows:

$P_CHBFRMASK = $MC_CHBFRAME_RESET_MASK

$P_NCBFRMASK = $MC_CHBFRAME_RESET_MASK

Example:

```
$P_NCBFRMASK = 'H81' ;$P_NCBFRAME[0] : $P_NCBFRAME[7]
$P_CHBFRMASK = 'H11' ;$P_CHBFRAME[0] : $P_CHBFRAME[4]
```

$P_IFRAME Current settable frame

The predefined frame variable $P_IFRAME can be used to read and write the current settable frame, which is valid in the channel, in the part program. The written settable frame is immediately included in the calculation.

In the case of NCU global settable frames, the modified frame acts only in the channel in which the frame was programmed. If the frame is to be modified for all channels of an NCU, $P_UIFR[n] and $P_IFRAME must be written simultaneously. The other channels must then activate the corresponding frame, e.g. with G54.

$P_PFRAME Current programmable frame

$P_PFRAME is the programmable frame that results from the programming of TRANS/ ATRANS, G58/G59, ROT/AROT, SCALE/ASCALE, MIRROR/AMIRROR or from the assignment of CTRANS, CROT, CMIRROR, CSCALE to the programmable frame.

Current, programmable frame variable that establishes the reference between the settable zero system (SZS) and the workpiece coordinate system (WCS).

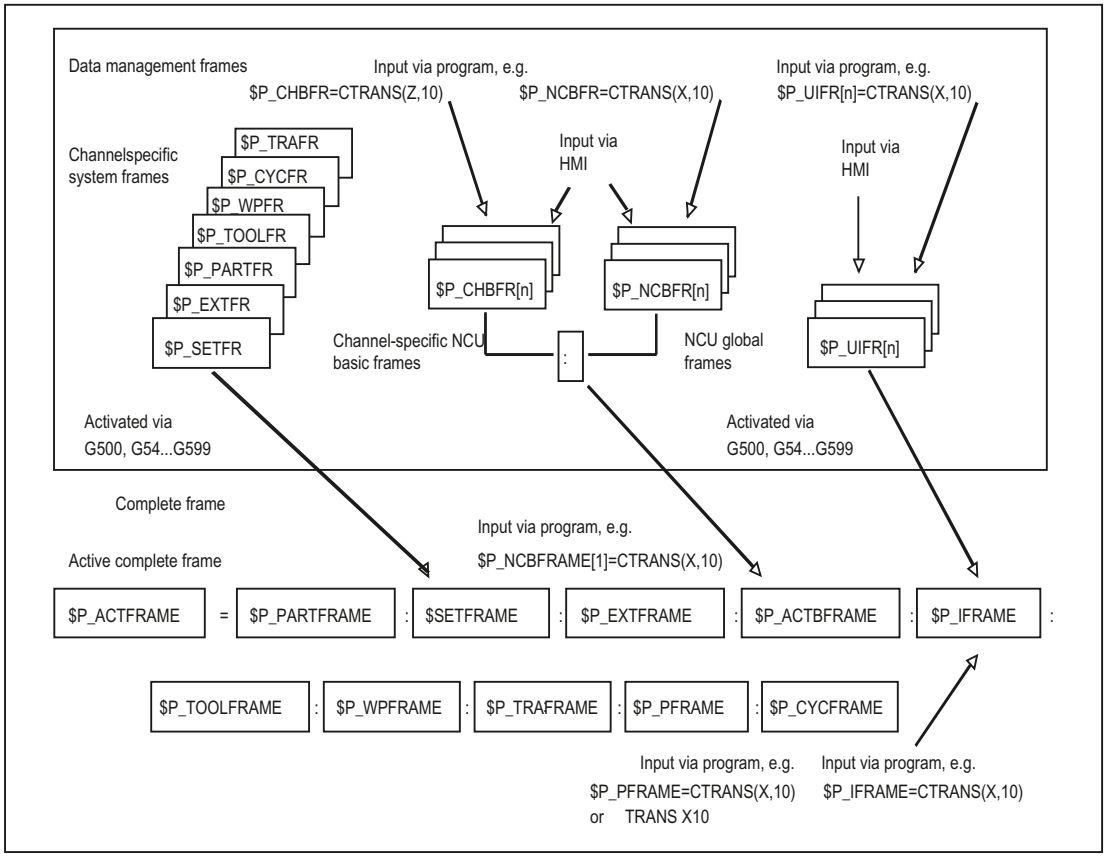
P_ACTFRAME Current complete frame

The resulting current complete frame $P_ACTFRAME is now a chain of all basic frames, the current settable frame and the programmable frame. The current frame is always updated whenever a frame component is changed.

`$P_ACTFRAME` corresponds to:

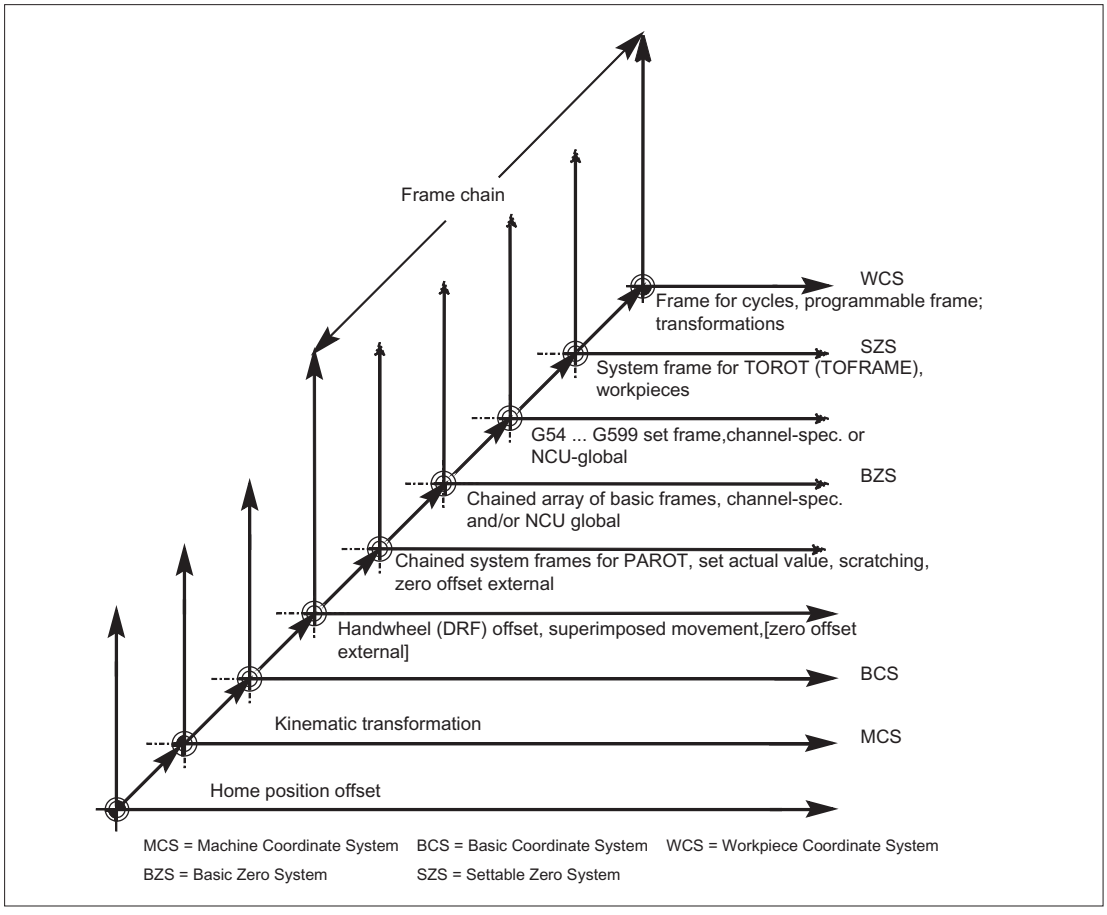
`$P_PARTFRAME : $P_SETFRAME : $P_EXTFRAME : $P_ACTBFRAME : $P_IFRAME :`

`$P_TOOLFRAME : $P_WPFRAME : $P_TRAFRAME : $P_PFRAME : $P_CYCFRAME`



Frame chaining

The current frame is composed of the complete basic frame, the settable frame, the system frame and the programmable frame in accordance with the current complete frame specified above.



Transformations 7

7.1 General programming of transformation types

General function

You can choose to program transformation types with suitable parameters in order to adapt the controller to various machine kinematics. These parameters can be used to declare both the orientation of the tool in space and the orientation movements of the rotary axes accordingly for the selected transformation.

In three-, four-, and five-axis transformations, the programmed positional data always relates to the tip of the tool, which is tracked orthogonally to the machined surface in space. The Cartesian coordinates are converted from the basic coordinate system to the machine coordinate system and relate to the geometry axes. These describe the operating point. Virtual rotary axes describe the orientations of the tool in space and are programmed with TRAORI.

In the case of kinematic transformation, positions can be programmed in the Cartesian coordinate system. The controller maps the Cartesian coordinate system traversing movements programmed with TRANSMIT, TRACYL and TRAANG to the traversing movements of the real machine axes.

Programming

Three, four and five axis transformations (TRAORI)

The orientation transformation declared is activated with the TRAORI command and the three possible parameters for transformation number, orientation vector and rotary axis offsets.

```
TRAORI(transformation number, orientation vector, rotary axis
offsets)
```

Kinematic transformations

`TRANSMIT(transformation number)` declared transformations are examples of kinematic transformation.

```
TRACYL(working diameter, transformation number)
```

```
TRAANG(angle of offset axis, transformation number)
```

Deactivate active transformation

`TRAFOOF` can be used to deactivate the currently active transformation.

Orientation transformation

Three, four and five axis transformations (TRAORI)

For the optimum machining of surfaces configured in space in the working area of the machine, machine tools require other axes in addition to the three linear axes X, Y and Z. The additional

axes describe the orientation in space and are called orientation axes in subsequent sections. They are available as rotary axes on four types of machine with varying kinematics.

1. Two-axis swivel head, e.g. cardanic tool head with one rotary axis parallel to a linear axis on a fixed tool table.
2. Two-axis rotary table, e.g. fixed swivel head with tool table, which can rotate about two axes.
3. Single-axis swivel head and single-axis rotary table, e.g. one rotatable swivel head with rotated tool for tool table, which can rotate about one axis.
4. Two-axis swivel head and single-axis rotary table, e.g. on tool table, which can rotate about one axis, and one rotatable swivel head with tool, which can rotate about itself.

3- and 4-axis transformations are special types of 5-axis transformation and are programmed in the same way as 5-axis transformations.

The functional scope of "**generic 3-/4-/5-/6-axis transformation**" is suitable both for transformations for orthogonal rotary axes and transformations for the universal milling head and, like all other orientation transformations, can also be activated for these four machine types with TRAORI. In generic 5-/6-axis transformation, tool orientation has an additional third degree of freedom, whereby the tool can be rotated about its own axis relative to the tool direction so that it can be directed as required in space.

References: /FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformation (F2)

Initial tool orientation setting regardless of kinematics

ORIRESET

If an orientation transformation is active using TRAORI, then ORIRESET can be used to specify the initial settings of up to 3 orientation axes with the optional parameters A, B, C. The order in which the programmed parameters are assigned to the round axes depends on the orientation axis order defined by the transformation. Programming ORIRESET(A, B, C) results in the orientation axes moving in linear and synchronous motion from their current position to the specified initial setting position.

Kinematic transformations

TRANSMIT and TRACYL

For milling on turning machines, either

1. Face machining in the turning clamp with TRANSMIT or
2. Machining of grooves with any path on cylindrical bodies with TRACYL

can be programmed for the transformation declared.

TRAANG

If the option of setting the infeed axis for inclined infeed is required (for grinding technology, for example), TRAANG can be used to program a configurable angle for the transformation declared.

Cartesian PTP travel

Kinematic transformation also includes the so-called "Cartesian PTP travel" for which up to 8 different articulated joint positions STAT= can be programmed. Although the positions are programmed in a Cartesian coordinate system, the movement of the machine occurs in the machine coordinates.

References:
/FB2/ Function Manual, Extended Functions; Kinematic Transformation (M1)

Chained transformations

Two transformations can be switched one after the other. For the second transformation chained here, the motion parts for the axes are taken from the first transformation.

The first transformation can be:

- Orientation transformation TRAORI
- Polar transformation TRANSMIT
- Cylinder transformation TRACYL
- Inclined axis transformation TRAANG

The second transformation must be a TRAANG type transformation for an inclined axis.

7.1.1 Orientation movements for transformations

Travel movements and orientation movements

The traversing movements of the programmed orientations are determined primarily by the type of machine. For three-, four-, and five-axis type transformations with TRAORI, the rotary axes or pivoting linear axes describe the orientation movements of the tool.

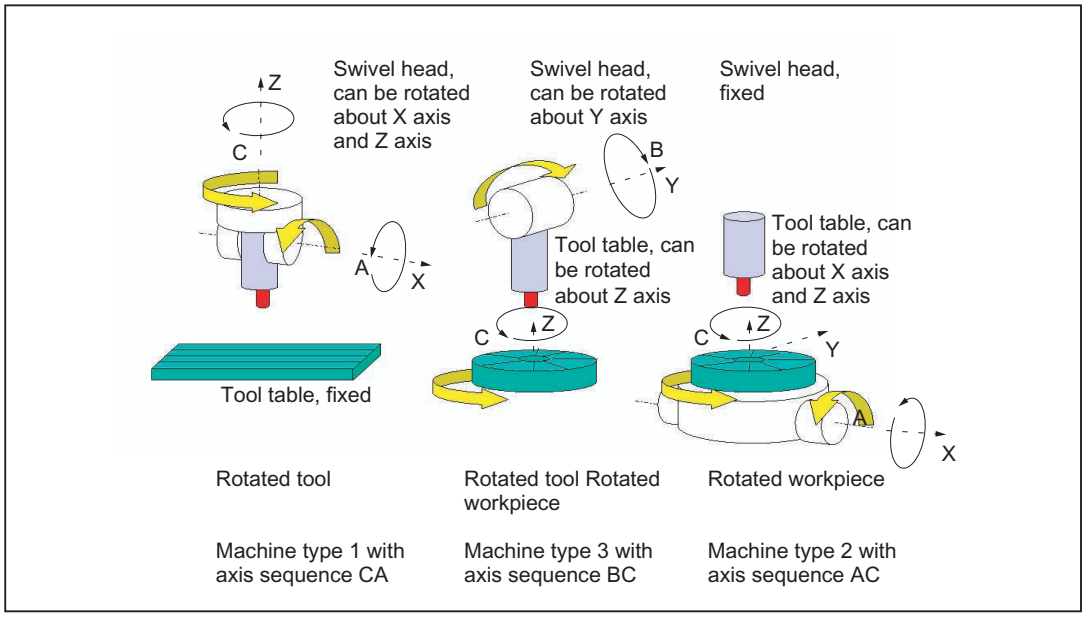
Changes in the position of the rotary axes involved in the orientation transformation will induce compensating movements on the remaining machine axes. The position of the tool tip remains unchanged.

Orientation movements of the tool can be programmed using the rotary axis identifiers A…, B…, C… of the virtual axes as appropriate for the application either by entering Euler or RPY angles or directional or surface normal vectors, normalized vectors for the rotary axis of a taper or for intermediate orientation on the peripheral surface of a taper.

In the case of kinematic transformation with TRANSMIT, TRACYL and TRAANG, the controller maps the programmed Cartesian coordinate system traversing movements to the traversing movements of the real machine axes.

Machine kinematics for three, four and five axis transformation (TRAORI)

Either the tool or the tool table can be rotatable with up to two rotary axes. A combination of swivel head and rotary table (single-axis in each case) is also possible.



Machine type	Programming of orientation
Three-axis transformation machine types 1 and 2	Programming of tool orientation only in the plane, which is perpendicular to the rotary axis. There are **two** translatory axes (linear axes) and **one** axis of rotation (rotary axis).
Four-axis transformation machine types 1 and 2	Programming of tool orientation only in the plane, which is perpendicular to the rotary axis. There are **three** translatory axes (linear axes) and **one** axis of rotation (rotary axis).
Five-axis transformation machine types 3	Programming of orientation transformation. Kinematics with **three** linear axes and **two** orthogonal rotary axes.
Single-axis swivel head and single-axis rotary table	The rotary axes are parallel to two of the three linear axes. The first rotary axis is moved by two Cartesian linear axes. It rotates the third linear axis with the tool. The second rotary axis rotates the workpiece.

Generic 5/6-axis transformations

Machine type	Programming of orientation transformation
Generic five/six-axis transformation machine types 4 Two-axis swivel head with tool which rotates around itself and single-axis rotary table	Programming of orientation transformation. Kinematics with **three** linear axes and **three** orthogonal rotary axes. The rotary axes are parallel to two of the three linear axes. The first rotary axis is moved by two Cartesian linear axes. It rotates the third linear axis with the tool. The second rotary axis rotates the workpiece. The basic tool orientation can also be programmed with additional rotation of the tool around itself with the THETA rotary angle.

When calling "generic three-, four-, and five/six-axis transformation", the basic orientation of the tool can also be transferred. The restrictions in respect of the directions of the rotary axes no longer apply. If the rotary axes are not exactly vertical to one another or existing rotary axes are not exactly parallel with the linear axes, "generic five-/six-axis transformation" can provide better results in respect of tool orientation.

Kinematic transformations TRANSMIT, TRACYL and TRAANG

For milling on turning machines or an axis that can be set for inclined infeed during grinding, the following axis arrangements apply by default in accordance with the transformation declared:

TRANSMIT	Activation of polar transformation
Face machining in the turning clamp	A rotary axis An infeed axis vertical to the axis of rotation A longitudinal axis parallel to the axis of rotation

TRACYL	Activation of the cylinder surface transformation
Machining of grooves with any path on cylindrical bodies	A rotary axis An infeed axis vertical to the axis of rotation A longitudinal axis parallel to the axis of rotation

TRAANG	Activation of the inclined axis transformation
Machining with an oblique infeed axis	A rotary axis An infeed axis with parameterizable angle A longitudinal axis parallel to the axis of rotation

Cartesian PTP travel

The machine moves in machine coordinates and is programmed with:

TRAORI	Activation of transformation
PTP Point-to-point motion	Approach position in Cartesian coordinate system (MCS)
CP	Path motion of Cartesian axes in the BCS

TRAORI	Activation of transformation
STAT	Position of the articulated joints is dependent on the transformation
TU	The angle at which the axes traverse on the shortest path

PTP transversal with generic 5/6-axis transformation

The machine is moved using machine coordinates and the tool orientation, where the movements can be programmed both using round axis positions and using Euler and/or RPY angle vectors irrespective of the kinematics or the direction vectors.

Round axis interpolation, vector interpolation with large circle interpolation or interpolation of the orientation vector on a peripheral surface of a taper are possible in such cases.

Example: Three- to five-axis transformation on a universal milling head

The machine tool has at least five axes:

- Three translatory axes for movements in straight lines, which move the operating point to any position in the working area.
- Two rotary swivel axes arranged at a configurable angle (usually 45 degrees) allow the tool to swivel to positions in space that are limited to a half sphere in a 45-degree configuration.

7.1.2 Overview of orientation transformation TRAORI

Programming types available in conjunction with TRAORI

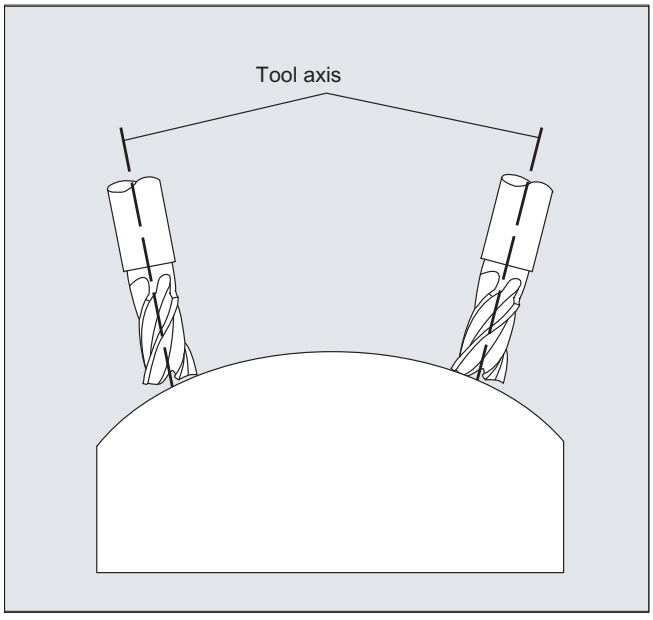
Machine type	Programming with active transformation TRAORI
Machine types 1, 2, or 3 two-axis swivel head or two-axis rotary table or a combination of single-axis swivel head and single-axis rotary table.	The axis sequence of the orientation axes and the orientation direction of the tool can either be configured on a **machine-specific** basis using machine data depending on the machine kinematics or on a **workpiece-specific** basis with programmable orientation independently of the machine kinematics. The directions of rotation of the orientation axes in the reference system are programmed with: - ORIMKS reference system = machine coordinate system - ORIWKS reference system = workpiece coordinate system The default setting is ORIWKS. Programming of orientation axes with: A, B, C of the machine axis position direct A2, B2, C2 angle programming virtual axes with - ORIEULER via Euler angle (standard) - ORIRPY via RPY angle - ORIVIRT1 via virtual orientation axes 1st definition - ORIVIRT2 via virtual orientation axes 2nd definition with differentiation between the interpolation type: **linear interpolation** - ORIAXES of orientation axes or machine axes **large radius circle interpolation** (interpolation of the orientation vector) - ORIVECT from orientation axes Programming orientation axes by specifying A3, B3, C3 of the vector components (direction/surface normal) Programming the resulting tool orientation A4, B4, C4 of the vector surface normal at the beginning of the block A5, B5, C5 of the vector perpendicular to the surface at the end of the block LEAD leading angle for tool orientation TILT tilt angle for the tool orientation

Machine type	Programming with active transformation TRAORI
	Interpolation of the orientation vector on a taper peripheral surface Orientation changes to a taper peripheral surface **anywhere in space** using interpolation: - ORIPLANE in the plane (large radius circle interpolation) - ORICONCW on a taper peripheral surface in the clockwise direction - ORICONCCW on a taper peripheral surface in the counter-clockwise direction A6, B6, C6 director vector (axis of rotation of the taper) -OICONIO interpolation on a taper peripheral surface with: A7, B7, C7 intermediate vectors (initial and ultimate orientation) or - ORICONTO on the peripheral surface of a taper, tangential transition Changes in orientation in relation **to a path** with - ORICURVE specification of the movement of two contact points using PO[XH]=(xe, x2, x3, x4, x5) orientation polynomials up to the fifth degree PO[YH]=(ye, y2, y3, y4, y5) orientation polynomials up to the fifth degree PO[ZH]=(ze, z2, z3, z4, z5) orientation polynomials up to the fifth degree - ORIPATHS smoothing of orientation characteristic with A8, B8, C8 reorientation phase of tool corresponding to: direction and path length of tool during retraction movement
Machine types 1 and 3 Other machine types with additional tool rotation around itself require a 3rd rotary axis Orientation transformation, e.g. generic 6-axis transformation. Rotations of orientation vector.	Programming **of rotations** for tool orientation with LEAD angle, angle relative to surface normal vector PO[PHI] programming of a polynomial up to the fifth degree TILT angle rotation about path tangent (Z direction) PO[PSI] programming of a polynomial up to the fifth degree THETA angle of rotation (rotation about tool direction in Z) THETA= value reached at end of block THETA=AC(...) absolute non-modal switching to dimensions THETA=IC(...) non-modal switching to chain dimensions THETA=Θ_e interpolate programmed angle G90/G91 PO[THT]=(..) programming of a polynomial up to the fifth degree programming of the rotation vector - ORIROTA rotation, absolute - ORIROTR relative rotation vector - ORIROTT tangential rotation vector
Orientation relative to the path for orientation changes relative to the path or rotation of the rotary vector tangentially to the path	Changes in orientation **relative to the path** with - ORIPATH tool orientation relative to the path - ORIPATHS also in the event of a blip in the orientation characteristic programming of rotation vector - ORIROTC tangential rotation vector, rotation to path tangent

7.2 Three, four and five axis transformation (TRAORI)

7.2.1 General relationships of universal tool head

To obtain optimum cutting conditions when machining surfaces with a three-dimensional curve, it must be possible to vary the setting angle of the tool.

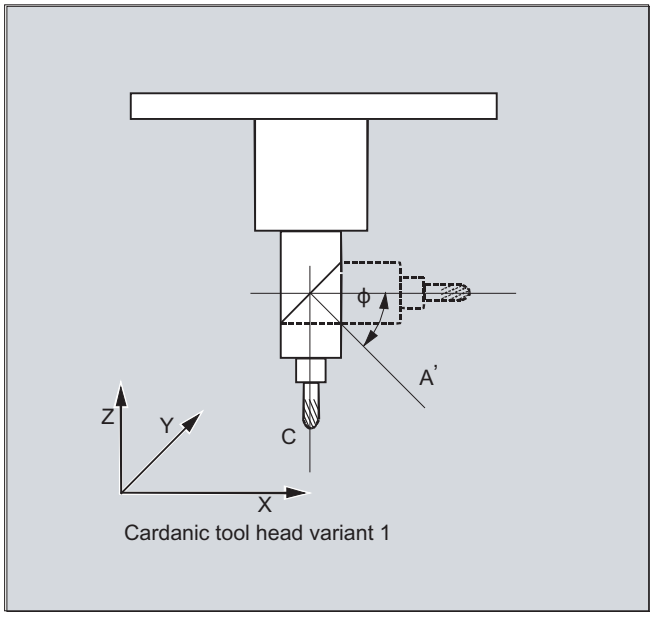


The machine design to achieve this is stored in the axis data.

5-axis transformation

Cardanic tool head

Three linear axes (X, Y, Z) and two orientation axes (C, A) define the setting angle and the operating point of the tool here. One of the two orientation axes is created as an inclined axis, in our example A' - in many cases, placed at 45°.

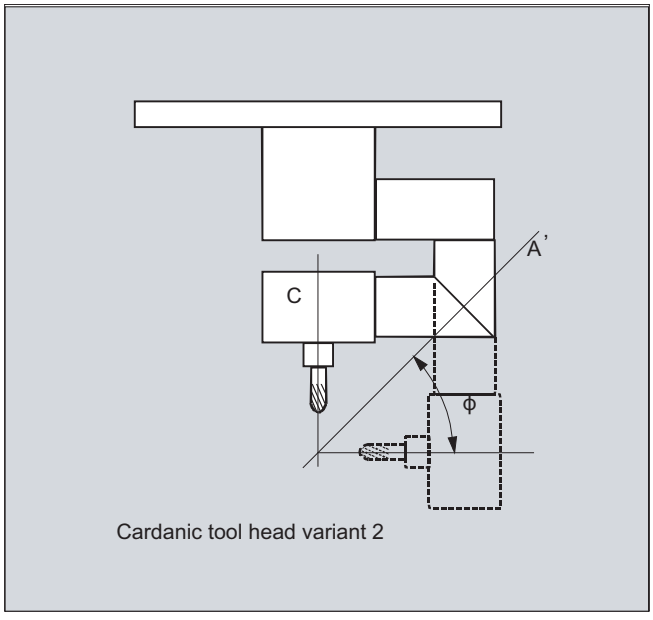


Cardanic tool head variant 1

In the examples shown here, you can see the arrangements as illustrated by the CA machine kinematics with the Cardanic tool head!

Machine manufacturer

The axis sequence of the orientation axes and the orientation direction of the tool can be set up using the machine data as appropriate for the machine kinematics.



Cardanic tool head variant 2

In this example, A' lies below the angle φ to the X axis.

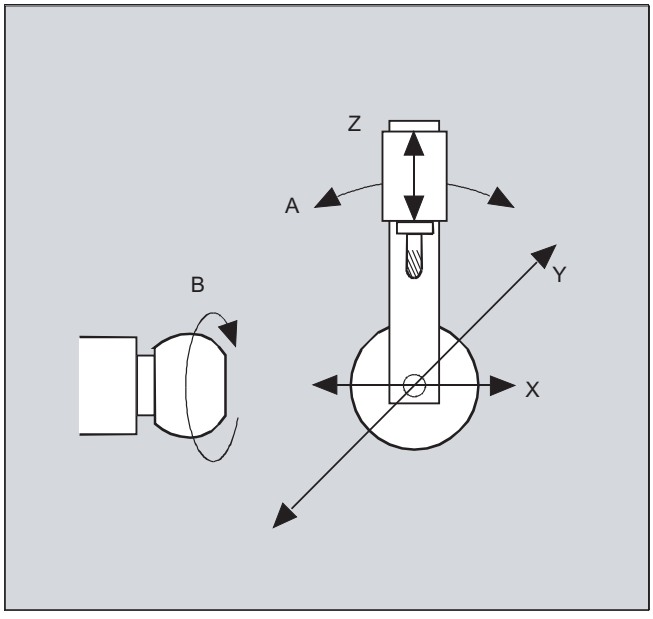
The following possible relations are generally valid:

A' lies below the angle φ to the	X axis
B' lies below the angle φ to the	Y axis
C' lies below the angle φ to the	Z axis

Angle φ can be configured in the range 0° to +89° using machine data.

With swiveling linear axis

This is an arrangement with a moving workpiece and a moving tool. The kinematics consists of three linear axes (X, Y, Z) and two orthogonally arranged rotary axes. The first rotary axis is moved, for example, over a compound slide of two linear axes, the tool standing parallel to the third linear axis. The second rotary axis turns the workpiece. The third linear axis (swivel axis) lies in the compound slide plane.



The axis sequence of the rotary axes and the orientation direction of the tool can be set up using the machine data as appropriate for the machine kinematics.

There are the following possible relationships:

Axes:	Axis sequences:
1. rotary axis	A A B B C C
2. rotary axis	B C A C A B
Swiveled linear axis	Z Y Z X Y X

For more detailed information about configurable axis sequences for the orientation direction of the tool, see

References: /FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformations (F2), Section Universal milling head, "Parameterization".

7.2.2 Three, four and five axis transformation (TRAORI)

The user can configure two or three translatory axes and one rotary axis. The transformations assume that the rotary axis is orthogonal on the orientation plane.

Orientation of the tool is possible only in the plane perpendicular to the rotary axis. The transformation supports machine types with movable tool and movable workpiece.

Three- and four-axis transformations are configured and programmed in the same way as five-axis transformations.

Reference:
Function Manual, Special Functions; Multi-Axis Transformations (F2)

Syntax

```
TRAORI(<n>)
TRAORI(<n>,<X>,<Y>,<Z>,<A>,<B>)
TRAFOOF
```

Meaning

TRAORI:	Activates the first specified orientation transformation	
TRAORI(<n>):	Activates the orientation transformation specified by n	
<n>:	Number of the transformation	
	Value:	1 or 2
	Example: TRAORI(1) activates orientation transformation 1	
<X>,<Y>,<Z>:	Component of orientation vector to which tool points	
<A>,<B>:	Programmable offset for the rotary axes	
TRAFOOF:	Deactivate transformation	

Tool orientation

Depending on the orientation direction selected for the tool, the active working plane (G17, G18, G19) must be set in the NC program in such a way that tool length offset works in the direction of tool orientation.

Note

When the transformation is enabled, the positional data (X, Y, Z) always relates to the tip of the tool. Changing the positions of the rotary axes involved in the transformation causes compensating motion of the remaining machine axes - which means that the position of the tool tip remains unchanged.

Orientation transformation always points from the tool tip to the tool adapter.

Offset for orientation axes

When orientation transformation is activated an additional offset can be programmed directly for the orientation axes.

Parameters can be omitted if the correct sequence is used in programming.

Example:

```
TRAORI(, , , ,A,B)  ; If only a single offset is to be entered
```

As an alternative to direct programming, the additional offset for orientation axes can also be transferred automatically from the zero offset currently active. Transfer is configured in the machine data.

Examples

```
TRAORI(1,0,0,1)          ; The basic orientation of the tool is in the Z direction
TRAORI(1,0,1,0)          ; The basic orientation of the tool is in the Z direction
TRAORI(1,0,1,1)          ; The basic orientation of the tool is in the Y/Z direction
                         (corresponds to the position -45°)
```

7.2.3 Variants of orientation programming and initial setting (ORIRESET)

Orientation programming of tool orientation with TRAORI

In conjunction with a programmable TRAORI orientation transformation, in addition to the linear axes X, Y, Z, the axis identifiers A.., B..., C... can also be used to program axis positions or virtual axes with angles or vector components. Various types of interpolation are possible for orientation and machine axes. Regardless of which PO[angle] orientation polynomials and PO[axis] axis polynomials are currently active, a number of different types of polynomial can be programmed. These include G1, G2, G3, CIP or POLY.

Changes in tool orientation can even be programmed using orientation vectors in some cases. In such cases, the ultimate orientation of each block can be set either by means of direct programming of the vector or by programming the rotary axis positions.

Note

Variants of orientation programming for three- to five-axis transformation

In respect of three- to five-axis transformation, the following variants:

1. A, B, C direct entry of machine axis positions
2. A2, B2, C2 angular programming of virtual axes using Euler angle or RPY angle
3. A3 ,B3, C3 entry of vector components
4. LEAD, TILT entry of lead and tilt angles relative to the path and surface
5. A4, B4, C4 and A5, B5, C5 surface normal vector at start of block and end of block
6. A6, B6, C6 and A7, B7, C7 interpolation of orientation vector on a peripheral surface of a taper
7. A8, B8, C8 reorientation of tool, direction and path length of retracting movement

are mutually exclusive.

If an attempt is made to program mixed values, alarm messages are output.

Initial tool orientation setting ORIRESET

By programming ORIRESET (A, B, C), the orientation axes are moved in linear and synchronous motion from their current position to the specified initial setting position.

If an initial setting position is not programmed for an axis, a defined position from the associated machine data $MC_TRAFO5_ROT_AX_OFFSET_1/2 is used. Any active frames of round axles which may be present are ignored.

Note

Only if an orientation transformation is active with TRAORI(...), can an initial setting for the tool orientation regardless of kinematics be programmed without alarm 14101 using ORIRESET(...).

Examples

```
1. Example of machine kinematics CA (channel axis names C, A)
ORIRESET(90, 45)       ;C at 90 degrees, A at 45 degrees
ORIRESET(, 30)         ;C at $MC_TRAFO5_ROT_AX_OFFSET_1/2[0], A at 30 degrees
ORIRESET( )            ;C at $MC_TRAFO5_ROT_AX_OFFSET_1/2[0],
                       ;A at $MC_TRAFO5_ROT_AX_OFFSET_1/2[1]
2. Example of machine kinematics CAC (channel axis names C, A, B)
ORIRESET(90, 45, 90)   ;C at 90 degrees, A at 45 degrees, B at 90 degrees
ORIRESET( )            ;C at $MC_TRAFO5_ROT_AX_OFFSET_1/2[0],
                       ;A at $MC_TRAFO5_ROT_AX_OFFSET_1/2[1],
                       ;B at $MC_TRAFO5_ROT_AX_OFFSET_1/2[2]
```

Programming LEAD, TILT and THETA rotations

In respect of three- to five-axis transformation, tool orientation rotations are programmed with the LEAD and TILT angles.

In respect of a transformation with third rotary axis, additional programming settings for C2 (rotations of the orientation vector) are permitted for both orientation with vector components and with entry of the LEAD, TILT angles.

With an additional third rotary axis, the rotation of the tool about itself can be programmed with the THETA rotary angle.

7.2.4 Programming the tool orientation (A..., B..., C..., LEAD, TILT)

The following options are available when programming tool orientation:

1. Direct programming the motion of rotary axes. The change of orientation always occurs in the basic or machine coordinate system. The orientation axes are traversed as synchronized axes.
2. Programming in Euler or RPY angles in accordance with angle definition using `A2, B2, C2`

3. Programming the direction vector using A3, B3, C3 The direction vector points from the tool tip toward the tool adapter.
4. Programming the surface normal vector at the start of the block with A4, B4, C4 and at the end of the block with A5, B5, C5 (face milling).
5. Programming using lead angle LEAD and tilt angle TILT
6. Programming the rotary axis of taper as normalized vector using A6, B6, C6 or of intermediate orientation on the peripheral surface of a taper using A7, B7, C7,
   see "Orientation programming along the peripheral surface of a taper (ORIPLANE, ORICONxx)".
7. Programming the reorientation, direction and path length of tool during retraction movement using A8, B8, C8,
   see "Smoothing the orientation characteristic (ORIPATHS A8=, B8=, C8=)"

Note

In all cases, orientation programming is only permissible if an orientation transformation is active.

Advantage: These programs can be transferred to any machine kinematics.

Definition of tool orientation via G code

Note

Machine manufacturer

Machine data can be used to switch between Euler or RPY angles. If the machine data is set accordingly, changeovers are possible both depending on the active G code of group 50 and irrespective of this. The following setting options can be selected:

1. If both machine data for defining the orientation axes and defining the orientation angle are set to zero via G code:
   The angles programmed using A2, B2, C2 are **dependent on machine data** The angle definition of orientation programming is either interpreted as Euler or RPY angles.
2. If the machine data for defining the orientation axes is set to one via G code, the changeover is
   dependent on the active G code of group 50:
   The angles programmed using A2, B2, C2 are interpreted in accordance with the active G codes ORIEULER, ORIRPY, ORIVIRT1, ORIVIRT2, ORIAXPOS and ORIPY2 .
   The values programmed with the orientation axes are also interpreted as orientation angles in accordance with the active G code of group 50.
3. If the machine data for defining the orientation angle is set to one via G code and the machine data for defining the orientation axes is set to zero via G code, the changeover is **not dependent** on the active G code of group 50:
   The angles programmed using A2, B2, C2 are interpreted in accordance with one of the active G codes ORIEULER, ORIRPY, ORIVIRT1, ORIVIRT2 ORIAXPOS and ORIPY2 . The values programmed with the orientation axes are always interpreted as round axis positions irrespective of the active G code of group 50.

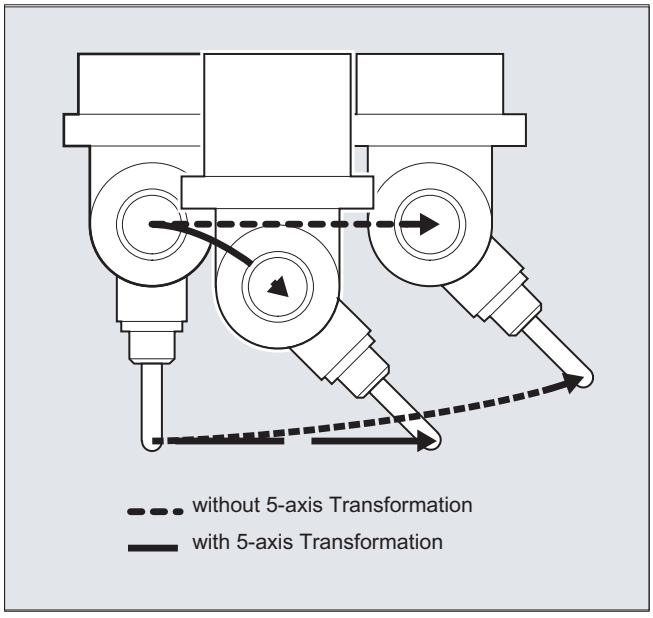
Programming

`G1 X... Y... Z... A... B... C...`	Programming the rotary axis motion
`G1 X... Y... Z... A2= B2= C2=`	Programming in Euler angles
`G1 X... Y... Z... A3== B3== C3==`	Programming the directional vector
`G1 X... Y... Z... A4== B4== C4==`	Programming the surface normal vector at block start
`G1 X... Y... Z... A5== B5== C5==`	Programming the surface normal vector at end of block
`LEAD=`	Lead angle for programming tool orientation
`TILT=`	Tilt angle for programming tool orientation

Parameters

`G...:`	Details of the rotary axis motion
`X... Y... Z...:`	Details of the linear axes
`A... B... C...:`	Details of the machine axis positions of the rotary axes
`A2= B2= C2=:`	Angle programming (Euler or RPY angle) of virtual axes or orientation axes
`A3== B3== C3== :`	Details of the direction vector components
`A4== B4== C4==:`	Details, for example, for the face milling, the component of the surface normal vector at block start
`A5== B5== C5== :`	Details, for example, for the face milling, the component of the surface normal vector at block end
`LEAD= :`	Angle relative to the surface normal vector in the plane that is defined by the path tangent and the surface normal vector
`TILT=:`	Angle in the plane, perpendicular to the path tangent relative to the surface normal vector

Example: Comparison without and with 5-axis transformation



Further information

5-axis programs are usually generated by CAD/CAM systems and not entered at the control. So the following explanations are directed mainly at programmers of postprocessors.

The type of orientation programming is defined in G code group 50:

G function	Orientation programming
ORIEULER:	Via Euler angle
ORIRPY:	Via RPY angle (rotation sequence ZYX)
ORIVIRT1:	Via virtual orientation axes (definition 1)
ORIVIRT2:	Via virtual orientation axes (definition 2)
ORIAXPOS:	Via virtual orientation axes with rotary axis positions
ORIPY2:	Via RPY angle (rotation sequence XYZ)

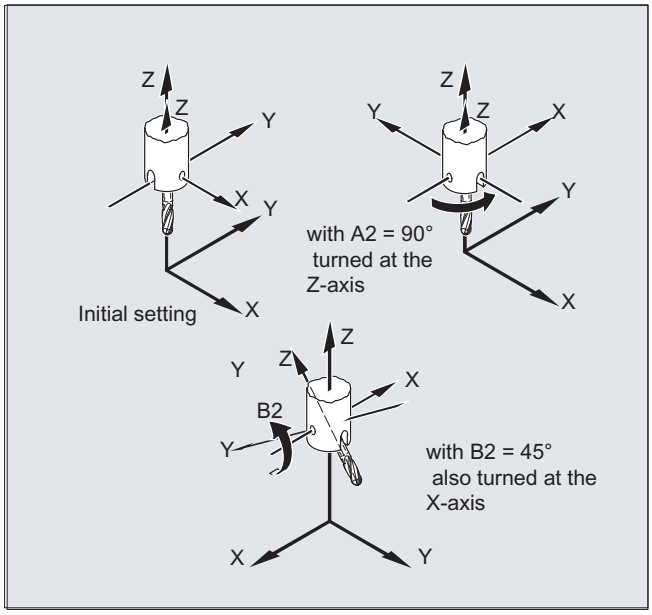
Note

The machine manufacturer can use machine data to define various variants. Please refer to the machine manufacturer's instructions.

Programming in Euler angles ORIEULER

The values programmed during orientation programming with A2, B2, C2 are interpreted as Euler angles (in degrees).

The orientation vector results from turning a vector in the Z direction firstly with A2 around the Z axis, then with B2 around the new X axis and lastly with C2 around the new Z axis.



In this case the value of C2 (rotation around the new Z axis) is meaningless and does not have to be programmed.

Programming in RPY angles ORIRPY

The values programmed with A2, B2, C2 for orientation programming are interpreted as an RPY angle (in degrees).

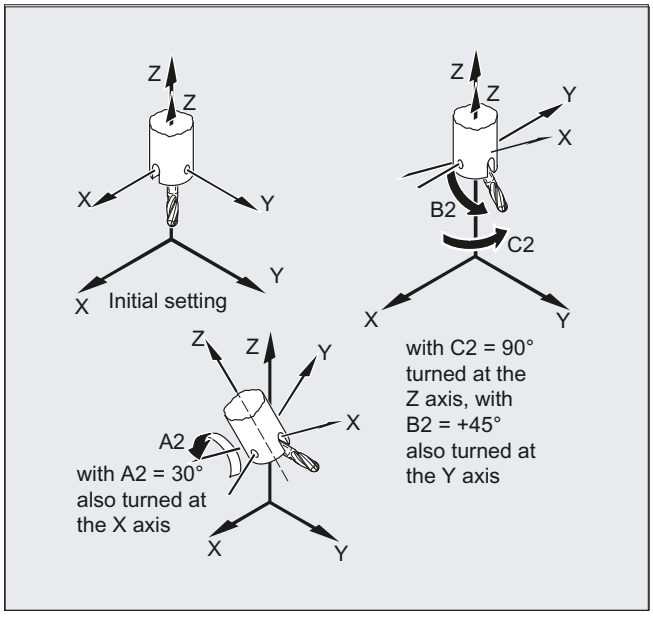
Note

In contrast to Euler angle programming, all three values here have an effect on the orientation vector.

When defining angles with orientation angle via RPY angle, the following applies for the orientation axes:

$MC_ORI_DEF_WITH_G_CODE = 0

The orientation vector results from turning a vector in the Z direction firstly with C2 around the Z axis, then with B2 around the new Y axis and lastly with A2 around the new X axis.



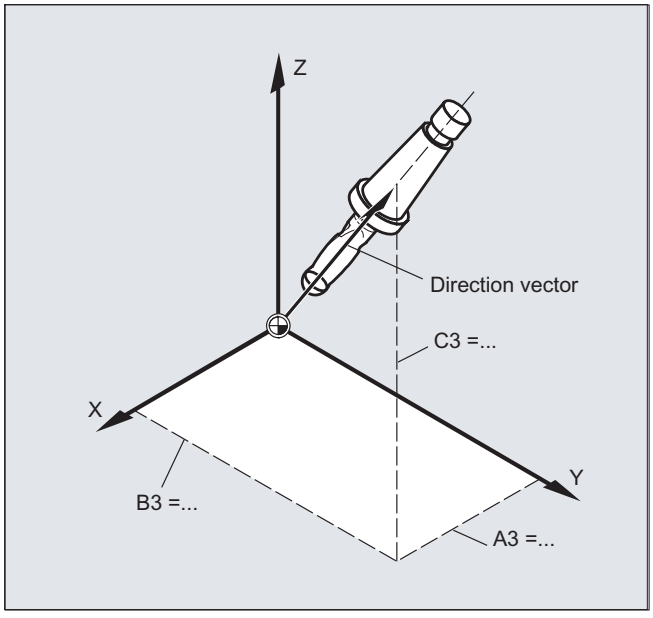
By defining the orientation axes via G code, if the machine data $MC_ORI_DEF_WITH_G_CODE = 1, then:

The orientation vector results from turning a vector in the Z direction firstly with A2 around the Z axis, then with B2 around the new Y axis and finally with C2 around the new X axis.

Programming the directional vector

The components of the direction vector are programmed with `A3, B3, C3`. The vector points towards the tool adapter; the length of the vector is of no significance.

Vector components that have not been programmed are set equal to zero.



Programming the tool orientation with LEAD= and TILT=

The resultant tool orientation is determined from:

- Path tangent
- Surface normal vector
  at the start of the block `A4, B4, C4` and at the end of the block `A5, B6, C5`
- Lead angle `LEAD`
  in the plane defined by the path tangent and surface normal vector
- Tilt angle `TILT` at the end of the block
  vertical to the path tangent and relative to the surface normal vector

Behavior at inside corners (for 3D tool offset)

If the block is shortened at an inside corner, the resulting tool orientation is also achieved at the end of the block.

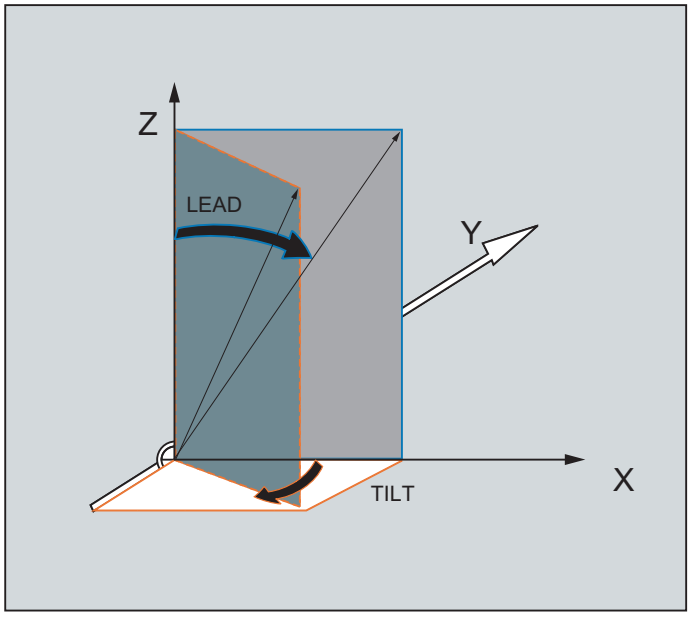
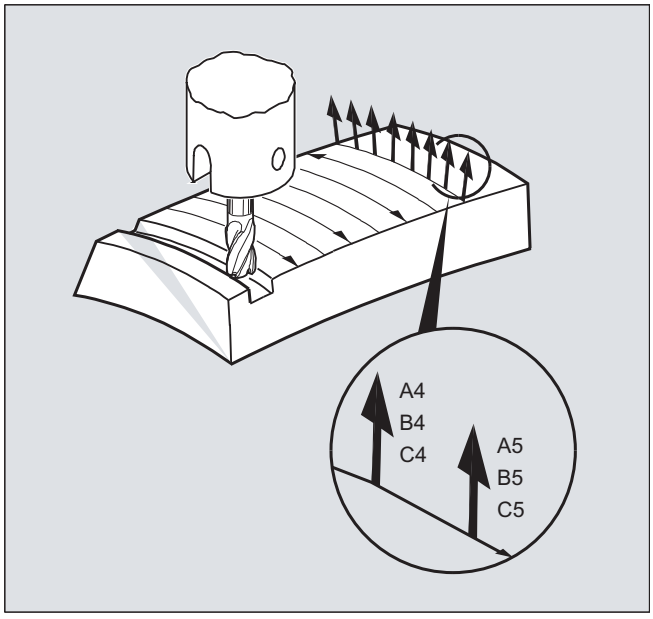


Figure 7-1 Definition of tool orientation with LEAD= and TILT=

7.2.5 Face milling (A4, B4, C4, A5, B5, C5)

Face milling is used to machine curved surfaces of any kind.



For this type of 3D milling, you will require the line-by-line description of the 3D paths on the workpiece surface.

The tool shape and dimensions are taken into account in the calculations, which are normally performed in CAM. The fully calculated NC blocks are then read into the control via postprocessors.

Programming the path curvature

Surface description

The path curvature is described by surface normal vectors with the following components:

`A4, B4, C4` Start vector at block start

`A5, B5, C5` End vector at block end

If a block only contains the start vector, the surface normal vector will remain constant throughout the block. If a block only contains the end vector, interpolation will run from the end value of the previous block via large-circle interpolation to the programmed end value.

If the start and end vectors are programmed, interpolation runs between the two directions, also via large-circle interpolation. This allows continuously smooth paths to be created.

Regardless of the active G17 to G19 level, in the initial setting, surface normal vectors point in the Z direction.

The length of a vector is meaningless.

Vector components that have not been programmed are set to zero.

When ORIWKS is active (see "Reference of the orientation axes (ORIWKS, ORIMKS): (Page 314)"), the surface normal vectors refer to the active frame and are also rotated with frame rotation.

Machine manufacturer

The surface normal vector must be perpendicular to the path tangent, within a limit value set via machine data, otherwise an alarm will be output.

7.2.6 Reference of the orientation axes (ORIWKS, ORIMKS):

For orientation programming in the workpiece coordinate system using

- Euler or RPY angle or
- Orientation vector

the course of the rotary motion can be set using `ORIMKS`/`ORIWKS`.

Note

Machine manufacturer

The type of interpolation for the orientation is specified with machine data:

MD21104 $MC_ORI_IPO_WITH_G_CODE

= FALSE: The reference is provided by the G functions ORIWKS und ORIMKS.

= TRUE: The reference is provided by the G functions in the 51st group (ORIAXES, ORIVECT, ORIPLANE, etc.).

Syntax

```
ORIMKS=...
ORIWKS=...
```

Meaning

`ORIMKS`:	Rotation in the machine coordinate system
`ORIWKS`:	Rotation in the workpiece coordinate system

Note

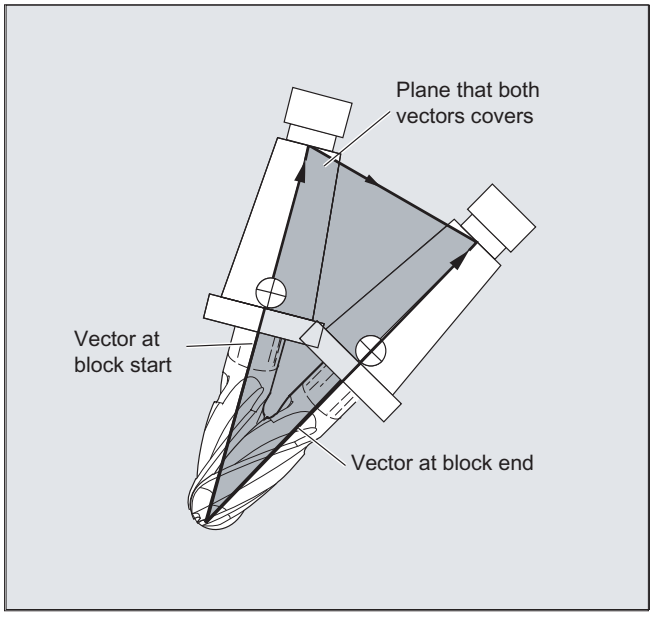
ORIWKS is the basic setting. In the case of a 5-axis program, if it is not immediately obvious on which machine it is to run, ORIWKS must always be selected. Which movements the machine actually executes depend on the machine kinematics.

`ORIMKS` can be used to program actual machine movements (to avoid collisions with devices or similar, for example).

Further information

With ORIMKS, the movement executed by the tool **depends** on the machine kinematics. In the case of a change in orientation of a tool tip at a fixed point in space, linear interpolation takes place between the rotary axis positions.

With ORIWKS, the movement executed by the tool **does not depend** on the machine kinematics. With an orientation change with a fixed tool tip, the tool moves in the plane set up by the start and end vectors.



Singular positions

Note

ORIWKS

Orientation movements in the singular setting area of the 5-axis machine require vast movements of the machine axes. (For example, with a rotary swivel head with C as the rotary axis and A as the swivel axis, all positions with A = 0 are singular.)

Machine manufacturer

To avoid overloading the machine axes, the velocity control vastly reduces the tool path velocity near the singular positions.

With machine data

$MC_TRAFO5_NON_POLE_LIMIT

$MC_TRAFO5_POLE_LIMIT

the transformation can be parameterized in such a way that orientation movements close to the pole are put through the pole and rapid machining is possible.

Singular positions are handled only with the MD $MC_TRAFO5_POLE_LIMIT.

References:
/FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformation (F2),
"Singular Points and How to Deal with Them" section.

7.2.7 Programming orientation axes (ORIAXES, ORIVECT, ORIEULER, ORIRPY, ORIRPY2, ORIVIRT1, ORIVIRT2)

The "Orientation axes" function describes the orientation of the tool in space and is achieved by programming the offset for the rotary axes. An additional, third degree of freedom can be achieved by also rotating the tool about itself. In this case, the tool is oriented in space via a third rotary axis for which 6-axis transformation is required. The rotation of the tool about itself is defined using the THETA angle of rotation in accordance with the type of interpolation of the rotation vectors (see "Rotations of the tool orientation (ORIROTA, ORIROTR, ORIROTT, ORIROTC, THETA) (Page 325)").

Axis identifiers A2, B2 and C2 are used to program the orientation axes.

Syntax

```
N... ORIAXES/ORIVECT            ; Linear or large-circle interpolation
N... G1 X Y Z A B C

N... ORIPLANE                   ; Orientation interpolation of the plane

N... ORIEULER/ORIRPY/ORIRPY2    : Orientation angle Euler/RPY angle
N... G1 X Y Z A2= B2= C2=       ; Angle programming of virtual axes

N... ORIVIRT1/ORIVIRT2          ; Virtual orientation axes def. 1/2
N... G1 X Y Z A3= B3= C3=       ; Direction vector programming
```

Note

Other rotary axis offsets of the orientation axes can be programmed for orientation changes along the peripheral surface of a taper in space, see "Orientation programming along the peripheral surface of a taper (ORIPLANE, ORICONCW, ORICONCCW, ORICONTO, ORICONIO) (Page 318)".

Meaning

ORIAXES:	Linear interpolation of machine or orientation axes
ORIVECT:	Large-circle interpolation (identical to ORIPLANE)

ORIMKS: ORIWKS:	Rotation in the machine coordinate system Rotation in the workpiece coordinate system For a description, see "Reference of the orientation axes (ORIWKS, ORIMKS): (Page 314)".
A= B= C=:	Programming the machine axis position
ORIEULER:	Orientation programming via Euler angle
ORIRPY:	Orientation programming via RPY angle The rotation sequence is XYZ and: • A2 is the angle of rotation around X • B2 is the angle of rotation around Y • C2 is the angle of rotation around Z
ORIRPY2:	Orientation programming via RPY angle The rotation sequence is ZYX and: • A2 is the angle of rotation around Z • B2 is the angle of rotation around Y • C2 is the angle of rotation around X
A2= B2= C2=:	Angle programming of virtual axes
ORIVIRT1/ORIVIRT2:	Orientation programming using virtual orientation axes Definition 1: Definition according to MD21120 $MC_ORIAX_TURN_TAB_1 Definition 2: Definition according to MD21130 $MC_ORIAX_TURN_TAB_2
A3= B3= C3=:	Direction vector programming of direction axis

Further information

Machine manufacturer

MD21102 $MC_ORI_DEF_WITH_G_CODE specifies how the programmed angles A2, B2, C2 are defined:

The definition is according to MD21100 $MC_ORIENTATION_IS_EULER (standard) or the definition is according to G group 50 (ORIEULER, ORIRPY, ORIVIRT1, ORIVIRT2).

MD21104 $MC_ORI_IPO_WITH_G_CODE defines which interpolation mode type is active: ORIWKS/ORIMKS or ORIAXES/ORIVECT.

JOG mode

Interpolation for orientation angles in this mode of operation is always linear. During continuous and incremental traversal via the traversing keys, only one orientation axis can be traversed. Orientation axes can be traversed simultaneously using the handwheels.

For manual travel of the orientation axes, the channel-specific feed override switch or the rapid traverse override switch work at rapid traverse override.

A separate velocity setting is possible with the following machine data:

MD21160 $MC_JOG_VELO_RAPID_GEO

MD21165 $MC_JOG_VELO_GEO

MD21150 $MC_JOG_VELO_RAPID_ORI

MD21155 $MC_JOG_VELO_ORI

Note

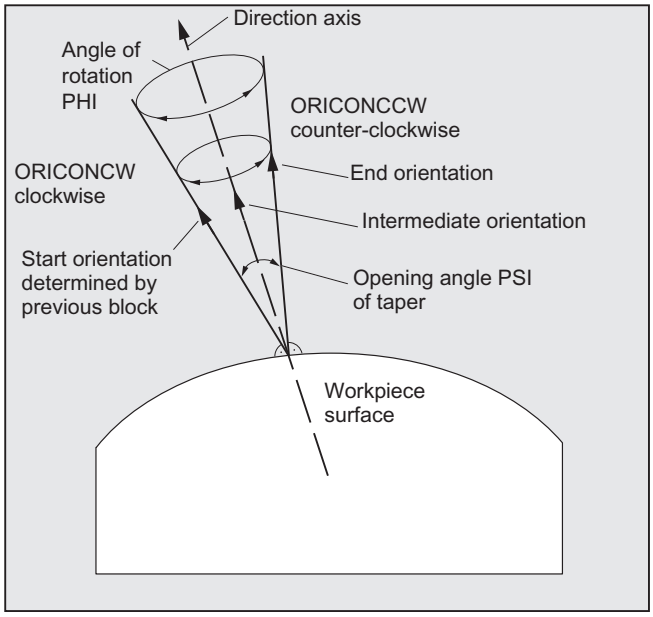
SINUMERIK 840D sl with "handling transformation package"

Using the "Cartesian manual traverse" function, in the JOG mode, the translation of geometry axes can be set separately from one another in the reference systems MCS, WCS and TCS.

References:
Function Manual Extended Functions; Kinematic Transformation (M1)

7.2.8 Orientation programming along the peripheral surface of a taper (ORIPLANE, ORICONCW, ORICONCCW, ORICONTO, ORICONIO)

With extended orientation it is possible to execute a change in orientation along the peripheral surface of a taper in space. The orientation vector is interpolated on the peripheral surface of a taper using the ORICONxx modal command. The end orientation can be programmed with ORIPLANE for interpolation on a plane. The start orientation is usually defined by the previous blocks.



Programming

The end orientation is either defined by specifying the angle programming in the Euler or RPY angle using A2, B2, C2 or by programming the rotary axis positions using A, B, C. Further programming details are needed for orientation axes along the peripheral surface of a taper:

- Rotary axis of taper as a vector with `A6, B6, C6`
- Opening angle `PSI` with identifier `NUT`
- Intermediate orientation outside of the taper with `A7, B7, C7`

Note

Programming direction vector A6, B6, C6 for the rotary axis of the taper

The programming of an end orientation is not absolutely necessary. If no end orientation is specified, a full outside taper with 360 degrees is interpolated.

Programming the opening angle of the taper with NUT=angle

An end orientation must be specified.

A complete outside taper with 360 degrees cannot be interpolated in this way.

Programming the intermediate orientation A7, B7, C7 on the outside of the taper

An end orientation must be specified. The change in orientation and the direction of rotation is defined uniquely by the three vectors Start orientation, End orientation and Intermediate orientation. All three vectors must be different. If the programmed intermediate orientation is parallel to the start or end orientation, a linear large-circle interpolation of the orientation is executed in the plane that is defined by the start and end vector.

Extended orientation interpolation on the peripheral surface of a taper

`N... ORICONCW` or `ORICONCCW`	**Interpolation** on the outside of a taper with
`N... A6= B6= C6= A3= B3= C3=`	direction vector in the clockwise/counter-clockwise direction of the taper and end orientation or
or	
`N... ORICONTO`	
`N... G1 X Y Z A6= B6= C6=`	tangential transition and specification of end orientation
or	or
`N... ORICONIO`	specification of end orientation and intermediate orientation on the outside of the taper with
`N... G1 X Y Z A7= B7= C7=`	
`N... PO[PHI]=(a2, a3, a4, a5)`	polynomials for angle of rotation and
`N... PO[PSI]=(b2, b3, b4, b5)`	polynomials for opening angle

Parameters

`ORIPLANE`:	Interpolation in the plane (large-circle interpolation)
`ORICONCW`:	Interpolation on the peripheral surface of a taper in the clockwise direction
`ORICONCCW`:	Interpolation on the peripheral surface of a taper in the counter-clockwise direction

ORICONTO:	Interpolation on the peripheral surface of a taper with tangential transition
A6= B6= C6=:	Programming of a rotary axis of the taper (normalized vector)
NUT=angle:	Opening angle of taper in degrees
NUT=+179:	Traverse angle less than or equal to 180 degrees
NUT=-181:	Traverse angle greater than or equal to 180 degrees
ORICONIO:	Interpolation on the peripheral surface of a taper
A7= B7= C7=:	Intermediate orientation (programming as normalized vector)
PHI:	Angle of rotation of the orientation about the direction axis of the taper
PSI:	Opening angle of the taper
Possible polynomials PO[PHI]=(a2, a3, a4, a5) PO[PSI]=(b2, b3, b4, b5)	Apart from the different angles, polynomials can also be programmed up to the 5th degree

Example: Different changes to orientation

Program code	Comment
…	
N10 G1 X0 Y0 F5000	
N20 TRAORI(1)	; Orientation transformation ON
N30 ORIVECT	; Interpolate tool orientation as a vector.
…	; Tool orientation in the plane.
N40 ORIPLANE	; Select large-circle interpolation.
N50 A3=0 B3=0 C3=1	
N60 A3=0 B3=1 C3=1	; Orientation in the Y/Z plane is rotated through 45 degrees, orientation (0,1/√2,1/√2) is reached at the end of the block.
…	
N70 ORICONCW	; Orientation programming on the outside of the taper:
N80 A6=0 B6=0 C6=1 A3=0 B3=0 C3=1	The orientation vector is interpolated on the outside of a taper with the direction (0,0,1) up to the orientation (1/√2,0,1/√2) in the clockwise sense, the angle of rotation is 270 degrees.
N90 A6=0 B6=0 C6=1	; The tool orientation goes through a full revolution on the outside of the same taper.

Further information

If changes of orientation along the peripheral surface of a taper anywhere in space are to be described, the vector about which the tool orientation is to be rotated must be known. The start and end orientation must also be specified. The start orientation results from the previous block and the end orientation has to be programmed or defined via other conditions.

Programming in the ORIPLANE plane corresponds to ORIVECT

The programming of large-radius circular interpolation together with angle polynomials corresponds to the linear and polynomial interpolation of contours. The tool orientation is interpolated in a plane that is defined by the start and end orientation. If additional polynomials are programmed, the orientation vector can also be tilted out of the plane.

Programming of circles in a plane G2/G3, CIP and CT

The extended orientation corresponds to the interpolation of circles in a plane. For the corresponding programming options for circles with centers or radii such as G2/G3, circle via intermediate point CIP and tangential circles CT, see

References: Programming Manual Fundamentals, "Programming motion commands".

Orientation programming

Interpolation of the orientation vector on the peripheral surface of a taper ORICONxx

Four different types of interpolation from G-code group 51 can be selected for interpolating orientations on the peripheral surface of a taper:

1. Interpolation on the outside of a taper in the clockwise direction `ORICONCW` with specification of end orientation and taper direction, or opening angle. The direction vector is programmed with identifiers `A6, B6, C6` and the opening angle of the taper with identifier NUT= value range in interval 0 degrees to 180 degrees.
2. Interpolation on the outside of a taper in the counterclockwise direction `ORICONCCW` with specification of end orientation and taper direction, or opening angle. The direction vector is programmed with identifiers `A6, B6, C6` and the opening angle of the taper with identifier NUT= value range in interval 0 degrees to 180 degrees.
3. Interpolation on the outside of a taper `ORICONIO` with specification of end orientation and an intermediate orientation, which is programmed with identifiers `A7, B7, C7`.
4. Interpolation on the outside of a taper `ORICONTO` with tangential transition and specification of end orientation. The direction vector is programmed with identifiers `A6, B6, C6`.

7.2.9 Specification of orientation for two contact points (ORICURVE, PO[XH]=, PO[YH]=, PO[ZH]=)

Programming the change in orientation using the second curve in space ORICURVE

Another way to program changes in orientation, besides using the tool tip along a curve in space, is to program the motion of a second contact point of the tool using `ORICURVE`. In this way, changes in tool orientation can be defined uniquely, as when programming the tool vector itself.

Machine manufacturer

Please refer to the machine manufacturer's notes on axis identifiers that can be set via machine data for programming the second orientation path of the tool.

Programming

This type of interpolation can be used to program points (using `G1`) or polynomials (using `POLY`) for the two curves in space. Circles and involutes are not permitted. A `BSPLINE` spline interpolation and the "Combine short spline blocks" function can also be activated.

References:
Function Manual, Basic Functions; Continuous-Path Mode, Exact Stop, Look Ahead (B1), Section: Combine short spline blocks

The other spline types, `ASPLINE` and `CSPLINE`, and compressor activation using `COMPON`, `COMPCURV` or `COMPCAD` are not permitted.

The motion of the two contact points of the tool can be predefined up to the 5th degree when programming the orientation polynomials for coordinates.

Extended orientation interpolation with additional curve in space and polynomials for coordinates

```
N... ORICURVE
N... PO[XH]=(xe, x2, x3, x4, x5)
N... PO[YH]=(ye, y2, y3, y4, y5)
N... PO[ZH]=(ze, z2, z3, z4, z5)
```

Specification of the motion of the second contact point of the tool and additional polynomials of the coordinates in question

Parameters

`ORICURVE`	`Interpolation of the orientation specifying a movement between two contact points of the tool.`
`XH YH ZH`	`Identifiers of the coordinates of the second contact point of the tool of the additional contour as a curve in space`
`Possible polynomials PO[XH]=(xe, x2, x3, x4, x5) PO[YH]=(ye, y2, y3, y4, y5) PO[ZH]=(ze, z2, z3, z4, z5)`	`Apart from using the appropriate end points, the curves in space can also be programmed using polynomials.`
`xe, ye, ze`	`End points of the curve in space`
`xi, yi, zi`	`Coefficients of the polynomials up to the 5th degree`

Note

Identifiers XH YH ZH for programming a second orientation path

The identifiers must be selected such that no conflict arises with the other identifiers or linear axes

X Y Z axes

and rotary axes such as

A2 B2 C2 Euler angle or RPY angle

A3 B3 C3 direction vectors

A4 B4 C4 or A5 B5 C5 surface normal vectors

A6 B6 C6 rotation vectors or A7 B7 C7 intermediate point coordinates

or other interpolation parameters.

7.3 Orientation polynomials (PO[angle], PO[coordinate])

Irrespective of the polynomial interpolation from G-code group 1 that is currently active, two different types of orientation polynomial can be programmed up to the 5th degree for a 3-axis to 5-axis transformation.

1. Polynomials for **angles:** lead angle LEAD, tilt angle TILT
   in relation to the plane that is defined by the start and end orientation.
2. Polynomials for **coordinates:** XH, YH, ZH of the second curve in space for the tool orientation of a reference point on the tool.

With a 6-axis transformation, the rotation of rotation vector THT can be programmed with polynomials up to the 5th degree for rotations of the tool itself, in addition to the tool orientation.

Syntax

Type 1 orientation polynomials for **angles**

```
N… PO[PHI]=(a2, a3, a4, a5)
N… PO[PSI]=(b2, b3, b4, b5)
```

3-axis to 5-axis transformation

Type 2 orientation polynomials for **coordinates**

```
N… PO[XH]=(xe, x2, x3, x4, x5)
N… PO[YH]=(ye, y2, y3, y4, y5)
N… PO[ZH]=(ze, z2, z3, z4, z5)
```

Identifiers for the coordinates of the second orientation path for tool orientation

In both cases, with 6-axis transformations, a polynomial can also be programmed for the **rotation** using

`N… PO[THT]=(c2, c3, c4, c5)`	Interpolation of the rotation relative to the path
or	
`N… PO[THT]=(d2, d3, d4, d5)`	Interpolation absolute, relative and tangential to the change of orientation

of the orientation vector. This is possible if the transformation supports a rotation vector with an offset that can be programmed and interpolated using the THETA angle of rotation.

Meaning

`PO[PHI]`	Angle in the plane between start and end orientation
`PO[PSI]`	Angle describing the tilt of the orientation from the plane between start and end orientation
`PO[THT]`	Angle of rotation created by rotating the rotation vector of one of the G codes of group 54 that is programmed using THETA
`PHI`	Lead angle LEAD
`PSI`	Tilt angle TILT
`THETA`	Rotation about the tool direction in Z
`PO[XH]`	X coordinate of the reference point on the tool
`PO[YH]`	Y coordinate of the reference point on the tool
`PO[ZH]`	Z coordinate of the reference point on the tool

Further information

Orientation polynomials cannot be programmed:

- If ASPLINE, BSPLINE, CSPLINE spline interpolations are active.
  Type 1 polynomials for orientation angles are possible for every type of interpolation except spline interpolation, that is, linear interpolation with rapid traverse G00 or with feedrate G01 with polynomial interpolation using POLY and
  circular/involute interpolation G02, G03, CIP, CT, INVCW and INCCCW
  .
  However, type 2 polynomials for orientation coordinates are only possible if
  linear interpolation with rapid traverse G00 or with feedrate G01 or
  polynomial interpolation with POLY is active.
- If the orientation is interpolated using ORIAXES axis interpolation. In this case, polynomials can be programmed directly with PO[A] and PO[B] for orientation axes A and B.

Type 1 orientation polynomials with ORIVECT, ORIPLANE and ORICONxx

Only type 1 orientation polynomials are possible for large-radius circular interpolation and interpolation outside of the taper with ORIVECT, ORIPLANE and ORICONxx.

Type 2 orientation polynomials with ORICURVE

If interpolation with the additional curve in space ORICURVE is active, the Cartesian components of the orientation vector are interpolated and only type 2 orientation polynomials are possible.

7.4 Rotations of the tool orientation (ORIROTA, ORIROTR, ORIROTT, ORIROTC, THETA)

If you also want to be able to change the orientation of the tools on machine types with movable tools, program each block with end orientation. Depending on the machine kinematics you can either program the orientation direction of the orientation axes or the direction of rotation of orientation vector THETA. Different interpolation types can be programmed for these rotation vectors:

- ORIROTA: Angle of rotation to an absolute direction of rotation.
- ORIROTR: Angle of rotation relative to the plane between the start and end orientation.
- ORIROTT: Angle of rotation relative to the change in the orientation vector.
- ORIROTC: Tangential angle of rotation to the path tangent.

Syntax

Only if interpolation type ORIROTA is active can the angle of rotation or rotation vector be programmed in all four modes as follows:

1. Directly as rotary axis positions `A, B, C`
2. Euler angles (in degrees) with `A2, B2, C2`
3. RPY angles (in degrees) with `A2, B2, C2`
4. Direction vector via `A3, B3, C3` (angle of rotation using `THETA=<value>`)

If ORIOTR or ORIOTT is active, the angle of rotation can only be programmed directly with `THETA`.

A rotation can also be programmed in a separate block without an orientation change taking place. In this case, ORIROTR and ORIROTT are irrelevant. In this case, the angle of rotation is always interpreted with reference to the absolute direction (ORIROTA).

`N... ORIROTA`	Define the interpolation of the rotation vector
`N... ORIROTR`	
`N... ORIROTT`	
`N... ORIROTC`	
`N... A3= B3= C3= THETA=<value>`	Define the rotation of the orientation vector
`N... PO[THT]=(`d_2`, `d_3`, `d_4`, `d_5`)`	Interpolate angle of rotation with a 5th order polynomial

Meaning

`ORIROTA`:	Angle of rotation to an absolute direction of rotation
`ORIROTR`:	Angle of rotation relative to the plane between the start and end orientation
`ORIROTT`:	Angle of rotation as a tangential rotation vector to the change of orientation
`ORIROTC`:	Angle of rotation as a tangential rotation vector to the path tangent
`THETA`:	Rotation of the orientation vector

THETA=<value>:	Angle of rotation in degrees reached by the end of the block
THETA=Θe:	Angle of rotation with end angle Θ_e of rotation vector
THETA=AC(…):	Non-modal switchover to absolute dimensions
THETA=AC(…):	Non-modal switchover to incremental dimensions
Θe:	End angle of rotational vector both absolute with G90 and relative with G91 (incremental dimensioning) is active
PO[THT]=(....):	Polynomial for angle of rotation

Example: Rotations of the orientations

Program code	Comment
N10 TRAORI	; Activate orientation transformation
N20 G1 X0 Y0 Z0 F5000	; Tool orientation
N30 A3=0 B3=0 C3=1 THETA=0	; In Z direction with angle of rotation 0
N40 A3=1 B3=0 C3=0 THETA=90	;In X direction and rotation about 90 degrees
N50 A3=0 B3=1 C3=0 PO[THT]=(180,90)	;Orientation
N60 A3=0 B3=1 C3=0 THETA=IC(-90)	;In Y direction and rotation about 180 degrees
N70 ORIROTT	;Remains constant and rotation to 90 degrees
N80 A3=1 B3=0 C3=0 THETA=30	;Angle of rotation relative to change of orientation
	;Rotation vector in angle 30 degrees to X/Y plane

When interpolating block N40, the angle of rotation from initial value of 0 degrees to final value of 90 degrees is interpolated linearly. In block N50, the angle of rotation changes from 90 degrees to 180 degrees, according to parabola θ(u) = +90u2. In N60, a rotation can also be executed without a change in orientation taking place.

With N80, the tool orientation is rotated from the Y direction toward the X direction. The change in orientation takes place in the X/Y plane and the rotation vector describes an angle of 30 degrees to this plane.

Further information

ORIROTA

The angle of rotation THETA is interpolated with reference to an absolute direction in space. The basic direction of rotation is defined in the machine data.

ORIROTR

The angle of rotation THETA is interpreted relative to the plane defined by the start and end orientation.

ORIROTT

The angle of rotation THETA is interpreted relative to the change in orientation. For THETA=0 the rotation vector is interpolated tangentially to the change in orientation and only differs from ORIROTR if at least one polynomial has been programmed for "tilt angle PSI" for the orientation. The result is a change in orientation that is not executed in the plane. An additional angle of

rotation THETA can then be used to interpolate the rotation vector such that it always produces a specific value referred to the change in orientation.

ORIROTC

The rotation vector is interpolated relative to the path tangent with an offset that can be programmed using the THETA angle. A polynomial PO[THT]=(c2, c3, c4, c5) up to the 5th degree can also be programmed for the offset angle.

7.5 Orientations relative to the path

7.5.1 Orientation types relative to the path

By using this expanded function, relative orientation is not only achieved at the end of the block, but across the entire trajectory. The orientation achieved in the previous block is transferred to the programmed end orientation using large-circle interpolation. There are basically two ways of programming the desired orientation relative to the path:

1. Like the tool rotation, the tool orientation is interpolated relative to the path using ORIPATH, ORPATHTS.
2. The orientation vector is programmed and interpolated in the usual manner. The rotation of the orientation vector is initiated relative to the path tangent using ORIROTC.

Syntax

The type of interpolation of the orientation and the rotation of the tool is programmed using:

N... ORIPATH	Orientation relative to the path
N... ORIPATHS	Orientation relative to the path with smoothing of orientation characteristic
N... ORIROTC	Interpolation of the rotation vector relative to the path

An orientation blip caused by a corner on the trajectory can be smoothed using ORIPATHS. The direction and path length of the retracting movement is programmed via the vector using the components A8=X, B8=Y C8=Z.

ORIPATH/ORIPATHS can be used to program various references to the path tangent via the three angles

- LEAD= Specification of lead angle relative to the path and surface
- TILT= Specification of tilt angle relative to the path and surface
- THETA= Angle of rotation

for the entire trajectory. Polynomials up to the 5th degree can be programmed in addition to the THETA angle of rotation using PO[THT]=(...).

Note

Machine manufacturer

Please refer to the machine manufacturer's instructions. Other settings can be made for orientations relative to the path via configurable machine and setting data. For more detailed information, please refer to

References:
/FB3/ Function Manual, Special Functions; 3 to 5-Axis Transformation (F2),
Section "Orientation"

Meaning

Various settings can be made for the interpolation of angles LEAD and TILT via machine data:

- The tool-orientation reference programmed using LEAD and TILT is retained for the entire block.
- Lead angle LEAD: rotation about the direction vertical to the tangent and normal vector TILT: rotation of the orientation about the normal vector.
- Lead angle LEAD: rotation about the direction vertical to the tangent and normal vector Tilt angle TILT: rotation of the orientation in the direction of the path tangent.
- Angle of rotation THETA: rotation of the tool about itself with an additional third rotary axis acting as an orientation axis in 6-axis transformation.

Note

Orientation relative to the path not permitted in conjunction with OSC, OSS, OSSE, OSD and OST

Orientation interpolation relative to the path, that is ORIPATH or ORIPATHS and ORIOTC, cannot be programmed in conjunction with orientation characteristic smoothing with a G code from group 34. OSOF has to be active for this.

7.5.2 Rotation of the tool orientation relative to the path (ORIPATH, ORIPATHS, angle of rotation)

With a 6-axis transformation, the tool can be rotated about itself with a third rotary axis to orientate the tool as desired in space. With a rotation of the tool orientation relative to the path using ORIPATH or ORIPATHS, the additional rotation can be programmed via the THETA angle of rotation. Alternatively, the LEAD and TILT angles can be programmed using a vector, which is located in the plane vertical to the tool direction.

Machine manufacturer

Please refer to the machine manufacturer's instructions. The interpolation of the LEAD and TILT angles can be set differently using machine data.

Syntax

Rotation of tool orientation and tool

The type of tool orientation relative to the path is activated using ORIPATH or ORIPATHS.

`N... ORIPATH`	Activate type of orientation relative to the path
`N... ORIPATHS`	Activate type of orientation relative to the path with smoothing of the orientation characteristic

Activating the three angles that can be rotated:

`N... LEAD=`	Angle for the programmed orientation relative to the surface normal vector
`N... TILT=`	Angle for the programmed orientation in the plane, vertical to the path tangent relative to the surface normal vector
`N... THETA=`	Angle of rotation relative to the change of orientation in the tool direction of the third rotary axis

The values of the angles at the end of block are programmed using `LEAD=value`, `TILT=value` or `THETA=value`. In addition to the constant angles, polynomials can be programmed for all three angles up to the 5th degree.

`N... PO[PHI]=(a2, a3, a4, a5)`	Polynomial for the leading angle LEAD
`N... PO[PSI]=(b2, b3, b4, b5)`	Polynomial for the tilt angle TILT
`N... PO[THT]=(d2, d3, d4, d5)`	Polynomial for the angle of rotation THETA

The higher polynomial coefficients, which are zero, can be omitted when programming. Example: `PO[PHI]=a2` results in a parabola for the `LEAD` angle.

Meaning

Tool orientation relative to the path

`ORIPATH`:	Tool orientation in relation to path
`ORIPATHS`:	Tool orientation in relation to path, blips in the orientation characteristic are smoothed
`LEAD`:	Angle relative to the surface normal vector in the plane that is defined by the path tangent and the surface normal vector
`TILT`:	Rotation of orientation in the Z direction or rotation about the path tangent
`THETA`:	Rotation about the tool direction toward Z
`PO[PHI]`:	Orientation polynomial for the LEAD angle
`PO[PSI]`:	Orientation polynomial for the TILT angle
`PO[THT]`:	Orientation polynomial for the THETA angle of rotation

Note

Angle of rotation THETA

A 6-axis transformation is required to rotate a tool with a third rotary axis that acts as an orientation axis about itself.

7.5.3 Interpolation of the tool rotation relative to the path (ORIROTC, THETA)

Interpolation with rotation vectors

The rotation vector of the tool rotation, programmed with ORIROTC, relative to the path tangent can also be interpolated with an offset that can be programmed using the THETA angle of rotation. A polynomial can, therefore, be programmed up to the 5th degree for the offset angle using PO[THT].

Syntax

`N... ORIROTC`	Initiate the rotation of the tool relative to the path tangent
`N... A3= B3= C3= THETA=value`	Define the rotation of the orientation vector
`N... A3= B3= C3= PO[THT]=(c2, c3, c4, c5)`	Interpolate offset angle with polynomial up to 5th degree

A rotation can also be programmed in a separate block without an orientation change taking place.

Meaning

Interpolation of the rotation of tool relative to the path in 6-axis transformation

`ORIROTC`:	Initiate tangential rotation vector relative to path tangent
`THETA=value`:	Angle of rotation in degrees reached by the end of the block
`THETA=θe`:	Angle of rotation with end angle Θ_e of rotation vector
`THETA=AC(…)`:	Non-modal switchover to absolute dimensions
`THETA=IC(…)`:	Non-modal switchover to incremental dimensions
`PO[THT]=(c2, c3, c4, c5)`:	Interpolate offset angle with polynomial of 5th degree

Note

Interpolation of the rotation vector ORIROTC

Initiating rotation of the tool relative to the path tangent in the opposite direction to the tool orientation, is only possible with a 6-axis transformation.

With active ORIROTC

Rotation vector ORIROTA cannot be programmed. If programming is undertaken, ALARM 14128 "Absolute programming of tool rotation with active ORIROTC" is output.

Orientation direction of the tool for 3-axis to 5-axis transformation

The orientation direction of the tool can be programmed via Euler angles, RPY angles or direction vectors as with 3-axis to 5-axis transformations. Orientation changes of the tool in space can also be achieved by programming the large-circle interpolation ORIVECT, linear interpolation of the orientation axes ORIAXES, all interpolations on the peripheral surface of a taper ORICONxx, and interpolation in addition to the curve in space with two contact points of the tool ORICURVE.

G....:	Details of the rotary axis motion
X, Y, Z:	Details of the linear axes
ORIAXES:	Linear interpolation of machine or orientation axes
ORIVECT:	Large-circle interpolation (identical to ORIPLANE)
ORIMKS: ORIWKS:	Rotation in the machine coordinate system Rotation in the workpiece coordinate system Description, see the Rotations of the tool orientation section
A= B= C=:	Programming the machine axis position
ORIEULER:	Orientation programming via Euler angle
ORIRPY:	Orientation programming via RPY angle
A2= B2= C2=:	Angle programming of virtual axes
ORIVIRT1: ORIVIRT2:	Orientation programming using virtual orientation axes (definition 1), definition according to MD $MC_ORIAX_TURN_TAB_1 (definition 2), definition according to MD $MC_ORIAX_TURN_TAB_2
A3= B3= C3=:	Direction vector programming of direction axis
ORIPLANE:	Interpolation in the plane (large-circle interpolation)
ORICONCW:	Interpolation on the peripheral surface of a taper in the clockwise direction
ORICONCCW:	Interpolation on the peripheral surface of a taper in the counter-clockwise direction
ORICONTO:	Interpolation on the peripheral surface of a taper with tangential transition
A6= B6= C6=:	Programming of a rotary axis of the taper (normalized vector)
NUT=angle	Opening angle of taper in degrees
NUT=+179	Traverse angle less than or equal to 180 degrees
NUT=-181	Traverse angle greater than or equal to 180 degrees
ORICONIO:	Interpolation on the peripheral surface of a taper

A7= B7= C7=:	Intermediate orientation (programming as normalized vector)
ORICURVE XH YH ZH, e.g. with polynomials PO[XH]=(xe, x2, x3, x4, x5)	Interpolation of the orientation specifying a movement between two contact points of the tool. In addition to the end points, additional curve polynomials can also be programmed.

Note

If the tool orientation with active ORIAXES is interpolated via the orientation axes, the angle of rotation is only initiated relative to the path at the end of block.

7.5.4 Smoothing of orientation characteristic (ORIPATHS A8=, B8=, C8=)

Changes of orientation that take place with constant acceleration on the contour can cause unwanted interruptions to the path motions, particularly at the corner of a contour. The resulting blip in the orientation characteristic can be smoothed by inserting a separate intermediate block. If ORIPATHS is active during reorientation, the change in orientation occurs at a constant acceleration. The tool can be retracted in this phase.

Machine manufacturer

Please refer to the machine manufacturer's notes on any predefined machine and setting data used to activate this function.

Machine data can be used to set how the retracting vector is interpreted:

1. In the TCS, the Z coordinate is defined by the tool direction.
2. In the WCS, the Z coordinate is defined by the active plane.

For further explanations about the "Orientation relative to the path" function , see
References:
Function Manual, Special Functions; Multi-axis Transformations (F2)

Syntax

Further programming details are needed at the corner of the contour for constant tool orientations relative to the path as a whole. The direction and path length of this motion is programmed via the vector using the components A8=X, B8=Y C8=Z.

```
N... ORIPATHS A8=X B8=Y C8=Z
```

Meaning

ORIPATHS:	Tool orientation relative to the path; blip in orientation characteristic is smoothed
A8= B8= C8=:	Vector components for direction and path length
X, Y, Z:	Retracting movement in tool direction

Note

Programming direction vectors A8, B8, C8

If the length of this vector is exactly zero, no retracting movement is executed.

ORIPATHS

Tool orientation relative to the path is activated using ORIPATHS. The orientation is otherwise transferred from the start orientation to the end orientation by means of linear large-circle interpolation.

7.6 Compression of the orientation (COMPON, COMPCURV, COMPCAD, COMPSURF)

NC programs, in which orientation transformation (TRAORI) is active and tool orientations are programmed (no matter what type), can be compressed if kept within specified limits.

Programming

Tool orientation

If orientation transformation (TRAORI) is active, for 5-axis machines, tool orientation can be programmed in the following way (independent of the kinematics):

- Programming of the direction **vectors** via:
  `A3=<...> B3=<...> C3=<...>`
- Programming of the Euler**angles** or RPY-**angles** via:
  `A2=<...> B2=<...> C2=<...>`

Rotation of the tool

For **6-axis** machines you can program the tool rotation in addition to the tool orientation.

The angle of rotation is programmed with:

`THETA=<...>`

See " Rotation of tool orientation (Page 325) ".

Note

NC blocks, in which a rotation is also programmed, can only be compressed if the angle of rotation changes **linearly**. This means that it is not permissible that a polynomial with PO[THT]=(...) is programmed for the angle of rotation.

General structure of an NC block that can be compressed

The general structure of an NC block that can be compressed can therefore look like this:

`N... X=<...> Y=<...> Z=<...> A3=<...> B3=<...> C3=<...> THETA=<...> F=<...>`

or

```
N... X=<...> Y=<...> Z=<...> A2=<...> B2=<...> C2=<...> THETA=<...> F=<...>
```

Note

The position values can be entered directly (e.g. X90) or indirectly via parameter settings (e.g. X=R1*(R2+R3)).

Programming tool orientation using rotary axis positions

Tool orientation can be also specified using rotary axis positions, e.g. with the following structure:

```
N... X=<...> Y=<...> Z=<...> A=<...> B=<...> C=<...> THETA=<...> F=<...>
```

In this case, compression is executed in two different ways, dependent on whether large radius circular interpolation is executed. If no large radius circular interpolation takes place, then the compressed change in orientation is represented in the usual way by axial polynomials for the rotary axes.

Contour accuracy

Depending on the selected compression mode (MD20482 $MC_COMPRESSOR_MODE) either the configured axis-specific tolerances (MD33100 $MA_COMPRESS_POS_TOL) or the following channel-specific tolerances – set using setting data – are effective for the geometry axes and orientation axes for compression:

SD42475 $SC_COMPRESS_CONTUR_TOL (maximum contour deviation)

SD42476 $SC_COMPRESS_ORI_TOL (maximum angular deviation for tool orientation)

SD42477 $SC_COMPRESS_ORI_ROT_TOL (maximum angular deviation for the angle of rotation of the tool) (only available on 6-axis machines)

References:
Function Manual Basic Functions; 3 to 5-Axis Transformation (F2),
Section: "Compression of the orientation"

Activation/deactivation

Compressor functions are activated by modal G commands `COMPON`, `COMPCURV`, `COMPCAD` or `COMPSURF`.

`COMPOF` terminates the compressor function.

See " NC block compression (COMPON, COMPCURV, COMPCAD) (Page 239) ".

Note

Orientation motion is only compressed when large radius circular interpolation is active (i.e. tool orientation is changed in the plane which is determined by start and end orientation).

Large radius circular interpolation is executed under the following conditions:

- MD21104 $MC_ORI_IPO_WITH_G_CODE = 0,
  ORIWKS is active and
  the orientation is programmed as a vector (with A3, B3, C3 or A2, B2, C2).
- MD21104 $MC_ORI_IPO_WITH_G_CODE = 1 and
  ORIVECT or ORIPLANE is active.
  The tool orientation can be programmed either as a direction vector or with rotary axis positions. If one of the G-codes `ORICONxx` or `ORICURVE` is active, or if polynomials for the orientation angles (`PO[PHI]` and `PO[PSI]`) are programmed, no large circle interpolation will be executed.

Example

In the example program below, a circle approached by a polygon definition is compressed. The tool orientation moves on the outside of the taper at the same time. Although the programmed orientation changes are executed one after the other, but in an unsteady way, the compressor function generates a smooth motion of the orientation.

```
Programming                                  Comment
DEF INT NUMBER=60
DEF REAL RADIUS=20
DEF INT COUNTER
DEF REAL ANGLE
N10 G1 X0 Y0 F5000 G64

$SC_COMPRESS_CONTUR_TOL=0.05                 ; Maximum deviation of the contour = 0.05 mm
$SC_COMPRESS_ORI_TOL=5                       ; Maximum deviation of the orientation
                                             = 5 degrees


TRAORI
COMPCURV                                     ; The movement describes a circle generated
                                             from polygons. The orientation moves on a
                                             taper around the Z axis with an opening an-
                                             gle of 45 degrees.
N100 X0 Y0 A3=0 B3=-1 C3=1
N110 FOR COUNTER=0 TO NUMBER
N120 ANGLE=360*COUNTER/NUMBER
N130 X=RADIUS*cos(angle) Y=RADIUS*sin(angle)
A3=sin(angle) B3=-cos(angle) C3=1
N140 ENDFOR
```

7.7 Smoothing the orientation characteristic (ORISON, ORISOF)

The "Smoothing the orientation characteristic (ORISON)" function can be used to smooth oscillations affecting orientation over several blocks. The aim is to achieve a smooth characteristic for both the orientation and the contour.

Requirement

The "Smoothing the orientation characteristic (ORISON)" function is only available in systems with 5-/6-axis transformation.

Syntax

```
ORISON
...
ORISOF
```

Meaning

ORISON:	Smoothing of the orientation characteristic ON	
	Effective:	Modal
ORISOF:	Smoothing of the orientation characteristic OFF	
	Effective:	Modal

Setting data

The orientation characteristic is smoothed conforming to:

- A predefined maximum tolerance (maximum angular deviation of the tool orientation in degrees)
  and
- A predefined maximum path distance

These specifications are defined in setting data:

- SD42678 $SC_ORISON_TOL (tolerance for smoothing the orientation characteristic)
- SD42680 O$SC_ORISON_DIST (path distance for smoothing the orientation characteristic)

Example

Program code	Comment
...	
TRAORI()	; Activation of orientation transformation.
ORISON	; Activation of orientation smoothing.
$SC_ORISON_TOL=1.0	; Orientation tolerance smoothing = 1.0 degrees.
G91	

```
Program code                        Comment
X10 A3=1 B3=0 C3=1
X10 A3=-1 B3=0 C3=1
X10 A3=1 B3=0 C3=1
X10 A3=-1 B3=0 C3=1
X10 A3=1 B3=0 C3=1
X10 A3=-1 B3=0 C3=1
X10 A3=1 B3=0 C3=1
X10 A3=-1 B3=0 C3=1
X10 A3=1 B3=0 C3=1
X10 A3=-1 B3=0 C3=1
...
ORISOF                              ; Deactivation of orientation smoothing.
...
```

The orientation is pivoted through 90 degrees on the XZ plane from -45 to +45 degrees. Due to the smoothing of the orientation characteristic the orientation is no longer able to reach the maximum angle values of -45 or +45 degrees.

Further information

Number of blocks

The orientation characteristic is smoothed over a configured number of blocks set in machine data MD28590 $MC_MM_ORISON_BLOCKS.

Note

If smoothing of the orientation characteristic is activated with `ORISON` without sufficient block memory having been configured for it (MD28590 < 4), an alarm signal will be output and the function cannot be executed.

Maximum block path length

The orientation characteristic is only smoothed in blocks whose traversing distance is shorter than the configured maximum block path length (MD20178 $MC_ORISON_BLOCK_PATH_LIMIT). Blocks with longer traversing distances interrupt smoothing and are traversed as programmed.

7.8 Kinematic transformation

7.8.1 Activate face end transformation (TRANSMIT)

The front face transformation (TRANSMIT) is activated in the part program or synchronized action using the `TRANSMIT` statement.

Syntax

```
TRANSMIT
TRANSMIT(<n>)
```

Meaning

`TRANSMIT`:	Activate TRANSMIT with the first TRANSMIT data set
`TRANSMIT(n)`:	Activate TRANSMIT with the nth TRANSMIT data set

Note

A TRANSMIT transformation active in the channel is activated with:

- Deactivate transformation: `TRAFOOF`
- Activation of another transformation: E.g. `TRACYL`, `TRAANG`, `TRAORI`

7.8.2 Activate cylinder surface transformation (TRACYL)

The cylinder surface transformation (TRACYL) is activated in the part program or synchronized action using the `TRACYL` statement.

Syntax

```
TRACYL(<d>)
TRACYL(<d>,<n>)
TRACYL(<d>,<n>,<k>)
```

Meaning

`TRACYL(<d>)`:	Activate TRACYL with the first TRACYL data set and working diameter <d>	
`TRACYL (<d>,<n>)`:	Activate TRACYL with the <n>th TRACYL data set and working diameter <d>	
`<d>`:	Reference or working diameter The value must be greater than 1.	
`<n>`:	TRACYL data set number (optional)	
	Range of values:	1, 2

<k>:	The parameter <k> is only relevant for transformation type 514	
	k = 0:	without groove side correction
	k = 1:	with groove side correction
	If the parameter is not specified, then the parameterized basic position applies: $MC_TRACYL_DEFAULT_MODE_<n> With <n> = TRACYL data set number	

Note

A TRACYL transformation active in the channel is switched-off with:

- Deactivate transformation: `TRAFOOF`
- Activation of another transformation: E.g. `TRAANG`, `TRANSMIT`, `TRAORI`

Example

Program code	Comment
...	
N40 TRACYL(40.)	; Activate TRACYL with the first TRACYL data set and working diameter 40 mm.
...	

Further information

Program structure

A part program for milling a groove with TRACYL transformation 513 (TRACYL with groove side offset) generally comprises the following steps:

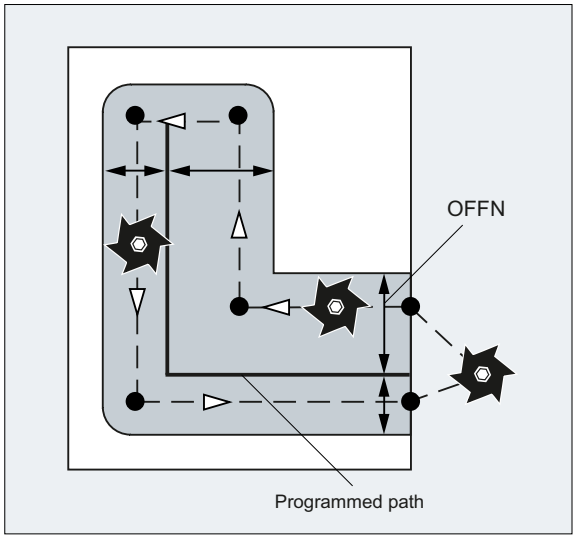
1. Select tool.
2. Select TRACYL.
3. Select suitable coordinate offset (frame).
4. Positioning.
5. Program OFFN.
6. Select TRC.
7. Approach block (position TRC and approach groove side).
8. Groove center line contour.
9. Deselect TRC.
10. Retraction block (retract TRC and move away from groove side).
11. Positioning.
12. TRAFOOF.
13. Reselect original coordinate shift (frame).

Contour offset (OFFN)

In order to mill grooves using TRACYL transformation 513, the center line of the groove and **half of the groove width** via the OFFN address are programmed in the part program.

To avoid damage to the groove side, OFFN acts only when the tool radius compensation is active.

It is possible to change OFFN within a part program. This allows the groove center line to be offset from the center:



Note

OFFN should be at least as large as the tool radius to avoid damage occurring to the opposite side of the groove wall.

Note

OFFN acts differently with TRACYL than it does without TRACYL. Since, even without TRACYL, OFFN is included when TRC is active, OFFN should be reset to zero after TRAFOOF.

NOTICE
Effect of OFFN depends on the transformation type
For TRACYL transformation 513 (TRACYL with groove side offset), half the groove width is programmed for OFFN.
For TRACYL transformation 512 (TRACYL with groove side offset), the value of OFFN acts as an allowance for the TRC.

Tool radius compensation (TRC)

For TRACYL transformation 513, the TRC is not taken into account relative to the groove side, but to the programmed center of the groove. In order that the tool travels to the left of the groove side, statement G42 must be programmed instead of G41 or the value of OFFN specified with a negative sign.

Tool diameter

With TRACYL and a tool whose diameter is less than the groove width, the same groove side geometry is not generated as with a tool whose diameter is the same as the groove width. To improve the precision, it is recommended that the tool diameter is selected to be only slightly less than the groove width.

Axis utilization

Note

The following axes cannot be used as a positioning axis or a reciprocating axis:

- The geometry axis in the peripheral direction of the cylinder peripheral surface (Y axis).
- The additional linear axis for groove side compensation (Z axis).

7.8.3 Activate skew angle transformation (TRAANG)

The oblique angle transformation (TRAANG) is activated in the part program or synchronized action using the `TRAANG` statement.

Syntax

```
TRAANG
TRAANG()
TRAANG(, <n>)
TRAANG(<α>)
TRAANG(<α>,<n>)
```

Meaning

`TRAANG`/`TRAANG()`:	Activate TRAANG with the first TRAANG data set and last valid angle <α>	
`TRAANG(, <n>)`:	Activate TRAANG with the <n>th TRAANG data set and last valid angle <α>	
`TRAANG(<α>)`:	Activate TRAANG with the first TRAANG data set and angle <α>	
`TRAANG(<α>,<n>)`:	Activate TRAANG with the <n>th TRAANG data set and angle <α>	
<α>:	Angle of the inclined axis (optional)	
	Range of values:	-90° < α < + 90°
	If the parameter is not specified, then the parameterized basic position applies: $MC_TRAANG_ANGLE_<n> With <n> = TRAANG data set number	
<n>:	TRAANG data set number (optional)	
	Range of values:	1, 2

Note

A TRAANG transformation active in the channel is switched-off with:

- Deactivate transformation: TRAFOOF
- Activation of another transformation: E.g. TRACYL, TRANSMIT, TRAORI

Example

```
Program code          Comment
...
N20 TRAANG(45)        ; Activate TRAANG with the first TRAANG data set and angle 45°
...
```

7.8.4 Oblique plunge-cutting on grinding machines (G5, G7)

The G commands G7 and G5 are used to simplify programming of oblique plunge-cutting on grinding machines with "inclined axis (TRAANG)", so that when plunge cutting, only the inclined axis is traversed.

Only the required end position of the plunge-cutting motion has to be programmed in X and Z. For G7, starting from the actual position of the X axis, the NC calculates and approaches the programmed end position and angle α of the inclined axis.

The starting position is calculated from the point where the two straight lines intersect:

- Straight line parallel to the Z axis, at a distance from the actual position of the X axis
- Straight line parallel to the inclined axis through the programmed end position

With the subsequent G5, the inclined axis is traversed to the programmed end position.

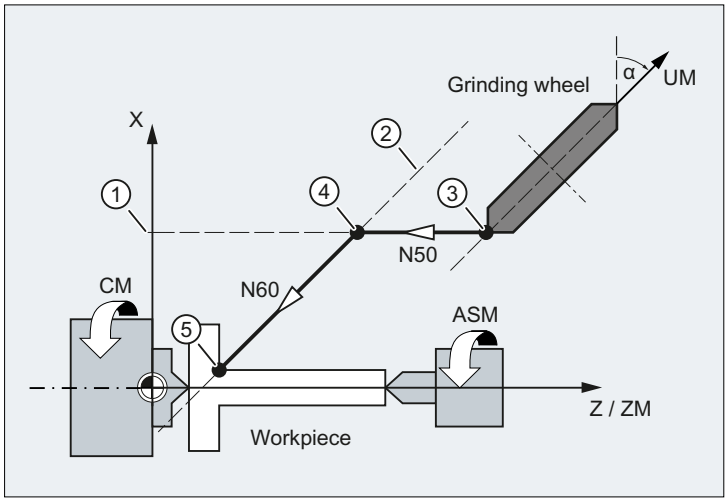
Syntax

```
G7 <Endpos_X> <Endpos_Z>
G5 <Endpos_X>
```

Meaning

G7:	Calculate the starting position for the oblique plunge-cutting and approach.
G5:	Traverse the inclined axis to the programmed end position
<Endpos_X>:	X axis end position
<Endpos_Z>:	End position of the Z axis

Example



① Parallel to the Z axis, at a distance from the actual position of the X axis
② Parallel to the inclined axis through the programmed end position
③ Starting position
④ Plunge-cutting: Starting position
⑤ Plunge-cutting: End position
X Geometry axis
Z Geometry axis
ZM Machine axis
UM Machine axis

```
Program code                    Comment
N... G18                        ; Select XZ plane.
N40 TRAANG (45.0)               ; Activate TRAANG transformation, angle = 45°
N50 G7 X40 Z70 F4000            ; Calculate the starting position and approach
N60 G5 X40 F100                 ; Traverse inclined axis to the end position.
N70 ...
```

7.9 Activate concatenated transformation (TRACON)

The TRAANG transformation is activated in the part program or synchronized action using the `TRACON` statement.

Syntax

```
TRACON(<Trafo_No>,<Par_1>,<Par_2,...)
...
TRAFOOF
```

Meaning

TRACON:	Activate concatenated transformation If another transformation was previously activated, it is implicitly deactivated by TRACON().		
<Trafo_No>:	Number of the concatenated transformation:		
	Type:	INT	
	Range of values:	0 ... 2	
	Value:	0, 1	First/only concatenated transformation
		2	Second concatenated transformation
		Not specified	Same meaning as with 0 or 1
		Note: Values not equal to 0, 1, 2 generate an error alarm.	
<Par_1>,<Par_2,...:	Parameters for the concatenated transformations (e.g. angle of the inclined axis) If parameters are not set, the defaults or the parameters last used take effect. Commas must be used to ensure that the specified parameters are evaluated in the sequence in which they are expected, if default settings are to be effective for previous parameters. In particular, a comma is required before at least one parameter, even though it is not necessary to specify <Trafo_No>. For example TRACON(, 3.7).		
TRAFOOF:	Deactivate the last activated (concatenated) transformation		

Example

```
Program code                    Comment
...
N230 TRACON(1,45.)              ; Activate first concatenated transformation.
                                ; The previously active transformation is automatically deselected.
                                ; The parameter for the inclined axis is 45°.
...
N330 TRACON(2,40.)              ; Activate second concatenated transformation.
                                ; The parameter for the inclined axis is 40°.
...
N380 TRAFOOF                    ; Deactivate second concatenated transformation.
...
```

7.10 Cartesian PTP travel

7.10.1 Activating/deactivating Cartesian PTP travel (PTP, PTPG0, PTPWOC, CP)

The Cartesian point-to-point or PTP travel is activated/deactivated in the NC program using G group 49 commands.

The commands are modal. The default setting is travel with Cartesian path motion (CP).

Contrary to CP, for active PTP travel, only the Cartesian target point is transformed, and the machine axes are traversed in synchronism.

In order that the Cartesian target point can be uniquely converted into machine axis values, in addition to position and angular data, information is also necessary that identifies the axis positions. This data is retrieved from the adjustable addresses STAT (Page 345) andTU (Page 349).

Precondition

TRAORI/TRANSMIT transformation is active.

Syntax

```
PTP / PTPG0 / PTPWOC
...
CP
```

Meaning

PTP:	Activating point-to-point motion PTP The programmed Cartesian position in G0 and G1 blocks is approached with synchronous axis motion.
PTPG0:	Activating point-to-point motion PTPG0 Only in G0 blocks is the programmed Cartesian position approached with synchronous axis motion. In G1 blocks, a switchover is made to CP path motion.
PTPWOC:	Activating PTPWOC point-to-point motion Just the same as PTP, however, without any compensationary motion, which is caused by motion of rotary axes and orientation axes.
CP:	Deactivating point-to-point motion and activating path motion CP Cartesian path motion is executed with CP.

Examples

See:

- Example 1: PTP and TRAORI (Page 351)
- Example 2: PTPG0 and TRANSMIT (Page 352)

7.10.2 Specify the position of the joints (STAT)

Position data with Cartesian coordinates and specification of the tool orientation is not sufficient in order to uniquely identify the machine position, as for the same tool orientation, several joint positions are possible. Depending on the kinematics involved, there can be as many as 8 different joint positions. These different joint positions are transformation-specific.

In an order to avoid any ambiguity, the joint positions are specified under the STAT address.

Note

The control only takes into account programmed STAT values for PTP motion. CP motion is ignored because when traversing with active transformation, a position change is normally not possible. When traversing with active CP, the position for the target point is taken from the starting point.

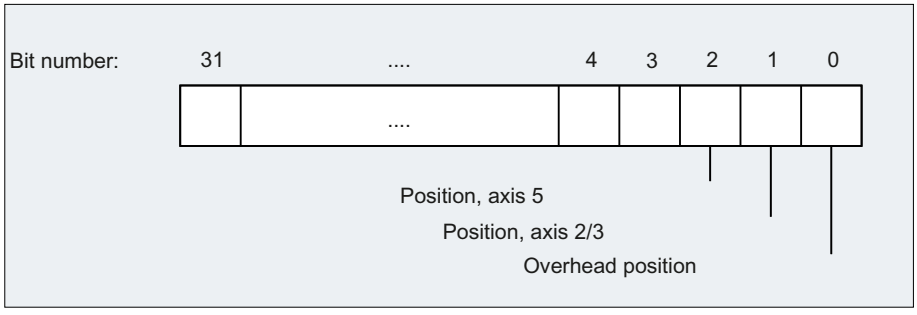
Syntax

```
STAT=<Value>
```

Meaning

`STAT`:	Adjustable address to specify joint positions
`<Value>`:	Binary or decimal value Contains one bit for each possible position. The significance of the bits is defined by the particular transformation.

The following assignments apply for transformations, which are included in the "Handling transformation package":



The meaning of the bits will be demonstrated using an example of a robot with a 6-axis joint kinematics:

Bit 0	Position of the intersection points of the manual axes (A4, A5, A6)	
	= 0	Basic range The robot is in the basic range if the X value of the intersection point of the manual axes is positive referred to the A1 coordinate system.
	= 1	Overhead range The robot is in the overhead range if the X value of the intersection point of the manual axes is negative referred to the A1 coordinate system.

Bit 1	Position of axis 3 The angle where the value of bit 1 changes depends on the particular robot type. The following applies for robots whose axes 3 and 4 intersect:	
	= 0	A3 < 0°
	= 1	A3 ≥ 0°
	Note: For robots with an offset between axes 3 and 4, the angle where the value of bit 1 changes is dependent on the magnitude of this offset.	
Bit 2	Position of axis 5	
	= 0	A5 ≤ 0°
	= 1	A5 > 0°
Bit 3	Is not used. Value is always 0.	
Bit 4	Specifies as to whether the point was taught in with a robot with absolute precision – or not.	
	= 0	The point was not taught in with a robot with absolute precision.
	= 1	The point was taught in with a robot with absolute precision.
	Note: Independent of the value of the bit, the point can be approached with robots that have absolute precision – as well as robots that do not have absolute precision. Bit 4 is for information purposes only, and has no influence as to how the robot control calculates the point. Bit 4 can therefore be neglected if a robot is programmed offline.	

TRANSMIT

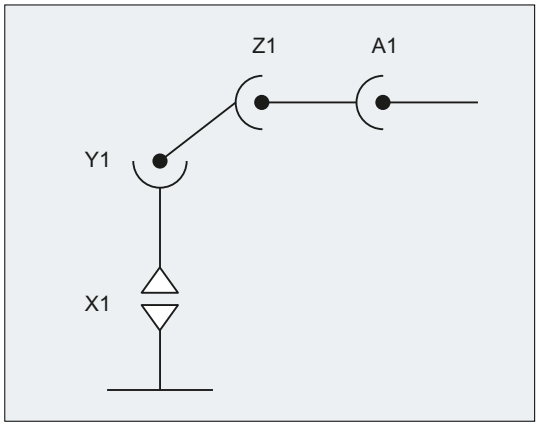
For TRANSMIT, address STAT is used to resolve the ambiguity regarding the pole.

If the rotary axis must rotate through 180º or for CP, the contour would go through the pole, the following applies:

Bit 0	Only relevant for $MC_TRANSMIT_POLE_SIDE_FIX_1/2 = 1 or 2:	
	= 0	Rotary axis traverses through +180º or rotates clockwise.
	= 1	Rotary axis rotates through -180º or rotates counterclockwise.
Bit 1	Only relevant for $MC_TRANSMIT_POLE_SIDE_FIX_1/2 = 0:	
	= 0	The axis traverses through the pole. The rotary axis does not rotate.
	= 1	The axis rotates around the pole. Bit 0 of STAT is relevant.

Examples

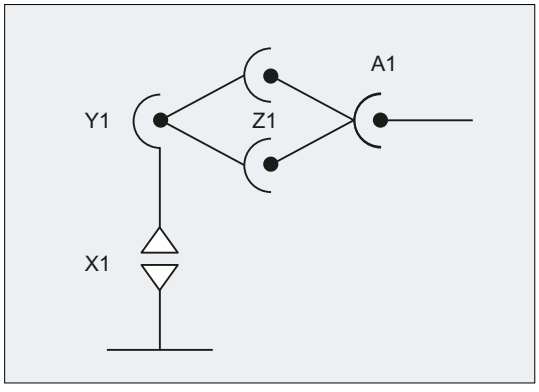
The following diagrams show examples for the ambiguity as a result of different joint positions for 6-axis joint kinematics. Joint positions only become clear and unique as a result of the STAT data.



Ambiguity, overhead range

Position 1:
X1 = 0°
Y1 = 45°
Z1 = 45°

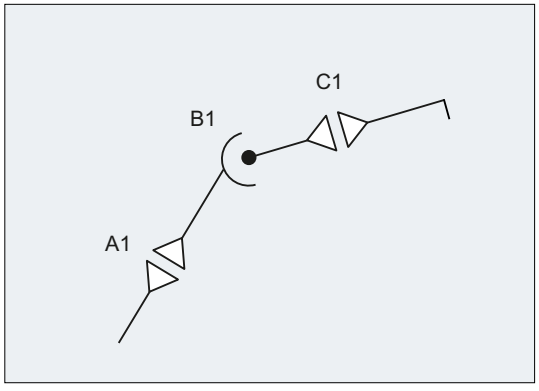
Position 2:
X1 = -180°
Y1 = -45°
Z1 = -45°
STAT = 'B01'



Ambiguity, elbow up or down

Position 1:
Y1 = -30°
Z1 = 60°
A1 = -30°
STAT = 'B00'

Position 2:
Y1 = 30°
Z1 = -60°
A1 = 30°
STAT = 'B10'



Ambiguity, axis B1

Position 1:
A1 = 0°
B1 = 40°
C1 = 0°
STAT = 'B00'

Position 2:
A1 = 180°
B1 = -40°
C1 = 180°
STAT = 'B100'

7.10.3 Specify the sign of the axis angle (TU)

In order that rotary axes can also approach axis angles exceeding +180° or less than -180° without requiring a special traversing strategy (e.g. intermediate point), the sign of the axis angle must be specified under the adjustable address TU.

Note

The control only takes into account programmed TU values for PTP motion. CP motion is ignored.

Syntax

```
TU=<Value>
```

Meaning

`TU`:	Adjustable address to specify axis angle signs		
`<Value>`:	Binary or decimal value For each axis that is involved in the transformation, there is a bit that indicates the sign of the axis angle (θ), and therefore the traversing direction.		
	Bit	= 0	0° ≤ θ < 360°
		= 1	- 360° < θ < 0°

The meaning of the bits will be demonstrated using an example of a robot with a 6-axis joint kinematics:

Bit 0	Sign of axis angle A1	
	= 0	A1 ≥ 0°
	= 1	A1 < 0°
Bit 1	Sign of axis angle A2	
	= 0	A2 ≥ 0°
	= 1	A2 < 0°
Bit 2	Sign of axis angle A3	
	= 0	A3 ≥ 0°
	= 1	A3 < 0°
Bit 3	Sign of axis angle A4	
	= 0	A4 ≥ 0°
	= 1	A4 < 0°
Bit 4	Sign of axis angle A5	
	= 0	A5 ≥ 0°
	= 1	A5 < 0°
Bit 5	Sign of axis angle A6	
	= 0	A6 ≥ 0°
	= 1	A6 < 0°

TU=19 (corresponds to TU='B010011) would therefore signify:

Bit	Value		Axis angle
0	= 1	①	A1 < 0°
1	= 1	①	A2 < 0°
2	= 0	①	A3 ≥ 0°
3	= 0	①	A4 ≥ 0°
4	= 1	①	A5 < 0°
5	= 0	①	A6 ≥ 0°

Note

In the case of axes with a traversing range > ±360°, the axis always moves along the shortest path because the axis position cannot be specified uniquely by the TU information.

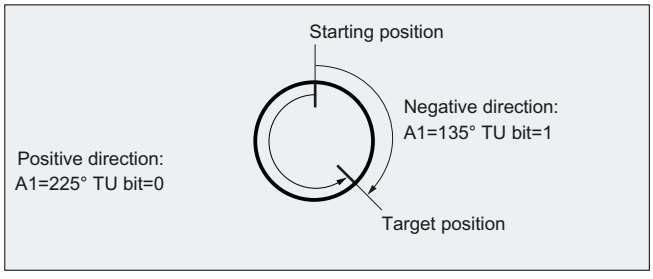
If no TU is programmed for a position, the axis always traverses via the shortest possible route.

TRANSMIT

For PTP travel with TRANSMIT active, the address of TU has no meaning!

Example

The rotary axis position shown in the following diagram can be approached in the negative or positive direction. The angular position is programmed under address A1. The traversing direction is only absolutely clear when TU is specified.



7.10.4 Example 1: PTP and TRAORI

Example 1

A sequence from a program example for the following kinematics is listed below:

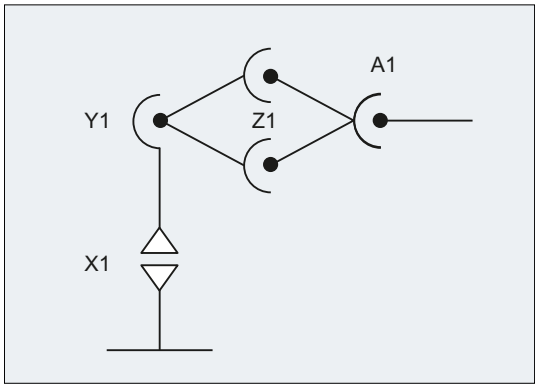


Figure 7-2 Elbow up or down

Program code	Comment
N10 G0 X0 Y-30 Z60 A-30 F10000	; Initial position → elbow up
N20 TRAORI(1)	; Transformation ON
N30 X1000 Y0 Z400 A0	
N40 X1000 Z500 A0 STAT='B10' TU='B100' PTP	; Re-orientation without transformation
	; → elbow down
N50 X1200 Z400 CP	; Transformation active again
N60 X1000 Z500 A20	
N70 M30	

Example 2

PTP travel for generic 5-axis transformation

Assumption: Right-angled CA kinematics used as basis.

Program code	Comment
TRAORI	;Transformation CA kinematics ON
PTP	; Activate PTP traversal
N10 A3=0 B3=0 C3=1	; rotary axis positions C=0 A=0
N20 A3=1 B3=0 C3=1	; rotary axis positions C=90 A=45
N30 A3=1 B3=0 C3=0	; rotary axis positions C=90 A=90
N40 A3=1 B3=0 C3=1 STAT=1	; rotary axis positions C=270 A=-45

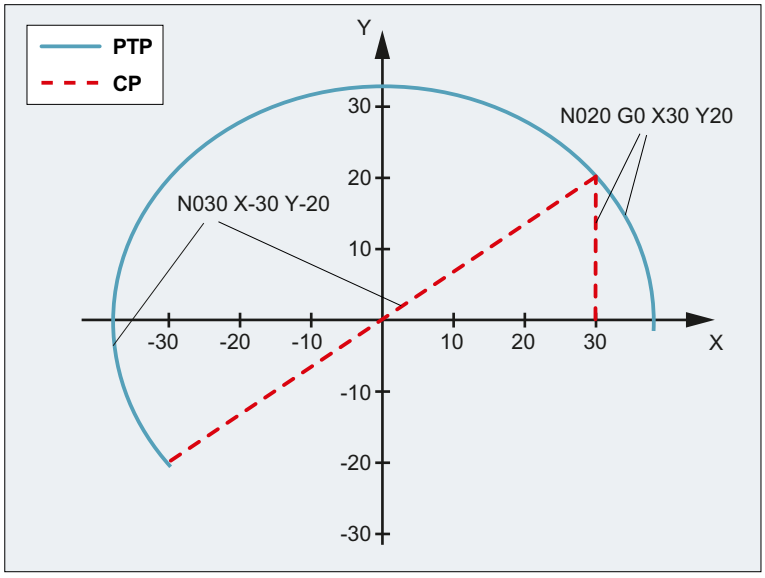
Select clear approach position of rotary axis position:

In block N40, the rotary axes – as a result of the programming of STAT=1 – travel the longer distance from their start point (C=90, A=90) to the end point (C=270, A=–45). On the other hand, with STAT=0, the rotary axes would travel along the shortest path to the end point (C=90, A=45).

7.10.5 Example 2: PTPG0 and TRANSMIT

Example 1

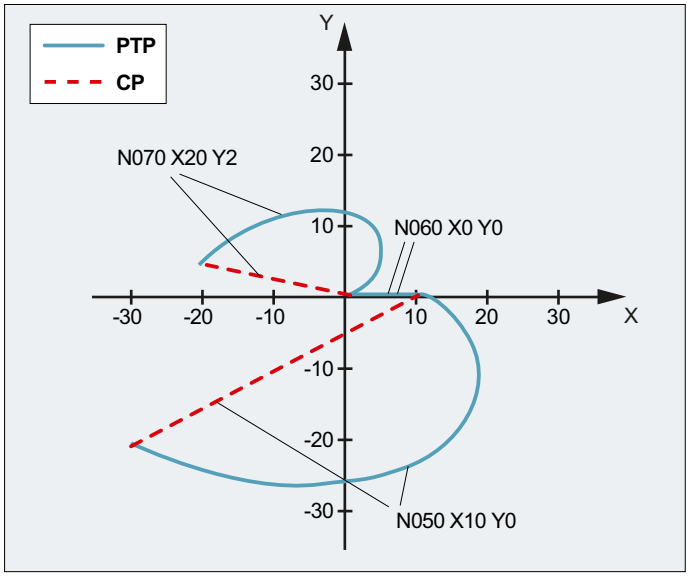
Traversing around the pole with PTPG0 and TRANSMIT



Program code	Comment
N001 G0 X30 Z0 F10000 T1 D1 G90	;Initial setting absolute dimension
N002 SPOS=0	
N003 TRANSMIT	;TRANSMIT transformation
N010 PTPG0	; for each G0 block, automatically PTP – and then CP again.
N020 G0 X30 Y20	
N030 X-30 Y-20	
N120 G1 X30 Y20	
N110 X30 Y0	
M30	

Example 2

Traversing from the pole with PTPG0 and TRANSMIT



Programming	Comment
N001 G0 X90 Z0 F10000 T1 D1 G90	;Initial setting
N002 SPOS=0	
N003 TRANSMIT	;TRANSMIT transformation
N010 PTPG0	; for each G0 block, automatically PTP – and then CP again.
N020 G0 X90 Y60	
N030 X-90 Y-60	
N040 X-30 Y-20	
N050 X10 Y0	
N060 X0 Y0	
N070 X-20 Y2	
N170 G1 X0 Y0	
N160 X10 Y0	
N150 X-30 Y-20	
M30	

7.11 Constraints when selecting a transformation

Function

Transformations can be selected via a part program or MDA. Please note:

- No intermediate movement block is inserted (chamfer/radii).
- Spline block sequences must be excluded; if not, a message is displayed.

- Fine tool compensation must be deselected (FTOCOF); if not a message is displayed.
- Tool radius compensation must be deselected (G40); if not a message is displayed.
- An activated tool length offset is included in the transformation by the control.
- The control deselects the current frame active before the transformation.
- The control deselects an active operating range limit for axes affected by the transformation (corresponds to WALIMOF).
- Protection zone monitoring is deselected.
- Continuous path control and rounding are interrupted.
- All the axes specified in the machine data must be synchronized relative to a block.
- Axes that are exchanged are exchanged back; if not, a message is displayed.
- A message is output for dependent axes.

Tool change

Tools may only be changed when the tool radius compensation function is deselected.

A change in tool length offset and tool radius compensation selection/deselection must not be programmed in the same block.

Frame change

All statements, which refer exclusively to the base coordinate system, are permissible (FRAME, tool radius compensation). However, a frame change with G91 (incremental dimension) – unlike with an inactive transformation – is not handled separately. The increment to be traveled is evaluated in the workpiece coordinate system of the new frame – regardless of which frame was effective in the previous block.

Exceptions

Axes affected by the transformation cannot be used

- as a preset axis (alarm),
- for approaching a checkpoint (alarm),
- for referencing (alarm).

7.12 Deselecting a transformation (TRAFOOF)

The predefined TRAFOOF procedure deactivates all active transformations and frames.

Note

For deselecting the transformation, the same secondary conditions (Page 353) apply as for selecting.

Frames required after this must be activated by renewed programming.

Syntax

```
...
TRAFOOF
```

Meaning

TRAFOOF:	Deactivating all active transformations/frames

Kinematic chains 8

8.1 Deletion of components (DELOBJ)

The `DELOBJ()` function "deletes" components by resetting the assigned system variables to their default values:

- Elements from kinematic chains
- Protection areas, protection area elements and collision pairs
- Transformation data

Syntax

```
[<RetVal>=] DELOBJ(<CompType>[,,,<NoAlarm>)])
[<RetVal>=] DELOBJ(<CompType>,<Index1>[,,<NoAlarm>])
[<RetVal>=] DELOBJ(<CompType>[,<Index1>][,<Index2>][,<NoAlarm>])
```

Meaning

DELOBJ:	Deletion of elements from kinematic chains, protection areas, protection area elements, collision pairs and transformation data
<CompType>:	Component type to be deleted
	Data type: STRING
	Value: "KIN_CHAIN_ELEM" **Meaning**: System variables of all kinematic elements: $NK_...
	Value: "KIN_CHAIN_SWITCH" **Meaning**: System variable $NK_SWITCH[<i>]
	Value: "KIN_CHAIN_ALL" **Meaning**: All kinematic elements and switches. Is the same as the successive call of DELOBJ with "KIN_CHAIN_ELEM" and "KIN_CHAIN_SWITCH"
	Value: "PROT_AREA" **Meaning**: System variables of the protection areas: • $NP_PROT_NAME • $NP_CHAIN_NAME • $NP_CHAIN_ELEM • $NP_1ST_PROT
	Value: "PROT_AREA_ELEM" **Meaning**: System variables of the protection area elements of machine protection areas and/or automatic tool protection areas: • $NP_NAME • $NP_NEXT • $NP_NEXTP • $NP_COLOR • $NP_D_LEVEL • $NP_USAGE • $NP_TYPE • $NP_FILENAME • $NP_PARA • $NP_OFF • $NP_DIR • $NP_ANG
	Value: "PROT_AREA_COLL_PAIRS" **Meaning**: System variables of the collision pairs: • $NP_COLL_PAIR • $NP_SAFETY_DIST
	Value: "PROT_AREA_ALL" **Meaning**: All protection areas, protection area elements and collision pairs (system variable $NP_...) Is the same as the successive call of DELOBJ with "PROT_AREA," "PROT_AREA_ELEM," and "PROT_AREA_COLL_PAIRS"
	Value: "TRAFO_DATA" **Meaning**: System variables of all transformations $NT_...

<Index1>:	Index of the first component to be deleted (**optional**)	
	Data type:	INT
	Default value:	-1
	Range of values:	-1 ≤ x ≤ (maximum number of configured components -1)
	Value	**Meaning**
	0, 1, 2, ...	Index of the component to be deleted.
	-1	All components of the specified type are deleted. <Index2> is not evaluated.
<Index2>:	Index of the last components to be deleted (**optional**) If <Index2> is not programmed, only the system variables of the component referenced in <Index1> are deleted.	
	Data type:	INT
	Default value:	Only the system variables of the component referenced in <Index1> are deleted.
	Range of values:	<Index1> < x ≤ (max. number of configured components -1)
<NoAlarm>:	Alarm suppression (**optional**)	
	Data type:	BOOL
	Default value:	FALSE
	Value	**Meaning**
	FALSE	In the event of an error (<RetVal> < 0), program processing is stopped and an alarm displayed.
	TRUE	In the event of an error, the program processing is not stopped and no alarm displayed. Application: User-specific reaction corresponding to the return value
<RetVal>:	Function return value	
	Data type:	INT
	Range of values:	0, -1, -2, ... -7
	Value	**Meaning**
	0	No error occurred.
	-1	Call of the function without parameters. At least parameter <CompType> must be specified.
	-2	<CompType> identifies an unknown component
	-3	<Index1> is less than -1
	-4	<Index1> is greater than the configured number of components
	-5	<Index1> has a value not equal to -1 when deleting a component **group**
	-6	<Index2> is less than <Index1>
	-7	<Index2> is greater than the configured number of components

8.2 Index determination by means of names (NAMETOINT)

User-specific names are entered in the system variable arrays of type STRING. Based on the identifier of the system variables and the name, the `NAMETOINT()` function determines the index value belonging to the name under which it is stored in the system variable array.

Syntax

```
<RetVal> = NAMETOINT(<SysVar>,<Name>[,<NoAlarm>])
```

Meaning

`NAMETOINT`:	Determining the system variable index	
`<SysVar>`:	Name of the system variable array of typeSTRING	
	Data type:	STRING
	Range of values:	Name of all NC system variable arrays of type STRING
`<Name>`:	Character string or name for which the system variable index is to be determined.	
	Data type:	STRING
`<NoAlarm>`:	Alarm suppression (**optional**)	
	Data type:	BOOL
	Default value:	FALSE
	Value	**Meaning**
	TRUE	In the event of an error, the program processing is not stopped and no alarm displayed. Application: User-specific reaction corresponding to the return value
	FALSE	In the event of an error (<RetVal> < 0), program processing is stopped and an alarm displayed.
`<RetVal>`:	System variable index or error message	
	Data type:	INT
	Range of values:	-1 ≤ x ≤ (max. number of configured components -1)
	Value	**Meaning**
	≥ 0	The sought name has been found under the specified system variable index.
	-1	The sought name has not been found or an error has occurred.

Example

```
Program code                                                    Comment
DEF INT INDEX
$NP_PROT_NAME[27]="Cover"
...
INDEX = NAMETOINT("$NP_PROT_NAME","Cover")                      ; INDEX == 27
```

Collision avoidance with kinematic chains 9

Note

Protection areas

The protection areas specified in the following chapters refer to the "Geometric machine modeling" function

References:
Function Manual, Special Functions, Chapter "Geometric machine modeling"

9.1 Check for collision pair (COLLPAIR)

In collision avoidance with kinematic chains, the COLLPAIR function can be used to determine whether two protection zones form a collision pair, i.e. whether they have to be monitored for collision when both are active.

Syntax

```
[<RetVal>=]COLLPAIR(<Name_1>,<Name_2>[,<NoAlarm>)])
```

Meaning

COLLPAIR:	Check whether part of a collision pair		
<RetVal>:	Function return value		
	Data type:	INT	
	Value:	≥ 0	The two protection zones form a collision pair. Return value == collision pair index m
		-1	Either two strings have not been specified or at least one of the two is the zero string.
		-2	The protection zone specified in the first parameter has not been found.
		-3	The protection zone specified in the second parameter has not been found.
		-4	Neither of the two specified protection zones has been found.
		-5	Both specified protection zones have been found, but not together in a collision pair.
<Name_1>:	Name of the first protection zone		
	Data type:	STRING	
	Range of values:	Parameterized protection zone names	

<Name_2>:	Name of the second protection zone		
	Data type:	STRING	
	Range of values:	Parameterized protection zone names	
<NoAlarm>:	Alarm suppression (optional)		
	Data type:	BOOL	
	Value:	FALSE	In the event of an error (<RetVal> < 0), the program processing is stopped and an alarm displayed.
		TRUE	In the event of an error, the program processing is not stopped and no alarm displayed. Application: User-specific reaction corresponding to the return value
	Default value:	FALSE	

References

Function Manual, Special Functions; Section K9: Collision avoidance

9.2 Requesting a recalculation of the collision model (PROTA)

The collision model can be changed by the writing of system parameters for kinematic chains, geometric machine modeling and/or collision avoidance in the part program or via the OPI. For the changes to take effect, a recalculation of the model must be requested with the PROTA procedure.

Protection zone status

With the call of the PROTA procedure, a recalculation of the collision model is performed **without** changing the protection zone status. A **newly** defined protection zone is then in the initialization status. A protection zone whose name has been changed is **not** considered to be newly defined. Its status remains unchanged.

With the call of the PROTA procedure with parameter "R", a recalculation of the collision model is performed **with** change of the protection zone status. All protection zones are reset to their respective initialization status corresponding to $NP_INIT_STAT.

Note

Tool change and automatically created tool protection zones

After a tool change, the collision model is implicitly recalculated for automatically created tool protection zones.

Syntax

```
PROTA
PROTA(<Par>)
```

Meaning

PROTA:	Request for a recalculation of the collision model	
	Preprocessing stop:	Yes
<Par>:	Parameter	
	Data type:	STRING
	Range of values:	"R" With the recalculation of the collision model, all protection zones are reset to their respective initialization status.

Supplementary conditions

Simulation

The PROTA procedure must not be used in part programs in conjunction with the simulation (simNCK).

Example:

```
Program code                    Comment
...
IF $P_SIM == FALSE              ; IF simulation not active
   PROTA                        ; THEN recalculate collision model
ENDIF                           ; ENDIF
...
```

References

Function Manual, Special Functions; Section K9: Collision avoidance

9.3 Setting the protection zone status (PROTS)

The status of protection zones can be set for collision avoidance with kinematic chains with the PROTS procedure.

Syntax

```
PROTS(<State>)
PROTS(<State>,<Name_1>)
PROTS(<State>,<Name_1>,...,<Name_n>)
```

Meaning

PROTS:	Setting the protection zone status		
	Preprocessing stop:	No	
	Alone in the block:	Yes	
<State>:	Status to which the specified protection zones are to be set		
	Data type:	CHAR	
	Value:	"A"or "a"	Status: Active
		"I"or "i"	Status: Inactive
		"P"or "p"	Status: Preactivated (activation performed via NC/PLC interface)
		"R"or "r"	Status = $NP_INIT_STAT (initialization status)
<Name_1> ... <Name_n>:	Name of one or more protection zones that are to be set to the specified status **Note:** The maximum number of protection zones that can be specified as parameters depends only on the maximum possible number of characters per program line.		
	Data type:	STRING	
	Range of values:	Parameterized protection zone names **Note:** If no name is specified, the specified status is set for all defined protection zones.	

References

Function Manual, Special Functions; Section K9: Collision avoidance

9.4 Determining the clearance of two protection zones (PROTD)

The clearance of two protection areas can be calculated for collision avoidance with kinematic chains with the PROTD function.

Function properties:

- The clearance calculation is performed independent of the protection area status (activated, deactivated, preactivated).
- The clearance calculation is performed with the positions valid at the end of the previous block.
- Overlays that are included in the main run calculation (e.g. DRF offset or external zero offset) are included in the clearance calculation with the values valid at the function **interpretation time**.

Note

Synchronization

If the PROTD function is used, it is the sole responsibility of the user to synchronize the main run and preprocessing, if required, with the STOPRE preprocessing stop.

Function value for collision

There is a collision between two protection areas when the following applies for the clearance:

- 0.0 < clearance < MD10619 $MN_COLLISION_TOLERANCE (tolerance for collision check)

If there is a collision between the specified protection areas, the PROTD function returns a value of 0.0.

Note

Safety clearance

The safety clearance for the collision check (MD10622 $MN_COLLISION_SAFETY_DIST) is not taken into account in the clearance calculation.

Syntax

```
[<RetVal>=] PROTD(<Name_1>,<Name_2>,VAR <Vector>[,<System>])
[<RetVal>=] PROTD(,,VAR <Vector>[,<System>])
```

Meaning

PROTD:	Calculates the clearance of the two specified protection areas.	
	Preprocessing stop:	No
	Alone in the block:	Yes
<RetVal>:	Function return value: Absolute clearance value of the two protection areas or 0.0 with collision (see above: Collision paragraph)	
	Data type:	REAL
	Range of values:	0.0 ≤ x ≤ +max. REAL value
<Name_1>, <Name_2>:	Names of the two protection areas whose clearance is to be calculated (**optional**).	
	Data type:	STRING
	Range of values:	Parameterized protection area names
	Default value:	If no protection areas have been specified, the function calculates the current smallest clearance from all the activated and preactivated protection areas in the collision model.
<Vector>:	Return value: 3-dimensional clearance vector from protection area <Name_2> to protection area <Name_1> with: • <Vector>[0]: X coordinate in the world coordinate system • <Vector>[1]: Y coordinate in the world coordinate system • <Vector>[2]: Z coordinate in the world coordinate system With collision: <Vector> == zero vector	
	Data type:	VAR REAL[3]
	Range of values:	<Vector>[n]: 0.0 ≤ x ≤ ±max. REAL value

<System>:	Measuring system (inch/metric) for clearance and clearance vector (**optional**)		
	Data type:	BOOL	
	Value:	FALSE	Measuring system corresponding to the currently active G function from G group 13 (G70, G71, G700, G710).
		TRUE	Measuring system corresponding to the set basic system: MD52806 $MN_ISO_SCALING_SYSTEM
	Default value:	FALSE	

References

Function Manual, Special Functions; Section K9: Collision avoidance

Tool offsets

10

10.1 Offset memory

Structure of the offset memory

Every data field can be invoked with a T and D number (except "Flat D No."); in addition to the geometrical data for the tool, it contains other information such as the tool type.

Flat D number structure

The "Flat D No. structure" is used if tool management takes place outside the NCK. In this case, the D numbers are created with the corresponding tool compensation blocks without assignment to tools.

T can continue to be programmed in the part program. However, this T has no reference to the programmed D number.

User cutting edge data

User cutting edge data can be configured via machine data. Please refer to the machine manufacturer's instructions.

Tool parameters

Note

Individual values in the offset memory

The individual values of the offset memory P1 to P25 can be read and written by the program via system variables. All other parameters are reserved.

The tool parameters $TC_DP6 to $TC_DP8, $TC_DP10 and $TC_DP11 as well as $TC_DP15 to $TC_DP17, $TC_DP19 and $TC_DP20 have another meaning depending on tool type.

Tool parameter number (DP)	Meaning of system variables	Remark
$TC_DP1	Tool type	For overview see list
$TC_DP2	Cutting edge position	Only for turning tools
Geometry	**Length compensation**	
$TC_DP3	Length 1	Allocation to
$TC_DP4	Length 2	Type and level
$TC_DP5	Length 3	
Geometry	**Radius**	
$TC_DP6 [1)] $TC_DP6 [2)]	Radius 1 / length 1 diameter d	Milling/turning/grinding tool Slotting saw
$TC_DP7 [1)] $TC_DP7 [2)]	Length 2 / corner radius, tapered milling tool Slot width b corner radius	Milling tools Slotting saw

Tool parameter number (DP)	Meaning of system variables	Remark
$TC_DP8 [1] $TC_DP8 [2]	Rounding radius 1 for milling tools projecting length k	Milling tools Slotting saw
$TC_DP9 [1] [3]	Rounding radius 2	Reserved
$TC_DP10 [1]	Angle 1 face end of tool	Tapered milling tools
$TC_DP11 [1]	Angle 2 tool longitudinal axis	Tapered milling tools
Wear	**Length and radius compensation**	
$TC_DP12	Length 1	
$TC_DP13	Length 2	
$TC_DP14	Length 3	
$TC_DP15 [1] $TC_DP15 [2]	Radius 1 / length 1 diameter d	Milling/turning/grinding tool Slotting saw
$TC_DP16 [1] $TC_DP16 [3]	Length 2 / corner radius, tapered milling tool, slot width b corner radius	Milling tools Slotting saw
$TC_DP17 [1] $TC_DP17 [2]	Rounding radius 1 for milling tools projecting length k	Milling / 3D face milling Slotting saw
$TC_DP18 [1] [3]	Rounding radius 2	Reserved
$TC_DP19[1]	Angle 1 face end of tool	Tapered milling tools
$TC_DP20[1]	Angle 2 tool longitudinal axis	Tapered milling tools
Tool base dimension/ adapter	**Length offsets**	
$TC_DP21	Length 1	
$TC_DP22	Length 2	
$TC_DP23	Length 3	
Technology		
$TC_DP24	Clearance angle	Only for turning tools
$TC_DP25		Reserved

[1] Also applies with milling tools for 3D face milling

[2] For slotting saw tool type

[3] Reserved (is not used by SINUMERIK 840D sl)

Remarks

Several entry components are available for geometric variables (e.g. length 1 or radius). These are added together to produce a value (e.g. total length 1, total radius), which is then used for the calculations.

Offset values not required must be assigned the value zero.

Tool parameters $TC-DP1 to $TC-DP23 with contour tools

Note

The tool parameters not listed in the table, such as $TC_DP7, are not evaluated, i.e. their content is meaningless.

Tool parameter number (DP)	Meaning	Cutting Dn		Remark
$TC_DP1	Tool type			400 to 599
$TC_DP2	Cutting edge position			
Geometry	**Length compensation**			
$TC_DP3	Length 1			
$TC_DP4	Length 2			
$TC_DP5	Length 3			
Geometry	**Radius**			
$TC_DP6	Radius			
Geometry	**Limit angle**			
$TC_DP10	Minimum limit angle			
$TC_DP11	Maximum limit angle			
Wear	**Length and radius compensation**			
$TC_DP12	Wear length 1			
$TC_DP13	Wear length 2			
$TC_DP14	Wear length 3			
$TC_DP15	Wear radius			
Wear	**Limit angle**			
$TC_DP19	Wear min. limit angle			
$TC_DP20	Wear max. limit angle			
Tool base dimension/ adapter	**Length offsets**			
$TC_DP21	Length 1			
$TC_DP22	Length 2			
$TC_DP23	Length 3			

Basic value and wear value

The resultant values are each a total of the basic value and wear value (e.g. $TC_DP6 + $TC_DP15 for the radius). The basic measurement ($TC_DP21 – $TC_DP23) is also added to the tool length of the first cutting edge. All the other parameters, which may also impact on effective tool length for a standard tool, also affect this tool length (adapter, orientational toolholder, setting data).

Limit angles 1 and 2

Limit angles 1 and 2 each relate to the vector of the cutting edge center point to the cutting edge reference point and are counted clockwise.

10.2 Additive offsets

10.2.1 Selecting additive offsets (DL)

Additive offsets can be considered as process offsets that can be programmed in the machining. They refer to the geometrical data of a cutting edge and are therefore a component of tool cutting data.

Data of an additive offset is addressed using a DL number (DL: Locationdependent; offsets regarding the location of use) and entered via the user interface.

Application

Dimension errors caused be the location of use can be compensated using additive offsets.

Syntax

```
DL=<number>
```

Meaning

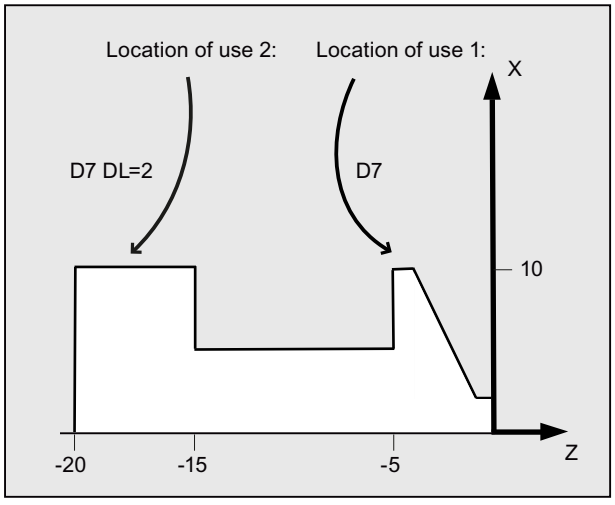
DL:	Command to activate an additive offset
<number>:	The additive tool offset data to be activated is specified using the <number> parameter

Note

The machine data is used to define the number of additive offsets and also activate them (→ carefully observe the machine OEM's data!).

Example

The same cutting edge is used for two bearing seats:



Program code	Comment
N110 T7 D7	; The revolver is positioned to location 7. D7 and DL=1 are activated and moved through in the next block.
N120 G0 X10 Z1	
N130 G1 Z-6	
N140 G0 DL=2 Z-14	; DL=2 is activated in addition to D7 and is moved through in the next block.
N150 G1 Z-21	
N160 G0 X200 Z200	; Approach tool change point.
...	

10.2.2 Specify wear and setup values ($TC_SCPxy[t,d], $TC_ECPxy[t,d])

Wear and setting-up values can be read and written to using system variables. The logic is based on the logic of the corresponding system variables for tools and tool noses.

System variables

$TC_SCPxy[<t>,<d>]:	Wear values that are assigned to the particular geometry parameters via xy, whereby x corresponds to the number of the wear value and y establishes the reference to the geometry parameter.
$TC_ECPxy[<t>,<d>]:	Setting-up values that are assigned to the particular geometry parameter via xy, whereby x corresponds to the number of the setting-up value and y establishes the reference to the geometry parameter.
<t>: T number of the tool <d>: D number of the tool cutting edge	

Note

The defined wear and setup values are added to the geometry parameters and the other offset parameters (D numbers).

Example

The wear value of length 1 is set to the value of 1.0 for the cutting edge <d> of tool <t>.

Parameter: $TC_DP**3** (length 1, with turning tools)

Wear values: $TC_SCP1**3** to $TC_SCP6**3**

Setup values: $TC_ECP1**3** to $TC_ECP6**3**

$TC_SCP4**3** [<t>,<d>] = 1.0

10.2.3 Delete additive offsets (DELDL)

The `DELDL` command deletes the additive offsets for the cutting edge of a tool (to release memory space). Both the defined wear values and the setup values are deleted.

Syntax

```
DELDL[<t>,<d>]
DELDL[<t>]
DELDL
<Status>=DELDL[<t>,<d>]
```

Meaning

`DELDL`:	Command to delete additive offsets	
`<t>`:	T number of the tool	
`<d>`:	D number of the tool cutting edge	
`DELDL[<t>,<d>]`:	All additive offsets of the cutting edges `<d>` of the tool `<t>` are deleted.	
`DELDL[<t>]`:	All additive offsets of all cutting edges of tool `<t>` are deleted.	
`DELDL`:	All additive offsets of all cutting edges of all tools of the TO unit are deleted (for the channel in which the command is programmed).	
`<Status>`:	Delete status	
	Value:	Meaning:
	0	Deletion was successfully completed.
	-	Offsets have not been deleted (if the parameter settings specify exactly one tool edge), or not deleted completely (if the parameter settings specify several cutting edges).

Note

Wear and setting-up values of active tools cannot be deleted (essentially the same as the delete behavior of D or tool data).

10.3 Special handling of tool offsets

The evaluation of the sign for tool length and wear can be controlled using setting data SD42900 to SD42960.

The same applies to the behavior of the wear components when mirroring geometry axes or changing the machining plane, and also to temperature compensation in tool direction.

Wear values:

If reference is made to wear values in the following, then this should be understood as the sum of the actual wear values ($TC_DP12 to $TC_DP20) and the sum offsets with the wear values ($SCPX3 to $SCPX11) and setting-up values ($ECPX3 to $ECPX11).

For more information about summed offsets, refer to:
References:
Function Manual, Tool Management

Setting data

SD42900 $SC_MIRROR_TOOL_LENGTH	Mirroring of tool-length components and components of the tool base dimension.
SD42910 $SC_MIRROR_TOOL_WEAR	Mirroring of wear values of the tool-length components.
SD42920 $SC_WEAR_SIGN_CUTPOS	Evaluating the sign of the wear components as a function of the cutting edge position.
SD42930 $SC_WEAR_SIGN	Inverts the sign of wear dimensions.
SD42935 $SC_WEAR_TRANSFORM	Transformation of wear values.
SD42940 $SC_TOOL_LENGTH_CONST	Assignment of tool length components to geometry axes.
SD42950 $SC_TOOL_LENGTH_TYPE	Assignment of the tool length components independent of tool type.
SD42960 $SC_TOOL_TEMP_COMP	Temperature compensation value in tool direction. Also operative when tool orientation is programmed.

References

Function Manual Basic Functions; Tool Offset (W1)

Further information

Activation of modified setting data

When the setting data described above is modified, the tool components are not reevaluated until the next time a tool edge is selected. If a tool is already active and the data of this tool is to be reevaluated, the tool must be selected again.

The same applies in the event that the resulting tool length is modified due to a change in the mirroring status of an axis. The tool must be selected again after the mirror command, in order to activate the modified tool-length components.

Orientable toolholders and new setting data

Setting data SD42900 to SD42940 has no effect on the components of an active toolholder with orientation capability. However, the calculation with an orientable toolholder always allows for a tool with its total resulting length (tool length + wear + tool base dimension). All modifications initiated by the setting data are included in the calculation of the resulting total length, i.e. vectors of the orientable toolholder are independent of the machining plane.

Note

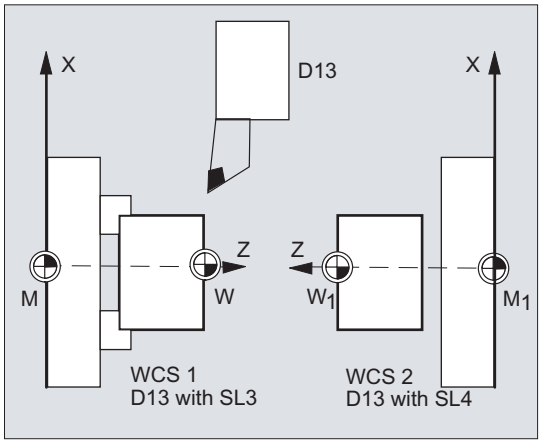
When orientable toolholders are used, it is frequently practical to define all tools for a non-mirrored basic system, even those which are only used for mirrored machining. When machining with mirrored axes, the toolholder is then rotated such that the actual position of the tool is described correctly. All tool-length components then automatically act in the correct direction, dispensing with the need for control of individual component evaluation via setting data, depending on the mirroring status of individual axes.

Further application options

The use of orientable toolholder functionality can also be useful if there is no physical option of turning tools on the machine, even though tools with different orientations are permanently installed. Tool dimensioning can then be performed uniformly in a basic orientation, where the dimensions relevant for machining are calculated according to the rotations of a virtual toolholder.

10.3.1 Mirroring of tool lengths

When setting data SD42900 $SC_MIRROR_TOOL_LENGTH and SD42910 $SC_MIRROR_TOOL_WEAR are not set to zero, then you can mirror the tool length components and components of the basis dimensions with wear values and their associated axes.



SD42900 $SC_MIRROR_TOOL_LENGTH

Setting data **not equal to** zero:

The tool length components ($TC_DP3, $TC_DP4 and $TC_DP5) and the components of the basis dimensions ($TC_DP21, $TC_DP22 and $TC_DP23) are mirrored against their associated axes, also mirrored – by inverting the sign.

The wear values are **not** mirrored. If these are also be be mirrored, then setting data SD42910 $SC_MIRROR_TOOL_WEAR must be set.

SD42910 $SC_MIRROR_TOOL_WEAR

Setting data **not equal to** zero:

The wear values of the tool length components - whose associated axes are mirrored - are also mirrored by inverting the sign.

10.3.2 Wear sign evaluation

When setting data SD42920 $SC_WEAR_SIGN_CUTPOS and SD42930 $SC_WEAR_SIGN are set not equal to zero, then you can invert the sign evaluation of the wear components.

SD42920 $SC_WEAR_SIGN_CUTPOS

Setting data **not equal to** zero:

For tools with the relevant cutting edge position (turning and grinding tools, tool types 400), then the sign evaluation of the wear components in the machining plane depends on the cutting edge position. This setting data is of no significance for tool types without relevant cutting edge position.

In the following table, the dimensions, whose sign is inverted using SD42920 (not equal to zero), are designed using an X:

Cutting edge position	Length 1	Length 2
1		
2		X
3	X	X
4	X	
5		
6		
7		X
8	X	
9		

Note

The sign evaluation using SD42920 and SD42910 are independent of one another. If, for example, the sign of a dimension is changed using both setting data, then the resulting sign remains unchanged.

SD42930 $SC_WEAR_SIGN

Setting data **not equal to** zero:

Inverts the sign of all wear dimensions. This affects both the tool length and other variables such as tool radius, rounding radius, etc.

If a positive wear dimension is entered, the tool becomes "shorter" and "thinner", refer to Chapter "tool offset, special handling", activating changed setting data".

10.3.3 Coordinate system of the active machining operation (TOWSTD, TOWMCS, TOWWCS, TOWBCS, TOWTCS, TOWKCS)

Depending on the kinematics of the machine or the availability of an orientable toolholder, the wear values measured in one of these coordinate systems are converted or transformed to a suitable coordinate system.

Coordinate systems of active machining operation

The following coordinate systems can produce tool length offsets that can be used to incorporate the tool length component "wear" into an active tool via the corresponding G code of Group 56.

- Machine coordinate system (MCS)
- Basic coordinate system (BCS)
- Workpiece coordinate system (WCS)
- Tool coordinate system (TCS)
- Tool coordinate system of kinematic transformation (KCS)

Syntax

```
TOWSTD
TOWMCS
TOWWCS
TOWBCS
TOWTCS
TOWKCS
```

Meaning

TOWSTD:	Initial setting value for offsets in tool length wear value
TOWMCS:	Offsets in tool length in MCS
TOWWCS:	Offsets in tool length in WCS
TOWBCS:	Offsets in tool length in BCS
TOWTCS:	Offsets in tool length at toolholder reference point (orientable toolholder)
TOWKCS:	Offsets in tool length at tool head (kinematic transformation)

Further information

Distinguishing features

The most important distinguishing features are shown in the following table:

G code	Wear value	Active orientable toolholder
TOWSTD:	Initial value, tool length	Wear values are subject to rotation.
TOWMCS:	Wear value in MCS. TOWMCS is identical to TOWSTD if a tool holder that can be orientated is not active.	It only rotates the vector of the resultant tool length without taking into account the wear.
TOWWCS:	The wear value is converted to the MCS in the WCS.	The tool vector is calculated as for TOWMCS without taking into account the wear.
TOWBCS:	The wear value is converted to the MCS in the BCS.	The tool vector is calculated as for TOWMCS without taking into account the wear.
TOWTCS:	The wear value is converted to the MCS in the workpiece coordinate system.	The tool vector is calculated as for TOWMCS without taking into account the wear.

TOWWCS , TOWBCS, TOWTCS: The wear vector is added to the tool vector.

Linear transformation

The tool length can be defined meaningfully in the MCS only if the MCS is generated by linear transformation from the BCS.

Non-linear transformation

For example, if with TRANSMIT a non-linear transformation is active, then when specifying the MCS as requested coordinate system, BCS is automatically used.

No kinematic transformation and no orientable toolholder

If neither a kinematic transformation nor an orientable toolholder is active, then all the other four coordinate systems (except for the WCS) are combined. It is then only the WCS, which is different to the other systems. Since only tool lengths need to be evaluated, translations between the coordinate systems are irrelevant.

References:
For more information on tool compensation, see:
Function Manual Basic Functions; Tool Offset (W1)

Inclusion of wear values in calculation

The setting data **SD42935 $SC_WEAR_TRANSFORM** defines which of the three wear components:

- Wear
- Total offsets fine
- Total offsets coarse

should be subject to a rotation using adapter transformation or a tool holder that can be orientated if one of the following G codes is active:

- `TOWSTD` default setting
  For offsets in the tool length
- `TOWMCS` wear values
  In the machine coordinate system (MCS)
- `TOWWCS` wear values
  In the workpiece coordinate system (WCS)
- `TOWBCS` wear values (BCS)
  In the basic coordinate system
- `TOWTCS` wear values in the tool coordinate system at the tool holder reference (T tool holder reference)
- `TOWKCS` Wear values in the coordinate system of the tool head for kinematic transformation

Note

Evaluation of individual wear components (assignment to geometry axes, sign evaluation) is influenced by:

- The active plane
- The adapter transformation
- Following setting data:
  - SD42910 $SC_MIRROW_TOOL_WEAR
  - SD42920 $SC_WEAR_SIGN_CUTPOS
  - SD42930 $SC_WEAR_SIGN
  - SD42940 $SC_TOOL_LENGTH_CONST
  - SD42950 $SC_TOOL_LENGTH_TYPE

10.3.4 Tool length and plane change

When setting data SD42940 $SC_TOOL_LENGTH_CONST is set not equal to zero, then you can assign the tool length components – such as lengths, wear and basic dimension – to the geometry axes for turning and grinding tools when changing the plane.

SD42940 $SC_TOOL_LENGTH_CONST

Setting data **not equal to** zero:

The assignment of tool length components (length, wear and tool base dimension) to geometry axes does not change when the machining plane is changed (`G17` - `G19`).

The following table shows the assignment of tool length components to geometry axes for turning and grinding tools (tool types 400 to 599):

Content	Length 1	Length 2	Length 3
17	Y	X	Z
*)	X	Z	Y
19	Z	Y	X
-17	X	Y	Z
-18	Z	X	Y
-19	Y	Z	X

*) Each value not equal to 0, which is not equal to one of the six listed values, is evaluated as value 18.

The following table shows the assignment of tool length components to geometry axes for all other tools (tool types < 400 or > 599):

Operating plane	Length 1	Length 2	Length 3
*)	Z	Y	X
18	Y	X	Z
19	X	Z	Y
-17	Z	X	Y
-18	Y	Z	X
-19	X	Y	Z

*) Each value not equal to 0, which is not equal to one of the six listed values, is evaluated as value 17.

Note

For representation in tables, it is assumed that geometry axes up to 3 are designated with X, Y, Z. The axis order and not the axis identifier determines the assignment between a compensation and an axis.

10.4 Online tool offset

10.4.1 Defining a polynomial function (FCTDEF)

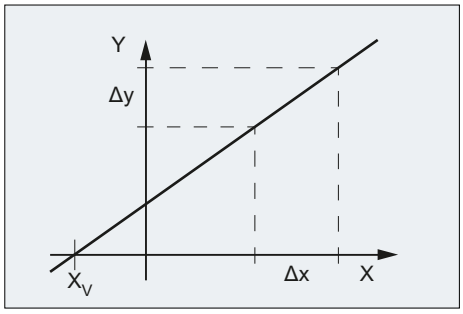
Certain dressing strategies (e.g. dressing roller) are characterized by the fact that the grinding wheel radius is continuously (linearly) reduced as the dressing roller is fed in. This strategy requires a linear function between infeed of the dressing roller and writing the wear value of each length. The linear function is defined using the predefined procedure FCTDEF(...) for up to third order polynomial functions.

Straight line equation

$y = f(x) = a_0 + a_1 * x_1$

a_1: Gradient of the straight line, with $a_1 = \Delta x / \Delta y$

a_0: Shift of the straight line along the X axis with $a_0 = -a1 * X_V$



Syntax

```
FCTDEF(<Func>,<LLimit>,<ULimit>,<a0>,<a1>,<a2>,<a3>)
```

Meaning

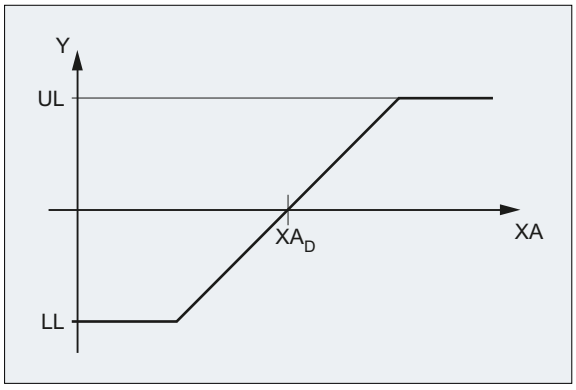
FCTDEF(...):	Defining a polynomial function for PUTFTOCF(...): $y = f(x) = a_0 + a_1*x + a_2*x^2 + a_3*x^3$	
<Func>:	Function number	
	Data type:	INT
	Range of values:	1, 2, 3
<LLimit>:	Lower limit value	
	Data type:	REAL
<ULimit>:	Upper limit value	
	Data type:	REAL
<a0>,<a1>,<a2>,<a3>:	Coefficients of polynomial function	
	Data type:	REAL

Example

Definitions

- Function number: 1
- Lower and upper limit value: -100, 100
- Gradient of the characteristic: $a_1 = 1$
- The operating point should be located at the center of the characteristic. Based on the setpoint position of axis XA in the WCS at the instant that the function is defined in the NC program, the characteristic must be shifted in the negative Y direction: $a_0 = -a_1 * XA_D = -1 * \$AA_IW$
- $a_2 = a_3 = 0$

Characteristic



UL	Upper limit value
LL	Lower limit value
XA_D	Setpoint of axis XA at the time that the function is defined in the NC program

Programming

Program code	Comment
`FCTDEF(1,-100,100,-$AA_IW[XA],1)`	`; Function definition`

10.4.2 Write online tool offset continuously (PUTFTOCF)

Using the predefined procedure PUTFTOCF(...), an online tool offset is executed based on a polynomial function previously defined with FCTDEF(...) (Page 380).

Note

The online tool offset can also be realized using a synchronized action.

For further information, see Function Manual Synchronized Actions.

Syntax

```
PUTFTOCF(<Func>,<RefVal>,<ToolPar>,<Chan>,<Sp>)
```

Meaning

PUTFTOCF(...):	Write online tool offset, continuously block-by-block using the polynomial function defined with FCTDEF(...)	
<Func>:	Function number, defined in the function definition with FCTDEF(...)	
	Data type:	INT
	Range of values:	1, 2, 3
<RefVal>:	Reference value, from which the offset is to be derived (e.g. setpoint of an axis).	
	Data type:	VAR REAL
<ToolPar>:	Number of the wear parameter (length 1, 2 or 3) in which the offset value is to be included.	
	Data type:	INT
<Chan>:	Number of the channel in which the online tool offset is to take effect. **Note:** Only required if the offset is not to take effect in the active channel.	
	Data type:	INT
<Sp>:	Number of the spindle for which the online tool offset is to take effect. **Note:** Only required if the offset is to be applied to a non-active grinding wheel rather than the active tool that is currently in use.	
	Data type:	INT

10.4.3 Write online tool offset, discrete (PUTFTOC)

Function

Using the predefined procedurePUTFTOC(...), an online tool offset is executed based on a fixed offset value.

Syntax

```
PUTFTOC(<CorrVal>,<ToolPar>,<Chan>,<Sp>)
```

Meaning

PUTFTOC(...):	Write online tool offset	
<CorrVal>:	Offset value, which is added to the wear parameter.	
	Data type:	REAL
<ToolPar>:	Number of the wear parameter (length 1, 2 or 3) in which the offset value is to be included.	
	Data type:	INT

<Chan>:	Number of the channel in which the online tool offset is to take effect. **Note:** Only required if the offset is not to take effect in the active channel.	
	Data type:	INT
<Sp>:	Number of the spindle for which the online tool offset is to take effect. **Note:** Only required if the offset is to be applied to a non-active grinding wheel rather than the active tool that is currently in use.	
	Data type:	INT

10.4.4 Activate/deactivate online tool offset (FTOCON/FTOCOF)

The online tool offset is activated or deactivated using the G commands FTOCON and FTOCOF.

Syntax

```
FTOCON
...
FTOCOF
```

Meaning

FTOCON:	Activate online tool offset The command must be programmed in the channel in which the online tool offset is to be activated.
FTOCOF:	Deactivate online tool offset The command must be programmed in the channel in which the online tool offset is to be deactivated. **Note:** On FTOCOF, the axis does not move further out for the tool offset. However, the value calculated with PUTFTOC/PUTFTOCF remains in the cutting-specific offset data. To finally deactivate the online tool offset, the tool (T...) must again be selected/ deselected after FTOCOF.

10.5 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

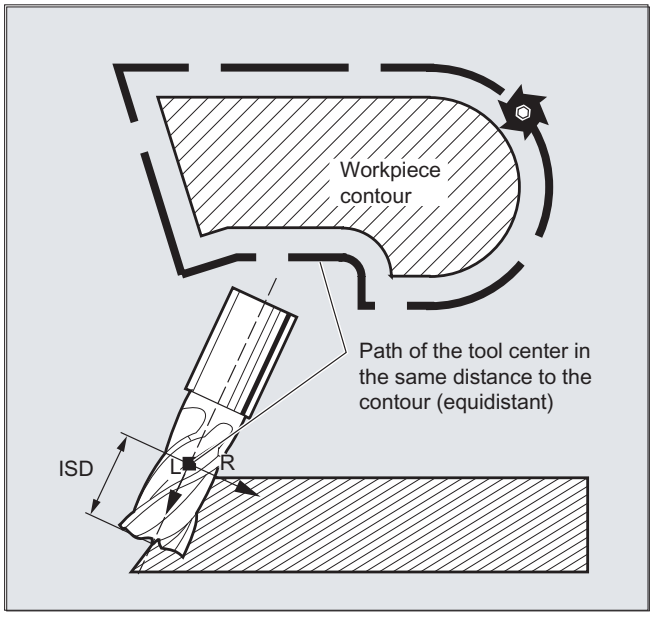
10.5.1 Activating 3D tool offsets (CUT3DC, CUT3DCD, CUT3DF, CUT3DFS, CUT3DFF, ISD)

For 3D tool radius compensation for cylindrical tools, contrary to 2½ D tool radius compensation, the changing tool orientation is taken into account.

The approach response is always controlled with NORM.

With G41/G42, the tool radius compensation is selected in motion direction left/right from the programmed path.

The 3D radius compensation is only effective when 5-axis transformation is selected. As a consequence, 3D tool radius compensation is also known as 5D compensation, as in this case, there are 5 degrees of freedom for the position of the tool in space.



Difference between 2½ D and 3D tool radius compensation

In 3D tool radius compensation tool orientation can be changed.

For 2½ D tool radius compensation, a tool with constant orientation is calculated.

Tool radius compensation referred to a differential tool

3D tool radius compensation, referred to a differential tool, is activated using the CUT3DCD command. It should be applied if the programmed contour refers to the center point path of a differential tool, and a tool other than a differential tool is used for machining. When calculating the 3D tool radius compensation, only the wear of the radius of the active tool ($TC_DP_15) and the possibly programmed tool offsetOFFN and TOFFR are taken into account. The basic radius ($TC_DP6) of the active tool is **not** taken into account.

Syntax

```
CUT3DC
CUT3DCD
CUT3DFS
CUT3DFF
CUT3DF
ISD=<Value>
```

Meaning

CUT3DC:	Activation of 3D radius offset for circumferential milling
CUT3DCD:	Activating the 3D radius compensation referred to a differential tool for circumferential milling
CUT3DFS:	D tool offset for face milling with constant orientation. The tool orientation is determined by G17 - G19 and is not influenced by frames.
CUT3DFF:	D tool offset for face milling with constant orientation. The tool orientation is the direction defined by G17 - G19 and, in some case, rotated by a frame.
CUT3DF:	D tool offset for face milling with orientation change (only with active 5-axes transformation).
G40 X... Y... Z...:	To deactivate: Linear block G0 / G1 with geometry axes
ISD:	Insertion depth

Note

The commands are modal, and are in the same group as CUT2D and CUT2DF. The command is not deselected until the next movement in the current plane is performed. This always applies for G40 and is independent of the CUT command.

Intermediate blocks are permitted with 3D tool radius compensation. The definitions for 2 1/2D tool radius compensation apply.

Supplementary conditions

G450 / G451 and DISC

A circular block is always inserted at out corners. G450/G451 have no significance.
The DISC command is not evaluated.

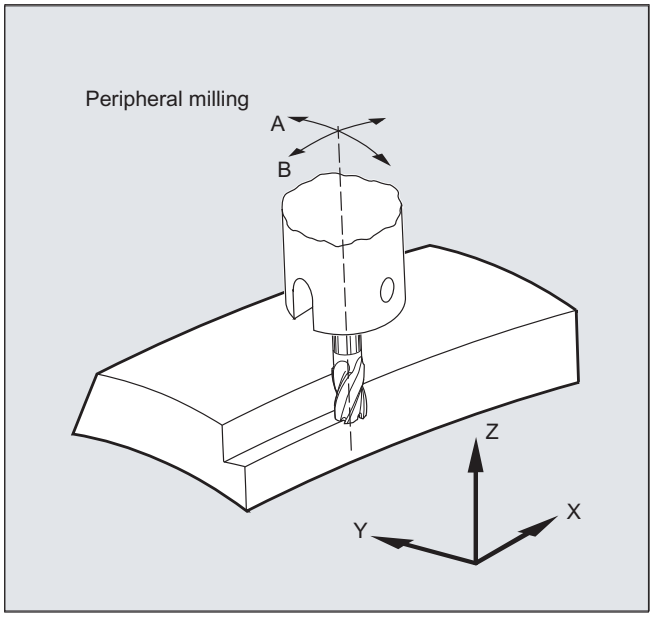
Example

```
Program code                    Comment
N10 A0 B0 X0 Y0 Z0 F5000
N20 T1 D1                       ; tool 1, cutting edge 1
N30 TRAORI(1)                   ; orientation transformation
N40 CUT3DC                      ; 3D tool radius compensation
N50 G42 X10 Y10                 ; tool radius compensation right of contour
N60 X60
N70 …
```

10.5.2 3D tool offset peripheral milling, face milling

Circumferential milling

The type of milling used here is implemented by defining a path (guide line) and the corresponding orientation. In this type of machining, the shape of the tool on the path is not relevant. Especially the radius at the tool intervention point is decisive.

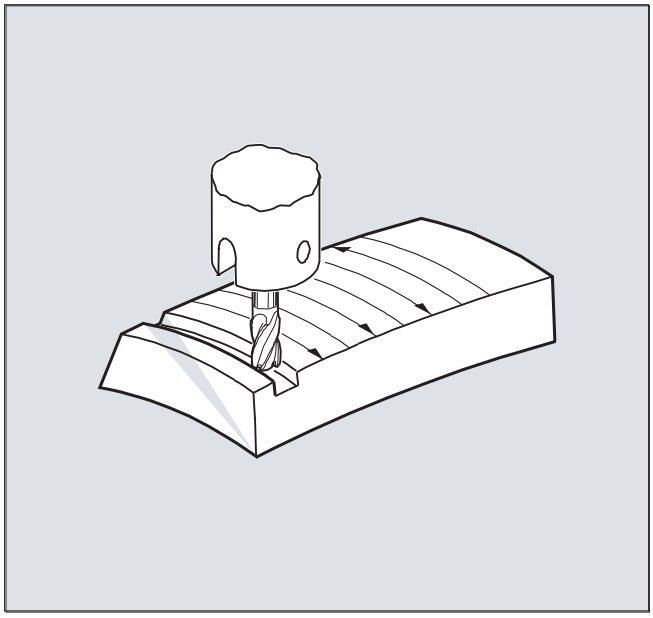


Note

The function 3D TRC is restricted to cylindrical tools.

Face milling

For this type of 3D milling, you will require the line-by-line description of the 3D paths on the workpiece surface. The tool shape and dimensions are taken into account in the calculations - which are normally performed in CAM. The post processor writes to the part program - in addition to NC blocks - to orientations (for active 5 axis transformation) and the G code for the required 3D tool offset. This means that the machine operator has the possibility of using tools that are slightly smaller - deviating from the tool used to calculate the NC paths.



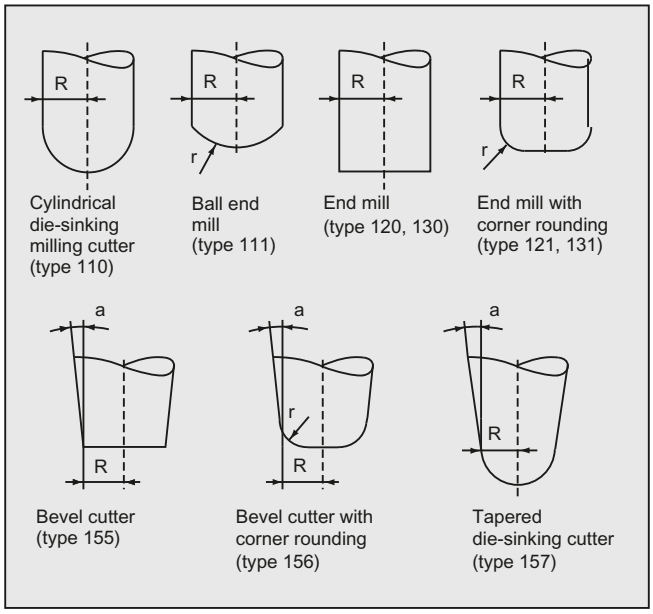
Example:

NC blocks were computed using a 10 mm milling tool. In this case, a milling tool diameter of 9.9 mm can be used for machining - whereby a modified roughness profile can be expended.

10.5.3 3D tool offset Tool shapes and tool data for face milling

Mill shapes, tool data

An overview of the tool shapes, which may be used for face milling operations and tool data limit values are listed in the following. The shape of the tool shaft is not taken into account. The effect of tool types 120 and 156 is identical.



If, in the NC program, a type number is specified that differs from that in the diagram, then the system automatically uses tool type 110 (cylindrical die-sinking milling tool). An alarm is output if the tool data limit values are violated.

Cutter type	Type No.	R	r	a
Cylindrical die-sinking milling cutter	110	> 0	-	-
Ball end mill	111	> 0	> R	-
End mill, angle head cutter	120, 130	> 0	-	-
End mill, angle head cutter with corner rounding	121, 131	> r	> 0	-
Bevel cutter	155	> 0	-	> 0
Bevel cutter with corner rounding	156	> 0	> 0	> 0
Tapered die-sinking cutter	157	> 0	-	> 0

R = shaft radius (tool radius)
r = corner radius
a = angle between the tool longitudinal axis and upper end of the torus surface
- = is not evaluated

Tool data	Tool parameters	
Tool dimensions	Geometry	Wear
R	$TC_DP6	$TC_DP15
r	$TC_DP7	$TC_DP16
a	$TC_DP11	$TC_DP20

Tool length offset

The tool tip is the reference point for length offset (intersection longitudinal axis/surface).

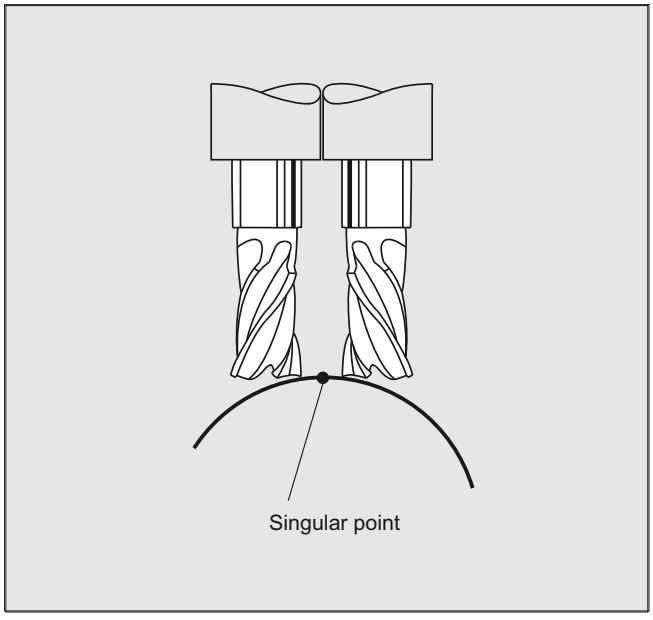
3D tool offset, tool change

A new tool with modified dimensions (R, r, a) or a different shaft may only be specified with the programming of `G41` or `G42` (transition `G40` to `G41` or `G42`, reprogramming of `G41` or `G42`). This rule does not apply to any other tool data, e.g. tool lengths, so that tools can be loaded without reprogramming `G41` or `G42`.

10.5.4 3D tool offset Offset on the path, path curvature, insertion depth (CUT3DC, CUT3DCD, ISD)

Offset on the path

With respect to face milling, it is advisable to examine what happens when the contact point "jumps" on the tool surface as shown in the example on the right where a convex surface is being machined with a vertically positioned tool. The application shown in the example should be regarded as a borderline case.



This borderline case is monitored by the control that detects abrupt changes in the machining point on the basis of angular approach motions between the tool and normal surface vectors. The control inserts linear blocks at these positions so that the motion can be executed.

These linear blocks are calculated on the basis of permissible angular ranges for the side angle stored in the machine data. An alarm is output if the limit values for the permitted angular ranges, defined in the machine data, are violated.

- MD21082 $MC_CUTCOM_PLANE_ORI_LIMIT (minimum angle between the surface normal vectors and tool orientation)
- MD21092 $MC_MAX_TILT_ANGLE (maximum absolute value of the perm. sideways angle for orientation progr.)

Path curvature

Path curvature is not monitored. In such cases, it is also advisable to use only tools of a type that do not violate the contour.

Insertion depth (ISD)

Insertion depth ISD is only evaluated when 3D tool radius compensation is active.

Command `ISD` (insertion depth) is used to program the tool insertion depth for circumferential milling operations. This makes it possible to change the position of the machining point on the outer surface of the tool.

Tool radius compensation referred to a differential tool

3D tool radius compensation, referred to a differential tool, is activated using the `CUT3DCD` command. It should be applied if the programmed contour refers to the center point path of a differential tool, and a tool other than a differential tool is used for machining. When calculating the 3D tool radius compensation, only the wear of the radius of the active tool ($TC_DP_15) and the possibly programmed tool offsetOFFN and TOFFR are taken into account. The basic radius ($TC_DP6) of the active tool is **not** taken into account.

Syntax

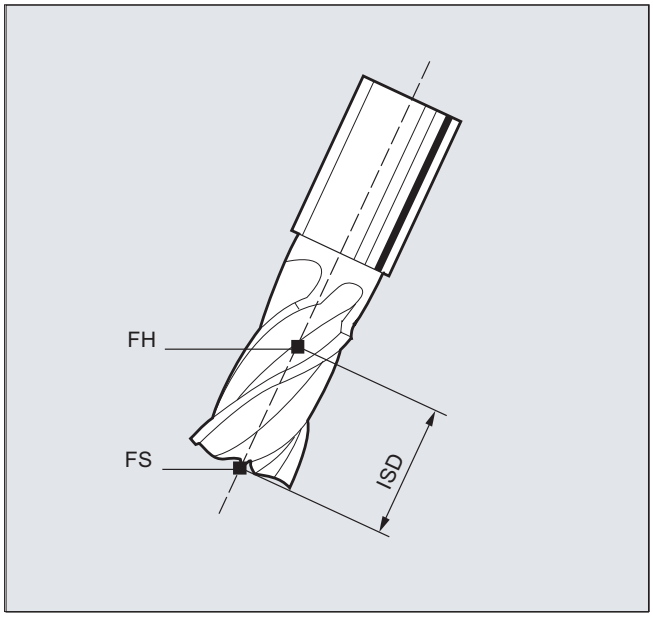
```
CUT3DC
CUT3DCD
ISD=<Value>
```

Meaning

`CUT3DC`:	Activation of 3D radius offset for circumferential milling
`CUT3DCD`:	Activating the 3D radius compensation referred to a differential tool for circumferential milling
`ISD`:	The clearance (`<value>`) between the milling tool tip (FS) and the milling tool construction point (FH) are specified using the command `ISD`.

Milling tool reference point

The milling tool reference point (FH) is obtained by projecting the programmed machining point onto the tool axis.



Further information

Pocket milling with inclined side walls for circumferential milling with CUT3DC

In this 3D tool radius compensation, a deviation of the mill radius is compensated by infeed toward the normals of the surface to be machined. The plane, in which the milling tool face is located, remains unchanged if the insertion depth `ISD` has remained the same. For example, a milling tool with a smaller radius than a standard tool would not reach the pocket base, which is also the limitation surface. For automatic tool infeed, this limitation surface must be known to the control, see section "3D circumferential milling with limitation surfaces".

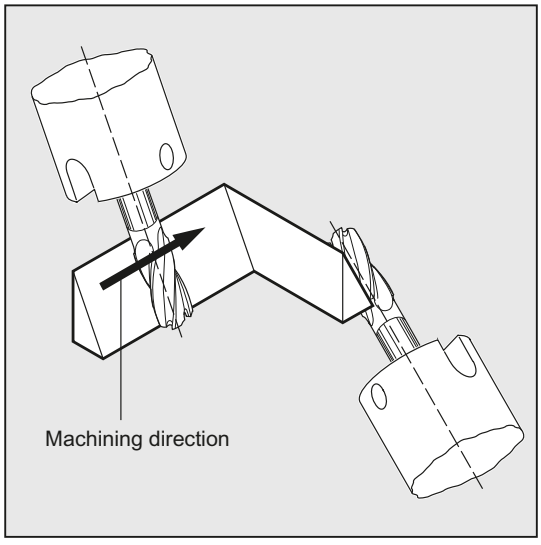
For additional information on collision monitoring, refer to:
References:
Programming Manual Basics; Chapter "Tool offsets".

10.5.5 3D tool offset Inside/outside corners and intersection procedure (G450/G451)

Inside corners / outside corners

Inside and outside corners are handled separately. The terms inner corner and outer corner are dependent on the tool orientation.

When the orientation changes at a corner, for example, the corner type may change while machining is in progress. Whenever this occurs, the machining operation is aborted with an error message.



Syntax

```
G450
G451
```

Meaning

G450:	Transition circle (tool travels round workpiece corners on a circular path)
G451:	Intersection of equidistant paths (tool backs off from the workpiece corner)

Further information

Intersection procedure for 3D compensation

With 3D circumferential milling, G code G450/G451 is now evaluated i.e. the point of intersection of the offset curves can be approached. Up to SW 4 a circle was always inserted at the outside corners. The intersection procedure is especially advantageous for 3D programs typically generated by CAD. These often consist of short straight blocks (to approximate smooth curves), where the transitions between adjacent blocks are almost tangential.

Up to now, with tool radius compensation on the outside of the contour, circles were generally inserted to circumnavigate the outside corners. These blocks can be very short with almost tangential transitions, resulting in undesired drops in velocity.

In these cases, analog to the 2 ½ D radius compensation, the two curves involved are extended; the intersection of both extended curves is approached.

The intersection is determined by extending the offset curves of the two participating blocks and defining the intersection of the two blocks at the corner in the plane perpendicular to the tool orientation. If there is not such intersection, the corner is handled as before – i.e. a circle is inserted.

For more information on the intersection procedure, see:
References:
Function Manual, Special Functions; 3D Tool Radius Compensation (W5)

10.5.6 3D tool offset 3D circumferential milling with limitation surfaces

Adaptation of 3D circumferential milling to the conditions for CAD programs

NC programs generated by CAD systems usually approximate the center path of a standard tool with a large number of short linear blocks. To ensure that the blocks of many part contours generated in this way map the original contour as precisely as possible, it is necessary to make certain changes to the part program.

Important information, which would be required to achieve optimum compensation, that is no longer available in the part program must be replaced using suitable measures. Here are some typical methods to compensate critical transitions, either directly in the part program or when determining the real contour (e.g. using tool infeed).

Applications

In addition to the typical applications for which instead of the standard tool, a real tool describes the center-point path, cylindrical tools with 3D tool compensation are also described. In this case, the programmed path refers to the contour on the machining surface. The associated limitation surface is independent of the tool. Just the same as for conventional tool radius compensation, the entire radius is used to calculate the perpendicular offset to the limitation surface.

10.5.7 3D tool offset Taking into consideration a limitation surface (CUT3DCC, CUT3DCCD)

3D circumferential milling with real tools

In 3D circumferential milling with a continuous or constant change in tool orientation, the tool center point path is frequently programmed for a defined standard tool. Because in practice suitable standard tools are often not available, a tool that does not deviate too much from a standard tool can be used.

`CUT3DCCD` takes account of a limitation surface for a real differential tool that the programmed standard tool would define. The NC program defines the center-point path of a standard tool.

`CUT3DCC` with the use of cylindrical tools takes account of a limitation surface that the programmed standard tool would have reached. The NC program defines the contour on the machining surface.

Syntax

```
CUT3DCCD
CUT3DCC
```

Meaning

CUT3DCCD:	Activation of the 3D tool offset for the circumferential milling with limitation surfaces with a differential tool on the tool center point path: Infeed to the limitation surface.
CUT3DCC:	Activation of the 3D tool offset for circumferential milling with limitation surfaces with 3D radius compensation: Contour on the machining surface

Note

Tool radius compensation with G41, G42

If tool radius compensation with G41, G42 is programmed when CUT3DCCD or CUT3DCC is active, the option "orientation transformation" must also be active.

Standard tools with corner rounding

Corner rounding with a standard tool is defined by the tool parameter $TC_DP7. Tool parameter $TC_DP16 describes the deviation of the corner rounding of the real tool compared with the standard tool.

Example

Tool dimensions of a toroidal miller with reduced radius as compared with the standard tool.

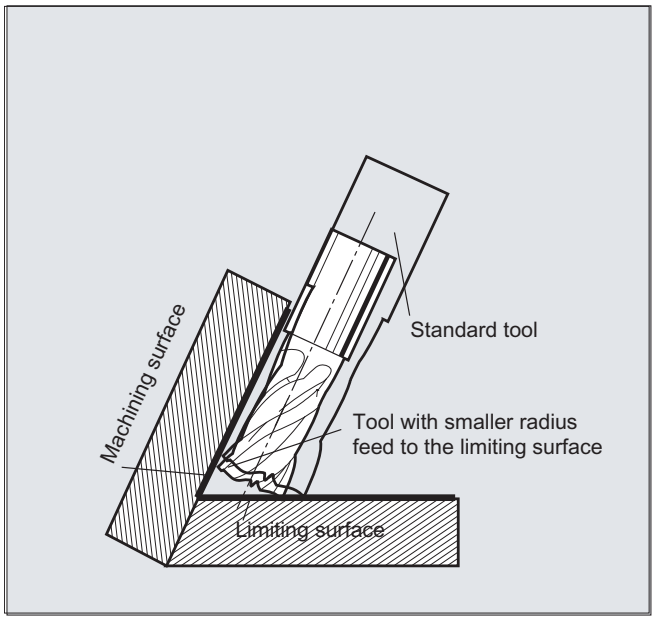
Tool type	R = shank radius	r = corner radius
Standard tool with corner rounding	R = $TC_DP6	r = $TC_DP7
Real tool with corner rounding: Tool types 121 and 131 torus milling tool (shaft milling tool)	R' = $TC_DP6 + $TC_DP15 + OFFN	r' = $TC_DP7 + $TC_DP16
In this example, both $TC_DP15 + OFFN and $TC_DP16 are negative. The tool type ($TC_DP1) is evaluated.		
Only cutter types with cylindrical shank (cylinder or end mill), toroidal millers (types 121 and 131) and, in the limit case, cylindrical die mills (type 110) are permitted.	For these approved cutter types, the corner radius r is identical to the shank radius R. All other permitted tool types are interpreted as cylindrical cutters and the dimensions specified for the corner rounding are not evaluated.	
All tool types of the numbers 1 - 399, with the exception of the numbers 111 and 155 to 157, are permitted.		

Further information

Tool center point path with infeed up to the limitation surface CUT3DCCD

If a tool with a smaller radius than the appropriate standard tool is used, machining is continued using a milling tool, which is infed in the longitudinal direction until it reaches the bottom (base) of the pocket. The tool removes as much material from the corner formed by the machining surface and limitation surface. This involves a machining type combining circumferential and

face milling. Analog to a tool with lower radius, for a tool with increased radius, the infeed is in the opposite direction.



Contrary to all other tool offsets of G code group 22, tool parameter `$TC_DP6` specified for `CUT3DCCD` does not influence the tool radius and the resulting compensation.

The compensation offset is the sum of:

- The wear value of the tool radius (tool parameter `$TC_DP15`) and
- A tool offset `OFFN` programmed to calculate the perpendicular offset to the limitation surface.

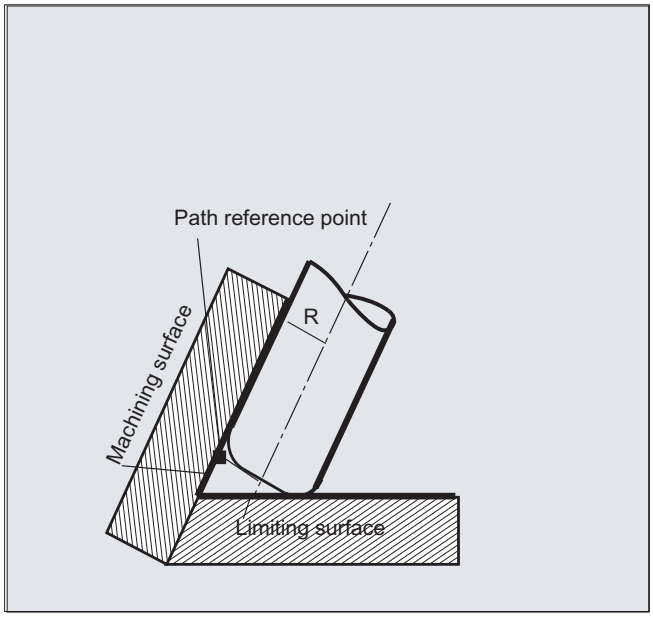
The generated part program does not specify whether the surface to be machined is to the right or left of the path. It is therefore assumed that the radius is a positive value and the wear value of the original tool is a negative value. A negative wear value always describes a tool with a lower diameter.

Using cylindrical tools

When cylindrical tools are used, infeed is only necessary if the machining surface and the surface of limitation form an acute angle (less than 90 degrees). If a torus milling tool (cylinder with rounded corners) is used, tool infeed in the longitudinal direction is required for both acute and obtuse angles.

3D radius compensation with CUT3DCC, contour on the machining surface

If `CUT3DCC` is active with a torus milling tool, the programmed path refers to a fictitious cylindrical milling tool having the same diameter. The resulting path reference point is shown in the following diagram for a torus milling tool.



The angle between the machining and limitation surfaces may change from an acute to an obtuse angle and vice versa even within the same block.

The tool actually being used may either be larger or smaller than the standard tool. However, the resulting corner radius must not be negative and the sign of the resulting tool radius must be kept.

For `CUT3DCC`, the NC part program refers to contour on the machining surface. As for conventional tool radius compensation, the total tool radius is used that comprises the sum of:

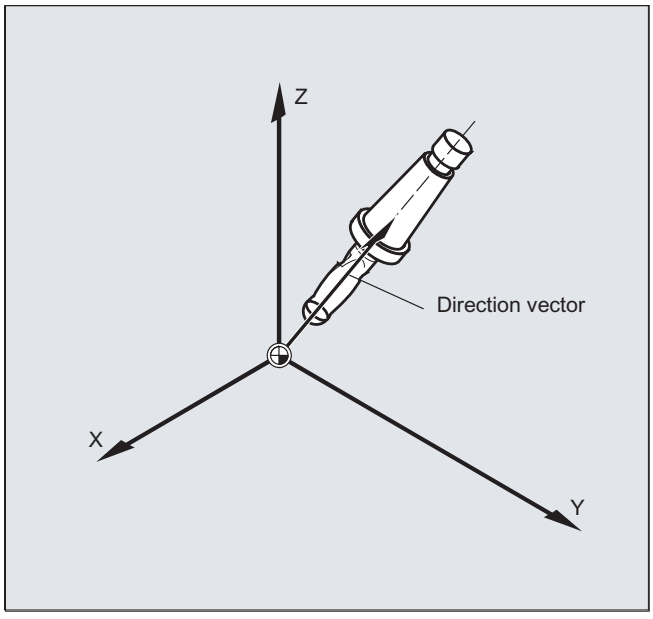
- Tool radius (tool parameter `$TC_DP6`)
- Wear value (tool parameter `$TC_DP15`) and
- A tool offset `OFFN` programmed to calculate the perpendicular offset to the limitation surface.

The position of the limitation surface is determined from the difference between these two values:

- Dimensions of the standard tool
- Tool radius (tool parameter `$TC_DP6`)

10.6 Tool orientation (ORIC, ORID, OSOF, OSC, OSS, OSSE, ORIS, OSD, OST)

The term tool orientation describes the geometric alignment of the tool in space. The tool orientation on a 5-axis machine tool can be set by means of program commands.



Orientation rounding movements activated with `OSD` and `OST` are formed differently depending on the type of interpolation for tool orientation.

If vector interpolation is active, the smoothed orientation characteristic is also interpolated using vector interpolation. On the other hand, if rotary axis interpolation is active, the orientation is smoothed directly using rotary axis movements.

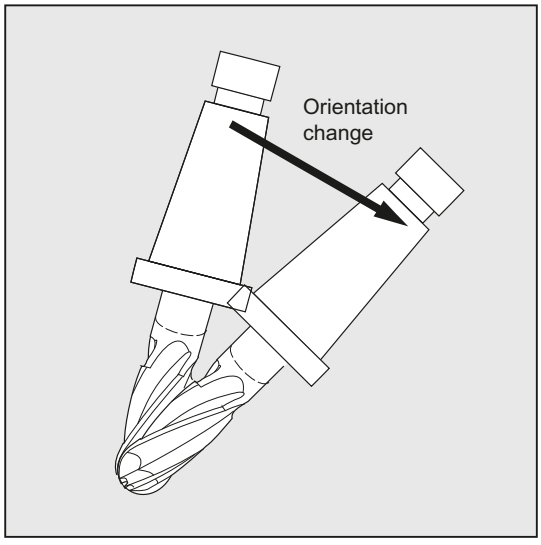
Programming

Programming a orientation change:

A change in tool orientation can be programmed by:

- Direct programming of rotary axes `A`, `B`, `C` (rotary axis interpolation)
- Euler or RPY angle
- Direction vector (vector interpolation by specifying `A3` or `B3` or `C3`)
- `LEAD/TILT` (face milling)

The reference coordinate system is either the machine coordinate system (`ORIMKS`) or the current workpiece coordinate system (`ORIWKS`).



Programming tool orientation:

ORIC:	Orientation and path movement in parallel
ORID:	Orientation and path movement consecutively
OSOF:	No orientation smoothing
OSC:	Orientation constantly
OSS:	Orientation smoothing only at beginning of block
OSSE:	Orientation smoothing at beginning and end of block
ORIS:	Velocity of the orientation change with orientation smoothing activated in degrees per mm (valid for OSS and OSSE)
OSD:	Smoothing of orientation by specifying smoothing distance with setting data: SD42674 $SC_ORI_SMOOTH_DIST
OST:	Smoothing of orientation by specifying angular tolerance in degrees for vector interpolation with setting data: SD42676 $SC_ORI_SMOOTH_TOL With rotary axis interpolation, the specified tolerance is assumed to be the maximum variance of the orientation axes.

Note

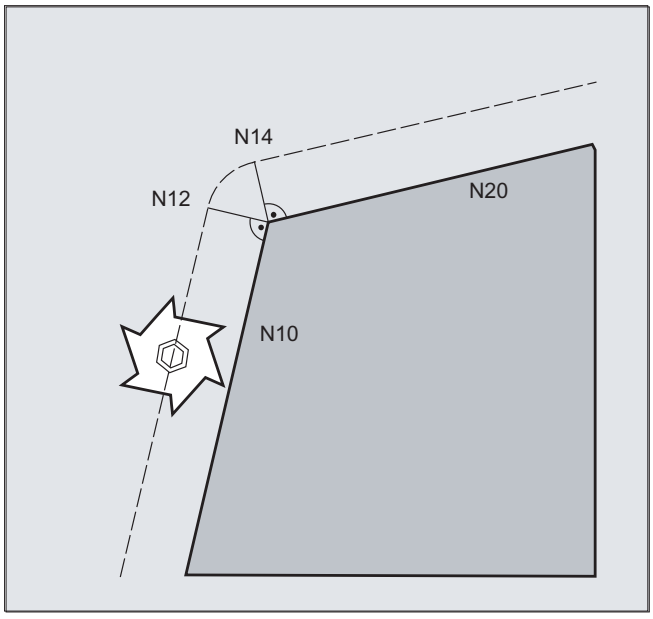
All commands for smoothing the tool orientation (OSOF, OSC, OSS, OSSE, OSD, and OST) are summarized in G function group 34. They are modal; in other words, only one of these commands can ever be effective at the same time.

Examples

Example 1: ORIC

If ORIC is active and there are two or more blocks with changes in orientation (e.g.A2=... B2=... C2=...) programmed between traversing blocks N10 and N20, then the inserted

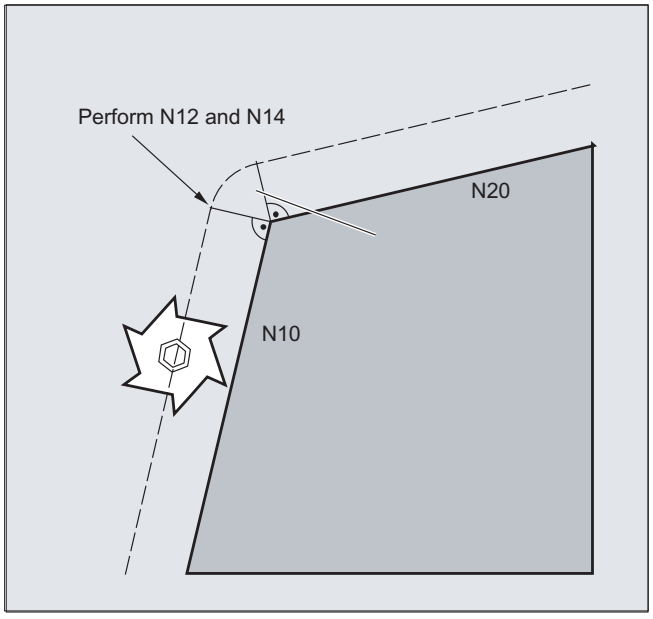
circle block is distributed among these intermediate blocks according to the absolute changes in angle.



Program code	Comment
ORIC	
N8 A2=… B2=… C2=…	
N10 X… Y… Z…	
N12 C2=… B2=…	; The circle block inserted at the external corner is distributed between N12 and N14, corresponding to the change in orientation. The circular motion and the orientation change are executed in parallel.
N14 C2=… B2=…	
N20 X =…Y=… Z=… G1 F200	

Example 2: ORID

If ORID is active, then all blocks between the two traversing blocks are executed at the end of the first traversing block. The circle block with constant orientation is executed immediately before the second traversing block.



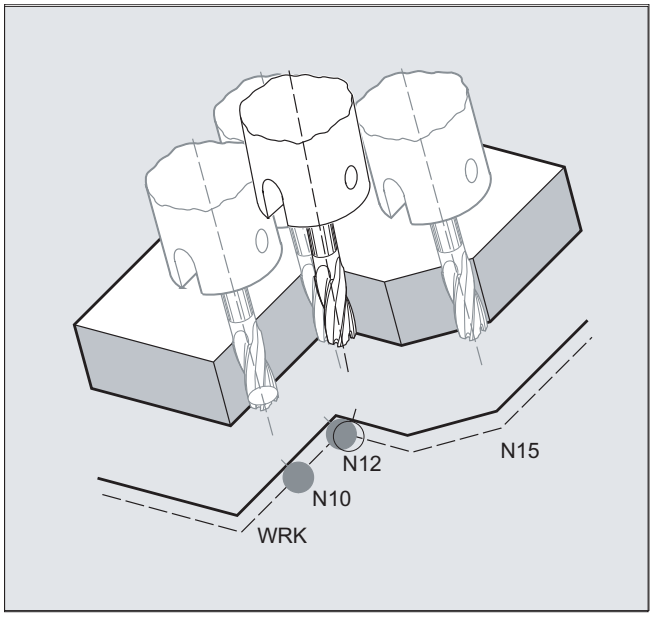
Program code	Comment
ORID	
N8 A2=… B2=… C2=…	
N10 X… Y… Z…	
N12 A2=… B2=… C2=…	; The N12 and N14 blocks are executed at the end of N10. The circle block is then executed with the actual orientation.
N14 M20	; Help functions, etc.
N20 X… Y… Z…	

Note

The method which is used to change orientation at an outer contour is determined using the program command that is active in the first traversing block of an outer corner.

Without change in orientation: If the orientation is not changed at the block boundary, the cross-section of the tool is a circle, which touches both of the contours.

Example 3: Change in orientation at an inside corner



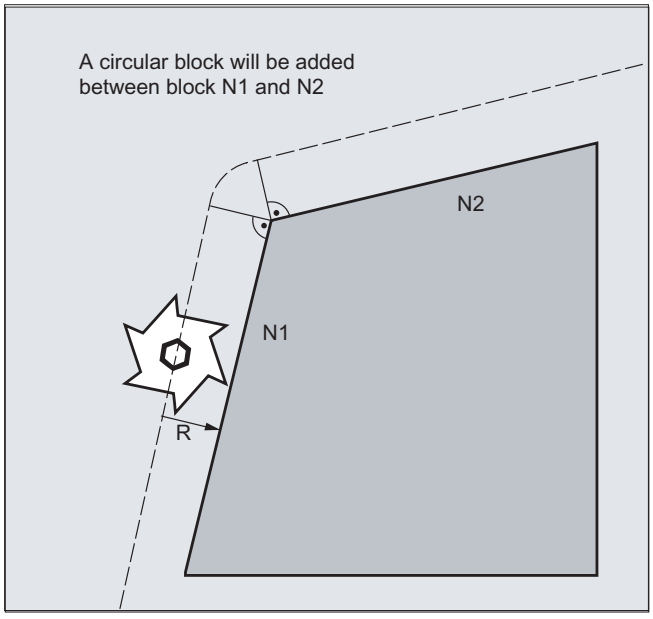

```
Program code
ORIC
N10 X …Y… Z… G1 F500
N12 X …Y… Z… A2=… B2=… C2=…
N15 X …Y… Z… A2=… B2=… C2=…
```

Further information

Behavior at outer corners

A circle block with the radius of the cutter is always inserted at an outside corner.

The `ORIC` and `ORID` program commands are used to determine whether changes in orientation programmed between block `N1` and `N2` are executed before the inserted circle block is processed or at the same time.



If an orientation change is required at outside corners, this can be performed either at the same time as interpolation or separately together with the path movement.

When ORID is programmed, the inserted blocks are executed first without path motion. The circle block generating the corner is inserted immediately before the second of the two traversing blocks.

If several orientation blocks are inserted at an external corner and ORIC is selected, the circular motion is distributed among the individual inserted blocks according to the absolute values of the orientation changes.

Smoothing orientation with OSD or OST

When blending with G642, the maximum variance for the contour axes and orientation axes cannot vary greatly. The smaller tolerance of the two determines the type of smoothing motion and/or angular tolerance to smooth the orientation characteristic relatively strongly without having to accept higher contour deviations.

OSD and OST can be activated to "generously" smooth very slight deviations from the orientation characteristics with a specified smoothing distance and angular tolerance without serious contour deviations.

Note

Unlike the process of rounding the contour (and orientation characteristics) with G642, when rounding the orientation with OSD and/or OST, a separate block is not formed, instead the rounding movement is added directly to the programmed original blocks.

With OSD and/or OST, block transitions cannot be rounded if there is a change in the type of interpolation for tool orientation (vector → rotary axis, rotary axis → vector). These block transitions can if necessary be rounded with the standard rounding functions G641, G642 and G643.

10.7 Free assignment of D numbers, cutting edge numbers

10.7.1 Free assignment of D numbers, cutting edge numbers (CE address)

D number

The D numbers can be used as contour numbers. You can also address the number of the cutting edge via the address CE. You can use the system variable $TC_DPCE to describe the cutting edge number.

Default: compensation no. == tool edge no.

Machine data are used to define the maximum number of D numbers (cutting edge numbers) and the maximum number of cutting edges per tool (→ machinery construction OEM). The following commands are only practical if the maximum cutting edge number (MD18105) was specified to be greater than the number of cutting edges per tool (MD18106). See machine manufacturer's specifications.

Note

In addition to relative D number allocation, the D numbers can also be assigned as "flat" or "absolute" D numbers (1-32000) without a reference to a T number (within the "Flat D number structure" function).

References

Function Manual Basic Functions; Tool Offset (W1)

10.7.2 Free assignment of D numbers: Checking D numbers (CHKDNO)

Using the `CKKDNO` command, you can check whether the existing D numbers were uniquely assigned. The D numbers of all tools defined within a TO unit may not occur more than once. No allowance is made for replacement tools.

Syntax

```
state=CHKDNO(Tno1,Tno2,Dno)
```

Meaning

`state`:	=TRUE:	The D numbers are assigned uniquely to the checked areas.
	= FALSE:	There was a D number collision or the parameters are invalid. Tno1, Tno2 and Dno return the parameters that caused the collision. These data can now be evaluated in the part program.

CHKDNO(Tno1,Tno2):	All D numbers of the part specified are checked.
CHKDNO(Tno1):	All D numbers of Tno1 are checked against all other tools.
CHKDNO:	All D numbers of all tools are checked against all other tools.

10.7.3 Free assignment of D numbers: Rename D numbers (GETDNO, SETDNO)

You must assign unique D numbers. Two different cutting edges of a tool must not have the same D number.

GETDNO

This command returns the D number of a particular cutting edge (ce) of a tool with tool number t. If no D number exists for the entered parameters, d=0 will be set. If the D number is invalid, a value greater than 32000 is returned.

SETDNO

This command assigns the value d of the D number to a cutting edge (ce) of tool t. The result of this statement is returned via state (TRUE or FALSE). If there is no data block for the specified parameter, the value FALSE is returned. Syntax errors generate an alarm. The D number cannot be set explicitly to 0.

Syntax

```
d = GETDNO (t,ce)
state = SETDNO (t,ce,d)
```

Meaning

d:	D number of the tool edge
t:	T number of the tool
ce:	Cutting edge number (CE number) of the tool
state:	Indicates whether the command could be executed (TRUE or FALSE).

Example for renaming a D number

```
Programming                                          Comment
$TC_DP2[1.2]=120
$TC_DP3[1,2]  = 5.5
$TC_DPCE[1,2] = 3                                    ; Cutting edge number CE
...
N10 def int DNoOld, DNoNew = 17
N20 DNoOld = GETDNO(1,3)
N30 SETDNO(1,3,DNoNew)
```

The new D value 17 is then assigned to cutting edge CE=3. Now the data for the cutting edge is addressed via D number 17; both via the system variables and in the programming with the NC address.

10.7.4 Free assignment of D numbers: Determine T number to the specified D number (GETACTTD)

You determine the T number associated with an absolute D number using the GETACTTD command. There is no check for uniqueness. If several D numbers within a TO unit are the same, the T number of the first tool found in the search is returned. This command is not suitable for use with "flat" D numbers, because the value "1" is always returned in this case (no T numbers in database).

Syntax

```
status=GETACTTD(Tnr,Dnr)
```

Meaning

Dnr:	D number for which the T number shall be searched.	
Tnr:	T number found	
status:	Value:	Meaning:
	0	The T number has been found. Tno contains the value of the T number.
	-1	No T number exists for the specified D number; Tno=0.
	-2	The D number is not absolute. Tno receives the value of the first found tool that contains the D number with the value Dno.
	-5	The function was not able to be executed for another reason.

10.7.5 Free assignment of D numbers: Invalidate D numbers (DZERO)

The DZERO command is used for support during retooling. Compensation data sets tagged with this command are no longer verified by the CHKDNO command. These data sets can be accessed again by setting the D number once more with SETDNO.

Syntax

```
DZERO
```

Meaning

DZERO:	Marks all D numbers of the TO unit as invalid.

10.8 Toolholder kinematics

Requirements

A toolholder can only orientate a tool in all possible directions in space if

- Two rotary axes V_1 and V_2 are present.
- The rotary axes are mutually orthogonal.
- The tool longitudinal axis is perpendicular to the second rotary axis V_2.

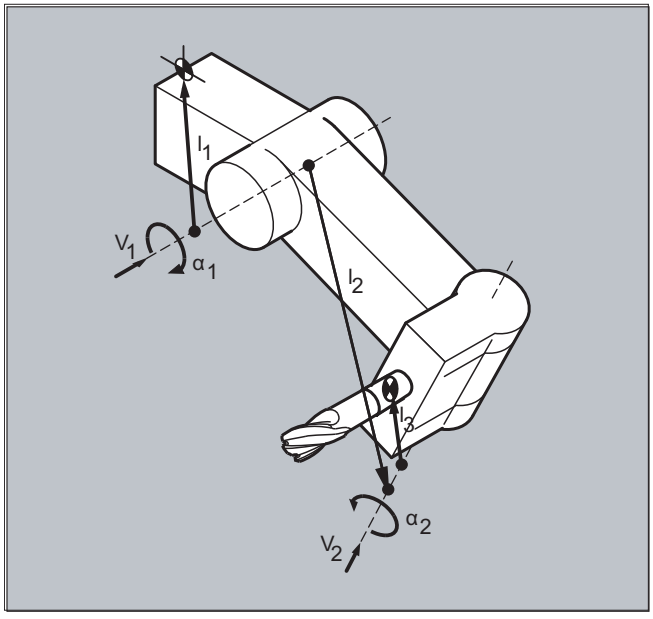
In addition, the following requirement is applicable to machines for which all possible orientations have to be settable:

- The tool longitudinal axis must be perpendicular to the first rotary axis V_1.

Function

The toolholder kinematics with a maximum of two rotary axes v_1 or v_2 are defined using the 17 system variables $TC_CARR1[m] to $TC_CARR17[m]. The description of the toolholder consists of:

- The vectoral distance from the first rotary axis of the toolholder I_1, the vectoral distance from the first rotary axis to the second rotary axis I_2, the vectoral distance from the second rotary axis to the reference point of the tool I_3.
- The direction vectors of both rotary axes V_1, V_2.
- The angles of rotation α1, α2 around the two axes. The rotation angles are counted in viewing direction of the rotary axis vectors, positive, in clockwise direction of rotation.



For machines with **resolved kinematics** (both the tool and the part can rotate), the system variables have been extended with the entries $TC_CARR18[m] to $TC_CARR23[m].

Parameters

Function of the system variables for orientable toolholders			
Designation	**x component**	**y component**	**z component**
l_1 offset vector	$TC_CARR1[m]	$TC_CARR2[m]	$TC_CARR3[m]
l_2 offset vector	$TC_CARR4[m]	$TC_CARR5[m]	$TC_CARR6[m]
v_1 rotary axis	$TC_CARR7[m]	$TC_CARR8[m]	$TC_CARR9[m]
v_2 rotary axis	$TC_CARR10[m]	$TC_CARR11[m]	$TC_CARR12[m]
α_1 angle of rotation α_2 angle of rotation	$TC_CARR13[m] $TC_CARR14[m]		
l_3 offset vector	$TC_CARR15[m]	$TC_CARR16[m]	$TC_CARR17[m]

Extensions of the system variables for orientable toolholders			
Designation	**x component**	**y component**	**z component**
l_4 offset vector	$TC_CARR18[m]	$TC_CARR19[m]	$TC_CARR20[m]
Axis identifier Rotary axis v_1 Rotary axis v_2	Axis identifier of the rotary axes v_1 and v_2 (initialized with zero) $TC_CARR21[m] $TC_CARR22[m]		
Kinematic type **T**ool **P**art **M**ixed mode	$TC_CARR23[m]		
	Kinematics type T ->	Kinematics type P ->	Kinematics type M
	Only the tool can rotate (default).	Only the part can rotate	Part and tool can rotate
Offset of the Rotary axis v_1 Rotary axis v_2	Angle in degrees of the rotary axes v_1 and v_2 on assuming the initial setting $TC_CARR24[m] $TC_CARR25[m]		
Angle offset of the rotary axis v_1 Rotary axis v_2	Offset of the Hirth tooth system in degrees for rotary axes v_1 and v_2 $TC_CARR26[m] $TC_CARR27[m]		
Angle increment v_1 rotary axis v_2 rotary axis	Offset of the Hirth tooth system in degrees for rotary axes v_1 and v_2 $TC_CARR28[m] $TC_CARR29[m]		
Min. position Rotary axis v_1 Rotary axis v_2	Software limit for the minimum position of the rotary axes v_1 and v_2 $TC_CARR30[m] $TC_CARR31[m]		
Max. position Rotary axis v_1 Rotary axis v_2	Software limits for the maximum position of the rotary axes v_1 and v_2 $TC_CARR32[m] $TC_CARR33[m]		
Toolholder name	A toolholder can be given a name instead of a number. $TC_CARR34[m]		
User: Axis name 1 Axis name 2 Identifier **Position**	Intended use in user measuring cycles $TC_CARR35[m] $TC_CARR36[m] $TC_CARR37[m]		
	$TC_CARR38[m]	$TC_CARR39[m]	$TC_CARR40[m]
Fine offset	Parameters that can be added **to the values in the basic parameters**.		
l_1 offset vector	$TC_CARR41[m]	$TC_CARR42[m]	$TC_CARR43[m]

Extensions of the system variables for orientable toolholders			
l_2 offset vector	$TC_CARR44[m]	$TC_CARR45[m]	$TC_CARR46[m]
l_3 offset vector	$TC_CARR55[m]	$TC_CARR56[m]	$TC_CARR57[m]
l_4 offset vector	$TC_CARR58[m]	$TC_CARR59[m]	$TC_CARR60[m]
v_1 rotary axis	$TC_CARR64[m]		
v_2 rotary axis	$TC_CARR65[m]		

Note

Explanations of parameters

"m" specifies the number of the toolholder to be programmed.

$TC_CARR47 to $TC_CARR54 and $TC_CARR61 to $TC_CARR63 are not defined and produce an alarm if read or write access is attempted.

The start/end points of the distance vectors on the axes can be freely selected. The rotation angles α_1, α_2 around the two axes are defined in the initial state of the toolholder by 0°. In this way, the kinematics of a toolholder can be programmed for any number of possibilities.

Toolholders with only one or no rotary axis at all can be described by setting the direction vectors of one or both rotary axes to zero.
With a toolholder without rotary axis the distance vectors act as additional tool offsets whose components cannot be affected by a change of machining plane (G17 to G19).

Parameter extensions

Parameters of the rotary axes

The system variables have been extended by the entries $TC_CARR24[m] to $TC_CARR33[m] and described as follows:

Offset of rotary axes v_1, v_2	Changing the position of the rotary axis v_1 or v_2 for the initial setting of the oriented toolholder.
The **angle offset/angle increment** of the rotary axes v_1, v_2	The offset or the angle increment of the Hirth tooth system of the rotary axes v_1 and v_2. Programmed or calculated angle is rounded up to the next value that results from phi = s + n * d when n is an integer.
The minimum and maximum position of the rotary axes v_1, v_2	The minimum and maximum position of the rotary axis limit angle (software limit) of the rotary axes v1 and v2.

Parameters for the user

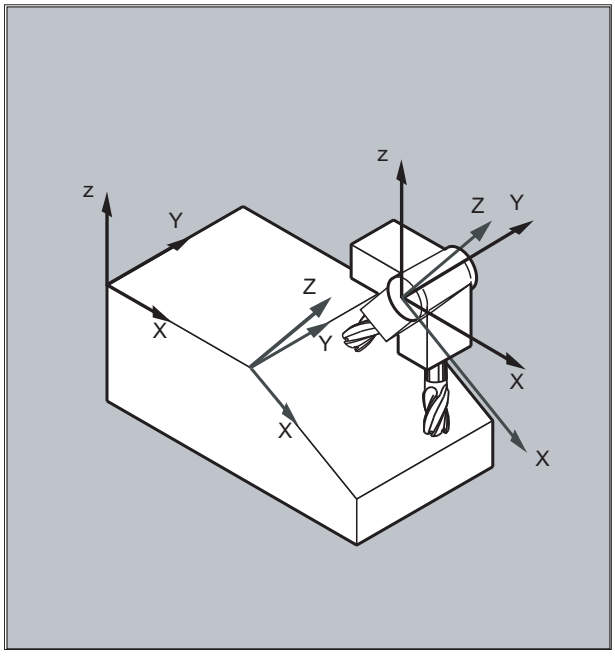
$TC_CARR34 to $TC_CARR40 contain parameters that are freely available to users and up to SW 6.4 were as standard, not further evaluated within the NCK or had no significance.

Fine offset parameters

$TC_CARR41 to $TC_CARR65 include fine offset parameters that can be added to the values in the basis parameters. The fine offset value assigned to a basic parameter is obtained when the value 40 is added to the parameter number.

Example

The toolholder used in the following example can be fully described by a rotation around the Y axis.



Program code	Comment
N10 $TC_CARR8[1]=1	; Definition of the Y component of the first rotary axis of toolholder 1.
N20 $TC_DP1[1,1] = 120	; Definition of a shaft miller.
N30 $TC_DP3[1,1]=20	; Definition of a shaft miller, 20 mm long.
N40 $TC_DP6[1,1]=5	; Definition of a shaft miller with 5 mm radius.
N50 ROT Y37	; Frame definition with 37° rotation around the Y axis.
N60 X0 Y0 Z0 F10000	; Approach start position.
N70 G42 CUT2DF TCOFR TCARR=1 T1 D1 X10	Set radius compensation, tool length compensation in rotated frame, select toolholder 1, tool 1.
N80 X40	; Perform machining under a rotation of 37°.
N90 Y40	
N100 X0	
N110 Y0	
N120 M30	

Further information

Resolved kinematics

For machines with resolved kinematics (both the tool as well as the workpiece can be rotated), the system variables have been expanded by the entries `$TC_CARR18[m]` up to `$TC_CARR23[m]` and are described as follows:

The rotatable tool table consisting of:

- The vectorial clearance of the second rotary axis V_2 to the reference point of a tool table that can be rotated I_4 of the third rotary axis.

The rotary axes consisting of:

- The two channel identifiers for the reference of the rotary axes $V_1$and V_2, whose position is, when required, accessed to determine the orientation of the toolholder that can be orientated.

The type of kinematics with one of the values T, P or M:

- Kinematics type T: Only tool can rotate.
- Kinematics type P: Only part can rotate.
- Kinematics type M: Tool and part can rotate.

Clearing the toolholder data

Data of all toolholder data sets can be deleted using `$TC_CARR1[0]=0`.

The kinematic type `$TC_CARR23[T]=T` must be assigned with one of the three permissible uppercase or lowercase letters (T,P,M) and for this reason, should not be deleted.

Changing the toolholder data

Each of the described values can be modified by assigning a new value in the part program. Any character other than T, P or M results in an alarm when an attempt is made to activate the toolholder that can be orientated.

Reading the toolholder data

Each of the described values can be read by assigning it to a variable in the part program.

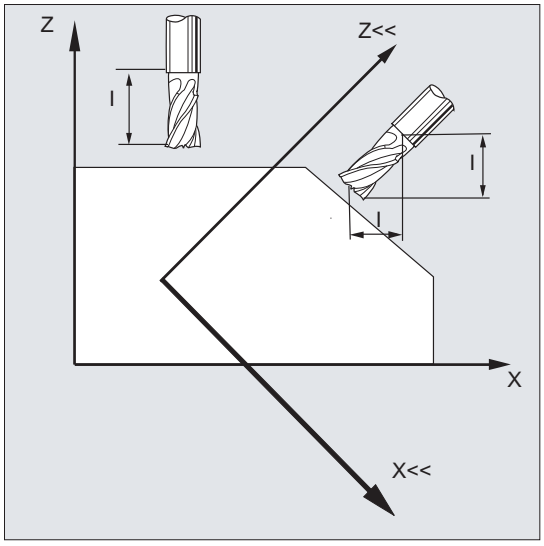
Fine offsets

An illegal fine offset value is only detected if a toolholder that can be orientated is activated, which contains such a value and at the same time setting data
SD42974 $SC_TOCARR_FINE_CORRECTION = TRUE.

The maximum permissible fine offset is limited to a permissible value in the machine data.

10.9 Tool length compensation for orientable toolholders (TCARR, TCOABS, TCOFR, TCOFRX, TCOFRY, TCOFRZ)

When the spatial orientation of the tool changes, its tool length components also change.



After a reset, e.g. through manual setting or change of the toolholder with a fixed spatial orientation, the tool length components also have to be determined again. This is performed using the TCOABS and TCOFR path commands.

For a toolholder of an active frame that can be orientated, when selecting the tool with TCOFRZ, TCOFRY and TCOFRX, it is possible to define the direction in which the tool should point.

Syntax

```
TCARR=[<m>]
TCOABS
TCOFR
TCOFRZ
TCOFRY
TCOFRX
```

Meaning

TCARR=[<m>]:	Request toolholder with the number "m"
TCOABS:	Determine tool length components from the orientation of the current toolholder
TCOFR:	Determine tool length components from the orientation of the active frame
TCOFRZ:	Orientable toolholder from active frame with a tool pointing in the Z direction
TCOFRY:	Orientable toolholder from active frame with a tool pointing in the Y direction
TCOFRX:	Orientable toolholder from active frame with a tool pointing in the X direction

Further information

Determine tool length offset from the orientation of the toolholder (TCOABS)

`TCOABS` calculates the tool length offset from the current orientation angles of the toolholder; saved in the system variables $TC_CARR13 and $TC_CARR14.

For a definition of toolholder kinematics with system variables, see "Toolholder kinematics (Page 406)".

In order to make a new calculation of the tool length offset when frames are changed, the tool has to be selected again.

Tool direction from active frame

The toolholder with orientation capability is set so that the tool points in the following directions:

- With `TCOFR` or `TCOFRZ` in the Z direction
- With `TCOFRY` in the Y direction
- With `TCOFRX` in the X direction

The tool length offset is re-calculated when changing over between `TCOFR` and `TCOABS`.

Request toolholder (TCARR)

With `TCARR`, the toolholder number m is requested with its geometry data (compensation memory).

With m=0, the active toolholder is deselected.

The geometry data of the toolholder only becomes active after a tool is called. The selected tool remains active after a toolholder change has taken place.

The current geometry data for the toolholder can also be defined in the part program via the corresponding system variables.

Recalculation of tool length offset (TCOABS) for a frame change

In order to make a new calculation of the tool length offset when frames are changed, the tool has to be selected again.

Note

The tool orientation must be manually adapted to the active frame.

When the tool length offset is calculated, the angle of rotation of the toolholder is calculated in an intermediate step. With toolholders with two rotary axes, there are generally two sets of rotation angles, which can be used to adapt the tool orientation to the active frame; therefore,

the rotation angle values stored in the system variables must at least correspond approximately to the mechanically set rotation angles.

Note

Tool orientation

It is not possible for the control to check whether the rotation angles calculated by means of the frame orientation are settable on the machine.

If the rotary axes of the toolholder are arranged such that the tool orientation calculated by means of the frame orientation cannot be reached, then an alarm is output.

The combination of tool precision compensation and the functions for tool length offset on movable toolholders is not permissible. If both functions are called simultaneously, an error message is issued.

The `TOFRAME` function allows a frame to be defined on the basis of the direction of orientation of the selected toolholder. For more information please refer to chapter "Frames".

When orientation transformation is active (3, 4 or 5-axis transformation), it is possible to select a toolholder with an orientation deviating from the zero position without causing output of an alarm.

Transfer parameter from standard and measuring cycles

For the transfer parameter of standard and measuring cycles, the following defined value ranges apply.

For angular value, the value range is defined as follows:

- Rotation around 1st geometry axis: -180 degrees to +180 degrees
- Rotation around 2nd geometry axis: -90 degrees to +90 degrees
- Rotation around 3rd geometry axis: -180 degrees to +180 degrees

Refer to Chapter Frames, "Programmable rotation (ROT, AROT, RPL)".

Note

When transferring angular values to a standard or measuring cycle, the following should be carefully observed:

Values less than the calculation resolution of the NC should be rounded-off to zero!

The calculation resolution of the NC for angular positions is defined in the machine data:

MD10210 $MN_INT_INCR_PER_DEG

10.10 Online tool length compensation (TOFFON, TOFFOF)

Use the system variable $AA_TOFF[<n>] to overlay the effective tool lengths in accordance with the three tool directions three-dimensionally in real time.

The three geometry axis identifiers are used as index <n>. Thus, the number of active direction offsets is determined by the geometry axes that are active at the same time.

All offsets can be active at the same time.

The online tool length offset function can be used for:

- Orientation transformation TRAORI
- Orientable toolholder TCARR

Note

Online tool length offset is an **option,** which must be enabled in advance. This function is only practical in conjunction with an active orientation transformation or an active orientable toolholder.

Syntax

```
TRAORI
TOFFON(<compensation direction>[,<offset value>])
WHEN TRUE DO $AA_TOFF[<compensation direction>]         ; In synchronized actions.
...
TOFFOF(<compensation direction>)
```

For more information about programming online tool length offset in motion-synchronous actions, see "Synchronized actions (Page 521)".

Meaning

TOFFON:	**Activate** online tool length offset	
	<compensation direction>:	Tool direction (X, Y, Z), in which the online tool length offset should be active.
	<offset value>:	When activating, an offset value can be specified for the relevant direction of compensation and this is immediately recovered.
TOFFOF:	**Reset** online tool length offset The compensation values in the specified compensation direction are reset and a preprocessing stop is initiated.	

Examples

Example 1: Selecting the tool length compensation

Program code	Comment
MD21190 $MC_TOFF_MODE = 1	; Absolute values are approached.
MD21194 $MC_TOFF_VELO[0] =1000	
MD21196 $MC_TOFF_VELO[1] =1000	
MD21194 $MC_TOFF_VELO[2] =1000	
MD21196 $MC_TOFF_ACCEL[0] =1	
MD21196 $MC_TOFF_ACCEL[1] =1	
MD21196 $MC_TOFF_ACCEL[2] =1	
N5 DEF REAL XOFFSET	
N10 TRAORI(1)	; Transformation on.
N20 TOFFON(Z)	; Activation of online tool length compensation for the Z tool direction.
N30 WHEN TRUE DO $AA_TOFF[Z]=10 G4 F5	; A TLC of 10 is interpolated for the Z tool direction.
...	
N100 XOFFSET=$AA_TOFF_VAL[X]	; Assigns actual compensation in the X direction.
N120 TOFFON(X,-XOFFSET) G4 F5	; For the X tool direction, the TLC is reduced back to 0.

Example 2: Deselect the tool length offset

Program code	Comment
N10 TRAORI(1)	; Transformation on.
N20 TOFFON(X)	; Activation of online tool length compensation for the X tool direction.
N30 WHEN TRUE DO $AA_TOFF[X]=10 G4 F5	; A TLC of 10 is interpolated for the X tool direction.
...	
N80 TOFFOF(X)	; Position offset of the X tool direction is deleted: ...$AA_TOFF[X]=0 No axis is moved. The position offset is added to the actual position in the Work corresponding to the actual orientation.

Further information

Block preparation

During block preparation in preprocessing, the current tool length offset active in the main run is also taken into consideration. To allow extensive use to be made of the maximum permissible axis velocity, it is necessary to stop block preparation with a `STOPRE` preprocessing stop while a tool offset is established.

The tool offset is always known at the time of run-in when the tool length offsets are not changed after program start or if more blocks have been processed after changing the tool length offsets than the IPO buffer can accommodate between run-in and main run.

Variable $AA_TOFF_PREP_DIFF

The dimension for the difference between the currently active compensation in the interpolator and the compensation that was active at the time of block preparation can be polled in the variable $AA_TOFF_PREP_DIFF[<n>].

Adjusting machine data and setting data

The following system data is available for online tool length offset:

- MD20610 $MC_ADD_MOVE_ACCEL_RESERVE (acceleration margin for overlaid motion)
- MD21190 $MC_TOFF_MODE
  Content of system variable $AA_TOFF[<n>] is moved through as absolute value or is integrated up.
- MD21194 $MC_TOFF_VELO (velocity of the online tool length offset)
- MD21196 $MC_TOFF_ACCEL (acceleration of the online tool length offset)
- Setting data for presetting limit values
  :
  SD42970 $SC_TOFF_LIMIT (upper limit of the tool length offset value)

Reference:
Function Manual, Special Functions; F2: Multi-axis transformations

10.11 Cutting data modification for tools that can be rotated (CUTMOD)

Using the function "cutting data modification for rotatable tools", the changed geometrical relationships, that are obtained relative to the workpiece being machined when rotating tools (predominantly turning tools, but also drilling and milling tools) can be taken into account with the tool compensation.

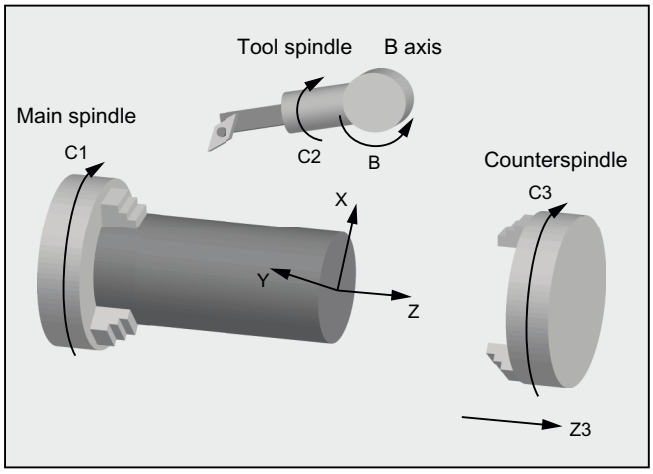


Figure 10-1 Rotatable tool for a lathe

The actual tool rotation is always determined from a currently active toolholder that can be orientated (refer to "Tool length offset for toolholders that can be orientated (Page 411)").

The function is activated using the command `_movePathCircular()`.

Syntax

```
CUTMOD=<value>
```

Meaning

CUTMOD:	Command to switch-in the function "cutting data modification for tools that can be rotated"	
<value>:	The following values can be assigned to the CUTMOD command:	
	0	The function is deactivated. The values supplied from system variables $P_AD... are the same as the corresponding tool parameters.
	> 0	The function is activated if a toolholder that can be orientated with the specified number is active, i.e. the activation is linked to a specific toolholder that can be orientated. The values supplied from system variables $P_AD... may be modified with respect to the corresponding tool parameters depending on the active rotation. The deactivation of the designated toolholder that can be orientated temporarily deactivates the function; the activation of another toolholder that can be orientated permanently deactivates it. This is the reason why in the first case, the function is re-activated when again selecting the same toolholder that can be orientated; in the second case, a new selection is required - even if at a subsequent time, the toolholder that can be orientated is re-activated with the specified number. The function is not influenced by a reset.
	-1	The function is always activated if a toolholder that can be orientated is active. When changing the toolholder or when de-selecting it and a subsequent new selection, CUTMOD does not have to be set again.
	-2	The function is always activated if a toolholder that can be orientated is active whose number is the same as the currently active toolholder that can be orientated. If a toolholder that can be orientated is not active, then this has the same significance as CUTMOD=0. If a toolholder that can be orientated is active, then this has the same significance as when directly specifying the actual toolholder number.
	< -2	Values less than -2 are ignored, i.e. this case is treated as if CUTMOD was not programmed. **Note:** This value range should not be used as it is reserved for possible subsequent expansions.

Note

SD42984 $SC_CUTDIRMOD

The CUTMOD command replaces the function that can be activated using the setting data SD42984 $SC_CUTDIRMOD. However, this function remains available unchanged. However, as it doesn't make sense to use both functions in parallel, it can only be activated if CUTMOD is equal to zero.

Example

The following example refers to a tool with cutting edge position 3 and a toolholder that can be orientated, which can rotate the tool around the B axis.

The numerical values in the comments specify the end of block positions in the machine coordinates (MCS) in the sequence X, Y, Z.

Program code	**Comment**		
N10 $TC_DP1[1,1]=500			
N20 $TC_DP2[1,1]=3	Cutting edge position		
N30 $TC_DP3[1,1]=12			
N40 $TC_DP4[1,1]=1			
N50 $TC_DP6[1,1]=6			
N60 $TC_DP10[1,1]=110	; Holder angle		
N70 $TC_DP11[1,1]=3	; Cut direction		
N80 $TC_DP24[1,1]=25	; Clearance angle		
N90 $TC_CARR7[2]=0 $TC_CARR8[2]=1 $TC_CARR9[2]=0	; B axis		
N100 $TC_CARR10[2]=0 $TC_CARR11[2]=0 $TC_CARR12[2]=1	; C axis		
N110 $TC_CARR13[2]=0			
N120 $TC_CARR14[2]=0			
N130 $TC_CARR21[2]=X			
N140 $TC_CARR22[2]=X			
N150 $TC_CARR23[2]="M"			
N160 TCOABS CUTMOD=0			
N170 G18 T1 D1 TCARR=2	X	Y	Z
N180 X0 Y0 Z0 F10000	12.000	0.000	1.000
N190 $TC_CARR13[2]=30			
N200 TCARR=2			
N210 X0 Y0 Z0	10.892	0.000	-5.134
N220 G42 Z-10	8.696	0.000	-17.330
N230 Z-20	8.696	0.000	-21.330
N240 X10	12.696	0.000	-21.330
N250 G40 X20 Z0	30.892	0.000	-5.134
N260 CUTMOD=2 X0 Y0 Z0	8.696	0.000	-7.330
N270 G42 Z-10	8.696	0.000	-17.330
N280 Z-20	8.696	0.000	-21.330
N290 X10	12.696	0.000	-21.330
N300 G40 X20 Z0	28.696	0.000	-7.330
N310 M30			

Explanations:

In block`N180`, initially the tool is selected for `CUTMOD=0` and non-rotated toolholders that can be orientated. As all offset vectors of the toolholder that can be orientated are 0, the position that corresponds to the tool lengths specified in `$TC_DP3[1,1]` and `$TC_DP4[1,1]` is approached.

The toolholder that can be orientated with a rotation of 30° around the B axis is activated in block `N200`. As the cutting edge position is not modified due to `CUTMOD=0`, the old cutting edge reference point is decisive just as before. This is the reason why in block `N210` the position is approached, which keeps the old tool nose reference point at the zero (i.e. the vector (1, 12) is rotated through 30° in the Z/X plane).

In block `N260`, contrary to block `N200 CUTMOD=2` is effective. As a result of the rotation of the toolholder that can be orientated, the modified cutting edge position becomes 8. The consequence of this is also the different axis positions.

The tool radius compensation (TRC) is activated in blocks `N220` and/or `N270`. The different cutting edge positions in both program sections has no effect on the end positions of the blocks in which the TRC is active; the corresponding positions are therefore identical. The different cutting edge positions only become effective again in the deselect blocks `N260` and/or `N300`.

Further information

Effectiveness of the modified cutting data

The modified cutting edge position and the modified cutting edge reference point are immediately effective when programming, even for a tool that is already active. A tool does not have to be re-selected for this purpose.

Influence of the active machining plane

To determine modified cutting edge position, cutting direction and holder or clearance angle, the evaluation of the cutting edge in the active plane (`G17` - `G19`) is decisive.

However, if setting data SD42940 $SC_TOOL_LENGTH_CONST (change of the tool length component when selecting the plane) has a valid non-zero value (plus or minus 17, 18 or 19), its contents define the plane in which the relevant quantities are evaluated.

System variables

The following system variables are available:

System variables	Meaning
$P_CUTMOD_ANG / $AC_CUTMOD_ANG	Supplies the (non-rounded) angle in the active machining plane that was used as basis for the modification of the cutting data (cutting edge position, cut direction, clearance angle and holder angle) for the functions activated using CUTMOD and/or $SC_CUTDIRMOD. $P_CUTMOD_ANG refers to the actual state in the preprocessing, $AC_CUTMOD_ANG to the actual main run block.
$P_CUTMOD / $AC_CUTMOD	Reads the currently valid value that was last programmed using the command CUTMOD (number of the toolholder that should be activated for the cutting data modification). If the last programmed CUTMOD value = -2 (activation with the currently active toolholder that can be orientated), then the value -2 is not returned in $P_CUTMOD, but rather the number of the active toolholder that can be orientated at the time of programming. $P_CUTMOD refers to the actual state in the preprocessing, $AC_CUTMOD to the actual main run block.
$P_CUT_INV / $AC_CUT_INV	Supplies the value TRUE if the tool is rotated so that the spindle direction of rotation must be inverted. To do this, the following four conditions must be fulfilled in the block to which the read operations refer: 1. If a turning or grinding tool is active (tool types 400 to 599 and / or SD42950 $SC_TOOL_LENGTH_TYPE = 2). 2. The cutting influence was activated using the language command CUTMOD. 3. A toolholder that can be orientated is active, which was designated using the numerical value of CUTMOD. 4. The toolholder that can be orientated rotates the tool around an axis in the machining plane (this is typically the C axis) so that the resulting perpendicular of the tool cutting edge is rotated with respect to the initial position by more than 90° (typically 180°). The contents of the variable is FALSE if at least one of the specified four conditions is not fulfilled. For tools whose cutting edge position is not defined, the value of the variable is always FALSE. $P_CUT_INV refers to the actual state in the preprocessing and $AC_CUT_INV to the actual main run block.

All main run variables ($AC_CUTMOD_ANG, $AC_CUTMOD and $AC_CUT_INV) can be read in synchronized actions. A read access operation from the preprocessing generates a preprocessing stop.

Modified cutting data:

If a tool rotation is active, the modified data is made available in the following system variables:

System variable	Meaning
$P_AD[2]	Cutting edge position
$P_AD[10]	Holder angle

System variable	Meaning
$P_AD[11]	Cut direction
$P_AD[24]	Clearance angle

Note

The data is always modified with respect to the corresponding tool parameters ($TC_DP2[..., ...] etc.) if the function "cutting data modification for rotatable tools" was activated using the command `CUTMOD` and a toolholder that can be orientated, which causes a rotation, is activated.

References

For additional information on the function "cutting data modification for rotatable tools", refer to:

Function Manual Basic Functions; Tool Offset (W1)

Path traversing behavior 11

11.1 Tangential control

11.1.1 Defining coupling (TANG)

Via the predefined procedure TANG(...), a tangential coupling between a rotary axis is defined as the following axis and two geometry axes as the leading axes. The following axis is continuously aligned with the path tangent of the leading axes.

Syntax

```
TANG(<following axis>, <leading axis_1>, <leading axis_2>, <coupling
factor>, <coordinate system>, <optimization>)
```

Meaning

TANG(...):	Define tangential coupling		
<following axis>:	Axis name of the following axis (rotary axis)		
	Data type:	AXIS	
	Range of values:	Channel axis names	
<leading axis_1> <leading axis_2>:	Axis names of the leading axes (geometry axes) [1]		
	Data type:	AXIS	
	Range of values:	Geometry axis names of the channel	
<coupling factor>:	Factor n of the angle change of the following axis for changing the path tangent of the leading axes: Angle change$_{\text{following axis}}$ = angle change$_{\text{path tangent}}$ * n		
	Data type:	REAL	
	Default value:	1.0	
<coordinate system>:	Active coordinate system [2]		
	Data type:	CHAR	
	Value:	"B":	Basic coordinate system (default value)
		"W":	Workpiece coordinate system (not available)

<optimization>:	Optimization type		
	Data type:	CHAR	
	Value:	"S":	Standard (default value) The dynamic response of the rotary axis has no effect on the leading axes If the dynamic response of the rotary axis is greater than required for tracking, this method is sufficiently precise. If the dynamic response of the rotary axis is not great enough to follow the change in the path tangent, the orientation of the rotary axis will deviate from the target orientation along an undefined rounding clearance.
		"P":	The dynamic response of the rotary axis is considered in the path planning of the leading axes. For this purpose, on activation of the tangential coupling with TANGON(), two additional parameters must be specified: • Rounding clearance • Angular tolerance See Section "Activating the coupling (TANGON) (Page 425)" **Note** With kinematic transformations, we recommend using optimization method "P."
Note Default values do not have to be programmed explicitly.			
[1] **Note** As the leading axes for tangential coupling, the geometry axes must be used that travel along the programmed path in the machine coordinate system (MCS) with reference to the initial position of the machine. For example, if swivel cycle CYCLE800 is used on a milling machine with a swivel head, depending on how the cycle is configured, interpolation will be performed in the WCS, e.g. with the geometry axes X and Y. The tangential coupling, however, must be defined with the geometry axes as the leading axes, which travel along the programmed path in the MCS. For this purpose, the geometry axes in the **non-swiveled** condition of the machine must be used as the leading axes.			
[2] **Note** The basic coordinate system (BCS) must not be rotated with respect to the MCS. For example, if the BCS is rotated with the ROT command or with the swivel cycle CYCLE800, the tangential control is no longer correct.			

11.1.2 Activating intermediate block generation (TLIFT)

If the tangent change of the following axis at any position along the programmed path of the leading axes exceeds the limit parameterized in machine data MD37400 $MA_EPS_TLIFT_TANG_STEP, further path planning will depend on the set behavior at corners. Without use of the predefined procedure TLIFT(...), the path is traversed in accordance with the rounding behavior programmed in connection with TANG(...) (Page 423) and TANGON(...) (Page 425).

Activating intermediate block generation

If TLIFT(...) is programmed after TANG(...), an intermediate block automatically generated by the control is inserted at this point when a corner is detected during preprocessing.

When the program is executed, the leading axes are stopped when the intermediate block is reached. In the intermediate block, the following axis is rotated with maximum axis dynamics toward the path tangent of the following block. The leading axes are then traversed further on the programmed path.

Deactivating intermediate block generation

To deactivate intermediate block generation, the tangential coupling must be defined again using TANG(...), but without subsequent activation of intermediate block generation by means of TLIFT(...).

Syntax

```
TLIFT(<following axis>)
```

Meaning

`TLIFT(...)`:	Activate corner detection with intermediate block calculation	
`<following axis>`:	Axis name of the following axis (rotary axis)	
	Data type:	AXIS
	Range of values:	Channel axis names

Speed of rotation of the following axis

Path axis

If the following axis had already been traversed as a path axis before tangential coupling was activated, the rotational movement is performed in the intermediate block as a path axis.

If you specify the reference radius with `FGREF[<axis>]=0.001`, the rotational movement will be performed with the parameterized maximum axis velocity:

MD32000 $MA_MAX_AX_VELO[<following axis>]

Positioning axis

If the following axis had not yet been traversed as a path axis before tangential coupling was activated, the rotation is performed in the intermediate block as a positioning axis.

The rotational movement is performed with the parameterized positioning axis velocity:

MD32060 $MA_POS_AX_VELO[<following axis>]

11.1.3 Activating the coupling (TANGON)

Via the predefined procedure TANGON(...), a tangential coupling previously defined with TANG(...) (Page 423) is activated. The following axis is then continuously aligned with the path tangent during subsequent travel.

Angle of the following axis

The angle of the following axis with respect to the path tangent depends on the transformation ratio specified in TANG(...), the offset angle parameterized in the machine data MD37402 $MA_TANG_OFFSET, and the offset angle specified for TANGON(...), which is applied additively.

Optimization "P"

If the value "P" was specified as the optimization parameter in the definition of the tangential coupling (TANG(...)), the parameter "rounding clearance" and optionally the parameter "angular tolerance" must be set when coupling is activated.

If the value 0 is specified as the angular tolerance, only the parameter "rounding clearance" will be active.

If a value greater than 0 is specified as the angular tolerance, the active rounding clearance results from the minimum of the parameterized rounding clearance and the rounding clearance based on the parameterized angular tolerance.

If the dynamic response of the following axis is not sufficient to follow the parameterized conditions, the path velocity of the leading axes will be reduced accordingly.

Syntax

```
TANGON(<following axis>, <offset angle>, <rounding clearance>,
<angular tolerance>)
```

Meaning

`TANGON(...)`:	Activate tangential coupling	
`<following axis>`:	Axis name of the following axis (rotary axis)	
	Data type:	AXIS
	Range of values:	Channel axis names
`<offset angle>`:	Offset angle of following axis with respect to the path tangent The reference point is the zero point of the rotary axis.	
	Data type:	REAL
`<rounding clearance>`:	Maximum permissible rounding clearance If the rounding clearance is increased due to the dynamic conditions, the path velocity of the leading axes is reduced.	
	Data type:	REAL
`<angular tolerance>`:	Maximum permissible tolerance with respect to the specified angle between the following axis zero setting and the path tangent	
	Data type:	REAL

11.1.4 Deactivating the coupling (TANGOF)

Via the predefined procedure TANGOF(...), a tangential coupling defined with TANG(...) (Page 423) and activated with TANGON(...) (Page 425) is deactivated. The following axis is then no longer aligned with the path tangent of the leading axis. However, the coupling of the following axis to the leading axes is retained even after deactivation, which prevents the following functions, for example:

- Plane change
- Geometry axis switchover
- Definition of a new tangential coupling for the following axis

Final cancellation of the connection of the coupling of the following axis to the leading axes is not completed until the coupling has been deleted with TANGDEL(...) (Page 427).

Programming

```
TANGOF(<following axis>)
```

Meaning

`TANGOF(...)`:	Deactivate a tangential coupling	
`<following axis>`:	Axis name of the following axis (rotary axis)	
	Data type:	AXIS
	Range of values:	Channel axis names

11.1.5 Deleting a coupling (TANGDEL)

A tangential coupling defined with TANG(...) (Page 423) will be retained even after deactivation of the tangential coupling with TANGOF(...) (Page 427). The existing tangential coupling then continues to prevent, for example, the following functions:

- Plane change
- Geometry axis switchover
- Definition of a new tangential coupling for the following axis

With the predefined procedure TANGDEL(...), the existing tangential coupling is deleted after the tangential coupling has been deactivated with TANGOF(...).

Syntax

```
TANGDEL(<following axis>)
```

Meaning

`TANGDEL(...)`:	Delete a tangential coupling defined with TANG()	
	Effective:	Non-modal

<following axis>:	Axis name of the following axis whose tangential coupling is to be deleted	
	Data type:	AXIS
	Range of values:	Channel axis names

Examples

Leading axis change

Before a new tangential coupling can be defined with another leading axis for the following axis, the existing tangential coupling must first be deleted.

```
Program code                    Comment
N10 TANG(A, X, Y, 1)            ; Define tangential couping for following axis A: A to
                                X and Y
N20 TANGON(A)                   ; Activate tangential coupling for following axis A
N30 X10 Y20
...
N80 TANGOF(A)                   ; Deactivate tangential coupling for following axis A
N90 TANGDEL(A)                  ; Delete tangential coupling for following axis A
...
N120 TANG(A, X, Z)              ; Define new tangential coupling for following axis A
N130 TANGON(A)                  ; Activate new tangential coupling for following axis A
...
```

Geometry axis switchover

Before geometry axis switchover can be performed for an existing coupling, the coupling must first be deleted.

```
Program code                    Comment
N10 GEOAX(2,Y1)                 ; 2nd geometry axis = machine axis Y1
N20 TANG(A, X, Y)               ; Define tangential coupling for following axis A
N30 TANGON(A, 90)               ; Activate tangential coupling for following axis A
N40 G2 F8000 X0 Y0 I0 J50       ; Motion block
N50 TANGOF(A)                   ; Deactivate tangential coupling for following axis A
N60 TANGDEL(A)                  ; Delete tangential coupling for following axis A
N70 GEOAX(2, Y2)                ; 2nd geometry axis = machine axis Y2
N80 TANG(A, X, Y)               ; Define new tangential coupling for following axis A
N90 TANGON(A, 90)               ; Activate new tangential coupling for following axis
                                A
...
```

11.2 Feedrate characteristic (FNORM, FLIN, FCUB, FPO)

To permit flexible definition of the feedrate characteristic, the feedrate programming according to DIN 66025 has been extended by linear and cubic characteristics.

The cubic characteristics can be programmed either directly or as interpolating splines. These additional characteristics make it possible to program continuously smooth velocity characteristics depending on the curvature of the workpiece to be machined.

These additional characteristics make it possible to program continuously smooth velocity characteristics depending on the curvature of the workpiece to be machined.

Syntax

```
F… FNORM
F… FLIN
F… FCUB
F=FPO(…,…,…)
```

Meaning

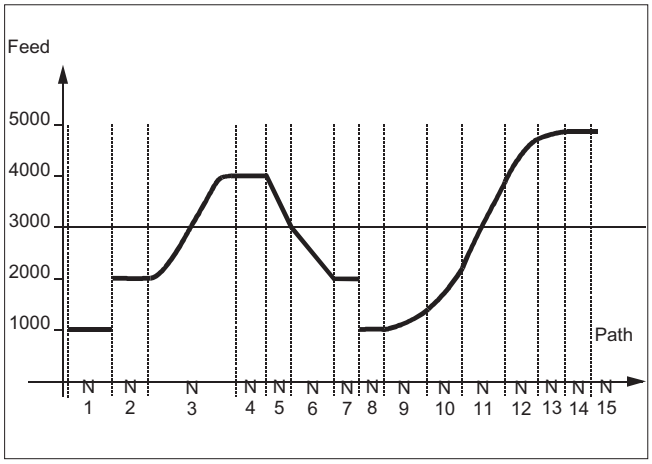
FNORM:	Basic setting. The feed value is specified as a function of the traverse path of the block and is then valid as a modal value.
FLIN:	Path velocity profile **linear:** The feed value is approached linearly via the traverse path from the current value at the block beginning to the block end and is then valid as a modal value. The response can be combined with G93 and G94.
FCUB:	Path velocity profile **cubic:** The blockwise programmed F values (relative to the end of the block) are connected by a spline. The spline begins and ends tangentially with the previous and following defined feedrate and takes effect with G93 and G94. If the F address is missing from a block, the last F value to be programmed is used.
F=FPO… :	**Polynomial** path velocity profile: The F address defines the feed characteristic via a polynomial from the current value to the block end. The end value is valid thereafter as a modal value.

Feed optimization on curved path sections

Feed polynomial F=FPO and feed spline FCUB should always be traversed at constant cutting rate CFC, thereby allowing a jerk-free setpoint feed profile to be generated. This enables creation of a continuous acceleration setpoint feed profile.

Example: Various feed profiles

This example shows you the programming and graphic representation of various feed profiles.



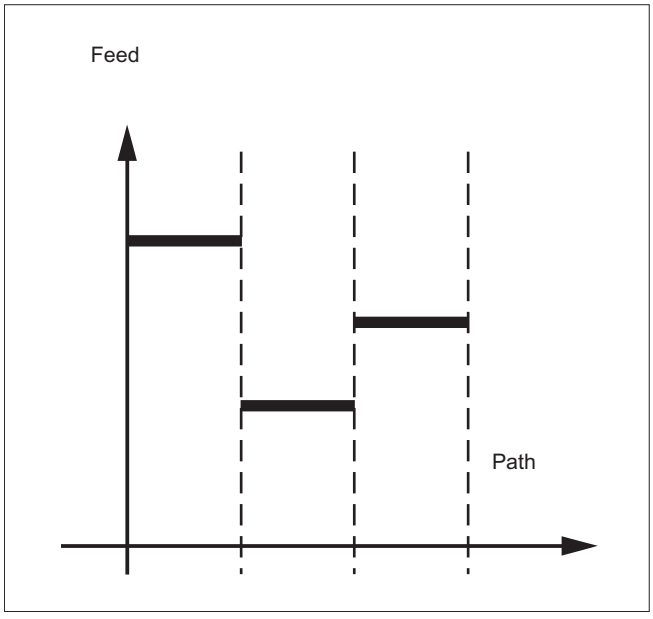
Program code	Comment
N1 F1000 FNORM G1 X8 G91 G64	; Constant feedrate profile, incremental dimension data
N2 F2000 X7	; Setpoint velocity step change
N3 F=FPO(4000, 6000, -4000)	; Feed profile via polynomial with feed 4000 at the end of the block
N4 X6	; Polynomial feedrate 4000 is valid as modal value
N5 F3000 FLIN X5	; Linear feedrate profile
N6 F2000 X8	; Linear feedrate profile
N7 X5	; Linear feedrate is valid as modal value
N8 F1000 FNORM X5	; Constant feedrate profile with acceleration step change
N9 F1400 FCUB X8	; All of the following F values programmed in blocks are connected with splines
N10 F2200 X6	
N11 F3900 X7	
N12 F4600 X7	
N13 F4900 X5	; Switch-out spline profile
N14 FNORM X5	
N15 X20	

Further information

FNORM

The feed address F defines the path feedrate as a constant value according to DIN 66025.

Please refer to Programming Manual "Fundamentals" for more detailed information on this subject.

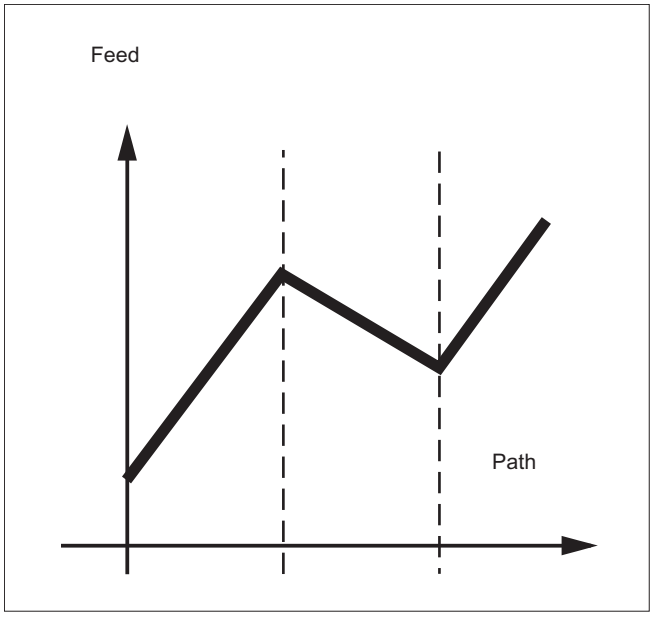


FLIN

The feedrate characteristic is approached linearly from the current feedrate value to the programmed F value until the end of the block.

Example:

```
N30 F1400 FLIN X50
```



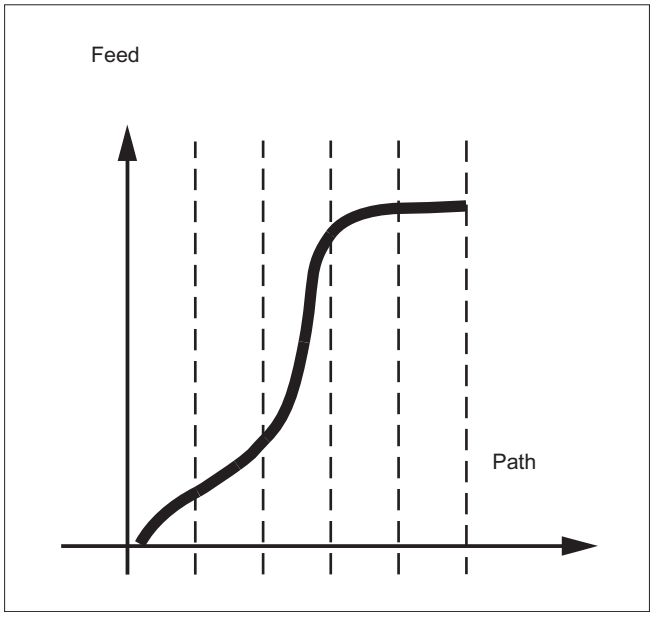
FCUB

The feedrate is approached according to a cubic characteristic from the current feedrate value to the programmed F value until the end of the block. The control uses splines to connect all

the feedrate values programmed non-modally that have an active FCUB. The feedrate values act here as interpolation points for calculation of the spline interpolation.

Example:

```
N50 F1400 FCUB X50
N60 F2000 X47
N70 F3800 X52
```



F=FPO(...,...,...)

The feedrate characteristic is programmed directly via a polynomial. The polynomial coefficients are specified according to the same method used for polynomial interpolation.

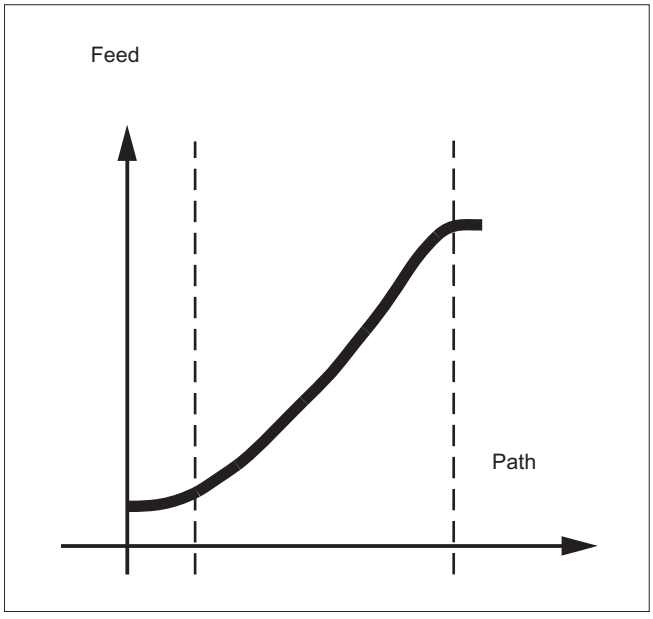
Example:

```
F=FPO(endfeed, quadf, cubf)
```

endfeed, quadf and cubf are previously defined variables.

endfeed:	Feedrate at block end
quadf:	Quadratic polynomial coefficient
cubf:	Cubic polynomial coefficient

With an active `FCUB`, the spline is linked tangentially to the characteristic defined via `FPO` at the block beginning and block end.



Supplementary conditions

- The functions for programming the path traversing characteristics apply regardless of the programmed feedrate characteristic.
- The programmed feedrate characteristic is always absolute regardless of `G90` or `G91`.
- The feed characteristic curve `FLIN` and `FCUB` does **not** act with `G93` and `G94` for `G95`, `G96/G961` and `G97/G971`.
- With an active compressor `COMPON` the following applies when several blocks are joined to form a spline segment:

`FNORM`:	The F word of the last block in the group applies to the spline segment.
`FLIN`:	The F word of the last block in the group applies to the spline segment. The programmed F value applies until the end of the segment and is then approached linearly.
`FCUB`:	The generated feedrate spline deviates from the programmed end points by an amount not exceeding the value set in machine data MD20172 $MC_COMPRESS_VELO_TOL.
`F=FPO(…,…,…)`:	These blocks are not compressed.

11.3 Acceleration behavior

11.3.1 Acceleration mode (BRISK, BRISKA, SOFT, SOFTA, DRIVE, DRIVEA)

The following part program commands are available for programming the current acceleration mode:

- "BRISK, BRISKA"
  The single axes or the path axes traverse with maximum acceleration until the programmed feedrate is reached (**acceleration without jerk limitation**).
- "SOFT, SOFTA"
  The single axes or the path axes traverse with constant acceleration until the programmed feedrate is reached (**acceleration with jerk limitation**).
- "DRIVE, DRIVEA"
  The single axes or the path axes traverse with maximum acceleration up to a programmed velocity limit (MD setting!). The acceleration rate is then reduced (MD setting) until the programmed feedrate is reached.

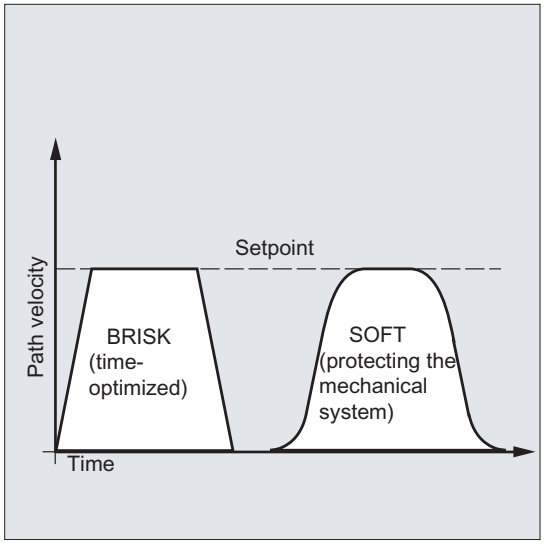


Figure 11-1 Path velocity curve with `BRISK` and `SOFT`

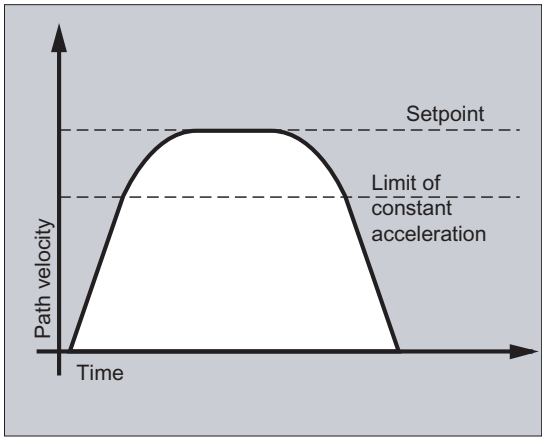


Figure 11-2 Path velocity curve with DRIVE

Syntax

```
BRISK
BRISKA(<axis1>,<axis2>,…)
SOFT
SOFTA(<axis1>,<axis2>,…)
DRIVE
DRIVEA(<axis1>,<axis2>,…)
```

Meaning

BRISK:	Command for activating the "acceleration without jerk limitation" for the path axes.
BRISKA:	Command for activating the "acceleration without jerk limitation" for single axis movements (JOG, JOG/INC, positioning axis, oscillating axis, etc.).
SOFT:	Command for activating the "acceleration with jerk limitation" for the path axes.
SOFTA:	Command for activating the "acceleration with jerk limitation" for single axis movements (JOG, JOG/INC, positioning axis, oscillating axis, etc.).
DRIVE:	Command for activating the reduced acceleration above a configured velocity limit (MD35220 $MA_ACCEL_REDUCTION_SPEED_POINT) for the path axes.
DRIVEA:	Command for activating the reduced acceleration above a configured velocity limit (MD35220 $MA_ACCEL_REDUCTION_SPEED_POINT) for single axis movements (JOG, JOG/INC, positioning axis, oscillating axis, etc.).
(<axis1>,<axis2>, etc.):	Single axes for which the called acceleration mode is to apply.

Supplementary conditions

Changing acceleration mode during machining

If the acceleration mode is changed in a part program during machining (BRISK ↔ SOFT), then there is a block change with exact stop at the end of the block during the transition even with continuous-path mode.

Examples

Example 1: SOFT and BRISKA

Program code

```
N10 G1 X… Y… F900 SOFT
N20 BRISKA(AX5,AX6)
...
```

Example 2: DRIVE and DRIVEA

Program code

```
N05 DRIVE
N10 G1 X… Y… F1000
N20 DRIVEA (AX4, AX6)
...
```

References

Function Manual, Basic Functions; Acceleration (B2)

11.3.2 Influence of acceleration on following axes (VELOLIMA, ACCLIMA, JERKLIMA)

In the case of axis couplings (tangential correction, coupled motion, master value coupling, electronic gear; see "Axis couplings (Page 467)").

The dynamics limits of the following axes/spindles can be manipulated using the VELOLIMA, ACCLIMA, and JERKLIMA functions from the part program or from synchronized actions, even if the axis coupling is already active.

Note

The JERKLIMA function is not available for all types of coupling.

References:

- Function Manual, Special Functions; Axis Couplings (M3)
- Function Manual, Extended Functions; Synchronous Spindle (S3)

Note

Availability for SINUMERIK 828D

The VELOLIMA, ACCLIMA and JERKLIMA functions can only be used with SINUMERIK 828D in conjunction with the "coupled motion" function!

Syntax

```
VELOLIMA(<axis>)=<value>
ACCLIMA(<axis>)=<value>
JERKLIMA(<axis>)=<value>
```

Meaning

VELOLIMA:	Command to correct the parameterized maximum **velocity**
ACCLIMA:	Command to correct the parameterized maximum **acceleration**
JERKLIMA:	Command to correct the parameterized maximum **jerk**
<axis>:	Following axis whose dynamics limits need to be corrected
<value>:	Percentage correction value

Examples

Example 1: Correction of the dynamics limits for a following axis (AX4)

Program code	Comment
...	
VELOLIMA[AX4]=75	; Limit correction to 75% of the maximum axial velocity stored in the machine data
ACCLIMA[AX4]=50	; Limit correction to 50% of the maximum axial acceleration stored in the machine data
JERKLIMA[AX4]=50	; Limit correction to 50% of the maximum axial jerk stored in the machine data
...	

Example 2: Electronic gear

Axis 4 is coupled to axis X via an "electronic gear" coupling. The acceleration capacity of the following axis is limited to 70% of the maximum acceleration. The maximum permissible velocity is limited to 50% of the maximum velocity. Once the coupling has been activated successfully, the maximum permissible velocity is restored to 100%.

Program code	Comment
...	
N120 ACCLIMA[AX4]=70	; Reduced maximum acceleration.
N130 VELOLIMA[AX4]=50	; Reduced maximum velocity.
...	
N150 EGON(AX4,"FINE",X,1,2)	; Activation of the EG coupling.

Program code	Comment
...	
N200 VELOLIMA[AX4]=100	; Full maximum velocity.
...	

Example 3: Influencing master value coupling by static synchronized action

Axis 4 is coupled to X by master value coupling. The acceleration response is limited to position 80% by static synchronized action 2 from position 100.

Program code	Comment
...	
N120 IDS=2 WHENEVER $AA_IM[AX4] > 100 DO ACCLIMA[AX4]=80	; Synchronized action
N130 LEADON(AX4, X, 2)	; Master value coupling on
...	

11.3.3 Activation of technology-specific dynamic values (DYNNORM, DYNPOS, DYNROUGH, DYNSEMIFIN, DYNFINISH)

Using the "Technology" G group, the appropriate dynamic response can be activated for five varying technological machining steps.

Dynamic values and G commands can be configured and are, therefore, dependent on machine data settings (→ machine manufacturer).

References:
Function Manual, Basic Functions; Continuous-Path Mode, Exact Stop, Look Ahead (B1)

Syntax

Activate dynamic values:

```
DYNNORM
DYNPOS
DYNROUGH
DYNSEMIFIN
DYNFINISH
```

Note

The dynamic values are already active in the block in which the associated G command is programmed. Machining is not stopped.

Read or write a specific field element:

```
R<m>=$MA...[n,X]
$MA...[n,X]=<value>
```

Meaning

DYNNORM:	G command for activating the **normal dynamic response**		
DYNPOS:	G command for activating the dynamic response for **positioning mode**, **tapping**		
DYNROUGH:	G command for activating the dynamic response for **roughing**		
DYNSEMIFIN:	G command for activating the dynamic response for **finishing**		
DYNFINISH:	G command for activating the dynamic response for **smooth-finishing**		
R<m>:	R-parameter with number <m>		
$MA...[n,X]:	Machine data with field element affecting dynamic response		
<n>:	Array index		
	Range of values:	0 ... 4	
	0	Normal dynamic response (DYNNORM)	
	1	Dynamic response for positioning mode (DYNPOS)	
	2	Dynamic response for roughing (DYNROUGH)	
	3	Dynamic response for finishing (DYNSEMIFIN)	
	4	Dynamic response for smooth-finishing (DYNFINISH)	
<X> :	Axis address		
<value>:	Dynamic value		

Examples

Example 1: Activate dynamic values

Program code	Comment
DYNNORM G1 X10	; Initial setting
DYNPOS G1 X10 Y20 Z30 F…	; Positioning mode, tapping
DYNROUGH G1 X10 Y20 Z30 F10000	;Roughing
DYNSEMIFIN G1 X10 Y20 Z30 F2000	;Finishing
DYNFINISH G1 X10 Y20 Z30 F1000	; Smooth finishing

Example 2: Read or write a specific field element

Maximum acceleration for roughing, axis X

Program code	Comment
R1=$MA_MAX_AX_ACCEL[2,X]	; reading
$MA_MAX_AX_ACCEL[2,X]=5	; writing

11.4 Traversing with feedforward control (FFWON, FFWOF)

The feedforward control reduces the velocity-dependent overtravel when contouring towards zero. Traversing with feedforward control permits higher path accuracy and thus improved machining results.

Syntax

```
FFWON
FFWOF
```

Meaning

FFWON:	Command to **activate** the feedforward control
FFWOF:	Command to **deactivate** the feedforward control

Note

The type of feedforward control and which path axes are to be traversed with feedforward control is specified via machine data.

Default: Velocity-dependent feedforward control

Option: Acceleration-dependent feedforward control

Example

Program code
N10 FFWON
N20 G1 X… Y… F900 SOFT

11.5 Programmable contour accuracy (CPRECON, CPRECOF)

The "Programmable contour accuracy" function reduces the path error on curved contours through automatic adaptation of the velocity.

The contour accuracy to be maintained is specified depending on the configuration of the machine (MD20470 $MC_MC_CPREC_WITH_FFW; see machine manufacturer specifications) either via the setting data $SC_CONTPREC or via the programmed contour tolerance CTOL. The smaller the value and the smaller the K_V factor of the geometry axes, the greater the path feedrate is reduced on curved contours.

The "Programmable contour accuracy" function is activated or deactivated via the operations CPRECON and CPRECOF in the NC program.

Syntax

```
CPRECON
...
CPRECOF
```

Meaning

CPRECON:	G function call: Switch "Programmable contour accuracy" **on**	
	Effective:	Modal
CPRECOF:	G function call: Switch "Programmable contour accuracy" **off**	
	Effective:	Modal

Together CPRECON and CPRECOF form the G function group 39 (programmable contour accuracy).

Note

The user can specify a minimum velocity for the path feedrate via the setting data $SC_MINFEED (minimum path feedrate with CPRECON).

The feedrate is not limited below this value, unless a lower F value has been programmed or the dynamic limits of the axes require a lower path velocity.

Note

The "Programmable contour accuracy" function only considers the geometry axes of the path. It has no effect on the velocities of positioning axes.

Example

Program code	**Comment**
N10 G0 X0 Y0	
N20 CPRECON	; Activate the "programmable contour accuracy".
N30 G1 G64 X100 F10000	; Machining with 10 m/min in the continuous-path mode.
N40 G3 Y20 J10	; Automatic feed limitation in circular block.
N50 G1 X0	; Feedrate again without limitation (10 m/min).
...	
N100 CPRECOF	; Deactivate the "programmable contour accuracy".
N110 G0 ...	

References

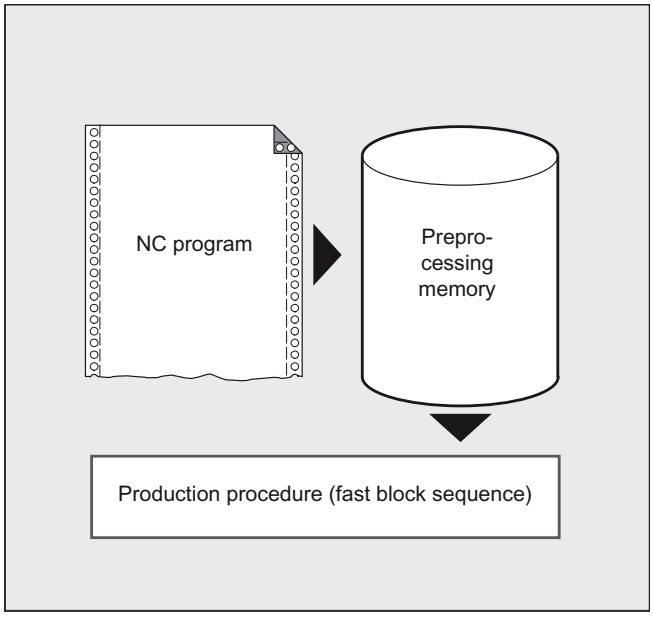
For the programming of CTOL, see "Programmable contour/orientation tolerance (CTOL, OTOL, ATOL) (Page 461)"

For more detailed information on the "Programmable contour accuracy" function, see:

Function Manual, Special Functions; Contour Tunnel Monitoring (K6), Section: "Programmable contour accuracy"

11.6 Program sequence with preprocessing memory (STOPFIFO, STARTFIFO, FIFOCTRL, STOPRE)

Depending on its expansion level, the control system has a certain quantity of so-called preprocessing memory in which prepared blocks are stored prior to program execution and then output as high-speed block sequences while machining is in progress. These sequences allow short paths to be traversed at a high velocity. Provided that there is sufficient residual control time available, the preprocessing memory is always filled.



Designate machining step

The beginning and end of the machining step to be buffered in the preprocessing memory are identified in the part program with "STOPFIFO" and "STARTFIFO" respectively. The processing of the preprocessed and buffered blocks starts only after the "STARTFIFO" command or if the preprocessing memory is full.

Automatic preprocessing memory control

Automatic preprocessing memory control is called with the "FIFOCTRL" command. "FIFOCTRL" acts initially just like "STOPFIFO". Whatever the programming, processing will not start until the preprocessing memory is full. However, the response to the emptying of the preprocessing memory does differ: With "FIFOCTRL", the path velocity is reduced increasingly once the fill level reaches 2/3 in order to prevent complete emptying and deceleration to standstill.

Preprocessing stop

Programming the "STOPRE" command in a block will stop block preprocessing and buffering. The following block is not executed until all preprocessed and saved blocks have been executed in full. The preceding block is halted in exact stop (as with G9).

NOTICE

Program abort

If tool offset or spline interpolations are active, a "STOPRE" command should not be programmed, as this will lead to contiguous block sequences being interrupted.

Syntax

Table 11-1 Identify machining step:

```
STOPFIFO
...
STARTFIFO
```

Table 11-2 Automatic preprocessing memory control:

```
...
FIFOCTRL
...
```

Table 11-3 Preprocessing stop:

```
...
STOPRE
...
```

Note

The "STOPFIFO", "STARTFIFO", "FIFOCTRL" and "STOPRE" commands must be programmed in their own block.

Meaning

STOPFIFO:	"STOPFIFO" identifies the start of a machining step to be buffered in the preprocessing memory. "STOPFIFO" stops processing and fills the preprocessing memory until: • "STARTFIFO" or "STOPRE" is recognized or • The preprocessing memory is full or • The end of the program is reached
STARTFIFO:	"STARTFIFO" starts rapid processing of the machining step; the preprocessing memory is filled in parallel to this.
FIFOCTRL:	Activation of automatic preprocessing memory control
STOPRE:	Stop preprocessing

Note

The preprocessing memory is not filled or filling is interrupted if the machining step contains commands that require unbuffered operation (search for reference, measuring functions, etc.).

Note

The control generates an internal preprocessing stop in the event of access to status data ($SA...).

Example: Stop preprocessing

Program code	Comment
...	
N30 MEAW=1 G1 F1000 X100 Y100 Z50	; Measurement block with probe at first measuring input and linear interpolation.
N40 STOPRE	; Preprocessing stop.
...	

11.7 Influencing the stop events by stop delay areas (DELAYFSTON, DELAYFSTOF)

The definition of a stop delay area is made in the part program with the predefined DELAYFSTON and DELAYFSTOF procedures.

Note

DELAYFSTON and DELAYFSTOF are **not** permitted in synchronized actions!

Syntax

```
DELAYFSTON
...
DELAYFSTOF
```

Note

DELAYFSTON and DELAYFSTOF are programmed separately in a part program line.

Meaning

DELAYFSTON:	Define the start of a range in which "soft" stops are delayed up to the end of the stop delay area.
DELAYFSTOF:	Define the end of the stop delay area

System variables

The following system variables can be queried to determine whether part program processing is currently in a stop delay area:

- in the part program with $P_DELAYFST
- in synchronized actions with $AC_DELAYFST

Value	Meaning
0	Delay stop area not active
1	Delay stop area active

Supplementary conditions

End of subprogram

DELAYFSTOF is activated implicitly at the end of the subprogram in which DELAYFSTON is called.

Nesting

If subprogram 1 calls subprogram 2 in a stop delay area, the whole of subprogram 2 is a stop delay area. In particular, DELAYFSTOF in subprogram 2 has no effect.

Example:

The following program section shows an example for nesting stop delay areas in two program levels:

```
Program code                    Comment
N10010  DELAYFSTON()            ; Blocks with N10xxx program level 1.
N10020  R1 = R1 + 1
```

```
Program code                Comment
N10030  G4 F1               ; Stop delay area starts.
...
N10040  subprogram2
...
...                         ; Interpretation of subprogram 2.
N20010  DELAYFSTON()        ; Ineffective, repeated start, 2nd level.
...
N20020  DELAYFSTOF()        ; Ineffective, end at another level.
N20030  RET
N10050  DELAYFSTOF()        ; Stop delay area end at the same level.
...
N10060  R2 = R2 + 2
N10070  G4 F1               ; Stop delay area ends. From now, stops act immediately.
```

11.8 Prevent program position for SERUPRO (IPTRLOCK, IPTRUNLOCK)

For some complicated mechanical situations on the machine it is necessary to the stop block search SERUPRO.

By using a programmable interruption pointer it is possible to intervene before an untraceable point with "Search at point of interruption".

It is also possible to define untraceable sections in part program sections that the NCK cannot yet re-enter. When the program is interrupted, the NCK notes the last block that was processed that can then be searched for via the HMI user interface.

Syntax

```
IPTRLOCK
IPTRUNLOCK
```

The commands are located in a part program line and allow a programmable interruption pointer

Meaning

IPTRLOCK:	Start of untraceable program section
IPTRUNLOCK:	End of untraceable program section

Both commands are only permitted in part programs, but **not** in synchronous actions.

Example

Nesting of untraceable program sections in two program levels with implicit "IPTRUNLOCK". Implicit "IPTRUNLOCK" in subprogram 1 ends the untraceable section.

```
Program code                    Comment
N10010  IPTRLOCK()
N10020  R1 = R1 + 1
N10030  G4 F1                   ; Hold block of the search-suppressed program sec-
                                tion starts.
...
N10040 subprogram2
...                             ; Interpretation of subprogram 2.
N20010  IPTRLOCK ()             ; Ineffective, repeated start.
...
N20020  IPTRUNLOCK ()           ; Ineffective, end at another level.
N20030  RET
...
N10060  R2 = R2 + 2
N10070  RET                     ; End of search-suppressed program section.
N100 G4 F2                      ; Main program is continued.
```

The interruption pointer then produces an interruption at 100 again.

Further information

Acquiring and finding untraceable sections

Non-searchable program sections are identified with language commands "IPTRLOCK" and "IPTRUNLOCK".

Command "IPTRLOCK" freezes the interruption pointer at a single block executable in the main run (SBL1). This block will be referred to as the hold block below. If the program is aborted after "IPTRLOCK", this hold block can be searched for from the HMI user interface.

Continuing from the current block

The interruption pointer is placed on the current block with "IPTRUNLOCK" as the interruption point for the following program section.

Once the search target is found a new search target can be repeated with the hold block.

An interrupt pointer edited by the user must be removed again via the HMI.

Rules for nesting:

The following points govern the interaction between language commands "IPTRLOCK" and "IPTRUNLOCK" with nesting and the subprogram end.

1. "IPTRLOCK" is activated implicitly at the end of the subprogram in which "IPTRUNLOCK" is called.
2. "IPTRLOCK" in a search-suppressed section has no effect.
3. If subprogram 1 calls subprogram 2 in an untraceable section, the whole of subprogram 2 remains untraceable. "IPTRUNLOCK" in particular has no effect in subprogram 2.

For more information, see
/FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1).

System variable

An untraceable section can be detected in the part program with "$P_IPTRLOCK".

Automatic interrupt pointer

The automatic interrupt pointer automatically defines a previously defined coupling type as untraceable. The machine data for

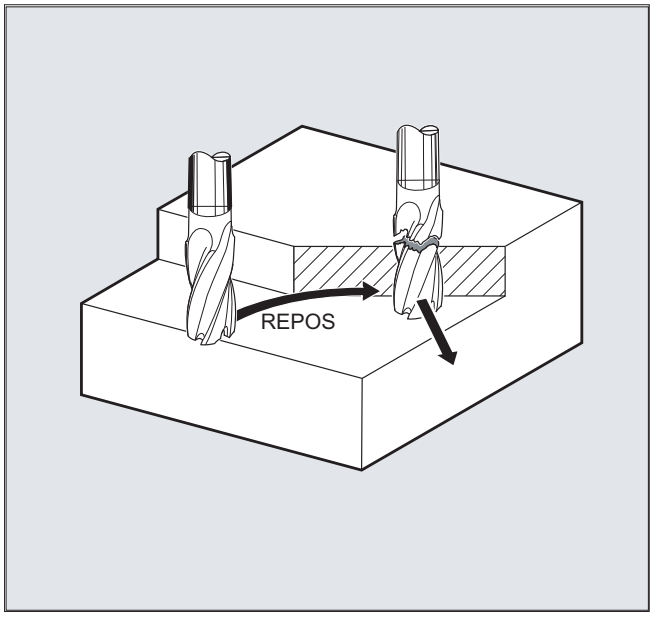
- Electronic gear for "EGON"
- Axial master value coupling for "LEADON"

are used to activate the automatic interrupt pointer. If the programmed interrupt pointer and interrupt pointer activated with automatic interrupt pointers overlap, the largest possible untraceable section will be generated.

11.9 Repositioning to the contour (REPOSA, REPOSL, REPOSQ, REPOSQA, REPOSH, REPOSHA, DISR, DISPR, RMIBL, RMBBL, RMEBL, RMNBL)

If you interrupt the program run and retract the tool during the machining operation – because, for example, the tool has broken or you wish to measure the workpiece – you can reposition at any selected point on the contour under control by the program.

The command `REPOS` acts in an ASUB as a subprogram return (e.g. M17). The following blocks are not executed. For information on interrupting program runs, see also "Interrupt routine (ASUB) (Page 118)."



Syntax

```
REPOSA RMIBL DISPR=…
REPOSA RMBBL
REPOSA RMEBL
REPOSA RMNBL
REPOSL RMIBL DISPR=…
REPOSL RMBBL
REPOSL RMEBL
REPOSL RMNBL
REPOSQ RMIBL DISPR=… DISR=…
REPOSQ RMBBL DISR=…
REPOSQ RMEBL DISR=…
REPOSQA DISR=…
REPOSH RMIBL DISPR=… DISR=…
REPOSH RMBBL DISR=…
REPOSH RMEBL DISR=…
REPOSHA DISR=…
```

Meaning

Selecting the approach path

`REPOSA`:	Repositioning to the contour with the geometry axes along a straight line. All other channel axes are also repositioned.
`REPOSL`:	Repositioning to the contour with the geometry axes along a straight line. Other axes have to be programmed explicitly.
`REPOSQ DISR=… :`	Repositioning to the contour with the geometry axes along a quadrant of radius DISR. Other axes have to be programmed explicitly.

`REPOSQA DISR=… :`	Repositioning to the contour with the geometry axes along a quadrant of radius DISR. All other channel axes are also repositioned.
`REPOSH DISR=… :`	Repositioning to the contour with the geometry axes along a semicircle of diameter DISR. Other axes have to be programmed explicitly.
`REPOSHA DISR=… :`	Repositioning to the contour with the geometry axes along a semi-circle of radius DISR. All other channel axes are also repositioned.

Selecting the repositioning point

`RMIBL`:	Approach interruption point
`RMIBL DISPR=…`:	Entry point at distance DISPR in mm/inch in front of interruption point
`RMBBL`:	Approach block start point
`RMEBL`:	Approach end of block
`RMEBL DISPR=… :`	Approach block end point at distance DISPR in front of end point
`RMNBL`:	Approach at nearest path point
`A0 B0 C0 :`	Axes in which approach is to be made

Note

Compatibility

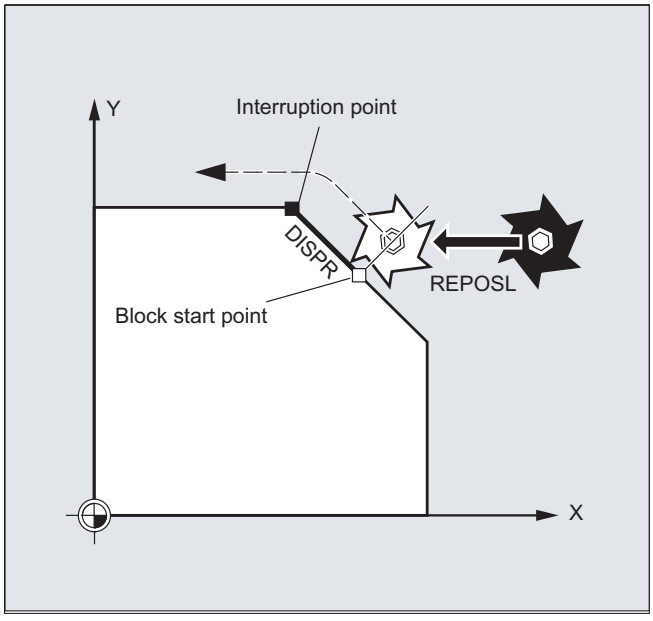
To remain compatible with older software versions, you can still program the `REPOS` approach mode via the modal commands `RMI`, `RMB`, `RME` and `RMN`. When used within an ASUB, this should be allocated the attribute `SAVE` in the `PROC` statement. Otherwise the modal `REPOS` approach mode used in the ASUB will take effect in subsequent `REPOS` processes, too, if it deviates from the preset `RMI`.

Repositioning to the contour along a straight line, REPOSA, REPOSL

The tool approaches the repositioning point along a straight line.

Example

```
REPOSL RMIBL DISPR=6 F400
```

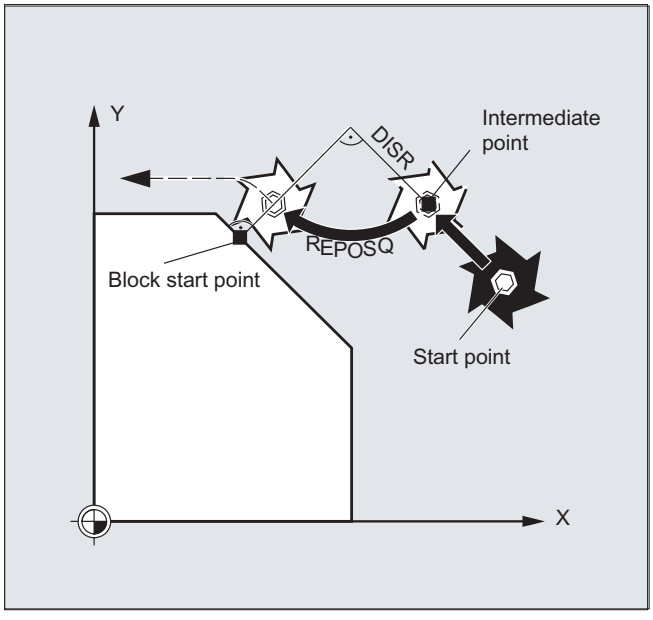


Repositioning to the contour along a quadrant, REPOSQ, REPOSQA

The tool approaches the repositioning point along a quadrant with a radius of `DISR=...`. The control automatically calculates the necessary intermediate point between the start and repositioning point.

Example

```
REPOSQ RMIBL DISR=10 F400
```

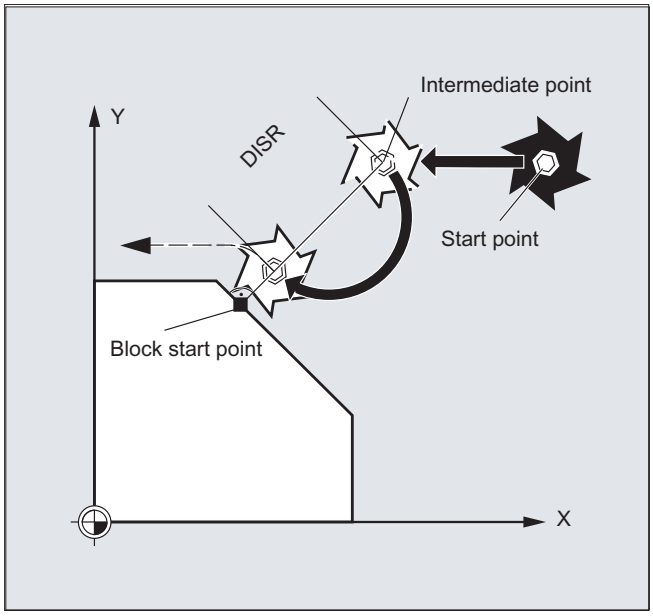


Repositioning to the contour along a semicircle, REPOSH, REPOSHA

The tool approaches the repositioning point along a semi-circle with a diameter of `DISR=…`. The control automatically calculates the necessary intermediate point between the start and repositioning point.

Example

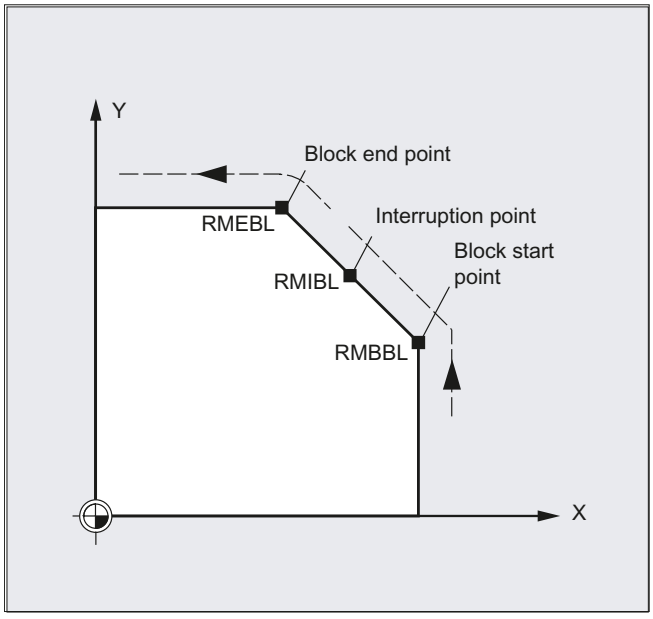
```
REPOSH RMIBL DISR=20 F400
```



Specifying the repositioning point (not for SERUPRO approaching with RMNBL)

With reference to the NC block in which the program run has been interrupted, it is possible to select one of three different repositioning points:

- RMIBL, interruption point
- RMBBL, block start point or last end point
- RMEBL, block end point



RMIBL DISPR=… or RME DISPR=… allows you to select a repositioning point which lies before the interruption point or the block end point.

DISPR=… allows you to describe the contour distance in mm/inch between the repositioning point and the interruption before the end point. Even for high values, this point cannot be further away than the block start point.

If no DISPR=… command is programmed, then DISPR=0 applies and with it the interruption point (with RMIBL) or the block end point (with RMEBL).

DISPR sign

The sign of DISPR is evaluated. In the case of a plus sign, the behavior is as previously.

In the case of a minus sign, approach is behind the interruption point or, with RMBBL, behind the block start point.

The distance between interruption point and approach point depends on the value of DISPR. Even for higher values, this point can lie in the block end point at the maximum.

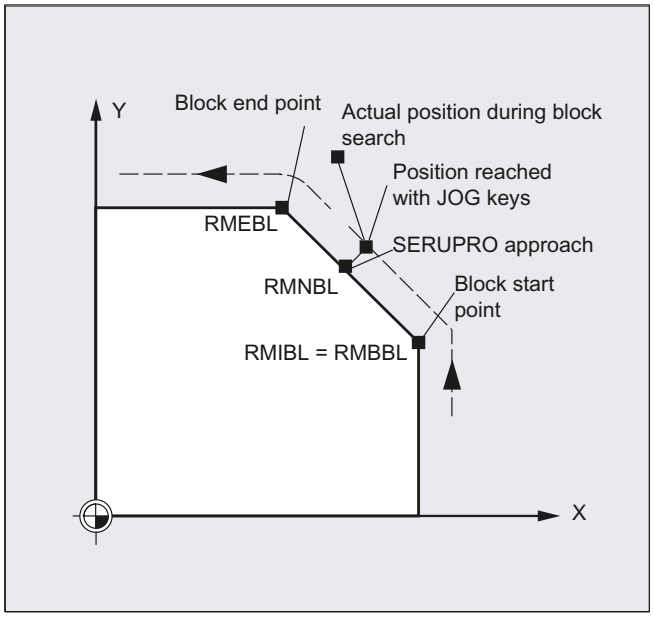
Application example:

A sensor will recognize the approach to a clamp. An ASUB is initiated to bypass the clamp.

Afterwards, a negative DISPR is repositioned on one point behind the clamp and the program is continued.

SERUPRO approach with RMNBL

If an abort is forced during machining at any position, the shortest path from the abort point is approached with SERUPRO approach and RMNBL so that afterward only the distance-to-go is processed. The user starts a SERUPRO process at the interruption block and uses the JOG keys to move in front of the problem component of the target block.



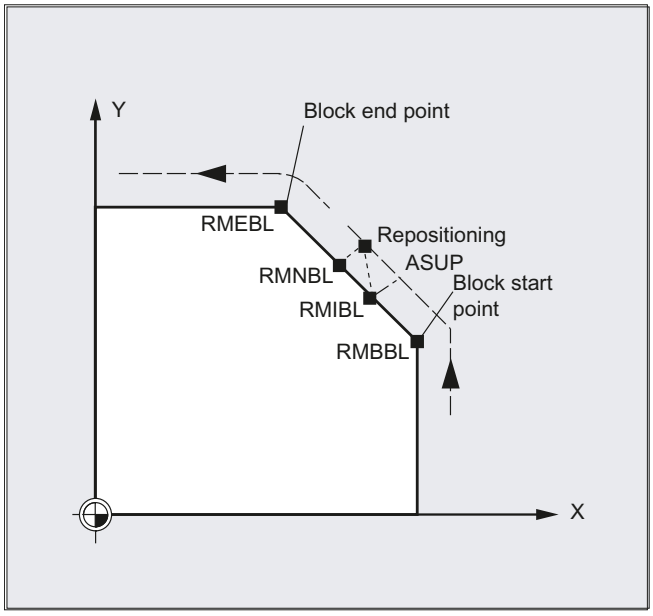
Note

SERUPRO

For SERUPRO, RMIBL and RMBBL are identical. RMNBL is not only limited to SERUPRO, but is generally valid.

Approach from the nearest path point RMNBL

When REPOSA is interpreted, the repositioning block with RMNBL is not started again in full after an interruption, but only the distance-to-go processed. The nearest path point of the interrupted block is approached.



Status for the valid REPOS mode

The valid REPOS mode of the interrupted block can be read with synchronized actions and variable $AC_ REPOS_PATH_MODE:

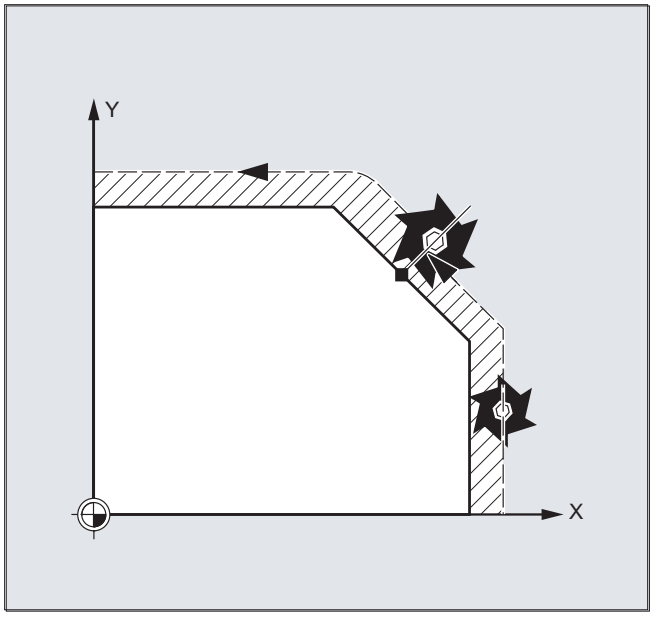
0 Approach not defined
1 RMBBL: Approach to beginning
2 RMIBL: Approach to point of interruption
3 RMEBL: Approach to end of block
4 RMNBL: Approach to next path point of the interrupted block

Approaching with a new tool

The following applies if you have stopped the program run due to tool breakage:

When the new D number is programmed, the machining program is continued with modified tool offset values at the repositioning point.

Where tool offset values have been modified, it may not be possible to reapproach the interruption point. In such cases, the point closest to the interruption point on the new contour is approached (possibly modified by DISPR).



Approach contour

The motion with which the tool is repositioned on the contour can be programmed. Enter zero for the addresses of the axes to be traversed.

The REPOSA, REPOSQA and REPOSHA commands automatically reposition all axes. Individual axis names need not be specified.

When the commands REPOSL, REPOSQ and REPOSH are programmed, all geometry axes are traversed automatically, i.e. they do not have to be specified in the command. All other axes must be specified in the commands.

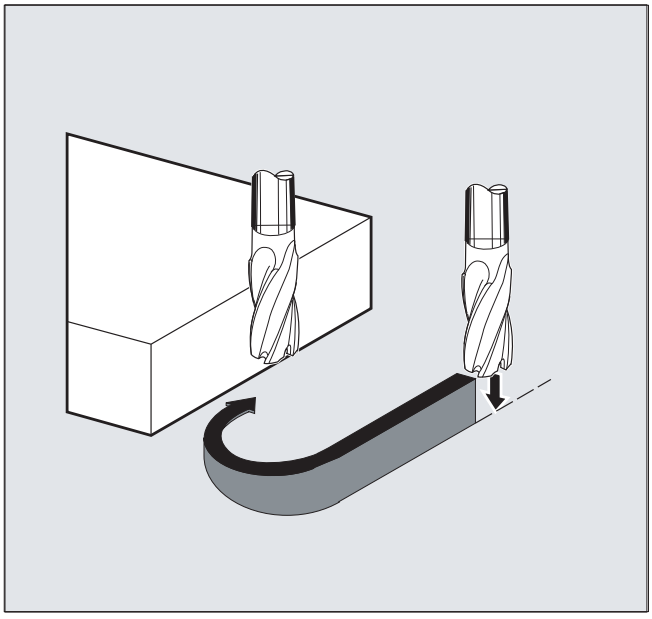
The following applies to the REPOSH and REPOSQ circular motions:

The circle is traversed in the specified working planes G17 to G19.

If you specify the third geometry axis (infeed direction) in the approach block, the repositioning point is approached along a helix in case the tool position and programmed position in the infeed direction do not coincide.

In the following cases, there is an automatic switchover to linear approach REPOSL:

- You have not specified a value for DISR.
- No defined approach direction is available (program interruption in a block without travel information).
- With an approach direction that is perpendicular to the current working plane.



11.10 Influencing the motion control

11.10.1 Percentage jerk correction (JERKLIM)

Using the NC command "JERKLIM", the maximum jerk of an axis for path motion - set using machine data - can be reduced or increased in critical program sections.

Requirement

The acceleration mode SOFT must be active.

Effectiveness

The function is effective:

- In the AUTOMATIC operating modes.
- Only on path axes.

Syntax

```
JERKLIM[<axis>]=<value>
```

Meaning

JERKLIM:	Command for jerk correction	
<axis>:	Machine axis whose jerk limit value is to be adapted.	
<value>:	Percentage correction value, referred to the configured maximum axis jerk for path motion (MD32431 $MA_MAX_AX_JERK).	
	Range of values:	1 ... 200
	Value 100 does not influence the jerk.	

Note

The behavior of JERKLIM at the end of the part program and channel reset is configured with bit 0 in machine data MD32320 $MA_DYN_LIMIT_RESET_MASK:

- Bit 0 = 0:
  The programmed value for JERKLIM is reset to 100% with channel reset/M30.
- Bit 0 = 1:
  The programmed value for JERKLIM is retained beyond the channel reset/M30.

Example

Program code	Comment
...	
N60 JERKLIM[X]=75	; The axis slide in the X direction should only be accelerated/decelerated with a maximum of 75% of the jerk permissible for the axis.
...	

11.10.2 Percentage velocity correction (VELOLIM)

The maximum possible velocity of an axis or the maximum possible gear-stage-dependent speed of a spindle set via machine data can be reduce with the VELOLIM command in the part program or synchronized action.

Effectiveness

The function is effective:

- In the AUTOMATIC operating modes.
- On path and positioning axes.
- On spindles in spindle/axis operations

Syntax

```
VELOLIM[<axis/spindle>]=<value>
```

Meaning

VELOLIM:	Command for velocity correction
<axis/spindle>:	Axis or spindle whose velocity or speed limit value should be adapted. **VELOLIM for spindles** Using machine data (MD30455 $MA_MISC_FUNCTION_MASK, bit 6), when programming in the part program, it can be set as to whether "VELOLIM" is effective independent of whether used as spindle or axis (bit 6 = 1) - or is able to be programmed separately for each operating mode (bit 6 = 0). If they are to be separately effective, then the selection is made using the identifier when programming: • Spindle identifier S<n> for spindle operating modes • Axis identifier, e.g. "C", for axis operation
<value>:	Percentage correction value The correction value refers to: • For axes/spindles in axis operation (MD30455 bit 6 == 0): To the configured maximum axis velocity (MD32000 $MA_MAX_AX_VELO). • For spindles in spindle or axis operation (MD30455 bit 6 == 1): To the maximum speed of the active gear unit stage (MD35130 $MA_GEAR_STEP_MAX_VELO_LIMIT[<n>])
	Range of values: 1 ... 100
	The value 100 does not influence the velocity or speed.

Note

Behavior at the end of the part program and for a channel reset

The behavior of "VELOLIM" at the end of the part program and channel reset can be set via the machine data: MD32320 $MA_DYN_LIMIT_RESET_MASK, bit 0

Detection of an active speed limitation in spindle operation

A speed limitation via "VELOLIM" (less than 100%) can be detected in spindle operation via the following system variables:

- $AC_SMAXVELO (maximum possible spindle speed)
- $AC_SMAXVELO_INFO (identifier for the speed-limiting cause)

Examples

Example 1: Velocity limitation, machine axis

Program code	Comment
...	

Program code	Comment
N70 VELOLIM[X]=80	; The axis slide in the X direction should only be traversed with a maximum of 80% of the velocity permissible for the axis.
...	

Example 2: Speed limitation, spindle

Program code	Comment
N05 VELOLIM[S1]=90	; Limiting the maximum speed of spindle 1 to 90% of 1000 rpm.
...	
N50 VELOLIM[C]=45	; Limiting the speed to 45% of 1000 rpm, C is the axis identifier of S1.
...	

Machine data settings for spindle 1 (AX5)

- Maximum speed of gear stage 1 = 1000 rpm:
  MD35130 $MA_GEAR_STEP_MAX_VELO_LIMIT[1, AX5] = 1000
- Programming "VELOLIM" acts together for spindle and axis operation independent of the programmed identifier:
  MD30455 $MA_MISC_FUNCTION_MASK[AX5], bit 6 = 1

11.10.3 Program example for JERKLIM and VELOLIM

The following program presents an application example for the percentage jerk and velocity limit:

Program code	Comments
N1000 G0 X0 Y0 F10000 SOFT G64	
N1100 G1 X20 RNDM=5 ACC[X]=20	
ACC[Y]=30	
N1200 G1 Y20 VELOLIM[X]=5	; The axis slide in the X direction should only be traversed with max. 5% of the velocity permissible for the axis.
JERKLIM[Y]=200	; The axis slide in the Y direction can be accelerated/decelerated with max. 200% of the jerk permissible for the axis.
N1300 G1 X0 JERKLIM[X]=2	; The axis slide in the X direction should only be accelerated/decelerated with max. 2% of the jerk permissible for the axis.
N1400 G1 Y0	
M30	

11.11 Programmable contour/orientation tolerance (CTOL, OTOL, ATOL)

The "CTOL", "OTOL" and "ATOL" commands can be used to adapt the machining tolerances defined for the compressor functions (COMPON, COMPCURV, COMPCAD), the smoothing types G642, G643, G645, OST, and the orientation ORISON using machine and setting data in the NC program.

The programmed values are valid until they are reprogrammed or deleted by means of the assignment of a negative value. They are also deleted at the end of a program, in the event of a channel reset, a mode group reset, an NCK reset (warm restart), and POWER ON (cold restart). On deletion of these values, the values from the machine and setting data are restored.

Syntax

```
CTOL=<value>
OTOL=<value>
ATOL[<axis>]=<value>
```

Meaning

CTOL:	Command for programming the **contour tolerance** "CTOL" is valid for: • All compressor functions • All rounding types except G641 and G644		
	<value>:	The value for the contour tolerance is specified as a length.	
		Type:	REAL
		Unit:	inch/mm (dependent on the current dimensions setting)
OTOL:	Command for programming the **orientation tolerance** "OTOL" is valid for: • All compressor functions • ORISON orientation smoothing • All rounding types except G641, G644, OSD		
	<value>:	The value for the orientation tolerance is specified as an angle.	
		Type:	REAL
		Unit:	degrees

ATOL:	Command for programming an **axis-specific tolerance** "ATOL" is valid for: • All compressor functions • ORISON orientation smoothing • All rounding types except G641, G644, OSD			
	<axis>:	Name of the axis for which an axis tolerance is to be programmed		
	<value>:	The value for the axis tolerance will be specified as a length or an angle dependent on the axis type (linear or rotary axis).		
		Type:	REAL	
		Unit:	For linear axes:	inch/mm (dependent on the current dimensions setting)
			For rotary axes:	degrees

Note

"CTOL" and "OTOL" have priority over "ATOL".

Supplementary conditions

Scaling frames

Scaling frames affect programmed tolerances in the same way as axis positions; in other words, the relative tolerance remains the same.

Example

Program code	Comment
COMPCAD G645 G1 F10000	; Activate COMPCAD compressor function.
X... Y... Z...	; The machine and setting data is applied here.
X... Y... Z...	
X... Y... Z...	
CTOL=0.02	; A contour tolerance of 0.02 mm is applied starting from here.
X... Y... Z...	
X... Y... Z...	
X... Y... Z...	
ASCALE X0.25 Y0.25 Z0.25	; A contour tolerance of 0.005 mm is applied starting from here.
X... Y... Z...	
X... Y... Z...	
X... Y... Z...	
CTOL=-1	; The machine and setting data is applied again starting from here.
X... Y... Z...	
X... Y... Z...	
X... Y... Z...	

Further information

Read tolerance values

For more advanced applications or for diagnostics, the currently valid tolerances for the compressor functions (COMPON, COMPCURV, COMPCAD), the smoothing types G642, G643, G645, OST, and the orientation smoothing ORISON can be read via system variables irrespective of how they might have come about.

- In synchronized actions or with preprocessing stop in the part program via system variables:

$AC_CTOL	Contour tolerance effective when the current main run record was preprocessed. If no contour tolerance is effective, `$AC_CTOL` will return the root from the sum of the squares of the tolerances of the geometry axes.
$AC_OTOL	Orientation tolerance effective when the current main run record was preprocessed. If no orientation tolerance is effective, `$AC_OTOL` will return the root from the sum of the squares of the tolerances of the orientation axes during active orientation transformation. Otherwise, it will return the value "-1".
$AA_ATOL[<axis>]	Axis tolerance effective when the current main run record was preprocessed. If no contour tolerance is active, `$AA_ATOL[<geometry axis>]` will return the contour tolerance divided by the root of the number of geometry axes. If an orientation tolerance and an orientation transformation are active `$AA_ATOL[<orientation axis>]` will return the orientation tolerance divided by the root of the number of orientation axes.

Note

If no tolerance values have been programmed, the $A variables will not be differentiated sufficiently to distinguish potential differences in the tolerances of the individual functions, since they can only declare one value.

Circumstances like this can occur if the machine data and the setting data set different tolerances for compressor functions, smoothing and orientation smoothing. The variables then return the greatest value prevailing with regard to the currently active functions.

if, for example, a compressor function is active with an orientation tolerance of 0.1° and ORISON orientation smoothing with 1°, the $AC_OTOL variable will return the value "1". If orientation smoothing is deactivated, only the value "0.1" will remain to be read.

- Without preprocessing stop in the part program via system variables:

$P_CTOL	Programmed contour tolerance
$P_OTOL	Programmed orientation tolerance
$PA_ATOL	Programmed axis tolerance

Note

If no tolerance values have been programmed, the $P variables will return the value "-1".

11.12 Tolerance for G0 motion (STOLF)

G0 tolerance factor

G0 motion (rapid traverse, infeed motion), contrary to workpiece machining, can be implemented with a higher tolerance. This has the advantage that the execution times for G0 motion are shortened.

For G0 motion, the tolerances are set by configuring the G0 tolerance factor (MD20560 $MC_G0_TOLERANCE_FACTOR).

The G0 tolerance factor is only effective, if:

- One of the following functions is active:
  - Compressor function: COMPON, COMPCURV and COMPCAD
  - Smoothing functions: G642 and G645
  - Orientation smoothing: OST
  - Orientation smoothing: ORISON
  - Smoothing for path-relevant orientation: ORIPATH
- Several (≥ 2) consecutive G0 blocks.
  For a single G0 block, the G0 tolerance factor is not effective, as **at the transition** from a non G0 motion to a G0 motion (and vice versa), the "**lower tolerance**" always applies (workpiece machining tolerance)!

Adapting the G0 tolerance factor from the part program

The configured G0 tolerance factor (MD20560) can be temporarily overwritten by programming STOLF in the part program. In this case, the value in MD20560 is not changed. After a reset or end of part program, the configured tolerance factor is effective again.

Syntax

```
STOLF=<tolerance factor>
```

Meaning

STOLF:	Command to program the G0 tolerance factor
<tolerance factor>:	G0 tolerance factor The factor can either be greater than 1 or less than 1. However, normally higher tolerances can be set for G0 motion. For STOLF=1.0 (this corresponds to the configured standard value), for G0 motion, the same tolerances are effective as for non-G0 motion.

System variables

The G0 tolerance factor, effective in the part program or in the actual IPO block, can be read using system variables.

- In synchronized actions or with preprocessing stop in the part program via system variable:

  $AC_STOLF Active G0 tolerance factor
  G0 tolerance factor, which was effective when processing the actual main run block.

- Without preprocessing stop in the part program via system variable:

  $P_STOLF Programmed G0 tolerance factor

If no value with STOLF is programmed in the active part program, then these two system variables supply the value set using MD20560 $MC_G0_TOLERANCE_FACTOR.

If no rapid traverse (G0) is active in a block, then these system variables always supply a value of 1.

Example

```
Program code                    Comment
COMPCAD G645 G1 F10000          ; Compressor function COMPCAD
X... Y... Z...                  ; The machine and setting data apply here.
X... Y... Z...
X... Y... Z...
G0  X... Y... Z...
G0 X... Y... Z...               ; Machine data $MC_G0_TOLERANCE_FACTOR (e.g. =3), is
                                effective here, i.e. a smoothing tolerance of
                                $MC_G0_TOLERANCE_FACTOR*$MA_COMPRESS_POS_TOL.
CTOL=0.02
STOLF=4
G1 X... Y... Z...               ; A contour tolerance of 0.02mm is applied starting
                                from here.
X... Y... Z...
X... Y... Z...
G0 X... Y... Z...
```

Program code	Comment
X... Y... Z...	; From here, a G0 tolerance factor of 4 applies, i.e. a contour tolerance of 0.08 mm.

11.13 Block change behavior with active coupling (CPBC)

The CPBC command specifies the block change criterion that must be satisfied so that a block change can be executed in the part program with active coupling.

Syntax

```
CPBC[<following axis>] = <criterion>
```

Meaning

CPBC:	Block change criterion with active coupling	
<following axis>:	Axis identifier of the following axis	
<criterion>:	Block change criterion	
	Type:	STRING
	Value	**Meaning:** Block change is performed
	"NOC"	Irrespective of the coupling status
	"IPOSTOP"	For setpoint synchronism
	"COARSE"	For actual value synchronism "coarse"
	"FINE"	For actual value synchronism "fine"

Example

```
; Block change takes place with:
;  - Coupling to following axis X2 == active
;  - Setpoint synchronism == active
CPBC[X2]="IPOSTOP"
```

Axis couplings

12

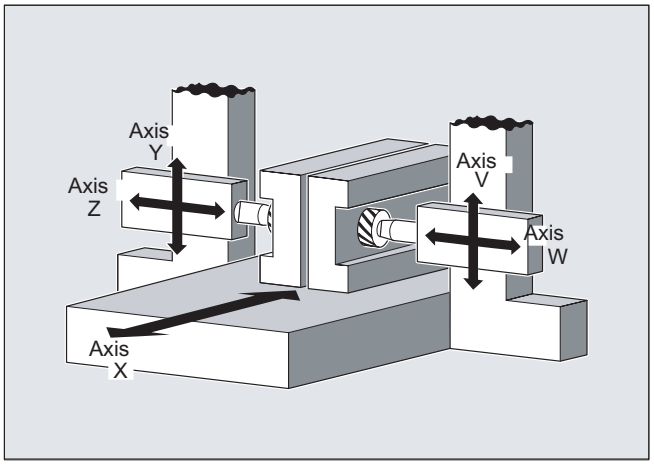
12.1 Coupled motion (TRAILON, TRAILOF)

When a defined leading axis is moved, the coupled motion axes (= following axes) assigned to it traverse through the distances described by the leading axis, allowing for a coupling factor.

Together, the leading axis and following axis represent coupled axes.

Applications

- Traversal of an axis by means of a simulated axis. The leading axis is a simulated axis and the coupled axis a real axis. In this way, the real axis can be traversed as a function of the coupling factor.
- Two-sided machining with two coupled motion groups:
  1. leading axis Y, coupled motion axis V
  2. leading axis Z, coupled motion axis W



Syntax

```
TRAILON(<following axis>,<leading axis>,<coupling factor>)
TRAILOF(<following axis>,<leading axis>,<leading axis 2>)
TRAILOF(<following axis>)
```

Meaning

TRAILON:	Command for activating and defining a coupled axis grouping	
	Effective:	Modal
<following axis>:	Parameter 1: Axis name of trailing axis **Note:** A coupled-motion axis can also act as the leading axis for other coupled-motion axes. In this way, it is possible to create a range of different coupled axis groupings.	

`<leasing axis>`:	Parameter 2: Axis name of trailing axis
`<coupling factor>`:	Parameter 3: Coupling factor The coupling factor specifies the desired relationship between the paths of the coupled-motion axis and the leading axis: <coupling factor> = path of coupled-motion axis/path of leading axis
	Type: REAL
	Default: 1
	The input of a negative value causes the master and coupled axes to traverse in opposition. If a coupling factor is not programmed, then coupling factor 1 automatically applies.
`TRAILOF`:	Command for deactivating a coupled axis grouping
	Effective: Modal

	`TRAILOF` with 2 parameters deactivates only the coupling to the specified leading axis:
	`TRAILOF(<following axis>,<leading axis>)`
	If a coupled-motion axis has two leading axes, `TRAILOF` can be called with three parameters to deactivate both couplings.
	`TRAILOF(<following axis>,<leading axis>,<leading axis 2>)`
	Programming `TRAILOF` without specifying a leading axis produces the same result:
	`TRAILOF(<following axis>)`

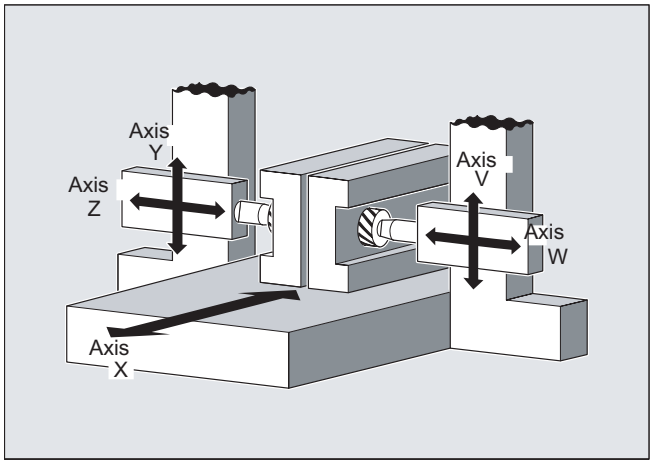
Note

Coupled axis motion is always executed in the base coordinate system (BCS).

The number of coupled axis groupings which may be simultaneously activated is limited only by the maximum possible number of combinations of axes on the machine.

Example

The workpiece is to be machined on two sides with the axis configuration shown in the diagram. To do this, you create two combinations of coupled axes.



Program code	Comment
…	
N100 TRAILON(V,Y)	; Activation of 1st coupled axis group.
N110 TRAILON(W,Z,-1)	; Activation of 2nd coupled axis grouping, Negative coupling factor: Coupled-motion axis traverses in the opposite direction from leading axis.
N120 G0 Z10	; Infeed of Z and W axes in opposite axial directions.
N130 G0 Y20	; Infeed of Y and V axes in same axis direction.
…	
N200 G1 Y22 V25 F200	; Overlaying of a dependent and independent movement of coupled motion axis V.
…	
TRAILOF(V,Y)	; Deactivation of 1st coupled axis grouping.
TRAILOF(W,Z)	; Deactivation of 2nd coupled axis grouping.

Further information

Axis types

A coupled axis grouping can consist of any desired combinations of linear and rotary axes. A simulated axis can also be defined as a leading axis.

Coupled-motion axes

Up to two leading axes can be assigned simultaneously to a trailing axis. The assignment is made in different combinations of coupled axes.

A coupled-motion axis can be programmed with the full range of available motion commands (G0, G1, G2, G3, etc.). The coupled axis not only traverses the independently defined paths, but also those derived from its leading axes on the basis of coupling factors.

Dynamics limit

The dynamics limit is dependent on the type of activation of the coupled axis grouping:

- Activation in part program
  If activation is performed in the part program and all leading axes are active as program axes in the activated channel, the dynamic response of all coupled-motion axes is taken into account during traversing of the leading axis to avoid overloading the coupled-motion axes.
  If activation is performed in the part program with leading axes that are not active as program axes in the activating channel ($AA_TYP ≠ 1), then the dynamic response of the coupled-motion axes is not taken into account during traversing of the leading axis. This can cause the overloading of coupled-motion axes with a dynamic response which is less than that required for the coupling.
- Activation in synchronized action
  If activation is performed in a synchronized action, the dynamic response of the coupled-motion axes is not taken into account during traversing of the leading axis. This can cause the overloading of coupled-motion axes with a dynamic response which is less than that required for the coupling.

⚠ CAUTION

Axis overload

If a coupled axis grouping is activated:

- In synchronized actions
- In the part program with leading axes that are not program axes in the channel of the coupled-motion axes

It is the specific responsibility of the user / machine manufacturer to take suitable action to ensure that the traversing of the leading axis will not cause the overloading of the coupled-motion axes.

Coupling status

The coupling status of an axis can be checked in the part program with the system variable:

$AA_COUP_ACT[<axis>]

Value	Meaning
0	No coupling active
8	Coupled motion active

Display of distance-to-go of the coupled-motion axis for modulo rotary axes

If the leading and coupled-motion axes are modulo rotary axes, traversing movements in the leading axis from n * 360° with n = 1, 2, 3 ... , add up in the distance-to-go display of the coupled-motion axis until the coupling is switched off.

Example: Program section with TRAILON and leading axis B and following axis C

Program code	Comment
`TRAILON(C,B,1)`	`; Activate coupling`
`G0 B0`	`; Starting position`
	`; Distance-to-go display at block start:`

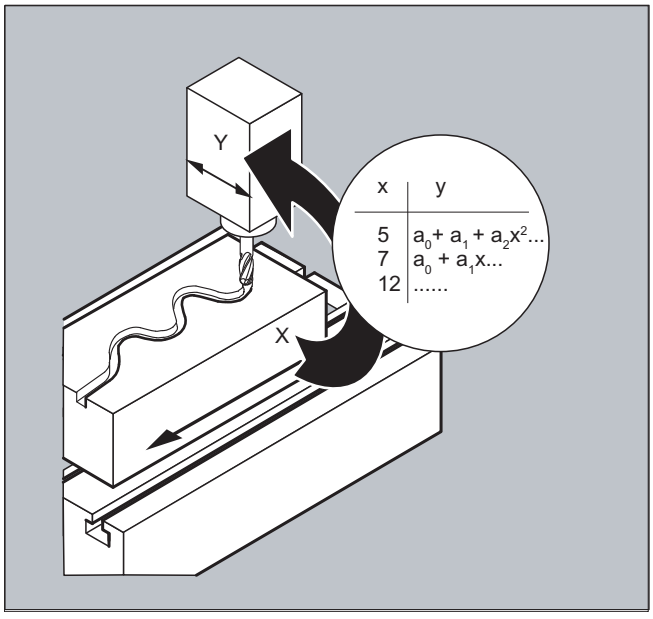
Program code	Comment
G91 B360	; B=360, C=360
G91 B720	; B=720, C=1080
G91 B360	; B=360, C=1440

12.2 Curve tables (CTAB)

Curve tables can be used to program position and velocity relationships between two axes (leading and following axis). Curve tables are defined in the part program.

Application

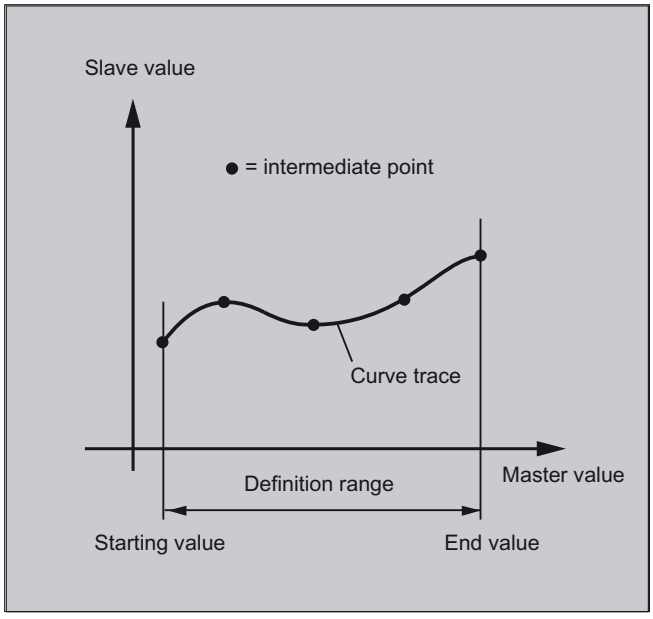
Curve tables replace mechanical cams. The curve table forms the basis for the axial master value coupling by creating the functional relationship between the leading and the following value: With appropriate programming, the control calculates a polynomial that corresponds to the cam from the relative positions of the leading and following axes.



12.2.1 Define curve tables (CTABDEF, CATBEND)

A curve table represents a part program or a section of a part program enclosed by CTABDEF at the start and CTABEND at the end.

Within this part program section, unique following axis positions are assigned to individual positions of the leading axis using motion operations; these following axis positions are used as intermediate points when calculating the curve definition in the form of a polynomial up to the 5th order.



Requirement

The MD must be configured accordingly to ensure that sufficient memory space is reserved for the definition of curve tables (→ machine manufacturer).

Syntax

```
CTABDEF(<following axis>,<leading axis>,<n>,<periodicity>[,<memory
location>])
...
CTABEND
```

Meaning

`CTABDEF( ):`	Start of curve table definition	
`CTABEND`:	End of curve table definition	
`<following axis>`:	Axis whose motion is to be calculated using the curve table	
`<leasing axis>`:	Axis providing the master values for the calculation of the following axis motion	
`<n>`:	Number (**ID**) of curve table The number of a curve table is unique and independent of the memory location. It is not possible for there to be tables with the same number in the static and dynamic NC memory.	
`<periodicity>`:	Table periodicity	
	`0`	Table is non-periodic (table is processed only once, even for rotary axes)
	`1`	Table is periodic with regard to the leading axis
	`2`	Table is periodic with regard to leading axis and following axis

<memory location>:	Specification of memory location (optional)	
	"SRAM"	The curve table is created in the **static** NC memory.
	"DRAM"	The curve table is created in the **dynamic** NC memory.
	Note: If a value is not programmed for this parameter, the default memory location set with MD20905 $MC_CTAB_DEFAULT_MEMORY_TYPE is used.	

Note

Overwrite

A curve table is overwritten as soon as its number (<n>) is used in another table definition. (exception: the curve table is either active in an axis coupling or locked with CTABLOCK). **No warning is output when curve tables are overwritten.**

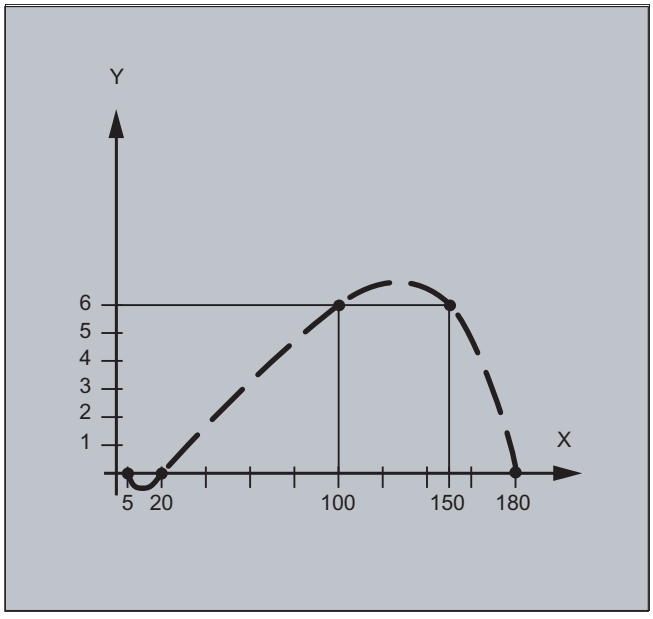
Examples

Example 1: Program section as curve table definition

A program section is to be used unchanged for defining a curve table. The STOPRE command for preprocessing stop can remain and is reactivated immediately as soon as the program section is no longer being used for table definition and CTABDEF and CTABEND have been removed.

```
Program code                    Comment
…
CTABDEF(Y,X,1,1)                ; Definition of a curve table.
…
IF NOT ($P_CTABDEF)
STOPRE
ENDIF
…
CTABEND
```

Example 2: Definition of a non-periodic curve table



Program code	Comment
N100 CTABDEF(Y,X,3,0)	; Beginning of the definition of a ;non-periodic curve table with number 3.
N110 X0 Y0	; 1st motion operation, defines the starting values and 1st intermediate point:
	Master value: 0, Following value: 0
N120 X20 Y0	; 2nd interpolation point:
	Master value: 0…20, Following value: starting value…0
N130 X100 Y6	; 3rd interpolation point:
	Master value: 20…100, Following value: 0…6
N140 X150 Y6	; 4th interpolation point:
	Master value: 100…150, Following value: 6…6
N150 X180 Y0	; 5th interpolation point:
	Master value: 150…180, Following value: 6…0
N200 CTABEND	; End of the definition. The curve table is generated in its internal representation as a polynomial of up to the 5th order. The calculation of the curve definition with the specified intermediate points is dependent on the modally selected interpolation type (circular, linear, spline interpolation). The part program state before starting the definition is restored.

Example 3: Definition of a periodic curve table

Definition of a periodic curve table with number 2, master value range 0 to 360, following axis motion from 0 to 45 and back to 0:

Program code	Comment
N10 DEF REAL DEPPOS	

```
Program code                                                    Comment
N20 DEF REAL GRADIENT
N30 CTABDEF(Y,X,2,1)                                            ; Start of definition.
N40 G1 X=0 Y=0
N50 POLY
N60 PO[X]=(45.0)
N70 PO[X]=(90.0) PO[Y]=(45.0,135.0,-90)
N80 PO[X]=(270.0)
N90 PO[X]=(315.0) PO[Y]=(0.0,-135.0,90)
N100 PO[X]=(360.0)
N110 CTABEND                                                    ; End of the definition.
;Test of the curve by coupling Y to X:
N120 G1 F1000 X0
N130 LEADON(Y,X,2)
N140 X360
N150 X0
N160 LEADOF(Y,X)

N170 DEPPOS=CTAB(75.0,2,GRADIENT)                               ; Read the table function for mas-
                                                                ter value 75.0.
N180 G0 X75 Y=DEPPOS                                            ; Positioning leading and follow-
                                                                ing axes.
;After activating the coupling, no synchronization of the following axis is required.
N190 LEADON(Y,X,2)
N200 G1 X110 F1000
N210 LEADOF(Y,X)
N220 M30
```

Further Information

Starting and end value of the curve table

The starting value for the beginning of the definition range of the curve table are the first associated axis positions specified (the first traverse statement) within the curve table definition. The end value of the definition range of the curve table is determined in accordance with the last traverse command.

Available language scope

Within the definition of the curve table, you have use of the entire NC language.

Note

The following entries are not permitted in curve table definitions:

- Preprocessing stop
- Jumps in the leading axis movement (e.g. on changing transformations)
- Traverse statement for the following axis only
- Reversal of the leading axis, i.e. position of the leading axis must always be unique
- CTABDEF and CTABEND statement on various program levels.

Effectiveness of modal operations

All modal statements that are made within the curve table definition are invalid when the table definition is completed. The part program in which the table definition is made is therefore before and after the table definition in the same state.

Assignments to R-parameters

Assignments to R-parameters in the table definition are reset after CTABEND.

Example:

Program code	**Comment**
...	
R10=5 R11=20	;R10=5
...	
CTABDEF	
G1 X=10 Y=20 F1000	
R10=R11+5	;R10=25
X=R10	
CTABEND	
...	;R10=5

Activating ASPLINE, BSPLINE, CSPLINE

If an ASPLINE, BSPLINE or CSPLINE is activated within a curve definition table CTABDEF ... CTABEND, at least a start point should be programmed before this spline activation. Immediate activation after CTABDEF should be avoided, otherwise the spline will depend on the current axis position before the curve table definition.

Example:

```
Program code
...
CTABDEF(Y,X,1,0)
X0 Y0
ASPLINE
X=5 Y=10
```

```
Program code
X10 Y40
...
CTABEND
```

Repeated use of curve tables

The functional relationship between the leading axis and the following axis calculated using the curve table will be retained under the selected table number after the end of the part program and POWER OFF if the table has been saved to the static NC memory (SRAM).

A table created in the dynamic memory (DRAM) will be deleted on POWER ON and may have to be regenerated.

Once created, the curve table can be applied to any axis combinations of leading and following axis and is independent of the axes used to create the curve table.

Overwriting curve tables

A curve table is overwritten as soon as its number is used in another table definition.

Exception: A curve table is either active in an axis coupling or locked with CTABLOCK.

Note

No warning is output when curve tables are overwritten.

Curve table definition active?

The `$P_CTABDEF` system variable can be used at any time in the part program to check whether a curve table definition is active.

Revoking the curve table definition

Once the operations relating to the curve table definition have been excluded, the part program section can be used as a real part program again.

Loading curve tables using "Execution from external source"

If curve tables are executed from an external source, the selection of the size of the reload buffer (DRAM) in MD18360 $MN_MM_EXT_PROG_BUFFER_SIZE has to support the simultaneous storage of the entire curve table definition in the reload buffer. If it is not, part program processing will be canceled with an alarm.

Jumps in the following axis

Depending on the setting in machine data
MD20900 $MC_CTAB_ENABLE_NO_LEADMOTION
, jumps in the following axis may be tolerated if a movement is missing in the leading axis.

12.2.2 Check for presence of curve table (CTABEXISTS)

The CTABEXISTS command can be used to check if a specific curve table number is present in the NC memory.

Syntax

```
CTABEXISTS(<n>)
```

Meaning

CTABEXISTS:	Checks for the presence of curve table number <n> in the static or dynamic NC memory.	
	0	Table does not exist
	1	Table exists
<n>:	Number (**ID**) of curve table	

12.2.3 Delete curve tables (CTABDEL)

CTABDEL can be used to delete curve tables.

Note

Curve tables that are active in an axis coupling cannot be deleted.

Syntax

```
CTABDEL(<n>)
CTABDEL(<n>,<m>)
CTABDEL(<n>,<m>,<memory location>)
CTABDEL()
CTABDEL(,,<memory location>)
```

Meaning

CTABDEL:	Command for deleting curve tables	
<n>:	Number (**ID**) of the curve table to be deleted When a curve table range CTABDEL(<n>,<m>) is deleted, <n> is used to specify the number of the first curve table in the range.	
<m>:	When a curve table range CTABDEL(<n>,<m>) is deleted, <m> is used to specify the number of the last curve table in the range. <m> has to be greater than <n>.	
<memory location>:	Specification of memory location (optional) In the case of deletion **without** a memory location being specified, the specified curve tables are deleted in the static and the dynamic NC memory. In the case of deletion **with** a memory location being specified, of the specified curve tables, only those located in the specified memory location are deleted. The rest are retained.	
	"SRAM"	Deletion in the **static** NC memory
	"DRAM"	Deletion in the **dynamic** NC memory

If CTABDEL is programmed without specification of the curve table to be deleted, then **all** curve tables or all curve tables in the specified memory will be deleted:

CTABDEL():	Deletes all curve tables in the static and the dynamic NC memory
CTABDEL(,,"SRAM"):	Deletes all curve tables in the static NC memory
CTABDEL(,,"DRAM"):	Deletes all curve tables in the dynamic NC memory

Note

If, in the case of multiple deletion with CTABDEL(<n>,<m>) or CTABDEL(), at least one of the of the curve tables to be deleted is active in a coupling, the delete command will not be executed; in other words, **none** of the addressed curve tables will be deleted.

12.2.4 Locking curve tables to prevent deletion and overwriting (CTABLOCK, CTABUNLOCK)

Locks can be set to protect curve tables against unintentional deletion and overwriting. Once a lock has been set, it can be revoked at any time.

Syntax

Lock:

```
CTABLOCK(<n>)
CTABLOCK(<n>,<m>)
CTABLOCK(<n>,<m>,<memory location>)
CTABLOCK()
CTABLOCK(,,<memory location>)
```

Unlock:

```
CTABUNLOCK(<n>)
CTABUNLOCK(<n>,<m>)
CTABUNLOCK(<n>,<m>,<memory location>)
CTABUNLOCK()
CTABUNLOCK(,,<memory location>)
```

Meaning

CTABLOCK:	Command for **setting** a lock to prevent deletion/overwriting
CTABUNLOCK:	Command for **revoking** a lock to prevent deletion/overwriting CTABUNLOCK unlocks the curve tables locked with CTABLOCK. Curve tables which are involved in an active coupling remain locked and cannot be deleted. The lock with CTABLOCK is unlocked as soon as the lock applied due to the active coupling is unlocked when the coupling is deactivated. This table can therefore be deleted. It is not necessary to call CTABUNLOCK again.

<n>:	Number (**ID**) of the curve table to be locked/unlocked When a curve table range CTABLOCK(<n>,<m>)/CTABUNLOCK(<n>,<m>) is locked/unlocked, <n> is used to specify the number of the first curve table in the range.	
<m>:	When a curve table range CTABLOCK(<n>,<m>)/CTABUNLOCK(<n>,<m>) is locked/unlocked, <m> is used to specify the number of the last curve table in the range. <m> has to be greater than <n>.	
<memory location>:	Specification of memory location (optional) In the case of locking/unlocking **without** a memory location being specified, the specified curve tables are locked/unlocked in the static and the dynamic NC memory. In the case of locking/unlocking **with** a memory location being specified, of the specified curve tables, only those located in the specified memory location are locked/unlocked. The rest are not locked/unlocked.	
	"SRAM"	Lock/unlock in the **static** NC memory
	"DRAM"	Lock/unlock in the **dynamic** NC memory

If CTABLOCK/CTABUNLOCK is programmed without specification of the curve table to be locked/unlocked, then **all** curve tables or all curve tables in the specified memory will be locked/unlocked:

CTABLOCK():	Locks all curve tables in the static and the dynamic NC memory
CTABLOCK(,,"SRAM"):	Locks all curve tables in the static NC memory
CTABLOCK(,,"DRAM"):	Locks all curve tables in the dynamic NC memory
CTABUNLOCK():	Unlocks all curve tables in the static and dynamic NC memory
CTABUNLOCK(,,"SRAM"):	Unlocks all curve tables in the static NC memory
CTABUNLOCK(,,"DRAM"):	Unlocks all curve tables in the dynamic NC memory

12.2.5 Curve tables: Determine table properties (CTABID, CTABISLOCK, CTABMEMTYP, CTABPERIOD)

These commands can be used to poll important properties of a curve table (table number, lock state, memory location, periodicity).

Syntax

```
CTABID(<p>)
CTABID(<p>,<memory location>)
CTABISLOCK(<n>)
CTABMEMTYP(<n>)
TABPERIOD(<n>)
```

Meaning

CTABID:	Returns the **table number** entered as the <p>th curve table in the specified memory. Example: CTABID(1,"SRAM") returns the number of the first curve table in the static NC memory. In this context the first curve table is the curve table with the highest table number. **Note:** If the sequence of curve tables in the memory changes between consecutive calls of CTABID, e.g. due to the deletion of curve tables with CTABDEL, CTABID(<p>,...) can return a different curve table with the same number <p>.	
CTABISLOCK:	Returns the **lock state** of curve table number <n>:	
	0	Table is not locked
	1	Table is locked by CTABLOCK
	2	Table is locked by active coupling
	3	Table is locked by CTABLOCK and active coupling
	-1	Table does not exist
CTABMEMTYP:	Returns the **memory location** of curve table number <n>:	
	0	Table in the static NC memory
	1	Table in the dynamic NC memory
	-1	Table does not exist
CTABPERIOD:	Returns the **periodicity** of curve table number <n>:	
	0	Table is not periodic
	1	Table is periodic in the leading axis
	2	Table is periodic in the leading and following axes
	-1	Table does not exist
<p>:	Entry number in memory	
<n>:	Number (**ID**) of curve table	
<memory location>:	Specification of memory location (optional)	
	"SRAM"	**Static** NC memory
	"DRAM"	**Dynamic** NC memory
	Note: If a value is not programmed for this parameter, the default memory location set with MD20905 $MC_CTAB_DEFAULT_MEMORY_TYPE is used.	

12.2.6 Read curve table values (CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABSSV, CTABSEV, CTAB, CTABINV, CTABTMIN, CTABTMAX)

The following curve table values can be read in the part program:

- Following axis and leading axis values at the start and end of a curve table
- Following axis values at the start and end of a curve segment
- Following axis value for a leading axis value

- Leading axis value for a following axis value
- Following axis minimum and maximum values
  - In the entire definition range of the curve table
    or
  - In a defined curve table interval

Syntax

```
CTABTSV(<n>,<gradient>[,<following axis>])
CTABTEV(<n>,<gradient>[,<following axis>])
CTABTSP(<n>,<gradient>[,<leading axis>])
CTABTEP(<n>,<gradient>[,<leading axis>])
CTABSSV(<master value>,<n>,<gradient>[,<following axis>])
CTABSEV(<master value>,<n>,<gradient>[,<following axis>])
CTAB(<master value>,<n>,<gradient>[,<following axis>,<leading axis>]
CTABINV(<following value>,<approximate
value>,<n>,<gradient>[,<following axis>,<leading axis>]
CTABTMIN(<n>[,<following axis>])
CTABTMAX(<n>[,<following axis>])
CTABTMIN(<n>,<a>,<b>[,<following axis>,<leading axis>])
CTABTMAX(<n>,<a>,<b>[,<following axis>,<leading axis>])
```

Meaning

CTABTSV:	Read following axis value at the **start** of curve table no. <n>
CTABTEV:	Read following axis value at the **end** of curve table no. <n>
CTABTSP:	Read leading axis value at the **start** of curve table no. <n>
CTABTEP:	Read leading axis value at the **end** of curve table no. <n>
CTABSSV:	Read following axis value at the **start** of the curve segment belonging to the specified leading axis value (<master value>)
CTABSEV:	Read following axis value at the **end** of the curve segment belonging to the specified leading axis value (<master value>)
CTAB:	Read following axis value for specified leading axis value (<master value>)
CTABINV:	Read leading axis value for specified following axis value (<following value>)
CTABTMIN:	Define following axis **minimum value**: • In the entire definition range of the curve table or • In a defined interval <a> ... <b>
CTABTMAX:	Define following axis **maximum value**: • In the entire definition range of the curve table or • In a defined interval <a> ... <b>
<n>:	Number (**ID**) of curve table
<gradient>:	The <gradient> parameter returns the **incline** of the curve table function at the calculated position.

<following axis>:	Axis whose motion is to be calculated using the curve table (optional)
<leasing axis>:	Axis providing the master values for the calculation of the following axis motion (optional)
<following value>:	Following axis value for reading the associated leading axis value for CTABINV
<leading value>:	Leading axis value: • For reading the associated following axis value with CTAB or • For the selection of the curve segment with CTABSSV/CTABSEV
<approximate value>:	The assignment of a leading axis value to a following axis value with CTABINV must not always be unique. CTABINV requires, therefore, an approximate value for the expected leading axis value as a parameter.
<a>:	**Lower** limit of the master value interval with CTABTMIN/CTABTMAX
<b>:	**Upper** limit of the master value interval with CTABTMIN/CTABTMAX
	Note: The master value interval <a> to <b> always has to be within the curve table's definition range.

Examples

Example 1:

Define following axis and leading axis values at the start and end of the curve table, along with the minimum and maximum values of the following axis in the entire definition range of the curve table.

```
Program code                              Comment
N10 DEF REAL STARTPOS
N20 DEF REAL ENDPOS
N30 DEF REAL STARTPARA
N40 DEF REAL ENDPARA
N50 DEF REAL MINVAL
N60 DEF REAL MAXVAL
N70 DEF REAL GRADIENT
...
N100 CTABDEF(Y,X,1,0)                     ; Start of table definition
N110 X0 Y10                               ; Start position 1st table segment
N120 X30 Y40                              ; End position 1st table segment =
                                          start position 2nd table segment
N130 X60 Y5                               ; End position 2nd table segment = ...
N140 X70 Y30
N150 X80 Y20
N160 CTABEND                              ; End of table definition.
...
N200 STARTPOS=CTABTSV(1,GRADIENT)         ; Following axis value at start of curve table
                                          = 10
N210 ENDPOS=CTABTEV(1,GRADIENT)           ; Following axis value at end of curve table
                                          = 20
```

```
Program code                                Comment
N220 STARTPARA=CTABTSP(1,GRADIENT)          ; Master axis value at start of curve table =
                                            0
N230 ENDPARA=CTABTEP(1,GRADIENT)            ; Master axis value at end of curve table = 80
N240 MINVAL=CTABTMIN(1)                     ; Minimum value of following axis with Y=5
N250 MAXVAL=CTABTMAX(1)                     ; Maximum value of following axis with Y=40
```

Example 2:

Determination of following axis values at the start and end of the curve segment associated with leading axis value X=30.

```
Program code                                 Comment
N10 DEF REAL STARTPOS
N20 DEF REAL ENDPOS
N30 DEF REAL GRADIENT
...
N100 CTABDEF(Y,X,1,0)                        ; Start of table definition.
N110 X0 Y0                                   ; Start position 1st table segment
N120 X20 Y10                                 ; End position 1st table segment =
                                             start position 2nd table segment
N130 X40 Y40                                 ; End position 2nd table segment = ...
N140 X60 Y10
N150 X80 Y0
N160 CTABEND                                 ; End of table definition.
...
N200 STARTPOS=CTABSSV(30.0,1,GRADIENT)  ; Start position Y in 2nd segment = 10
N210 ENDPOS=CTABSEV(30.0,1,GRADIENT)    ; End position Y in 2nd segment = 40
```

Further information

Use in synchronized actions

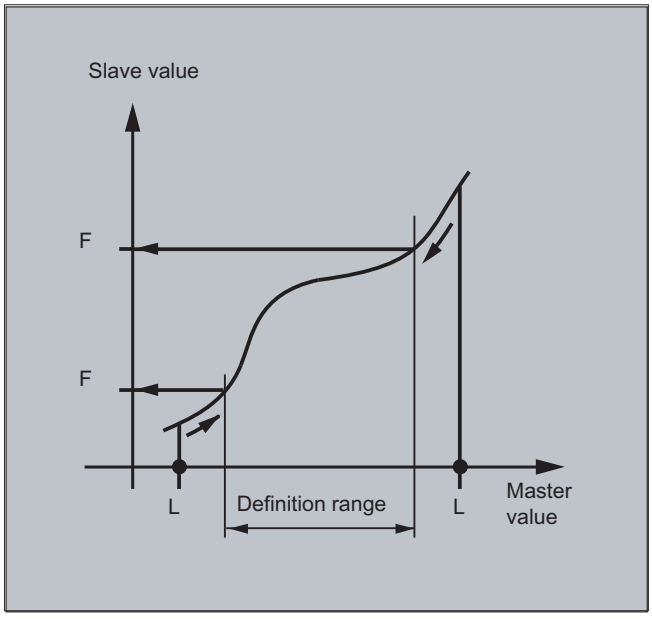
All commands for reading curve table values can also be used in synchronized actions (see also the chapter titled "Motion-synchronous actions").

When using the CTABINV, CTABTMIN, and CTABTMAX commands, make sure that:

- Sufficient NC power is available at the time of execution
  or
- The number of segments in the curve table is queried prior to the call, so that the table concerned can be subdivided if necessary

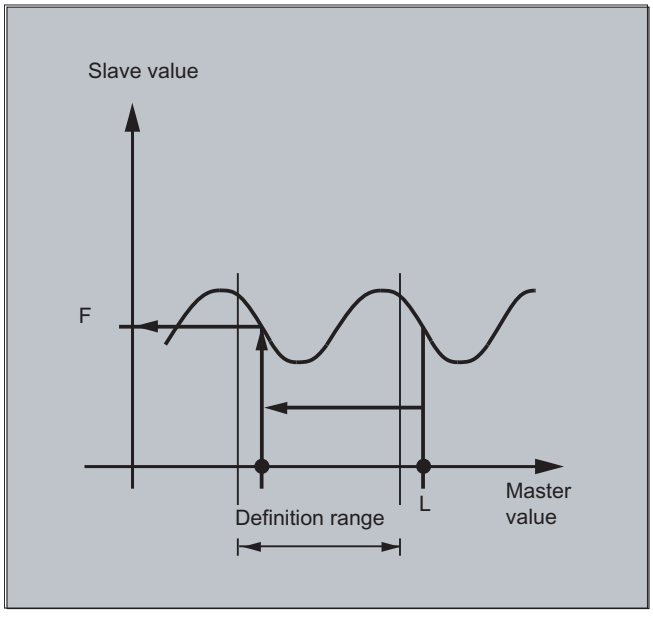
CTAB with non-periodic curve tables

If the specified <master value> is outside the definition range, the upper or lower limit will be output as the following value:



CTAB with periodic curve tables

If the specified <master value> is outside the definition range, the master value is evaluated modulo of the definition range and the corresponding following value is output:



Approximate value for CTABINV

The CTABINV command, therefore, requires an approximate value for the expected master value. CTABINV returns the leading value that is closest to the approximate value. The approximate value can be, for example, the master value from the previous interpolator clock cycle.

Incline of the curve table function

The output of the incline (<gradient>) makes it possible to calculate the velocity of the leading or following axis at the corresponding position.

Specification of the leading or following axis

The optional specification of the leading and/or following axis is important if the leading and following axes are configured in different length units.

CTABSSV, CTABSEV

The CTABSSV and CTABSEV commands are **not** suitable to query programmed segments in the following cases:

- Circles or involutes are programmed.
- Chamfer or rounding with CHF/ RND is active
- Smoothing with G643 is active
- NC block compression with COMPON/COMPCURV/COMPCAD is active

12.2.7 Curve tables: Check use of resources (CTABNO, CTABNOMEM, CTABFNO, CTABSEGID, CTABSEG, CTABFSEG, CTABMSEG, CTABPOLID, CTABPOL, CTABFPOL, CTABMPOL)

The programmer can use these commands to obtain up-to-date information about the use of resources for curve tables, table segments, and polynomials.

Syntax

```
CTABNO
CTABNOMEM(<memory location>)
CTABFNO(<memory location>)
CTABSEGID(<n>,<memory location>)
CTABSEG(<memory location>,<segment type>)
CTABFSEG(<memory location>,<segment type>)
CTABMSEG(<memory location>,<segment type>)
CTABPOLID(<n>)
CTABPOL(<memory location>)
CTABFPOL(<memory location>)
CTABMPOL(<memory location>)
```

Meaning

CTABNO:	Determine the total number of **defined** curve tables (in the static and the dynamic NC memory)
CTABNOMEM:	Determine the number of **defined** curve tables in the specified <memory location>
CTABFNO:	Determine the number of curve tables **remaining possible** in the specified <memory location>

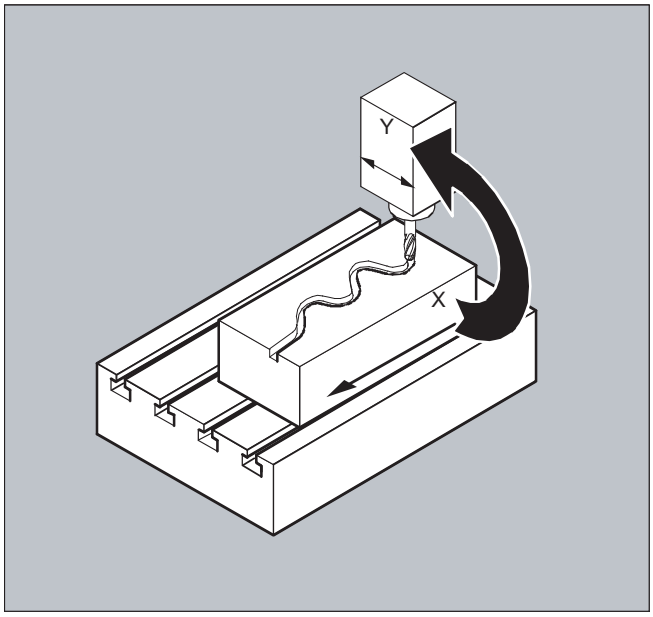
CTABSEGID:	Determine the number of curve segments of the specified <segment type> used by curve table number <n>	
CTABSEG:	Determine the number of curve segments of the specified <segment type> **used** in the specified <memory location>	
CTABFSEG:	Determine the number of curve segments of the specified <segment type> **remaining possible** in the specified <memory location>	
CTABMSEG:	Determine the **maximum possible** number of curve segments of the specified <segment type> in the specified <memory location>	
CTABPOLID:	Determine the number of curve polynomials used by curve table number <n>	
CTABPOL:	Determine the number of curve polynomials **used** in the specified <memory location>	
CTABFPOL:	Determine the number of curve polynomials **remaining possible** in the specified <memory location>	
CTABMPOL:	Determine the **maximum possible** number of curve polynomials in the specified <memory location>	
<n>:	Number (**ID**) of curve table	
<memory location>:	Specification of memory location (optional)	
	"SRAM"	**Static** NC memory
	"DRAM"	**Dynamic** NC memory
	Note: If a value is not programmed for this parameter, the default memory location set with MD20905 $MC_CTAB_DEFAULT_MEMORY_TYPE is used.	
<segment type>:	Specification of segment type (optional)	
	"L"	Linear segments
	"P"	Polynomial segments
	Note: If no value is programmed for this parameter, the sum of the linear and polynomial segments is output.	

12.3 Axial master value coupling (LEADON, LEADOF)

Note

This function is not available for SINUMERIK 828D!

With the axial master value coupling, a leading and a following axis are moved in synchronism. It is possible to assign the position of the following axis via a curve table or the resulting polynomial uniquely to a position of the leading axis – simulated if necessary.



The **leading axis** is the axis which supplies the input values for the curve table. The **following axis** is the axis, which takes the positions calculated by means of the curve table.

Actual value and setpoint coupling

The following can be used as the master value, i.e. as the output values for position calculation of the following axis:

- Actual values of the leading axis position: Actual value coupling
- Setpoints of the leading axis position: Setpoint value coupling

The master value coupling always applies in the basic coordinate system.

For information on the creation of curve tables, see Section "Curve tables".

Syntax

```
LEADON(<following axis>,<leading axis>,<n>)
LEADOF(<following axis>,<leading axis>)
```

or deactivation without specifying the leading axis:

```
LEADOF(<following axis>)
```

The master value coupling can be activated/deactivated both from the part program and also during motion from synchronized actions.

Meaning

`LEADON`:	Activate master value coupling
`LEADOF`:	Deactivate master value coupling
`<following axis>`:	Following axis
`<leasing axis>`:	Leading axis

<n>:	Curve table number
$SA_LEAD_TYPE:	Switching between setpoint and actual value coupling

Deactivate master value coupling, LEADOF

When you deactivate the master value coupling, the following axis becomes a normal command axis again!

Axial master value coupling and different operating states, RESET

Depending on the setting in the machine data, the master value couplings are deactivated with RESET.

Example of master value coupling from synchronous action

In a pressing plant, an ordinary mechanical coupling between a leading axis (stanchion shaft) and axis of a transfer system comprising transfer axes and auxiliary axes is to be replaced by an electronic coupling system.

It demonstrates how a mechanical transfer system is replaced by an electronic transfer system. The coupling and decoupling processes are implemented as **static synchronized actions**.

From the leading axis LV (stanchion shaft), transfer axes and auxiliary axes are controlled as following axes that are defined via curve tables.

Following axes

X feed or longitudinal axis
YL closing or transverse axis
ZL lifting axis
U roll feed, auxiliary axis
V guide head, auxiliary axis
W greasing, auxiliary axis

Actions

The actions that occur include, for example, the following synchronized actions:

- Activate coupling, `LEADON(<following axis>,<leading axis>,<curve table number>)`
- Deactivate coupling, `LEADOF(<following axis>,<leading axis>)`
- Set actual value, `PRESETON(<axis>,<value>)`
- Set marker, `$AC_MARKER[i]=<value>`
- Coupling type: real/virtual master value
- Approaching axis positions, `POS[<axis>]=<value>`

Conditions

Fast digital inputs, real-time variables $AC_MARKER and position comparisons are linked using the Boolean operator AND for evaluation as conditions.

Note

In the following example, line change, indentation and **bold** type are used for the sole purpose of improving readability of the program. For the control, everything that follows a line number constitutes a single line.

Comment

```
Program code                                                    Comment
                                                                ; Defines all static synchronized actions.
                                                                ; ****Reset marker
N2 $AC_MARKER[0]=0 $AC_MARKER[1]=0 $AC_MARKER[2]=0 $AC_MARKER[3]=0 $AC_MARKER[4]=0 $AC_MARKER[5]=0
$AC_MARKER[6]=0 $AC_MARKER[7]=0
                                                                ; **** E1 0=>1 transfer ON
N10 IDS=1 EVERY ($A_IN[1]==1) AND ($A_IN[16]==1) AND ($AC_MARKER[0]==0)
DO LEADON(X,LW,1) LEADON(YL,LW,2) LEADON(ZL,LW,3) $AC_MARKER[0]=1
                                                                ;**** E1 0=>1 coupling roller feed ON
N20 IDS=11 EVERY ($A_IN[1]==1) AND ($A_IN[5]==0) AND ($AC_MARKER[5]==0)
DO LEADON(U,LW,4) PRESETON(U,0) $AC_MARKER[5]=1
                                                                ; **** E1 0->1 coupling alignment head ON
N21 IDS=12 EVERY ($A_IN[1]==1) AND ($A_IN[5]==0) AND ($AC_MARKER[6]==0)
DO LEADON(V,LW,4) PRESETON(V,0) $AC_MARKER[6]=1
                                                                ; **** E1 0->1 lubrication coupling ON
N22 IDS=13 EVERY ($A_IN[1]==1) AND ($A_IN[5]==0) AND ($AC_MARKER[7]==0)
DO LEADON(W,LW,4) PRESETON(W,0) $AC_MARKER[7]=1
                                                                ; **** E2 0=>1 coupling OFF
N30 IDS=3 EVERY ($A_IN[2]==1)
DO LEADOF(X,LW) LEADOF(YL,LW) LEADOF(ZL,LW) LEADOF(U,LW) LEADOF(V,LW) LEADOF(W,LW) $AC_MARKER[0]=0
$AC_MARKER[1]=0 $AC_MARKER[3]=0 $AC_MARKER[4]=0 $AC_MARKER[5]=0 $AC_MARKER[6]=0 $AC_MARKER[7]=0
....
N110 G04 F01
N120 M30
```

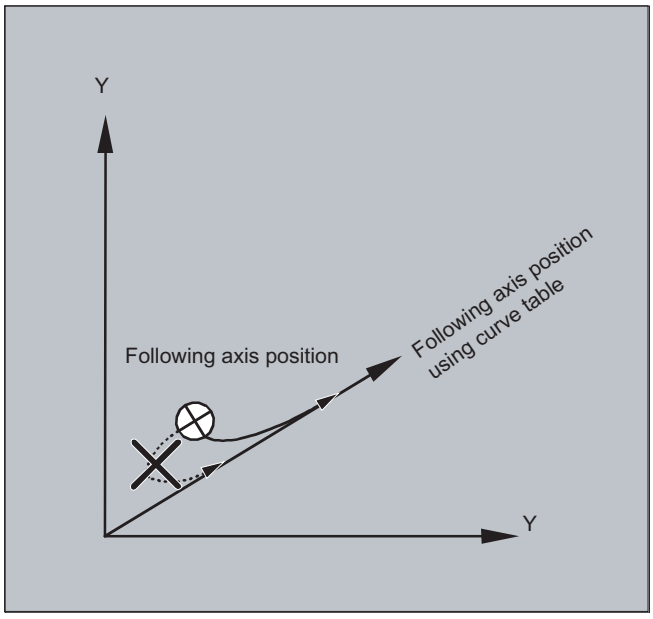
Description

Master value coupling requires synchronization of the leading and the following axes. This synchronization can only be achieved if the following axis is inside the tolerance range of the curve definition calculated from the curve table when the master value coupling is activated.

The tolerance range for the position of the following axis is defined via machine data MD 37200: COUPLE_POS_POL_COARSE A_LEAD_TYPE.

If the following axis is not yet at the correct position when the master value coupling is activated, the synchronization run is automatically initiated as soon as the position setpoint value calculated for the following axis is approximately the real following axis position. During the synchronization procedure the following axis is traversed in the direction that is defined by the

setpoint speed of the following axis (calculated from master spindle and using the CTAB curve table).

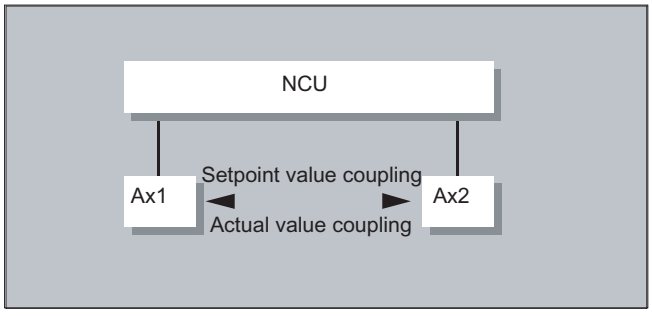


No synchronism

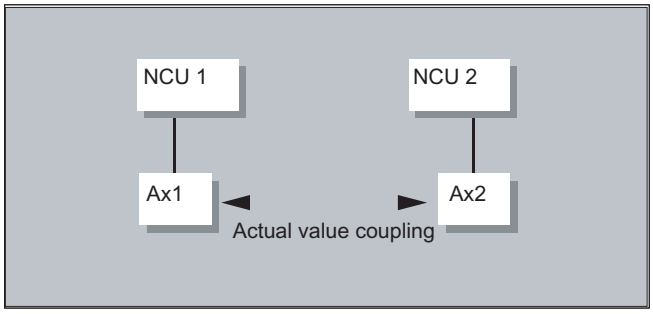
If the following axis position calculated moves away from the current following axis position when the master value coupling is activated, it is not possible to establish synchronization.

Actual value and setpoint coupling

Setpoint coupling provides better synchronization of the leading and following axis than actual value coupling and is therefore set by default.



Setpoint coupling is only possible if the leading and following axis are interpolated by the same NCU. With an external leading axis, the following axis can only be coupled to the leading axis via the actual values.



A **switchover** can be programmed via setting data $SA_LEAD_TYPE.

You must always switch between the actual-value and setpoint coupling when the following axis stops. It is only possible to resynchronize after switchover when the axis is motionless.

Application example

You cannot read the actual values without error during large machine vibrations. If you use master value coupling in press transfer, it might be necessary to switchover from actual-value coupling to setpoint coupling in the work steps with the greatest vibrations.

Master value simulation with setpoint coupling

Via machine data, you can disconnect the interpolator for the leading axis from the servo. In this way you can generate setpoints for setpoint coupling without actually moving the leading axis.

Master values generated from a setpoint link can be read from the following variables so that they can be used, for example, in synchronized actions:

- $AA_LEAD_P	Master value position
- $AA_LEAD_V	Master value velocity

Create master value

As an option, master values can be generated with other self-programmed methods. The master values generated in this way are written to and read from variables

- $AA_LEAD_SP	Master value position
- $AA_LEAD_SV	Master value velocity

Before you use these variables, the setting data $SA_LEAD_TYPE = 2 must be set.

Status of coupling

You can query the status of the coupling in the NC program with the following system variable:

```
$AA_COUP_ACT[[axis]]
```
0: No coupling active
16: Master value coupling active

Status management for synchronized actions

Switching and coupling events are managed via real-time variables:

```
$AC_MARKER[i] = n
```
managed with:

i flag number
n status value

12.4 Electronic gear (EG)

The "Electronic gear" function allows you to control the movement of a **following axis** according to linear traversing block as a function of up to five **leading axes**. The relationship between each leading axis and the following axis is defined by the coupling factor.

The following axis motion part is calculated by an addition of the individual leading axis motion parts multiplied by their respective coupling factors. When an EG axis grouping is activated, it is possible to synchronize the following axes in relation to a defined position. A gear group can be:

- Defined
- Activated
- Deactivated
- Deleted

.

The following axis movement can be optionally derived from

- Setpoints of the leading axes, as well as
- Actual values of leading axes.

Non-linear relationships between each leading axis and the following axis can also be realized as extension using **curve tables** (see "Path traversing behavior" section). Electronic gears can be cascaded, i.e., the following axis of an electronic gear can be the leading axis for a further electronic gear.

12.4.1 Defining an electronic gear (EGDEF)

An EG axis group is defined by specifying the following axis and at least one, however not more than five, leading axis, each with the relevant coupling type.

Requirement

Requirements for defining an EG axis group:

It is not permissible to define an axis coupling for the following axis (or an existing one must first be deleted with EGDEL).

Syntax

```
EGDEF(following axis,leading axis1,coupling type1,leading
axis2,coupling type2,...)
```

Meaning

`EGDEF`:	Definition of an electronic gear	
`Following axis`:	Axis that is influenced by the leading axes	
`Leading axis1 ,..., Leading axis5`	Axes that influence the following axis	
`Coupling type1 ,..., Coupling type5`	Coupling type The coupling type does not need to be the same for all leading axes and must be programmed separately for each individual master.	
	Value:	**Meaning:**
	0	The following axis is influenced by the **actual value** of the corresponding leading axis.
	1	The following axis is influenced by the **setpoint** of the corresponding leading axis.

Note

The coupling factors are preset to zero when the EG axis grouping is defined.

Note

EGDEF triggers preprocessing stop. The gearbox definition with EGDEF should also be used unaltered if, for systems, one or more leading axes affect the following axis via a **curve table**.

Example

Program code	**Comment**
`EGDEF(C,B,1,Z,1,Y,1)`	`; Definition of an EG axis group. Leading axes B, Z, Y influence the following axis C via the setpoint.`

12.4.2 Switch-in the electronic gearbox (EGON, EGONSYN, EGONSYNE)

There are 3 ways to switch-in an EG axis group.

Syntax

Variant 1:

The EG axis group is selectively switched-in without synchronization with:
`EGON(FA,"block change mode",LA1,Z1,N1,LA2,Z2,N2,...,LA5,Z5,N5)`

Variant 2:

The EG axis group is selectively activated with synchronization with:
`EGONSYN(FA,"block change mode",SynPosFA,[,LAi,SynPosLAi,Zi,Ni])`

Variant 3:

The EG axis group is selectively switched-in with synchronization and the approach mode specified with:

```
EGONSYNE(FA,"block change mode",SynPosFA,approach
mode[,LAi,SynPosLAi,Zi,Ni])
```

Meaning

Variant 1:

FA	Following axis	
`Block change mode`:	The following modes can be used:	
	`"NOC"`	Block change takes place immediately
	`"FINE"`	Block change is performed in "Fine synchronism"
	`"COARSE"`	Block change is performed in "Coarse synchronism"
	`"IPOSTOP"`	Block change is performed for setpoint-based synchronism
LA1, ... LA5	Leading axes	
Z1, ... Z5	Numerator for coupling factor i	
N1, ... N5	Denominator for coupling factor i Coupling factor i = numerator i/denominator i	

Only the leading axes previously specified with the EGDEF command may be programmed in the activation line. At least one leading axis must be programmed.

Variant 2:

`FA`	Following axis	
`Block change mode`:	The following modes can be used:	
	`"NOC"`	Block change takes place immediately
	`"FINE"`	Block change is performed in "Fine synchronism"
	`"COARSE"`	Block change is performed in "Coarse synchronism"
	`"IPOSTOP"`	Block change is performed for setpoint-based synchronism
`[,LAi,SynPosLAi,Zi,Ni]`	(do not write the square brackets) Min. 1, max. 5 sequences of:	
`LA1, ... LA5`	Leading axes	
`SynPosLAi`	Synchronized position for i-th leading axis	
`Z1, ... Z5`	Numerator for coupling factor i	
`N1, ... N5`	Denominator for coupling factor i Coupling factor i = numerator i/denominator i	

Only leading axes previously specified with the EGDEF command may be programmed in the activation line. Through the programmed "Synchronized positions" for the following axis (SynPosFA) and for the leading axes (SynPosLA), positions are defined for which the axis

grouping is interpreted as *synchronous*. If the electronic gear is not in the synchronized state when the grouping is switched on, the following axis traverses to its defined synchronized position.

Variant 3:

The parameters correspond to those of variant 2 plus:

`Approach mode`:	The following modes can be used:	
	`"NTGT"`	Approach next tooth gap time-optimized
	`"NTGP"`	Approach next tooth gap path-optimized
	`"ACN"`	Traverse rotary axis in negative direction absolute
	`"ACP"`	Traverse rotary axis in positive direction absolute
	`"DCT"`	Time-optimized for programmed synchronous position
	`"DCP"`	Distance-optimized to the programmed synchronous position

Variant 3 only affects modulo following axes that are coupled to modulo leading axes. Time optimization takes account of velocity limits of the following axis.

Further information

Description of the switch-in versions

Variant 1:

The positions of the leading axes and following axis at the instant the grouping is switched on are stored as "Synchronized positions". The "Synchronized positions" can be read with the system variable `$AA_EG_SYN`.

Variant 2:

If modulo axes are contained in the coupling group, their position values are modulus-reduced. This ensures that the next possible synchronized position is approached (so-called *relative synchronization*: e.g. the next tooth gap). The synchronized position is only approached if "Enable following axis override" interface signal DB(30 + axis number), DBX 26 bit 4 is issued for the following axis. If it is not issued, the program stops at the EGONSYN block and self-clearing alarm 16771 is output until the above mentioned signal is set.

Variant 3:

The tooth distance (deg.) is calculated like this: 360 * Zi/Ni. If the following axis is stopped at the time of calling, path optimization returns responds identically to time optimization.

If the following axis is already in motion, NTGP will synchronize at the next tooth gap irrespective of the current velocity of the following axis. If the following axis is already in motion, NTGT will synchronize at the next tooth gap depending on the current velocity of the following axis. The axis is also decelerated, if necessary.

Curve tables

If a **curve table** is used for one of the leading axes:

Ni The denominator of the coupling factor for linear coupling must be set to 0. (Denominator 0 would be illegal for linear couplings.) Denominator zero tells the control that

Zi is the number of the curve table to use. The curve table with the specified number must already be defined at POWER ON.

LAi The leading axis specified corresponds to the one specified for coupling via coupling factor (linear coupling).

For more information about using curve tables and cascading and synchronizing electronic gears, please refer to:
References:
Function Manual Special Functions; Coupled Axes and ESR (M3), "Coupled Motion and Leading Value Coupling".

Response of the Electronic gear at Power ON, RESET, mode change, block search

- **No** coupling is active after POWER ON.
- The status of active couplings is not affected by RESET or operating mode switchover.
- During block searches, commands for switching, deleting and defining the electronic gear are not executed or collected, but skipped.

System variables of the electronic gear

By means of the electronic gear's system variables, the part program can determine the current states of an EG axis grouping and react to them if required.

The system variables of the electronic gearbox are designated as follows:

$AA_EG_ ...

or

$VA_EG_ ...

References:
System Variables Manual

12.4.3 Switching-in the electronic gearbox (EGOFS, EGOFC)

There are 3 different ways to switch-out an active EG axis group.

Programming

Variant 1:

Syntax	Meaning
`EGOFS(following axis)`	The electronic gear is deactivated. The following axis is braked to a standstill. This call triggers a preprocessing stop.

Variant 2:

Syntax	Meaning
`EGOFS(following axis,leading axis1, …,leading axis5)`	This command parameter setting made it possible to **selectively** remove the influence of the individual leading axes on the following axis' motion.

At least one leading axis must be specified. The influence of the specified leading axes on the slave is selectively inhibited. This call triggers a preprocessing stop. If the call still includes active leading axes, then the slave continues to operate under their influence. If the influence of all leading axes is excluded by this method, then the following axis is braked to a standstill.

Variant 3:

Syntax	Meaning
`EGOFC(following spindle1)`	The electronic gear is deactivated. The following spindle continues to traverse at the speed/velocity that applied at the instant of deactivation. This call triggers a preprocessing stop.

Note

This variant is only permitted for spindles.

12.4.4 Deleting the definition of an electronic gear (EGDEL)

An EG axis group must be switched-out before its definition can be deleted.

Programming

Syntax	Meaning
`EGDEL(following axis)`	The coupling definition of the axis group is deleted. Additional axis groups can be defined by means of `EGDEF` until the maximum number of simultaneously activated axis groups is reached. This call triggers a preprocessing stop.

12.4.5 Rotational feedrate (G95) / electronic gear (FPR)

The FPR command can be used to specify the following axis of an electronic gear as the axis, which determines the revolutional feedrate. Please note the following with respect to this command:

- The feedrate is determined by the setpoint velocity of the following axis of the electronic gear.
- The setpoint velocity is calculated from the speeds of the leading spindles and modulo axes (which are not path axes) and from their associated coupling factors.
- Speed parts of linear or non-modulo leading axes and overlaid movement of the following axis are not taken into account.

12.5 Synchronous spindle

Synchronous operation involves a following spindle (FS) and a leading spindle (LS), referred to as the **synchronous spindle pair**. The following spindle imitates the movements of the leading spindle when a coupling is active (synchronous operation) in accordance with the defined functional interrelationship.

The synchronous spindle pairs for each machine can be assigned a fixed configuration by means of channel-specific machine data or defined for specific applications via the CNC part program. Up to two synchronized spindle pairs can be operated simultaneously on each NC channel.

Refer to the part program for the following coupling actions

- Defined or changed
- Activated
- Deactivated
- Deleted

.

In addition, depending on the software status

- It is possible to wait for the synchronism conditions
- The block change method can be changed
- Either the setpoint coupling or actual value coupling type is selected or the angular offset between master and following spindle specified
- When activating the coupling, previous programming of the following axis is transferred
- Either a measured or a known synchronism variance is corrected

12.5.1 Synchronous spindle: Programming (COUPDEF, COUPDEL, COUPON, COUPONC, COUPOF, COUPOFS, COUPRES, WAITC)

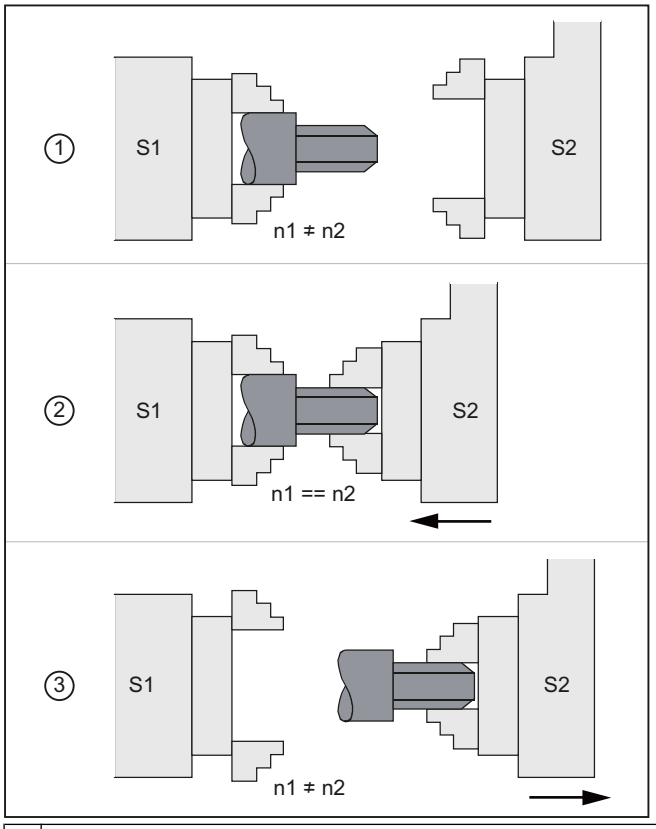
The "Synchronous spindle" enables the speed-synchronous traversing of the following spindle (FS) and leading spindle (LS) with a programmable transformation ratio.

The function supports the following modes:

- Speed synchronism ($n_{FS} = n_{LS}$)
- Position synchronism ($\phi_{FS} = \phi_{LS}$)
- Position synchronism with angular offset ($\phi_{FS} = \phi_{LS} + \Delta\phi$)

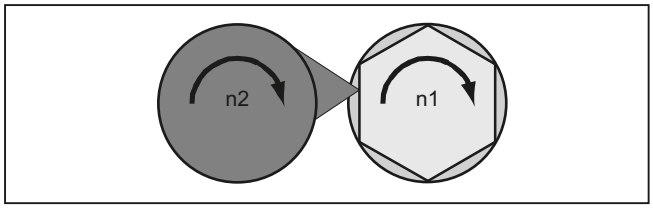
Application examples:

- Flying workpiece transfer, e.g. to machine the rear side, transformation ratio: 1:1



①	Synchronize the speed
②	Transfer the workpiece
③	Machine the rear side

- Multi-edge machining (polygonal turning), speed synchronism, transformation ratio: $n_1:n_2$



Syntax

```
COUPDEF(<FS>,<LS>,<ZFS>,<NLS>,<block change>,<coupling type>)
COUPON(<FS>,<LS>,<POSFS>)
COUPONC(<FS>,<LS>)
COUPOF(<FS>,<LS>,<POSFS>,<POSLS>)
COUPOFS(<FS>,<LS>)
COUPOFS(<FS>,<LS>,<POSFS>)
COUPRES(<FS>,<LS>)
COUPDEL(<FS>,<LS>)
WAITC(<FS>,<block change>,<LS>,<block change>)
```

Note

Abbreviated notation

A shorter notation without specification of the leading spindle is possible for the COUPOF, COUPOFS, COUPRES and COUPDEL statements.

Meaning

COUPDEF:	Define/change coupling on user-specific basis
COUPON:	Activate coupling. The following spindle synchronizes to the leading spindle based on the actual speed
COUPONC:	Coupling when activating with previous programming of M3 S... or M4 S.... A difference in speed for the following spindle is transferred immediately.
COUPOF:	Deactivate coupling. • With immediate block change: COUPOF(<S2>,<S1>) • Block change only after <POSFS> or <POSLS> deactivation position(s) has (have) been crossed: COUPOF(<S2>,<S1>,<POSFS>) COUPOF(<S2>,<S1>,<POSFS>,<POSLS>)
COUPOFS:	Deactivating a coupling with stop of following spindle. Block change as quickly as possible with immediate block change: COUPOFS(<S2>,<S1>) Block change only after passing the switch-off position: COUPOFS(<S2>,<S1>,<POSFS>)
COUPRES:	Reset coupling parameters to configured MD and SD
COUPDEL:	Delete user-defined coupling
WAITC:	Wait for synchronized run condition (NOC are increased to IPO during block changes)
<FS>:	Designation of following spindle
Optional parameters:	
<LS>:	Designation of main spindle Specification with spindle number: e.g. S2, S1

<ZFS>, <NLS>:	Transformation ratio between FS and LS. <ZFS>/<NLS> = numerator/denominator Default: <ZFS>/<NLS> = 1.0 ; specification of denominator optional	
<block change>:	Block change behavior The block change is:	
	"NOC"	Immediately
	"FINE"	On reaching "Synchronism fine"
	"COARSE"	On reaching "Synchronism coarse"
	"IPOSTOP"	On reaching IPOSTOP; in other words, after setpoint-based synchronism (default)
	The block change behavior is effective modally.	
<coupling type>:	Coupling type: Coupling between FS and LS	
	"DV"	Setpoint linkage (default)
	"AV"	Actual value coupling
	"VV"	Speed coupling
	The coupling type is modal.	
<POSFS>:	Angle offset between leading and following spindles	
	Range of values:	0°… 359,999°
<POSFS>,<POSLS>:	Switch-off positions of the following and leading spindles "The block change is enabled once POS_{FS}, POS_{LS} has been passed"	
	Range of values:	0°… 359,999°

Examples

Working with leading and following spindles

Program code	**Comment**
	Leading spindle = master spindle = spindle 1
	Following spindle = spindle 2
N05 M3 S3000 M2=4 S2=500	Leading spindle rotates at 3000 rpm, following spindle at 500 rpm.
N10 COUPDEF(S2,S1,1,1,"NOC","Dv")	Definition of the coupling (can also be configured).
...	
N70 SPCON	Bring leading spindle into closed-loop position control (setpoint coupling).
N75 SPCON(2)	Bring following spindle into closed-loop position control.
N80 COUPON(S2,S1,45)	On-the-fly coupling to offset position = 45 degrees.
...	
N200 FA[S2]=100	Positioning speed = 100 degrees/min
N205 SPOS[2]=IC(-90)	Traverse with 90 degrees overlay in negative direction.

Program code	Comment
N210 WAITC(S2,"Fine")	Wait for "fine" synchronism.
N212 G1 X... Y... F...	Machining
...	
N215 SPOS[2]=IC(180)	Traverse with 180 degrees overlay in the positive direction.
N220 G4 S50	Dwell time = 50 revolutions of the master spindle
N225 FA[S2]=0	Activate configured velocity (MD).
N230 SPOS[2]=IC(-7200)	20 revolutions. Move with configured velocity in the negative direction.
...	
N350 COUPOF(S2,S1)	Couple-out on-the-fly, S=S2=3000
N355 SPOSA[2]=0	Stop FS at zero degrees.
N360 G0 X0 Y0	
N365 WAITS(2)	Wait for spindle 2.
N370 M5	Stop FS.
N375 M30	

Programming a difference in speed

Program code	Comment
	Leading spindle = master spindle = spindle 1
	Following spindle = spindle 2
N01 M3 S500	Leading spindle rotates at 500 rpm.
N02 M2=3 S2=300	Following spindle rotates at 300 rpm.
...	
N10 G4 F1	Dwell time of master spindle.
N15 COUPDEF (S2,S1,-1)	Coupling factor with ratio -1:1
N20 COUPON(S2,S1)	Activate coupling. The speed of the following spindle results from the speed of the leading spindle and coupling factor.
...	
N26 M2=3 S2=100	Programming a difference in speed.

Examples of transfer of a movement for difference in speed

1. Activate coupling during previous programming of following spindle with COUPON

Program code	Comment
	Leading spindle = master spindle = spindle 1
	Following spindle = spindle 2
N05 M3 S100 M2=3 S2=200	Leading spindle rotates at 100 rpm, following spindle at 200 rpm.
N10 G4 F5	Dwell time = 5 seconds of master spindle
N15 COUPDEF(S2,S1,1)	Transformation ratio of FS to LS is 1.0 (default).
N20 COUPON(S2,S1)	On-the-fly coupling to the leading spindle.
N10 G4 F5	Following spindle rotates at 100 rpm.

2. Activate coupling during previous programming of following spindle with COUPONC

Program code	Comment
	Leading spindle = master spindle = spindle 1
	Following spindle = spindle 2
N05 M3 S100 M2=3 S2=200	Leading spindle rotates at 100 rpm, following spindle at 200 rpm.
N10 G4 F5	Dwell time = 5 seconds of master spindle
N15 COUPDEF(S2,S1,1)	Transformation ratio of FS to LS is 1.0 (default).
N20 COUPONC(S2,S1)	On-the-fly coupling to leading spindle and transfer previous speed to S2.
N10 G4 F5	S2 rotates at 100 rpm + 200 rpm = 300 rpm

3. Activate coupling with following spindle stationary with COUPON

Program code	Comment
	Leading spindle = master spindle = spindle 1
	Following spindle = spindle 2
N05 SPOS=10 SPOS[2]=20	Following spindle S2 in positioning mode.
N15 COUPDEF(S2,S1,1)	Transformation ratio of FS to LS is 1.0 (default).
N20 COUPON(S2,S1)	On-the-fly coupling to the leading spindle.
N10 G4 F1	Coupling is closed, S2 stops at 20 degrees.

4. Activate coupling with following spindle stationary with COUPONC

Note

Positioning or axis mode

If the following spindle is in positioning or axis mode before coupling, then the following spindle behaves the same for COUPON(<FS>,<LS>) and COUPONC(<FS>,<LS>).

Note

Leading spindle and axis operation

If, prior to the coupling being defined, the leading spindle is in axis operation, the velocity limit value from machine data

MD32000 $MA_MAX_AX_VELO (maximum axis velocity) will still apply even after the coupling is activated.

To avoid this behavior, the axis must be switched to spindle mode (M3 S... or M4 S...) prior to the coupling being defined.

Further information

Configured coupling

For the configured coupling, the LS and FS are defined via machine data. The configured spindles cannot be changed in the part program. The coupling can be parameterized in the part program using `COUPDEF` (on condition that no write protection is valid).

User-defined coupling

`COUPDEF` can be used to redefine or change a coupling in the part program. If a coupling is already active, it has to be deleted first with `COUPDEL` before a new coupling is defined.

A coupling is defined in its entirety by:

```
COUPDEF(<FS>,<LS>,<TFS>,<TLS>, block change behavior, coupling type)
```

Following spindle (FS) and leading spindle (LS)

The coupling is uniquely defined using the axis names for the FS and LS. The axis names have to be programmed with every `COUPDEF` statement. The other coupling parameters are modal and only have to be programmed if they change.

Example:

```
COUPDEF(S2,S1)
```

Transformation ratio

The transformation ratio is defined as the speed ratio between FS and LS:

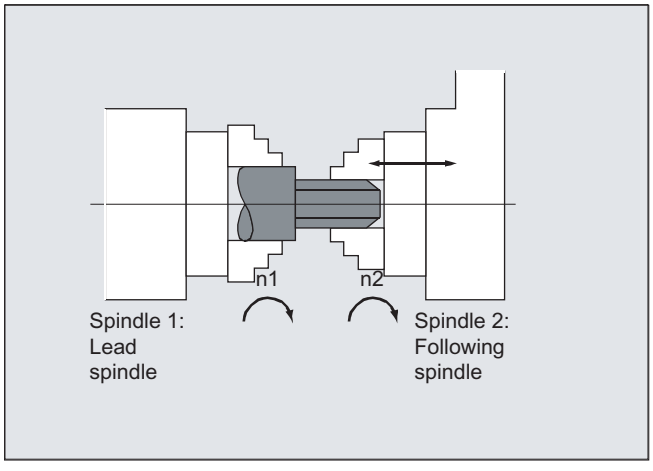
Following spindle / leading spindle = numerator/denominator

The numerator must be programmed. The denominator must not be programmed. The default value 1.0 is then set for the denominator.

Example:

Following spindle S2 and leading spindle S1, transformation ratio = 1/1

```
COUPDEF(S2, S1, 1.0)
```



Note

The transformation ratio can also be changed on-the-fly (when the coupling is active and the spindles are rotating).

Block change behavior NOC, FINE, COARSE, IPOSTOP

The following abbreviated notation can be used when programming the block change behavior:

- `"NO"`: Immediately (default)
- `"FI"`: On reaching "Synchronism fine"
- `"CO"`: On reaching "Synchronism coarse"
- `"IP"`: On reaching IPOSTOP; i.e. after setpoint-based synchronism

Type of coupling

Note

The coupling type may only be changed when the coupling is deactivated.

Activate synchronous mode COUPON, <POSFS>

- Activation of coupling with any angular offset between LS and FS:
  - `COUPON(S2,S1)`
  - `COUPON(S2)`
- Activation of coupling with angular offset <`POSFS`>
  `<POSFS>` refers to the 0° position of the leading spindle in the positive direction of rotation `<POSFS>` value range: 0°… 359,999°
  - `COUPON(S2,S1,30)`

Note

The angular offset can also be changed when the coupling is active.

Position the following spindle

Even with activated synchronous spindle coupling, the FS can be positioned in the range ±180° independently of the LS.

- Spindle positioning of the FS with `SPOS`
  Example: `SPOS[2]=IC(-90)`
  Further information on `SPOS` can be found in:
  References:
  Programming Manual, Fundamentals

Differential speed

A speed difference results in speed control mode and active synchronous spindle coupling through signed overlay of an FS speed because of LS movement and an FS speed because of spindle programming:

- Synchronous spindle coupling with `COUPONC`
- `S<FS>=<speed>` [`M<FS>=<direction of rotation>`]

Note

Supplementary conditions

- Speed `S...` must also be reprogrammed with direction of rotation `M3`/`M4`.
- Overlay of a spindle speed (`M<direction of rotation> S<FS>`) through the LS movement with synchronous spindle coupling `COUPONC` only becomes effective if the overlay has been enabled.
- The dynamic responses of the leading spindle have to be restricted to such an extent that when overlaying is applied to the following spindle, its dynamics limit values are not exceeded.

For more information about the speed difference, see:
References:
Function Manual, Extended Functions; Synchronous Spindle (S3)

Velocity, acceleration: FA, ACC, OVRA, VELOLIMA

Axial velocity and acceleration of a following spindle can be programmed with:

- `FA[SPI(S<n>)]` or `FA[S<n>]` (axial velocity)
- `ACC[SPI(S<n>)]` or `ACC[S<n>]` (axial acceleration)
- `OVRA[SPI(S<n>)]` and `OVRA[S<n>]` (axial override)
- `VELOLIMA[SPI(S<n>)]` and `VELOLIMA[S<n>]` (increase and reduction of axial velocity respectively)

When `<n>` = 1, 2, 3, ... (spindle numbers of the following spindles)

References:
Programming Manual, Fundamentals

Note

A reduction or increase of the maximum axial jerk has no effect with spindles.

Further information about the axial dynamic response is provided in:
References:
Function Manual, Extended Functions; Rotary Axes (R2)

Programmable block change behavior WAITC

`WAITC` can be used to define block change behavior, for example after a change to coupling parameters or positioning actions, with a variety of synchronism conditions (coarse, fine, `IPOSTOP`). If no synchronism conditions are specified, the block change behavior specified in the `COUPDEF` definition will apply.

Examples

- Wait for synchronism condition FINE to be fulfilled for following spindle S2 and COARSE to be fulfilled for following spindle S4: WAITC(S2,"FINE",S4,"COARSE")
- Wait for synchronism condition according to COUPDEF to be fulfilled: WAITC( )

Deactivate coupling COUPOF

COUPOF can be used to define the turn-off behavior of the coupling:

- Deactivation of coupling with immediate block change:
  - COUPOF(S2,S1) (with specification of leading spindle)
  - COUPOF(S2) (without specification of leading spindle)
- Deactivation of coupling after switch-off positions have been crossed. The block change takes place after the switch-off positions have been crossed.
  - COUPOF(S2,S1,150) (switch-off position FS: 150°)
  - COUPOF(S2,S1,150,30) (switch-off position FS: 150°, LS: 30°)

Deactivate coupling with following spindle stop COUPOFS

COUPOFS can be used to define the turn-off behavior of the coupling with following spindle stop:

- Deactivation of coupling with following spindle stop and immediate block change:
  - COUPOFS(S2,S1) (with specification of leading spindle)
  - COUPOFS(S2) (without specification of leading spindle)
- Deactivation of coupling after switch-off positions have been crossed with following spindle stop. The block change takes place after the switch-off positions have been crossed.
  - COUPOFS(S2,S1,150) (switch-off position FS: 150°)

Delete couplings COUPDEL

COUPDEL deletes the coupling:

- COUPDEL(S2,S1) (with specification of leading spindle)
- COUPDEL(S2) (without specification of leading spindle)

Reset coupling parameters, COUPRES

COUPRES activates the coupling values parameterized in the machine and setting data:

- COUPRES(S2,S1) (with specification of leading spindle)
- COUPRES(S2) (without specification of leading spindle)

System variables

- Current coupling status of following spindle
  The current coupling status of a following spindle can be read bit-coded via:
  <value> = $AA_COUP_ACT[<FS>]

Bit	<value>	Meaning
-	0	No coupling active
2	4	Synchronous spindle coupling active
Note • All other values refer to axis mode • If the spindle is a following spindle or several couplings, then the value of the coupling state of all couplings is returned as a total state.		

- Current angular offset
  The current angular offset of the following spindle to the leading spindle can be read via:
  - `$AA_COUP_OFFS[<FS>]` (angular offset on the setpoint side)
  - `$VA_COUP_OFFS[<FS>]` (angular offset on the actual value side)

  Application example
  Correction of the angular offset difference in the NC program after cancelling the follow-up mode:
  Angular offset difference = programmed angular offset - system variable

References

Detailed information on the system variables can be found in:

List Manual, System Variables

12.6 Generic coupling (CP...)

"Generic Coupling" is a general coupling function, combining all coupling characteristics of existing coupling types (coupled motion, master value coupling, electronic gearbox and synchronous spindle).

The function allows flexible programming:

- Users can select the coupling properties required for their applications (building block principle).
- Each coupling property can be programmed individually.
- The coupling properties of a defined coupling (e.g. coupling factor) can be changed.
- Later use of additional coupling properties is possible.
- The coordinate reference system of the following axis (base coordinate system or machine coordinate system) is programmable.
- Certain coupling properties can also be programmed with synchronous actions.
  References: Synchronized Actions Function Manual

Note

Previous coupling calls for coupled motion (TRAIL*), Master value coupling (LEAD*), Electronic Gearbox (EG*) and Synchronous spindle (COUP*) are supported via adaptive cycles.

Overview of all keywords and coupling characteristics

The following table gives an overview of all keywords of the generic coupling and the programmable coupling characteristics:

Keyword	Coupling characteristics / meaning	Syntax		
CPDEF	Creation of a coupling module	CPDEF=(<FAx>)		
CPDEL	Deletion of a coupling module	CPDEL=(<FAx>)		
CPLA	Definition of a leading axis	CPLA[<FAx>]=(<LAx>)		
CPLDEF	Definition of a leading axis and creation of a coupling module (also possible with CPDEF + CPLA)	CPLDEF[<FAx>]=(<LAx>) or CPDEF=(<FAx>) CPLA[<FAx>]=(<LAx>)		
CPLDEL	Deletion of a leading axis of a coupling module (also possible with CPDEF + CPLA)	CPLDEL[<FAx>]=(<LAx>) or CPDEL=(<FAx>) CPLA[<FAx>]=(<LAx>)		
CPON	Switching on a coupling module	CPON=(<FAx>)		
CPOF	Switching off a coupling module	CPOF=(<FAx>)		
CPLON	Switching on a leading axis of a coupling module	CPLON[<FAx>]=<LAx>		
CPLOF	Switching off a leading axis of a coupling module	CPLOF[<FAx>]=<LAx>		
CPLNUM	Numerator of the coupling factor	CPLNUM[FAx,LAx]=<value>		
CPLDEN	Denominator of the coupling factor	CPLDEN[FAx,LAx]=<value>		
CPLCTID	Number of the curve table	CPLCTID[FAx,LAx]=<value>		
CPLSETVAL	Coupling reference	CPLSETVAL[FAx,LAx]="<coupling reference>"		
		"<coupling reference>":	"CMDPOS"	Setpoint value coupling
			"CMDVEL"	Speed coupling
			"ACTPOS"	Actual value coupling
CPFRS	Coordinate reference system	CPFRS[FAx]="<coordinate reference>"		
		"<coordinate reference>":	"BCS"	Basic Coordinate System
			"MCS"	Machine Coordinate System

Keyword	Coupling characteristics / meaning	Syntax		
CPBC	Block change criterion	CPBC[FAx]="<block change criterion>"		
		"<block change criterion>":	"NOC"	Block change is performed irrespective of the coupling status.
			"IPOSTOP"	Block change is performed with setpoint synchronism.
			"COARSE"	Block change is performed with actual value synchronism “coarse”.
			"FINE"	Block change is performed with actual value synchronism "fine".
CPFPOS + CPON	Synchronized position of the following axis when switching on	CPON=FAx CPFPOS[FAx]=<value>		
CPLPOS + CPON	Synchronized position of the leading axis when switching on	CPLPOS[FAx,LAx]=<value>		

Keyword	Coupling characteristics / meaning	Syntax		
CPFMSON	Synchronization mode	CPFMSON[FAx]="<synchronization mode>"		
		"<synchronization mode>":	"CFAST"	The coupling is closed time-optimized.
			"CCOARSE"	The coupling is only closed when the following axis position, required according to the coupling rule, is in the range of the current following axis position.
			"NTGT"	The next tooth gap is approached time-optimized.
			"NTGP"	The next tooth gap is approached path-optimized.
			"NRGT"	The next segment is approached in a time-optimized manner, in accordance with the ratio of the number of gears to the number of teeth.
			"NRGP"	The next segment is approached in a path-optimized manner, in accordance with the ratio of the number of gears to the number of teeth.
			"ACN"	For rotary axes only! The rotary axis traverses to the synchronized position in the negative axis direction. Synchronization is realized immediately.
			"ACP"	For rotary axes only! The rotary axis traverses to the synchronized position in the positive axis direction. Synchronization is realized immediately.
			"DCT"	For rotary axes only! The rotary axis traverses to the programmed synchronized position time-optimized. Synchronization is realized immediately.
			"DCP"	For rotary axes only! The rotary axis traverses to the programmed synchronized position path-optimized. Synchronization is realized immediately.

Keyword	Coupling characteristics / meaning	Syntax		
CPFMON	Behavior of the following axis when switching on	CPFMON[FAx]= "<switch-on behavior>"		
		"<switch-on behavior>":	"STOP"	For spindles only! An active motion of the following spindle is stopped before switch-on.
			"CONT"	For spindles and main traverse axes only! The current motion of the following axis/spindle is taken over into the coupling as start motion.
			"ADD"	For spindles only! The motion components of the coupling operate in addition to the currently overlaid motion, i.e. the current motion of the following axis/spindle is retained as overlaid motion.
CPFMOF	Behavior of the following axis at complete switch-off	CPFMOF[FAx]="<switch-off behavior>"		
		"<switch-off behavior>":	"STOP"	Stop of a following axis/spindle. An active overlaid motion is also braked to standstill. The coupling is then opened
			"CONT"	For spindles and main traverse axes only! The following spindle continues to traverse at the speed/velocity that applied at the instant of deactivation.
CPFPOS + CPOF	Switch-off position of the following axis when switching off	CPOF=(FAx) CPFPOS[FAx]=<value>		

Keyword	Coupling characteristics / meaning	Syntax		
CPMRESET	Coupling behavior for RESET	CPMRESET[FAx]="<Reset behavior>"		
		"<reset behavior>":	"NONE"	The current state of the coupling is retained.
			"ON"	When the appropriate coupling module is created, the coupling is switched on. All defined leading axis relationships are activated. This is also performed when all or parts of these leading axis relationships are active, i.e. resynchronization is performed even with a completely activated coupling.
			"OF"	An active overlaid motion is also braked to standstill. The coupling is then deactivated. When the relevant coupling module was created without an explicit definition (CPDEF), the coupling module is deleted. Otherwise it is retained, i.e. it can still be used.
			"OFC"	Possible only in spindles! The following spindle continues to traverse at the speed/velocity that applied at the instant of deactivation. The coupling is switched off. When the relevant coupling module was created without an explicit definition (CPDEF), the coupling module is deleted. Otherwise it is retained, i.e. it can still be used.
			"DEL"	An active overlaid motion is also braked to standstill. The coupling is then deactivated and then deleted.
			"DELC"	Possible only in spindles! The following spindle continues to traverse at the speed/velocity that applied at the instant of deactivation. The coupling is deactivated and then deleted.

Keyword	Coupling characteristics / meaning	Syntax		
CPMSTART	Coupling behavior at part program start	CPMSTART[FAx]="<start behavior>"		
		"<start behavior>":	"NONE"	The current state of the coupling is retained.
			"ON"	Coupling switched-on. All defined leading axis relationships are activated. This is also performed when all or parts of these leading axis relationships are active, i.e. resynchronization is performed even with a completely activated coupling.
			"OF"	The coupling is switched off. When the relevant coupling module was created without an explicit definition (CPDEF), the coupling module is deleted. Otherwise it is retained, i.e. it can still be used.
			"DEL"	The coupling is deactivated and then deleted.
CPMPRT	Coupling response at part program start under search run via program test	CPMPRT[FAx]="<start behavior>"		
		"<start behavior>":	see CPMSTART	
CPLINTR	Offset value of the input value of a leading axis	CPLINTR[FAx,LAx]=<value>		
CPLINSC	Scaling factor of the input value of a leading axis	CPLINSC[FAx,LAx]=<value>		
CPLOUTTR	Offset value for the output value of a coupling	CPLOUTTR[FAx,LAx]=<value>		
CPLOUTSC	Scaling factor for the output value of a coupling	CPLOUTSC[FAx,LAx]=<value>		
CPSYNCOP	Threshold value of position synchronism "Coarse"	CPSYNCOP[FAx]=<value>		
CPSYNFIP	Threshold value of position synchronism "Fine"	CPSYNFIP[FAx]=<value>		
CPSYNCOP2	Second threshold value for the "Coarse" position synchronism	CPSYNCOP2[FAx]=<value>		
CPSYNFIP2	Second threshold value for the "Fine" position synchronism	CPSYNFIP2[FAx]=<value>		
CPSYNCOV	Threshold value of velocity synchronism "Coarse"	CPSYNCOV[FAx]=<value>		
CPSYNFIV	Threshold value of velocity synchronism "Fine"	CPSYNFIV[FAx]=<value>		

Keyword	Coupling characteristics / meaning	Syntax		
CPMBRAKE	Response of the following axis to certain stop signals and stop commands	CPMBRAKE[FAx]=<bit-coded value>		
CPMVDI	Response of the following axis to certain NC/PLC interface signals	CPMVDI[FAx]=<bit-coded value>		
CPMALARM	Suppression of special coupling-related alarm outputs	CPMALARM[FAx]=<bit-coded value>		
CPSETTYPE	Coupling type	CPSETTYPE[FAx]="<coupling type>"		
		"<coupling type>":	"CP"	Freely programmable
			"TRAIL"	Coupling type "Coupled motion"
			"LEAD"	Coupling type "Master Value Coupling"
			"EG"	Coupling type "Electronic gearbox"
			"COUP"	Coupling type "Synchronized spindle"

FAx: following axis/spindle

LAx: leading axis/spindle

Note

Coupling characteristics, which are not explicitly programmed (in part program of synchronous actions), become effective with their default settings.

Depending on the settings of the keyword CPSETTYPE instead of the default settings (CPSETTYPE="CP") preset coupling characteristics can become effective.

References

For detailed information on generic couplings, see:

- Function Manual, Special Functions; M3: Axis couplings, Chapter: "Generic coupling"

12.7 Master/slave coupling (MASLDEF, MASLDEL, MASLON, MASLOF, MASLOFS)

The "master/slave coupling" enables:

- The coupling of the slave axes to the master axis, when the axes involved are at standstill.
- The coupling/decoupling of **rotating**, speed-controlled spindles.
- The dynamic configuration.

Note

Positioning mode

For axes and spindles in the positioning mode, the coupling is only closed and opened at standstill.

Syntax

```
MASLON(<slave_1>,<slave_2>,...)
MASLOF(<slave_1>,<slave_2>,...)
MASLOFS(<slave_1>,<slave_2>,...)
```

Dynamic configuration:

```
MASLDEF(<slave_1>,<slave_2>, ... ,<master>)
MASLDEL(<slave_1>,<slave_2>,...)
```

Meaning

MASLON:	Activating a temporary master/slave coupling	
	<Slave_x>,...:	Slave axis 1 ... n
MASLOF:	Decoupling an active master/slave coupling	
	<slave_1>,...:	Slave axis 1 ... n
MASLOFS:	Decoupling a master/slave coupling and automatically braking slave spindles (see note "Coupling behavior for spindles! in speed control mode"!)	
	<slave_1>,...:	Slave axis 1 ... n
MASLDEF:	Creating/changing a master/slave group from the part program	
	<slave_1>,...:	Slave axis 1 ... n
	<master>:	Master axis
MASLDEL:	Separate master/slave coupling and delete the definition of the grouping	
	<slave_1>,...:	Slave axis 1 ... n
	Note: The master/slave definitions configured in the machine data are retained.	

Note

Coupling behavior for spindles in speed control mode

For spindles in the speed control mode, the coupling behavior of MASLON, MASLOF, MASLOFS and MASLDEL are specified explicitly via the following machine data:

MD37263 $MA_MS_SPIND_COUPLING_MODE

For the default setting with MD37263 = 0, the slave axes are coupled-in and coupled-out only when the axes involved are at standstill. MASLOFS corresponds to MASLOF.

For MD37263 = 1, the coupling instruction is immediately executed and therefore also the motion. For MASLON the coupling is immediately closed and for MASLOFS or MASLOF immediately opened. With MASLOF, the slave spindles rotating at this instant keep their speeds until a new speed is programmed. However, with MASLOFS, they are braked automatically.

Note

For MASLOF/MASLOFS, the implicit preprocessing stop is not required. Because of the missing preprocessing stop, the $P system variables for the slave axes do not provide updated values until next programming.

Note

For the slave axis, the actual value can be synchronized to the same value of the master axis using PRESETON. To do this, the permanent/slave coupling must be briefly switched off in order to set the actual value of the non-referenced slave axis to the value of the master/axis with Power On. Then the coupling is permanently re-established.

The permanent master/slave coupling is activated with the following MD setting:
MD37262 $MA_MS_COUPLING_ALWAYS_ACTIVE = 1
It has no effect on the language commands of the temporary coupling.

Examples

Example 1: Set actual value for the slave axis of a master/slave coupling

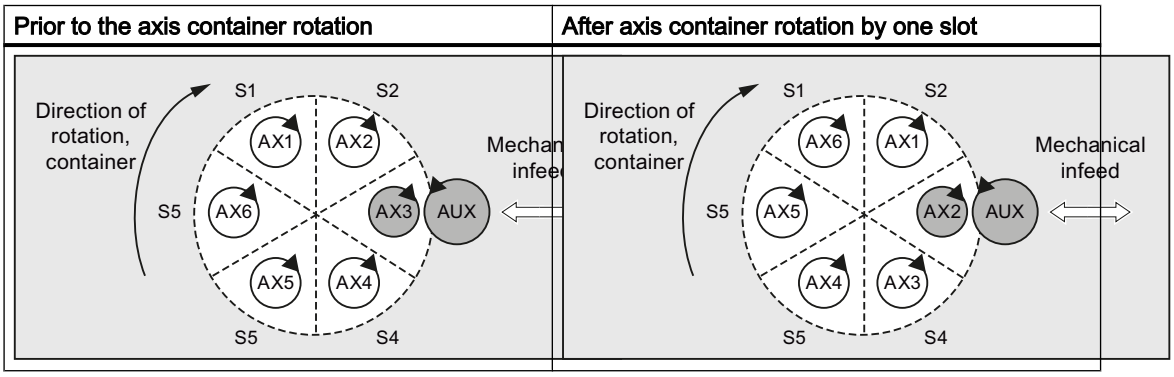
For a permanent master/slave coupling, PRESETON sets the actual value of the slave axis to the value of the master axis.

Program code	Comment
$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=0	; Deactivate the permanent coupling of the slave axis
NEWCONF	; Activate machine data change
STOPRE	
MASLOF(Y1)	; Deactivate temporary coupling
PRESETON(AX2,$VA_IM(M_AX))	; Actual value of the slave axis = actual value of the master axis
$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=1	; Activate the permanent coupling of the slave axis

Program code	Comment
NEWCONF	; Activate machine data change

Example 2: Dynamic configuration of a master/slave coupling

To enable coupling with another spindle after axis container rotation, the previous coupling must be uncoupled, the configuration cleared, and a new coupling configured.



Program code	Comment
MASLDEF(AUX,S3)	; AUX: Slave, S3: Master = AX3
MASLON(AUX)	; Coupling on
M3=3 S3=4000	; Rotate master
MASLDEL(AUX)	; Disconnect and delete coupling
AXCTSWE(CT1)	; Enable axis container rotation
MASLDEF(AUX,S3)	; AUX: Slave, S3: Master = AX2

References

- Function Manual Special Functions, Chapter "TE3: Speed/torque coupling, master-slave"
- Function Manual Extended Functions, Chapter "B3: Distributed systems - only 840D sl" > "NCU link" > "Axis container"

Synchronized actions 13

13.1 Definition of a synchronized action

A synchronized action is defined in a block of a part program. Any further commands that are not part of the synchronized action, may not be programmed within this block.

A synchronized action consists of the following components:

Validity, ID no. (optional)	Condition part (optional)			Action part		
	Frequency	G function (optional)	Condition	Keyword	G function (optional)	Actions
--- [1] ID=<no.> IDS=<no.>	--- [1] WHENEVER FROM WHEN EVERY	G...	Logical expression	DO	G...	Action 1 ... Action n

[1] Not programmed

Syntax

```
DO <action 1> ... <action n>
<frequency> [<G function>] <condition> DO <action 1> ... <action n>
ID=<No> <frequency> [<G function>] <condition> DO <action 1> ...
<action n>
IDS=<No> <frequency> [<G function>] <condition> DO <action 1> ...
<action n>
```

References

A detailed description of the functionality of synchronized actions can be found in:

Function Manual, Synchronized Actions

Oscillation 14

14.1 Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE, OSB)

An oscillating axis travels back and forth between two reversal points 1 and 2 at a defined feedrate, until the oscillating motion is deactivated.

Other axes can be interpolated as desired during the oscillating motion. A continuous infeed can be achieved via a path movement or with a positioning axis, however, there is **no relationship** between the oscillating movement and the infeed movement.

Properties of asynchronized oscillation

- Asynchronous oscillation is active on an axis-specific basis beyond block limits.
- Block-oriented activation of the oscillation movement is ensured by the part program.
- Combined interpolation of several axes and superimposing of oscillation paths are not possible.

Programming

The following commands can be used to activate and control asynchronous oscillation from the part program.

The programmed values are entered in the corresponding setting data with block synchronization during the main run and remain active until changed again.

Syntax

```
OSP1[<axis>]=<value> OSP2[<axis>]=<value>
OST1[<axis>]=<value> OST2[<axis>]=<value>
FA[<axis>]=<value>
OSCTRL[<axis>]=(<setting option>,<reset option>)
OSNSC[<axis>]=<value>
OSE[<axis>]=<value>
OSB[<axis>]=<value>
OS[<axis>] = 1
OS[<axis>] = 0
```

Meaning

<axis>:	Name of oscillating axis		
OS:	Activate/deactivate oscillation		
	Value:	1	Switch oscillation **on**
		0	Switch oscillation **off**
OSP1:	Define position of reversal point 1		

OSP2:	Define position of reversal point 2 **Note:** If incremental movement is active, the position will be calculated incrementally to the last corresponding reversal position programmed in the NC program.		
OST1:	Define stopping time in reversal point 1 in [s]		
OST2:	Define stopping time in reversal point 2 in [s]		
	<value>:	-2	Interpolation continues without wait for exact stop
		-1	Wait for exact stop coarse
		0	Wait for exact stop fine
		>0	Wait for exact stop fine and then wait for specified stopping time **Note:** The unit for the stopping time is identical to that of the stopping time programmed with G4.
FA:	Define feedrate The feedrate is the defined feedrate of the positioning axis. If no feedrate is defined, the value stored in the machine data applies.		
OSCTRL:	Specify setting and reset options Option values 0 to 3 encrypt the behavior at the reversal points on deactivation. One of the variants from 0 to 3 can be selected. The remaining settings can be combined at will with the selected variant. Multiple options are appended with plus characters (+).		
	<value>:	0	Stop at next reversal point on deactivation of oscillation (default) **Note:** Only possible if values 1 and 2 are reset.
		1	When the oscillation is deactivated, stop at reversal point 1
		2	When the oscillation is deactivated, stop at reversal point 2
		3	When the oscillation is deactivated, do not approach reversal point if no spark-out strokes are programmed
		4	Approach end position after spark-out
		8	If oscillation is canceled by deletion of distance-to-go, sparking-out strokes will then need to be executed and the end position approached if necessary.
		16	If oscillation is canceled by deletion of distance-to-go, the corresponding reversal point will need to be approached as is the case with shutdown.
		32	New feed is only active after the next reversal point
		64	FA equal to 0, FA = 0: Path overlay is active FA not equal to 0, FA <> 0: Speed overlay is active
		128	For rotary axis DC (shortest path)
		256	The sparking-out stroke is a dual stroke (default). 1=Single stroke.
		512	First approach start position
OSNSC:	Define number of sparking-out strokes		

OSE:	Define end position (in workpiece coordinate system) to be approached after deactivation of oscillation. **Note:** When programming "OSE" option 4 becomes effective implicitly for "OSCTRL".
OSB:	Define start position (in workpiece coordinate system) to be approached prior to activation of oscillation. The start position is approached before reversal point 1. If the start position coincides with reversal position 1, reversal position 2 is approached next. No stopping time applies when the start position is reached, even if this position coincides with reversal position 1; instead, the axis waits for the exact stop fine signal. Any exact stop condition configured is fulfilled. **Note:** Bit 9 in setting data SD43770 $SA_OSCILL_CTRL_MASK must be set to initiate an approach to the start position.

Examples

Example 1: Oscillating axis to oscillate between two reversal points

Oscillating axis Z is to oscillate between position 10 and 100. Reversal point 1 is to be approached with exact stop fine, reversal point 2 with exact stop coarse. The feedrate for the oscillating axis must be 250. 3 sparking-out strokes must be executed at the end of the machining operation and the oscillating must approach end position 200. The feedrate for the infeed axis must be 1 and the end of infeed in the X direction should be reached at position 15.

```
Program code                                Comment
WAITP(X,Y,Z)                                ; Initial setting.
G0 X100 Y100 Z100                           ; Switch over to positioning axis operation.
WAITP(X,Z)
OSP1[Z]=10 OSP2[Z]=100                      ; Reversal point 1, reversal point 2.
OSE[Z]=200                                  ; End position.
OST1[Z]=0 OST2[Z]=-1                        ;Stopping time at U1: Exact stop fine
                                            ; Stopping time at U2: Exact stop coarse
FA[Z]=250 FA[X]=1                           ; Feed for oscillating axis, infeed axis
OSCTRL[Z]=(4,0)                             ; Setting options.
OSNSC[Z]=3                                  ; 3 sparking-out strokes.
OS[Z]=1                                     ; Start oscillation.
WHEN $A_IN[3]==TRUE DO DELDTG(X)            ; Deletion of distance-to-go.
POS[X]=15                                   ; Starting position X axis.
POS[X]=50                                   ; End position X axis.
OS[Z]=0                                     ; Stop oscillation.
M30
```

Note

The "OSP1[Z]=..." to "OSNCS[Z]=..." command sequence can also be programmed in a block.

Example 2: Oscillation with online modification of the reversal position

The setting data necessary for asynchronous oscillation can be set in the part program.

If the setting data is described directly in the program, the change takes effect during preprocessing. A synchronized response can be achieved by means of a preprocessing stop (STOPRE).

Program code	Comment
$SA_OSCILL_REVERSE_POS1[Z]=-10	
$SA_OSCILL_REVERSE_POS2[Z]=10	
G0 X0 Z0	
WAITP(Z)	
ID=1 WHENEVER $AA_IM[Z] < $$AA_OSCILL_REVERSE_POS1[Z] DO $AA_OVR[X]=0	; If the actual value of the oscillating axis has exceeded the reversal point, then the infeed axis is stopped.
ID=2 WHENEVER $AA_IM[Z] < $$AA_OSCILL_REVERSE_POS2[Z] DO $AA_OVR[X]=0	
OS[Z]=1 FA[X]=1000 POS[X]=40	; Activate oscillation.
OS[Z]=0	; Deactivate oscillation.
M30	

Further information

Oscillating axis

The following apply to the oscillating axis:

- Every axis may be used as an oscillation axis.
- Several oscillation axes can be active at the same time (maximum: the number of the positioning axes).
- Linear interpolation G1is always active for the oscillating axis – irrespective of the G command currently valid in the program.

The oscillating axis can:

- Act as an input axis for dynamic transformation
- Act as a guide axis for gantry and coupled-motion axes
- Be traversed:
  - Without jerk limitation "BRISK"
    or
  - With jerk limitation "SOFT"
    or
  - With acceleration curve with a knee (as positioning axes)

Oscillation reversal points

The current offsets must be taken into account when oscillation positions are defined:

- Absolute specification
  "OSP1[Z]=<value>"
  Position of reversal point = sum of offsets + programmed value
- Relative specification
  "OSP1[Z]=IC(<value>)"
  Position of reversal point = reversal point 1 + programmed value

Example:

Program code

```
N10 OSP1[Z]=100 OSP2[Z]=110
...
N40 OSP1[Z]=IC(3)
```

WAITP

If oscillation is to be performed with a geometry axis, you must enable this axis for oscillation with "WAITP".

When oscillation has finished, "WAITP" is used to enter the oscillating axis as a positioning axis again, so that normal use can resume.

Oscillation with motion-synchronous actions and stopping times

Once the set stop times have expired, the internal block change is executed during oscillation (indicated by the new distances to go of the axes). The deactivation function is checked when the block changes. The deactivation function is defined according to the control setting for the motion sequence (OSCTRL). *This dynamic response can be influenced by the feed override.*

An oscillation stroke may then be executed before the sparking-out strokes are started or the end position approached. *Although it appears as if the deactivation response has changed, this is not in fact the case.*

14.2 Oscillation controlled by synchronized actions (OSCILL)

With this mode of oscillation, an infeed motion may only be executed at the reversal points or within defined reversal areas.

Depending on requirements, the oscillation movement can be

- Continued or
- Stopped until the infeed has finished executing.

Syntax

1. **Define parameters for oscillation**
2. **Define motion-synchronous actions**
3. **Assign axes, define infeed**

Meaning

`OSP1[<oscillating axis>]=`	Position of reversal point 1
`OSP2[<oscillating axis>]=`	Position of reversal point 2
`OST1[<oscillating axis>]=`	Stopping time at reversal point 1 in seconds
`OST2[<oscillating axis>]=`	Stopping time at reversal point 2 in seconds
`FA[<oscillating axis>]=`	Feed for oscillating axis
`OSCTRL[<oscillating axis>]=`	Set or reset options
`OSNSC[<oscillating axis>]=`	Number of sparking-out strokes
`OSE[<oscillating axis>]=`	End position
`WAITP(<oscillating axis>)`	Enable axis for oscillation

Axis assignment, infeed

```
OSCILL[<oscillating axis>]=(<infeed axis 1>,<infeed axis 2>,<infeed axis 3>)
POSP[<infeed axis>]=(<end position>,<partial length>,<mode>)
```

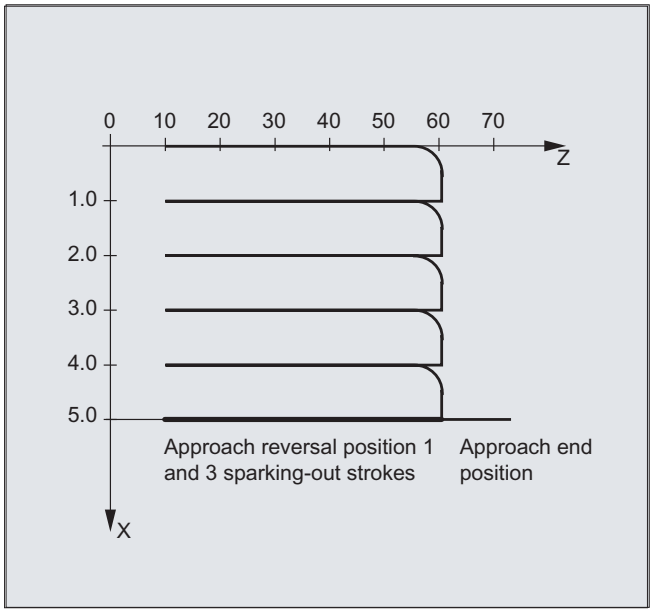
`OSCILL`:	Assign infeed axis or axes for oscillating axis
`POSP`:	Define complete and partial infeeds (see Section "File and Program Management")
`End position`:	End position for the infeed axis after all partial infeeds have been traversed.
`Partial length`:	Length of the partial infeed at reversal point/reversal area
`Mode`:	Division of the complete infeed into partial infeeds = Two residual steps of equal size (default); = All partial infeeds of equal size

Motion-synchronous actions

`WHEN… … DO`	when…, do…
`WHENEVER … DO`	whenever…, do…

Example

No infeed must take place at reversal point 1. At reversal point 2, the infeed is to start at a distance of ii2 before reversal point 2 and the oscillating axis is not to wait at the reversal point for the end of the partial infeed. Axis Z is the oscillation axis and axis X the infeed axis.



1. Parameters for oscillation

Program code	Comment
DEF INT ii2	; Define variable for reversal area 2
OSP1[Z]=10 OSP2[Z]=60	; Define reversal points 1 and 2
OST1[Z]=0 OST2[Z]=0	; Reversal point 1: Exact stop fine
	Reversal point 2: Exact stop fine
FA[Z]=150 FA[X]=0.5	; Oscillating axis Z feedrate, infeed axis X feedrate
OSCTRL[Z]=(2+8+16.1)	; Deactivate oscillating motion at reversal point 2; after delete DTG spark-out and approach end position; after delete DTG approach reversal position
OSNC[Z]=3	; Sparking-out strokes
OSE[Z]=70	; End position = 70
ii2=2	; Set reversal point range
WAITP(Z)	; Enable oscillation for Z axis

2. Motion-synchronous action

Program code	Comment
WHENEVER $AA_IM[Z]<$SA_OSCILL_REVERSE_POS2[Z] DO -> $AA_OVR[X]=0 $AC_MARKER[0]=0	; If the actual position of oscillating axis Z in MCS is less than the start of reversal range 2, then always set the axial override of the infeed axis X to 0% and the bit memory with index 0 to the value 0.

Program code	Comment
WHENEVER $AA_IM[Z]>=$SA_OSCILL_REVERSE_POS2[Z] DO $AA_OVR[Z]=0	; If the actual position of the oscillating axis Z in MCS is greater than the reversal position 2, then always set the axial override of the oscillating axis Z to 0%.
WHENEVER $AA_DTEPW[X] == 0 DO $AC_MARKER[0]=1	; If the remaining distance to go of the partial infeed is 0, then always set the bit memory with index 0 to the value 1.
WHENEVER $AC_MARKER[0]==1 DO $AA_OVR[X]=0 $AA_OVR[Z]=100	; Whenever the bit memory with index 0 is equal to 1, then set the axial override of the infeed axis X to 0%. As a consequence, a premature infeed is prevented (oscillating axis Z has still not left reversal area 2, but infeed axis X is ready for a new infeed). Set the axial override of oscillating axis Z from 0% (action of the 2nd synchronized action) back to 100% to move.

-> must be programmed in a single block

3. Start oscillation

Program code	Comment
OSCILL[Z]=(X) POSP[X]=(5,1,1)	; Start the axes Oscillating axis Z is assigned axis X as infeed axis. Up to end position 5, axis X should travel in steps of 1.
M30	; End of program

Further information

1. **Define oscillation parameters**
   The parameters for oscillation should be defined before the movement block containing the assignment of infeed and oscillating axes and the infeed definition (see "Asynchronized oscillation").
2. **Define motion-synchronized actions**
   The following synchronization conditions can be defined:
   Suppress infeed until the oscillating axis is located within a reversal area (ii1, ii2) or at a reversal point (U1, U2).
   Stop oscillation motion during infeed at reversal point.
   Restart oscillation movement on completion of partial infeed. Define **start of next partial infeed**.
3. **Assign oscillating and infeed axes** as well as **partial and complete infeed**.

Define oscillation parameters

Assignment of oscillating and infeed axes: OSCILL

```
OSCILL[<oscillating axis>]=(<infeed axis1>,<infeed axis2>,<infeed
axis3>)
```

The axis assignments and the start of the oscillation movement are defined with the "OSCILL" command.

Up to 3 infeed axes can be assigned to an oscillating axis.

Note

Before oscillation starts, the synchronization conditions must be defined for the behavior of the axes.

Define infeeds: POSP

```
POSP[<infeed axis>]=(<end position>,<partial length>,<mode>)
```

The following are declared to the control with the "POSP" command:

- Complete infeed (with reference to end position)
- The length of the partial infeed at the reversal point or in the reversal area
- The partial infeed response when the end position is reached (with reference to mode)

Mode = 0	The distance-to-go to the destination point for the last two partial infeeds is divided into two equal steps (default setting).
Mode = 1	All partial infeeds are of equal size. They are calculated from the complete infeed.

Define motion-synchronized actions

The synchronized-motion actions listed below are used for general oscillation.

You are given example solutions for individual tasks, which you can use as modules for creating user-specific oscillation movements

Note

In individual cases, the synchronization conditions can be programmed differentially.

Keywords

`WHEN … DO …`	when…, do…
`WHENEVER … DO`	whenever…, do…

Functions

You can implement the following functions with the language resources described in detail below:

1. Infeed at reversal point.
2. Infeed at reversal area.
3. Infeed at both reversal points.
4. Stop oscillation movement at reversal point.
5. Restart oscillation movement.
6. Do not start partial infeed too early.

The following assumptions are made for all examples of synchronized actions presented here:

- Reversal point 1 < reversal point 2
- Z = oscillating axis
- X = infeed axis

Note

For more details, see the "Motion-synchronous actions" section.

Assign oscillating and infeed axes as well as partial and complete infeed

Infeed in reversal point range

The infeed motion must start within a reversal area before the reversal point is reached.

These synchronized actions inhibit the infeed movement until the oscillating axis is within the reversal area.

The following instructions are used subject to the above assumptions:

Reversal range 1:

```
WHENEVER $AA_IM[Z]>$SA_OSCILL_RESERVE_POS1[Z]+ii1 DO $AA_OVR[X] = 0
```

Whenever the actual position of the oscillating axis in the MCS is greater than the start of reversal range 1, then set the axial override of the infeed axis to 0%.

Reversal range 2:

```
WHENEVER $AA_IM[Z]<$SA_OSCILL_RESERVE_POS2[Z]+ii2 DO $AA_OVR[X] = 0
```

Whenever the actual position of the oscillating axis in the MCS is less than the start of reversal range 2, then set the axial override of the infeed axis to 0%.

Infeed at reversal point

As long as the oscillation axis has not reached the reversal point, the infeed axis does not move.

The following instructions are used subject to the above assumptions:

Reversal range 1:

```
WHENEVER $AA_IM[Z]<>$SA_OSCILL_RESERVE_POS1[Z] DO $AA_OVR[X]=0
$AA_OVR[Z]=100
```

Whenever the actual position of oscillating axis Z in MCS is greater or less than the position reversal point 1, then set the axial override of the infeed axis X to 0% and the axial override of the oscillating axis Z to 100%.

Reversal range 2:

For reversal point 2:

```
WHENEVER $AA_IM[Z]<>$SA_OSCILL_RESERVE_POS2[Z] DO $AA_OVR[X]=0
$AA_OVR[Z]=100
```

Whenever the actual position of oscillating axis Z in MCS is greater or less than the position reversal point 2, then set the axial override of the infeed axis X to 0% and the axial override of the oscillating axis Z to 100%.

Stop oscillation movement at the reversal point

The oscillation axis is stopped at the reversal point, the infeed motion starts at the same time. The oscillating motion is continued when the infeed movement is complete.

At the same time, this synchronized action can be used to start the infeed movement if this has been stopped by a previous synchronized action, which is still active.

The following instructions are used subject to the above assumptions:

Reversal range 1:

```
WHENEVER $SA_IM[Z]==$SA_OSCILL_RESERVE_POS1[Z] DO $AA_OVR[X]=0
$AA_OVR[Z]=100
```

Whenever the actual position of the oscillating axis in the MCS is the same as the reversal position 1, then set the axial override of the oscillating axis to 0% and the axial override of the infeed axis to 100%.

Reversal range 2:

```
WHENEVER $SA_IM[Z]==$SA_OSCILL_RESERVE_POS2[Z] DO $AA_OVR[X]=0
$AA_OVR[Z]=100
```

Whenever the actual position of the oscillating axis Z in the MCS is the same as the reversal position 2, then set the axial override of the oscillating axis X to 0% and the axial override of the infeed axis to 100%.

Online evaluation of reversal point

If there is a main run variable coded with `$$` on the right of the comparison, then the two variables are evaluated and compared with one another continuously in the IPO cycle.

Note

Please refer to Section "Motion-synchronized actions" for more information.

Oscillation movement restarting

The purpose of this synchronized action is to continue the movement of the oscillation axis on completion of the part infeed movement.

The following instructions are used subject to the above assumptions:

```
WHENEVER $AA_DTEPW[X]==0 DO $AA_OVR[Z]= 100
```

Whenever the remaining distance for the partial infeed of infeed axis X in the WCS is equal to zero, then set the axial override of the oscillating axis to 100%.

Next partial infeed

When infeed is complete, a premature start of the next partial infeed must be inhibited.

A channel-specific marker (`$AC_MARKER[Index]`) is used for this purpose. It is enabled at the end of the partial infeed (partial distance-to-go ≡ 0) and deleted when the axis leaves the reversal area. The next infeed movement is then prevented by a synchronized action.

On the basis of the given assumptions, the following instructions apply for reversal point 1:

1. Set marker:

```
WHENEVER $AA_DTEPW[X] == 0 DO $AC_MARKER[1]=1
```

Whenever the remaining distance for the partial infeed of infeed axis X in the WCS is equal to zero, then set the bit memory with index 1 to 1.

2. Delete marker

```
WHENEVER $AA_IM[Z]<> $SA_OSCILL_RESERVE_POS1[Z] DO $AC_MARKER[1] = 0
```

Whenever the actual position of oscillating axis Z in the MCS is greater or less than the position of reversal point 1, then set the bit memory 1 to 0.

3. Inhibit infeed

```
WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0
```

Whenever bit memory 1 is the same, then set the axial override of the infeed axis X to 0%.

Punching and nibbling 15

15.1 Activation, deactivation

15.1.1 Punching and nibbling on or off (SPOF, SON, PON, SONS, PONS, PDELAYON, PDELAYOF, PUNCHACC)

Activate/deactivate punching and nibbling

PON and SON are used to activate the punching and nibble functions. SPOF terminates all punching- and nibble-specific functions. Modal commands PON and SON are mutually exclusive, i.e. PON deactivates SON and vice versa.

Punching/nibbling with leader

The functions SONS and PONS also switch-in the punching or nibbling functions.

Unlike SON/PON (stroke control at interpolation level), with these functions, signal-related control of stroke initiation is at servo level. This enables higher stroke frequencies and, as a result, increased punching capacity to be achieved.

While signals are being evaluated in the leader, all functions that cause the nibbling or punching axes to change position (e.g. handwheel travel, changes to frames via PLC, measurement functions) are inhibited.

Punching with delay

PDELAYON results in a delayed output of the punching stroke. The modally effective command has a preparatory function and therefore is generally located before PON. Normal punching resumes after PDELAYOF.

Note

The delay time is set in setting data SD42400 $SC_PUNCH_DWELLTIME.

Travel-dependent acceleration

PUNCHACC can be used to specify an acceleration characteristic defining different rates of acceleration dependent on the hole spacing.

Second punching interface

Machines which need to use a second punching interface (second punching unit or comparable medium) alternately can be switched over to a second pair of fast digital inputs and outputs on the control (I/O pair). Full punching/nibbling functionality is available on both interfaces. The

SPIF1 and SPIF2 commands are used to switch over between the first and second punching interface.

Note

Requirement: A second I/O pair has to be defined for the punching functionality in the machine data (→ see machine manufacturer's specifications).

Syntax

```
PON G... X... Y... Z...
SON G... X... Y... Z...
SONS G... X... Y... Z...
PONS G... X... Y... Z...
PDELAYON
PDELAYOF
PUNCHACC(<Smin>,<Amin>,<Smax>,<Amax>)
SPIF1/SPIF2
SPOF
```

Meaning

PON:	Activate punching.	
SON:	Activate nibbling	
PONS:	Activate punching with leader.	
SONS:	Activate nibbling with leader.	
SPOF:	Deactivate punching/nibbling.	
PDELAYON:	Activate punching with delay.	
PDELAYOF:	Deactivate punching with delay.	
PUNCHACC:	Activate travel-dependent acceleration. Parameter:	
	<Smin>	Minimum hole spacing
	<Amin>	Initial acceleration <Amin> can be greater than <Amax>.
	<Smax>	Maximum hole spacing
	<Amax>	Final acceleration <Amax> can be greater than <Amin>.
SPIF1:	Activate **first** punching interface. The stroke is controlled using the first pair of fast I/O.	
SPIF2:	Activate **second** punching interface. The stroke is controlled using the second pair of fast I/O.	
	Note: The first punch interface is always active after a RESET or control system power up. If only one punching interface is used, then it need not be programmed.	

Examples

Example 1: Activate nibbling

Program code	Comment
...	
N70 X50 SPOF	; Position without punch initiation.
N80 X100 SON	; Activate nibbling, initiate a stroke before the motion (X=50) and on completion of the programmed movement (X=100).
...	

Example 2: Punching with delay

Program code	Comment
...	
N170 PDELAYON X100 SPOF	; Position without punch initiation, activate delayed punch initiation.
N180 X800 PON	; Activate punching. The punch stroke is output with a delay when the end position is reached.
N190 PDELAYOF X700	; Deactivate punching with delay, normal punch initiation on completion of the programmed movement.
...	

Example 3: Punching with two punching interfaces

Program code	Comment
...	
N170 SPIF1 X100 PON	; At the end of the block, a stroke is initiated at the first fast output. The "Stroke active" signal is monitored at the first input.
N180 X800 SPIF2	; The second stroke is initiated at the second fast output. The "Stroke active" signal is monitored at the second input.
N190 SPIF1 X700	; All further strokes are controlled with the first interface.
...	

Further information

Punching and nibbling with leader (PONS/SONS)

Punching and nibbling with leader is not possible in more than one channel simultaneously. PONS or SONS can only be activated in one channel at a time.

Travel-dependent acceleration (PUNCHACC)

Example:
PUNCHACC(2,50,10,100)

Distance between holes less than 2 mm:

The axis accelerates at a rate corresponding to 50% of maximum acceleration.

Distance between holes from 2 mm to 10 mm:

Acceleration is increased to 100%, proportional to the spacing.

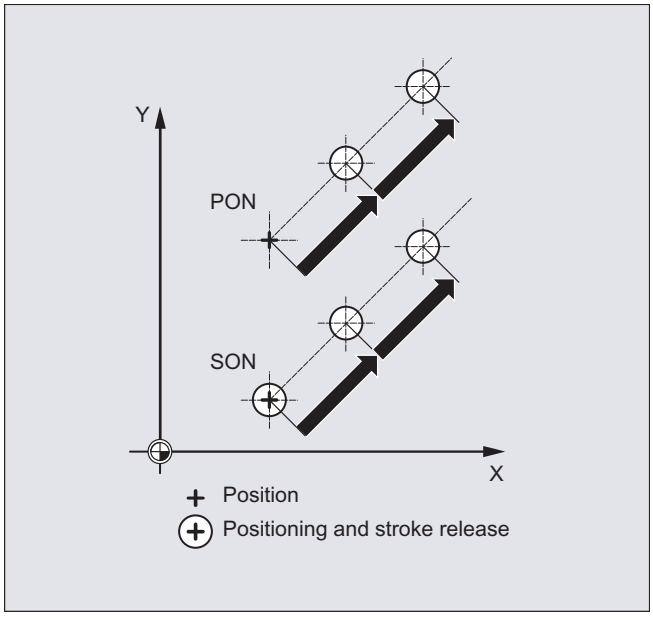
Distance between holes more than 10 mm:

Traverse at an acceleration of 100%.

Initiation of the first stroke

The instant at which the first stroke is initiated after activation of the function differs depending on whether nibbling or punching is selected:

- `PON/PONS`:
  - All strokes - even the one in the first block after activation - are executed at the block end.
- `SON/SONS`:
  - The first stroke after activation of the nibbling function is executed at the start of the block.
  - Each of the following strokes is initiated at the block end.



Punching and nibbling on the spot

A stroke is initiated only if the block contains traversing information for the punching or nibbling axes (axes in active plane).

However, to initiate a stroke at the same position, one of the punching/nibbling axes can be programmed with a traversing path of 0.

Machining with rotatable tools

Note

Use the tangential control function if you wish to position rotatable tools at a tangent to the programmed path.

Use of M commands

As in earlier versions, macro technology allows special M functions to be used instead of language commands (compatibility). The M functions and equivalent language commands as used in earlier systems are as follows:

```
M20, M23   ≙  SPOF
M22        ≙  SON
M25        ≙  PON
M26        ≙  PDELAYON
```

Example for macro file:

Program code	Comment
DEFINE M25 AS PON	; Punching on
DEFINE M125 AS PONS	; Punching with leader on
DEFINE M22 AS SON	; Nibbling on
DEFINE M122 AS SONS	; Nibbling with leader on
DEFINE M26 AS PDELAYON	; Punching with delay on
DEFINE M20 AS SPOF	; Punching, nibbling off
DEFINE M23 AS SPOF	; Punching, nibbling off

Programming example:

Program code	Comment
...	
N100 X100 M20	; Position without punch initiation.
N110 X120 M22	; Activate nibbling, initiate stroke before and after motion.
N120 X150 Y150 M25	; Activate punching, initiate stroke at end of motion.
...	

15.2 Automatic path segmentation

Segmentation into path segments

When punching or nibbling is activated, both SPP and SPN segment the total traversing section programmed for the path axes into a number of path segments with the same length (equidistant path segmentation). Internally, each path segment corresponds to a block.

Number of strokes

When punching, the first stroke is realized at the end point of the first path segment; but for nibbling, at the starting point of the first path segment. Therefore the following numbers are obtained over the complete traversing section:

Punching: Number of strokes = number of path segments

Nibbling: Number of strokes = number of path segments +1

Auxiliary functions

Auxiliary functions are executed in the first of the generated blocks.

Syntax

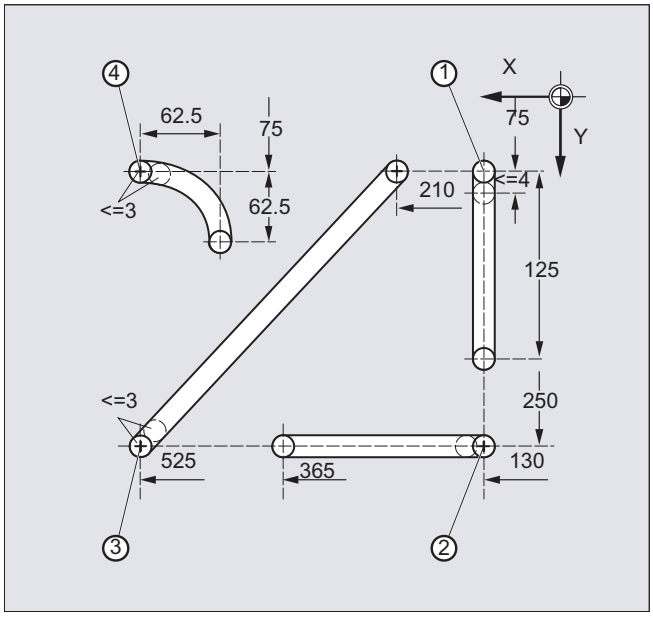
```
SPP=
SPN=
```

Meaning

SPP:	Size of path segment (maximum distance between strokes); modal
SPN:	Number of path segments per block; modally effective

Example 1

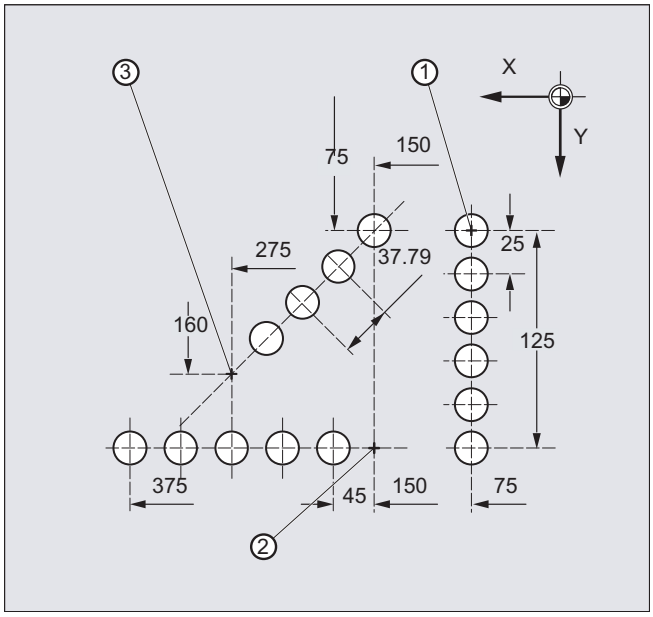
The programmed nibbling segments should be automatically split-up into path segments.



Program code	Comment
N100 G90 X130 Y75 F60 SPOF	; Positioning at starting point 1
N110 G91 Y125 SPP=4 SON	; Nibbling on; maximum path segment length for automatic path segmentation: 4 mm
N120 G90 Y250 SPOF	; Nibbling off; positioning to starting point 2
N130 X365 SON	; Nibbling on; maximum path segment length for automatic path segmentation: 4 mm
N140 X525 SPOF	; Nibbling off; positioning to starting point 3
N150 X210 Y75 SPP=3 SON	; Nibbling on; maximum path segment length for automatic path segmentation: 3 mm
N160 X525 SPOF	; Nibbling off; positioning to starting point 4
N170 G02 X-62.5 Y62.5 I J62.5 SPP=3 SON	; Nibbling on; maximum path segment length for automatic path segmentation: 3 mm
N180 G00 G90 Y300 SPOF	; Nibbling off

Example 2

Automatic path segmentation should be made for the individual series of holes. The maximum path segment length (SPP value) is specified for the segmentation.



Program code	Comment
N100 G90 X75 Y75 F60 PON	; Position to starting point 1; punch a single hole
N110 G91 Y125 SPP=25	; Maximum path segment length for automatic path segmentation: 25 mm
N120 G90 X150 SPOF	; Punching off; positioning to starting point 2
N130 X375 SPP=45 PON	; Punching on; maximum path segment length for automatic path segmentation: 45 mm
N140 X275 Y160 SPOF	; Punching off; positioning to starting point 3
N150 X150 Y75 SPP=40 PON	; Punching on, instead of the programmed path segment length of 40 mm, the calculated path segment length of 37.79 mm is used.
N160 G00 Y300 SPOF	; Punching off; positioning

15.2.1 Path segmentation for path axes

Length of SPP path segment

`SPP` is used to specify the maximum distance between strokes and thus the maximum length of the path segments in which the total traversing distance is to be divided. The command is deactivated with `SPOF` or `SPP=0`.

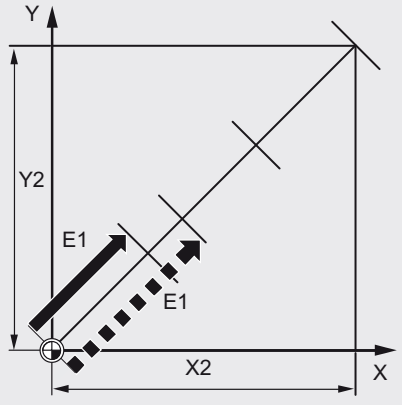
Example:

```
N10 SON X0 Y0
N20 SPP=2 X10
```

The total traversing distance of 10 mm will be divided into five path sections each of 2 mm (SPP=2).

Note

The path segments effected by SPP are always equidistant, i.e. all segments are equal in length. In other words, the programmed path segment size (SPP setting) is valid only if the quotient of the total traversing distance and the SPP value is an integer. If this is not the case, the size of the path segment is reduced internally such as to produce an integer quotient.



X2/Y2 Programmed traverse path (nibbling or punching block)
E1 Programmed path segment length
E1' Automatically rounded path segment length

Example:

```
N10 G1 G91 SON X10 Y10
N20 SPP=3.5 X15 Y15
```

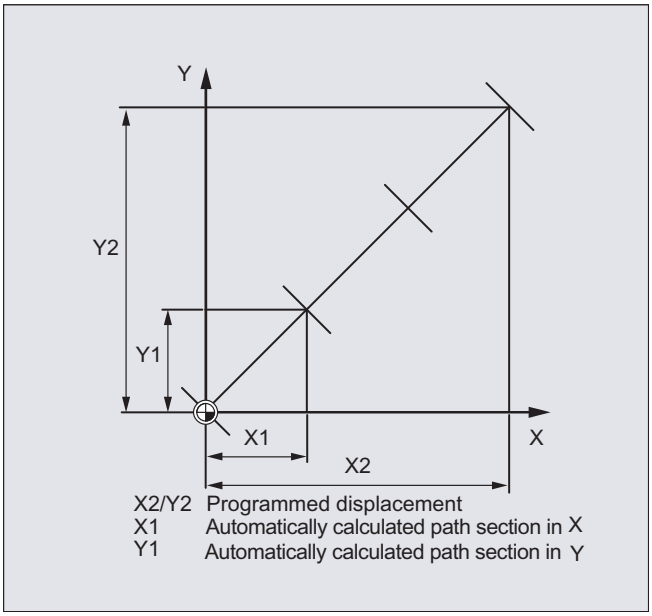
When the total traversing distance is 15 mm and the path segment length 3.5 mm, the quotient is not an integer value (4.28). In this case, the SPP value is reduced down to the next possible integer quotient. The result in this example would be a path segment length of 3 mm.

Number of SPN path segments

SPN defines the number of path segments to be generated from the total traversing distance. The length of the segments is calculated automatically. Since SPN is non-modal, punching or nibbling must be activated beforehand with PON or SON respectively.

SPP and SPN in the same block

If you program both the path segment length (SPP) and the number of path segments (SPN) in the same block, then SPN applies to this block and SPP to all the following blocks. If SPP was activated before SPN, then it takes effect again after the block with SPN.



Note

Provided that punching/nibbling functions are available in the controller, then it is possible to program the automatic path segmentation function with SPN or SPP even independent of this technology.

15.2.2 Path segmentation for single axes

If single axes are defined as punching/nibbling axes in addition to path axes, then the automatic path segmentation function can be activated for them.

Response of single axis to SPP

The programmed path segment length (SPP) basically refers to the path axes. For this reason, the SPP value is ignored in blocks which contain a single axis motion and an SPP value, but not a programmed path axis.

If both a single axis and a path axis are programmed in the block, then the single axis responds according to the setting of the appropriate machine data.

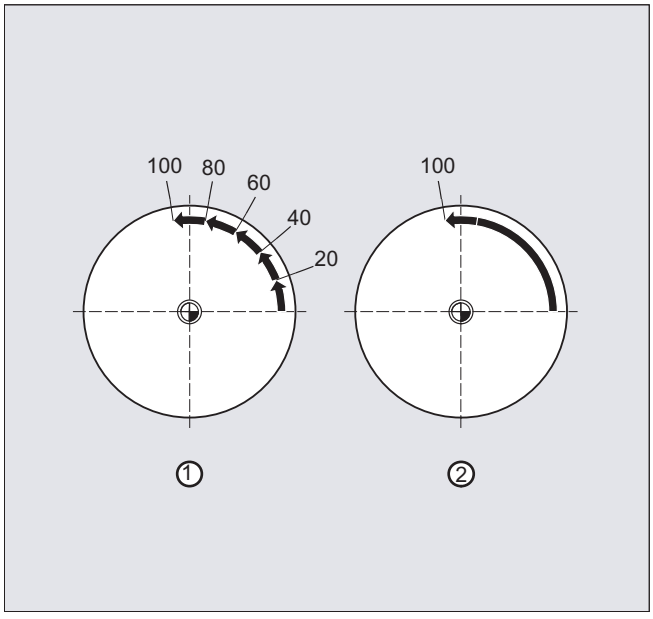
1. Standard setting
   The path traversed by the single axis is distributed evenly among the intermediate blocks generated by SPP.

Example:

```
N10 G1 SON X10 A0
N20 SPP=3 X25 A100
```

As a result of the programmed distance between strokes of 3 mm, five blocks are generated for the total traversing distance of the X axis (path axis) of 15 mm.

The A axis thus rotates through 20° in every block.



1. Single axis without path segmentation
   The single axis traverses the total distance in the first of the generated blocks.
2. With/without path segmentation
   The response of the single axis depends on the interpolation of the path axes:

- Circular interpolation: Path segmentation
- Linear interpolation: No path segmentation

Response to SPN

The programmed number of path segments is applicable even if a path axis is not programmed in the same block.

Requirement: The single axis is defined as a punching/nibbling axis.

Grinding 16

16.1 Activate/deactivate grinding-specific tool monitoring (TMON, TMOF)

With the predefined procedures TMON(...) and TMOF(...), the grinding-specific tool monitoring is activated or deactivated (geometry and speed monitoring).

Requirement

The tool-specific parameters $TC_TPG1 to $TC_TPG9 must be set.

Syntax

```
TMON(<TNo>)
...
TMOF(<TNo>)
```

Meaning

`TMON(...)`:	Activate grinding-specific tool monitoring The command must be programmed in the channel in which the grinding-specific tool monitoring is to be activated.
`TMOF(...)`:	Deactivate grinding-specific tool monitoring The command must be programmed in the channel in which the grinding-specific tool monitoring is to be deactivated.
`<TNo>`:	T number **Note:** Only required if the monitoring is to be switched on or off for an inactive grinding wheel rather than the active grinding wheel that is currently in use.
`TMOF(0)`:	Deactivate monitoring for all tools

Additional functions 17

17.1 Axis functions (AXNAME, AX, SPI, AXTOSPI, ISAXIS, AXSTRING, MODAXVAL)

"AXNAME" is used, e.g. to generate cycles that are generally valid, if the names of the axes are not known.

"AX" is used to indirectly program geometry and synchronous axes. The axis identifier is saved in a type AXIS variable or is supplied from a command such as "AXNAME" or "SPI".

"SPI" is used if axis functions are programmed for a spindle, e.g. a synchronous spindle.

"AXTOSPI" is used to convert an axis identifier into a spindle index (inverse function to "SPI").

"AXSTRING" is used to convert an axis identifier (data type AXIS) into a string (inverse function to "AXNAME").

"ISAXIS" is used in universal cycles in order to ensure that a specific geometry axis exists and thus that any following $P_AXNX call is not aborted with an error message.

"MODAXVAL" is used in order to determine the modulo position for modulo rotary axes.

Syntax

```
AXNAME("string")
AX[AXNAME("string")]
SPI(n)
```

`AXTOSPI(A)` or `AXTOSPI(B)` or `AXTOSPI(C)`

```
AXSTRING( SPI(n) )
ISAXIS(<geometry axis number>)
<Modulo position>=MODAXVAL(<axis>,<axis position>)
```

Meaning

`AXNAME`:	Converts an input string into axis identifiers; the input string must contain a valid axis name.
`AX`:	Variable axis identifier
`SPI`:	Converts the spindle number into an axis identifier; the transfer parameter must contain a valid spindle number.
`n`:	Spindle number
`AXTOSPI`:	Converts an axis identifier into an integer spindle index. "AXTOSPI" corresponds to the inverse function to "SPI".
`X, Y, Z`:	Axis identifier of AXIS type as variable or constant
`AXSTRING`:	The string is output with the associated spindle number.

ISAXIS:	Checks whether the specified geometry axis exists.
MODAXVAL:	For modulo rotary axes, determines the modulo position; this corresponds to the modulo rest referred to the parameterized modulo range (in the default setting, this is 0 to 360 degrees; the start and size of the modulo range can be changed using MD30340 MODULO_RANGE_START and MD30330 $MA_MODULO_RANGE).

Note

SPI extensions

The axis function SPI(n) can also be used to read and write frame components. This means that frames can be written, e.g. with the syntax `$P_PFRAME[SPI(1),TR]=2.22`.

An axis can be traversed by additionally programming axis positions using the address `AX[SPI(1)]=<axis position>`. The prerequisite is that the spindle is either in the positioning or axis mode.

Examples

Example 1: AXNAME, AX, ISAXIS

Program code	Comment
`OVRA[AXNAME("Transverse axis")]=10`	`; Override for transverse axis`
`AX[AXNAME("Transverse axis")]=50.2`	`; End position for transverse axis`
`OVRA[SPI(1)]=70`	`; Override for spindle 1`
`AX[SPI(1)]=180`	`; End position for spindle 1`
`IF ISAXIS(1) == FALSE GOTOF CONTINUE`	`; Abscissa available?`
`AX[$P_AXN1]=100`	`; Move abscissa`
`CONTINUE:`	

Example 2: AXSTRING

When programming with AXSTRING[SPI(n)], the axis index of the axis, which is assigned to the spindle, is no longer output as spindle number, but instead the string "Sn" is output.

Program code	Comment
`AXSTRING[SPI(2)]`	`; String "S2" is output.`

Example 3: MODAXVAL

The modulo position of modulo rotary axis A is to be determined.

Axis position 372.55 is the starting value for the calculation.

The parameterized modulo range is 0 to 360 degrees:

MD30340 MODULO_RANGE_START = 0

MD30330 $MA_MODULO_RANGE = 360

Program code	Comment
`R10=MODAXVAL(A,372.55)`	`; Calculated modulo position R10 = 12.55.`

Example 4: MODAXVAL

If the programmed axis identifier does not refer to a modulo rotary axis, then the value to be converted (<axis position>) is returned unchanged.

Program code	Comment
R11=MODAXVAL(X,372.55)	; X is a linear axis; R11 = 372.55.

17.2 Replaceable geometry axes (GEOAX)

The "Switchable geometry axes" function allows the geometry axes configured via machine data to be replaced by other channel axes.

Syntax

```
GEOAX(<n>,<channel axis>,<n>,<channel axis>,<n>,<channel axis>)
GEOAX()
```

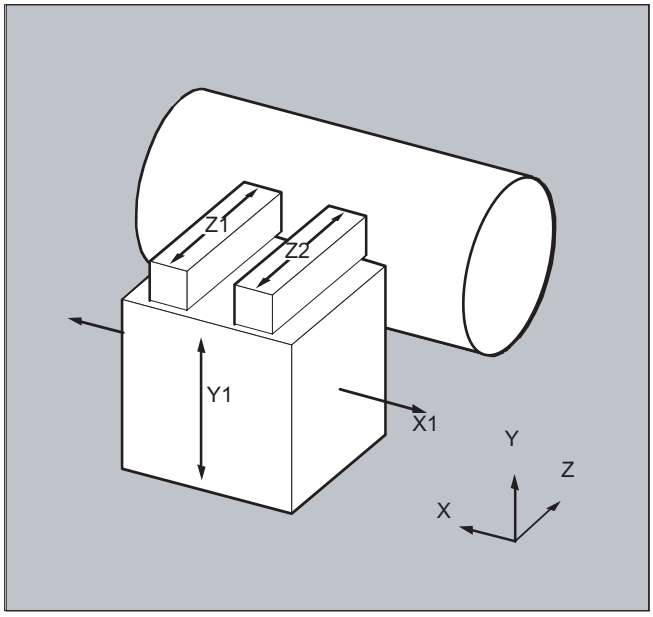
Meaning

GEOAX(...)	Function for switching geometry axes. **Note:** GEOAX() without parameter specification activates the basic configuration of the geometry axes parameterized in the machine data again.
<n>	Number of the geometry axis that is to be replaced by the specified channel axis. Range of values: 0, 1, 2, 3 **Note:** 0: The specified channel axis is removed from the geometry axis group without being replaced 1: 1. geometry axis ≙ coordinate axis X (abscissa) of the WCS 2: 2. geometry axis ≙ coordinate axis Y (ordinate) of the WCS 3: 3. geometry axis ≙ coordinate axis Z (applicate) of the WCS
<channel axis>	Name of the channel axis which is to added to the geometry axis group

Examples

Example 1: Switching two axes alternating as geometry axis

A tool slide can be traversed using channel axes X1, Y1, Z1, Z2:



The geometry axes are configured so that after powering-up, initially Z1 is effective as 3rd geometry axis under the geometry axis name "Z" and together with X1 and Y1 forms the geometry axis group.

Axes Z1 and Z2 should now be used, alternating, as geometry axis Z in the part program:

Program code	Comment
...	
N100 GEOAX(3,Z2)	; Channel axis Z2 acts as 3rd geometry axis (Z).
N110 G1 ...	
N120 GEOAX(3,Z1)	; Channel axis Z1 acts as 3rd geometry axis (Z).
...	

Example 2: Changing over the geometry axes for six channel axes

A machine has six channel axes with the names XX, YY, ZZ, U, V, W.

The basic setting of the geometry axis configuration via machine data is:

Channel axis XX = 1st geometry axis (X axis)

Channel axis YY = 2nd geometry axis (Y axis)

Channel axis ZZ = 3rd geometry axis (Z axis)

Program code	Comment
N10 GEOAX()	; The basic configuration of the geometry axes is effective.
N20 G0 X0 Y0 Z0 U0 V0 W0	; All axes in rapid traverse to position 0.
N30 GEOAX(1,U,2,V,3,W)	; Channel axis U becomes the first (X), V the second (Y)
	; and W the third geometry axis (Z).

Program code	Comment
N40 GEOAX(1,XX,3,ZZ)	; Channel axis XX becomes the first (X), ZZ the third ; geometry axis (Z). Channel axis V remains the second ; geometry axis (Y).
N50 G17 G2 X20 I10 F1000	; Full circle in the X/Y plane. Channel axes ; XX and V traverse.
N60 GEOAX(2,W)	; Channel axis W becomes the second geometry (Y).
N80 G17 G2 X20 I10 F1000	; Full circle in the X/Y plane. Channel axes ; XX and W traverse.
N90 GEOAX()	; Reset to the initial state.
N100 GEOAX(1,U,2,V,3,W)	; Channel axis U becomes the first (X), V the second ; (Y) and W the third geometry axis (Z).
N110 G1 X10 Y10 Z10 XX=25	; Channel axes U, V, W each traverse to ; position 10. XX as special axis traverses to position 25.
N120 GEOAX(0,V)	; V is removed from the geometry axis group. ; U and W remain the first (X) and third ; geometry axis (Z). ; The second geometry (Y) axis remains unassigned.
N130 GEOAX(1,U,2,V,3,W)	; Channel axis U remains the first (X), V becomes ; the second (Y), W remains the third geometry axis (Z).
N140 GEOAX(3,V)	; V becomes the third geometry axis (Z), whereby W ; is overwritten and therefore removed from the geometry ; axis group. The second geometry axis (Y) ; still remains unassigned.

Machine data

Axis configuration

Assignment of geometry, special and machine axes to channel axes:

- MD10000 $MN_AXCONF_MACHAX_NAME_TAB
- MD20050 $MC_AXCONF_GEOAX_ASIGN_TAB
- MD20060 $MC_AXCONF_GEOAX_NAME_TAB
- MD20070 $MC_AXCONF_MACHAX_USED
- MD20080 $MC_AXCONF_CHANAX_NAME_TAB
- MD35000 $MA_SPIND_ASSIGN_TO_MACHAX

Reset behavior

Reset behavior of changed geometry axis assignments:

- MD20110 $MC_RESET_MODE_MASK, bit 12
- MD20118 $MC_GEOAX_CHANGE_RESET

NC start behavior

- MD20112 $MC_START_MODE_MASK, bit 12

Notification to the PLC user program

Parameterization option of the M command which is output on the NC/PLC interface when the geometry axes are changed.

- MD22532 $MC_GEOAX_CHANGE_M_CODE

Supplementary conditions

No geometry axis changeover

- If one of the following functions is active, a geometry axis changeover is not possible:
  - Transformation
  - Spline interpolation
  - Tool radius compensation
  - Tool fine offset
- The geometry axis and another channel axis have the same name.
- One of the axes participating in the geometry axis changeover is involved in an action that goes beyond block limits, e.g. block-wide positioning axis or following axis of an axis coupling.

Rotary axes

Rotary axes cannot be programmed as geometry axes.

Axis state after replacing

An axis replaced by the changeover in the geometry axis group can be programmed as supplementary axis after the changeover operation via its channel axis names.

Frames, protection areas, working area limits

All frames, protection areas and working area limits are deleted after changing over the geometry axes.

Polar coordinates

Replacing the geometry axes with `GEOAX` sets analog to a level change with `G17`-`G19`, the modal polar coordinates to a value of 0.

DRF, WO

A possible handwheel offset (DRF) or an external work offset (WO) remains effective after the changeover.

Basic configuration of the geometry axes

The `GEOAX()` command calls the basic configuration of the geometry axis group.

The system automatically changes back to the basic configuration after POWER ON and when changing over into the "reference point approach" mode.

Tool length compensation

An active tool length compensation is also effective after the changeover operation. However, for geometry axes that have been newly added or those where the position has been replaced, it is still considered not to have been moved through. For the first motion command for these geometry axes, the resulting traversing distance correspondingly comprises the sum of the tool length compensation and the programmed traversing distance.

Geometry axes, which retain their position in the axis group after a replacement operation, also retain their status with respect to tool length compensation.

Geometry axis configuration for active transformation

- The geometry axis configuration parameterized for an active transformation via transformation machine data cannot be changed using the "Switchable geometry axes" function.
- Different data sets must be parameterized in the transformation machine data for a different geometry axis configuration for a transformation.
- A geometry axis configuration changed using `GEOAX` is deleted by activating a transformation.
- With regard to the geometry axes, the transformation-specific geometry axis parameterizations of active transformations have priority over the parameterizations relevant for the changeover of geometry axes.
  Example: A transformation is active. According to the machine data, the transformation should be retained at a channel reset. At the same time, the basic configuration of the geometry axes should be restored at a channel reset. The geometry axis configuration that has been specified for the transformation is retained.
- If a transformation is switched off, the parameterized basic setting of the geometry axis configuration takes effect again.

JOG mode, REF machine function

When switching over to the JOG mode, REF machine function (reference point approach), the geometry axis configuration parameterized in the machine data takes effect

17.3 Axis container (AXCTSWE, AXCTSWED, AXCTSWEC)

The "AXCTSWE" or "AXCTSWED" commands enable the rotation of the specified axis container.

Any previously set enable for axis container rotation is cancelled with the "AXCTSWEC" command.

Syntax

```
AXCTSWE(<ID>)
AXCTSWED(<ID>)
AXCTSWEC(<ID>)
```

Meaning

AXCTSWE:	Enable for rotation of the axis container The program processing is not stopped by "AXCTSWE". The rotation is performed as soon as all channels involved on the axis container have been enabled.	
AXCTSWED:	Enable to rotate the axis container without consideration of the other channels involved on the axis container **Note** • Command variant to simplify the commissioning of the part program or synchronized action. • The behavior with regard to the other channels involved on axis container can be specified via: MD12760 $MN_ AXCT_FUNCTION_MASK, bit 0	
AXCTSWEC:	Canceling the enable to rotate the axis container **Note** The enable for rotating an axis container can only be cancelled when the rotation has yet not been started: $AN_AXCTSWA[<axis container>] == 0 For system variable, see "Axis container (AXCTSWE, AXCTSWED, AXCTSWEC) (Page 555)"	
<ID>:	Identifier of the axis container or a container axis:	
	CT<number>:	Default identifier of an axis container: MD12750 $MN_AXCT_NAME_TAB Example: "CT1"
	<Container>:	User-specific identifier of an axis container: MD12750 $MN_AXCT_NAME_TAB Example: "CONTAINER_1"
	<axis>:	Identifier of a known container axis in the channel

Note

Increment

The increment of a axis container rotation is set via the setting data:

SD41700 $SN_AXCT_SWWIDTH

Further information

Diagnostics

The current status of a axis container can be read via the following system variables:

System variable	Type	Description
$AC_AXCTSWA[<name>]	BOOL	Channel-specific status of the axis container
$AN_AXCTSWA[<axis container>]	BOOL	NCU-specific status of the axis container

System variable	Type	Description
$AN_AXCTSWE[<axis container>]	INT	Slot-specific status of the axis container rotation The system variable supplies the status of the axis container slot **bitwise**. Each bit corresponds to a slot.
$AN_AXCTAS[<axis container>]	INT	Number of locations (slots) through which the axis container was just switched through.

Axis container rotation with implicit GET / GETD

The following machine data can be use to set that all container axes of the channel are brought into the channel by means of an implicit "GET/GETD" with the "AXCTSWE" command. An axis replacement is only possible again after the container rotation.

MD10722 $MN_AXCHANGE_MASK, bit 1 = 1

Note

An axis container rotation with implicit "GET/GETD" is **not** performed for an axis in the state "main run axis" (e.g. PLC axis) because the axis would have to exit the state for the axis container rotation.

17.4 Wait for valid axis position (WAITENC)

Using the language command "WAITENC", the NC program waits until the synchronized or restored axis positions are available for the axes configured with MD34800 $MA_WAIT_ENC_VALID = 1.

An interruption can take place in the wait state, e.g. by starting an ASUB or by changing the operating mode to JOG. When the program is continued, where relevant, the wait state is resumed.

Note

In the user interface, the wait state is displayed using the hold state "Wait for measuring system".

Syntax

"WAITENC" can be programmed in the program section of any NC program.

Programming must be realized in a dedicated block:

```
...
WAITENC
...
```

Example

"WAITENC" is for example used in an event-controlled user program, .../_N_CMA_DIR/_N_PROG_EVENT_SPF, as shown in the following application example.

Application example: Tool withdrawal after POWER OFF with orientation transformation

Machining with tool orientation was interrupted due to a power failure.
When powering up again, the event-controlled user program .../_N_CMA_DIR/_N_PROG_EVENT_SPF is called.

In the event-controlled user program, the system waits for synchronized or restored axis positions using "WAITENC"; in order to then be able to calculate a frame, which aligns the Work in the tool direction.

```
Program code                        Comment
...
IF $P_PROG_EVENT == 4               ; Run-up.
    IF $P_TRAFO <> 0                ; Transformation has been selected.
        WAITENC                     ; Wait for valid axis positions of the orientation
                                    axes.
        TOROTZ                      ; Rotate the Z axis of the WCS towards the tool axis.
    ENDIF
    M17
ENDIF
...
```

The tool can then be retracted in JOG mode by means of a retraction movement towards the tool axis.

17.5 Programmable parameter set changeover (SCPARA)

The changeover to a specific parameter set can be requested for an axis with the SCPARA command.

Note

No parameter set changeover during thread cutting

During thread cutting G33 and tapping G331/G332, the parameter set is selected by the control and cannot be changed.

Disabled parameter set changeover

A parameter set changeover can also be requested via the NC/PLC interface. In order to avoid changeover conflicts, the parameter set changeover of the NC (SCPARA) can be disabled via the NC/PLC interface:

DB31, ... DBX9.3 (parameter set specification disabled by NC)

Note

If a parameter set changeover is requested by SCPARA while the parameter set changeover is disabled via the NC/PLC interface, the changeover is rejected without an error message.

Syntax

```
SCPARA[<axis>]=<value>
```

Meaning

SCPARA:	Command: Change parameter set	
<axis>:	Axis identifier (channel axis)	
	Type:	AXIS
<value>:	Parameter set number: 1, 2, 3, ... max. parameter set number	

Example

Program code	Comment
...	
N110 SCPARA[X]= 3	; Select: Axis X, 3rd parameter set
...	

Further information

Enable of the parameter set changeover

The parameter set changeover of the axis must be explicitly enabled:

MD35590 $MA_PARAMSET_CHANGE_ENABLE[<axis>]

Read parameter set number

The number of the selected parameter set (specified parameter set) can be read via the system variable $AA_SCPAR.

References

Detailed information on the parameter sets can be found in:

Function Manual, Basic Functions; Section "Velocities, setpoint / actual value systems, closed-loop control (G2)" > "Closed-loop control" > "Parameter sets of the position controller"

17.6 Check scope of NC language present (STRINGIS)

Using the function "STRINGIS(...)" it can be checked as to whether the specified string is available as element of the NC programming language in the actual language scope.

Definition

```
INT STRINGIS(STRING <Name>)
```

Syntax

```
STRINGIS(<Name>)
```

Meaning

STRINGIS:	Function with return value
<name>:	Name of the NC programming language element to be checked
Return value:	The return value format is yxx (decimal).

Elements of the NC programming language

The following elements of the NC programming language can be checked:

- G codes of all existing G function groups, e.g. "G0", "INVCW", "POLY", "ROT", "KONT", "SOFT", "CUT2D", "CDON", "RMBBL", "SPATH"
- DIN or NC addresses, such as "ADIS", "RNDM", "SPN", "SR", "MEAS"
- Functions, e.g. "TANG(...)" or "GETMDACT"
- Procedures, e.g. "SBLOF".
- Keywords, e.g. "ACN", "DEFINE" or "SETMS"
- System data, e.g. machine data $M... , setting data $S... or option data $O...
- System variables $A... , $V... , $P...
- Arithmetic parameter R...
- Cycle names of activated cycles
- GUD and LUD variables
- Macro names
- Label names

Return value

Only the first three decimal positions of the return value are relevant. The return value format is yxx, with y = basis information and xx = detailed information.

Return value	Meaning	
000	The 'name' string is not known in this system [1)]	
100	The 'name' string is an element of the NC programming language, but currently cannot be programmed (option/function is inactive)	
2xx	The 'name' string is a programmable element of the NC programming language (option/function is active). The detailed information xx contains additional information about the element type:	
	xx	Meaning
	01	DIN address or NC address[2)]
	02	G code, e.g. G04, INVCW

Return value	Meaning	
	03	Function with return value
	04	Function without return value
	05	Keyword, e.g. DEFINE
	06	Machine ($M...), setting ($S...) or option data ($O...)
	07	System parameters, e.g. system variable ($...) or arithmetic parameter (R...)
	08	Cycle (the cycle must be loaded into the NCK and the cycle program must be active [3])
	09	GUD variable (the GUD variable must be defined in the GUD definition files and the GUD variables activated)
	10	Macro name (the macro must be defined in the macro definition files and macros activated) [4]
	11	LUD variable of the actual part program
	12	ISO G code (ISO language mode must be active)
400	The 'name' string is an NC address, that was not identified as xx == 01 or xx == 10 and is not G or R [2]	
y00	No specific assignment possible	

1) Depending on the control, under certain circumstances, only a subset of the Siemens NC language commands are known, e.g. SINUMERIK 802D sl. For these controls, for strings that are principally Siemens NC language commands, a value of 0 is returned. This behavior can be changed using MD10711 $MN_NC_LANGUAGE_CONFIGURATION. For MD10711 = 1, then a value of 100 is always returned for Siemens NC language commands.

2) NC addresses are the following letters: A, B, C, E, I, J, K, Q, U, V, W, X, Y, Z. These NC addresses can also be programmed with an address extension. The address extension can be specified when checking with STRINGIS. Example: 201 == STRINGIS("A1").
The letters: D, F, H, L, M, N, O, P, S, T are NC addresses or auxiliary functions that are defined by the user. A value of 400 is always returned for these. Example: 400 == STRINGIS("D"). These NC addresses cannot be specified with address extension when checking with STRINGIS.
Example: 000 == STRINGIS("M02"), but 400 == STRINGIS("M").

3) Names of cycle parameters cannot be checked with STRINGIS.
4) Address, defined as macro, e.g. G, H, M, L are identified as macro.

Examples

In the following examples it is assumed that the NC language elements specified as string - as long as nothing else is noted - can in principle be programmed in the control.

1. String "T" is defined as auxiliary function:
```
400 == STRINGIS("T")
000 == STRINGIS ("T3")
```
2. String "X" is defined as axis:
```
201 == STRINGIS("X")
201 == STRINGIS("X1")
```
3. String "A2" is defined as address with extension:
```
201 == STRINGIS("A")
201 == STRINGIS("A2")
```
4. String "INVCW" is defined as named G code:
```
202 == STRINGIS("INVCW")
```
5. String "$MC_GCODE_RESET_VALUES" is defined as machine data:
```
206 == STRINGIS("$MC_GCODE_RESET_VALUES")
```
6. String "GETMDACT" is an NC language function:
```
203 == STRINGIS("GETMDACT ")
```

7. String "DEFINE" is a keyword:
   `205 == STRINGIS("DEFINE")`
8. String "$TC_DP3" is a system parameter (tool length component):
   `207 == STRINGIS("$TC_DP3")`
9. String "$TC_TP4" is a system parameter (tool size):
   `207 == STRINGIS("$TC_TP4")`
10. String "$TC_MPP4" is a system parameter (magazine location state):
    - Tool magazine management is active: `207 == STRINGIS("$TC_MPP4") ;`
    - Tool magazine management is not active: `000 == STRINGIS("$TC_MPP4")`

    Also refer to the paragraph below: Tool magazine management.
11. String "MACHINERY_NAME" is defined as GUD variable:
    `209 == STRINGIS("MACHINERY_NAME")`
12. String "LONGMACRO" is defined as macro:
    `210 == STRINGIS("LONGMACRO")`
13. String "MYVAR" is defined as LUD variable:
    `211 == STRINGIS("MYVAR")`
14. String "XYZ" is a command that is not known in the NCK, GUD variable, macro or cycle name:
    `000 == STRINGIS("XYZ")`

Tool magazine management

If the tool magazine management function is not active, supplies STRINGIS for the system parameters of the tool magazine management, independent of the machine data

- MD10711 $MN_NC_LANGUAGE_CONFIGURATION

always a value of 000.

ISO mode

If the "ISO mode" function is active:

- MD18800 $MN_MM_EXTERN_LANGUAGE (activation, external NC languages)
- MD10880 $MN_ MM_EXTERN_CNC_SYSTEM (control system to be adapted)

STRINGIS checks the specified string initially as SINUMERIK G code. If the string is not a SINUMERIK G code, then it is subsequently checked as ISO G code.

Programmed switchovers (`G290` (SINUMERIK mode), `G291` (ISO Mode)) have no effect on STRINGIS.

Example

The machine data, relevant for the function STRINGIS(...), has the following values:

- MD10711 $MN_NC_LANGUAGE_CONFIGURATION = 2 (only the NC language commands whose options are set are considered to be known)
- MD19410 $ON_TRAFO_TYPE_MASK = 'H0' (option: transformations)
- MD10700 $MN_PREPROCESSING_LEVEL='H43' (preprocessing for cycles is active)

The following program example is executed without error message:

Program code	Comment
N1 R1=STRINGIS("TRACYL")	; R1 == 0, because TRACYL is identified as "not known" because of the missing transformation option
N2 IF STRINGIS("TRACYL") == 204	
N3 TRACYL(1,2,3)	; N3 is skipped
N4 ELSE	
N5 G00	; and instead, N5 is executed
N6 ENDIF	
N7 M30	

17.7 Interactively call the window from the part program (MMC)

You can use the "MMC" command to display user-defined dialog windows (dialog displays) on the HMI from the parts program.

The dialog window appearance is defined in a pure text configuration (COM file in cycles directory), while the HMI system software remains unchanged.

User-defined dialog windows cannot be called simultaneously in different channels.

Syntax

```
MMC(<command>,<acknowledgement mode>)
```

Meaning

MMC:	Subprogram identifier		
<command>:	Parameter of the STRING type		
	Contains the MMC command, e.g. in the following form: "CYCLES,PICTURE_ON,T_SK.COM,PICTURE,MGUD.DEF,PICTURE_3.AWB,TEST_1,A1"		
	CYCLES:		Operating area in which the configured user dialog boxes are implemented.
	PICTURE_ON: or PICTURE_OFF:		Command: Display selection or display deselection.
	T_SK.COM:		Com file: Name of the dialog display file (user cycles). The dialog display design is defined here. The dialog screen is used to display user variables and/or comment texts.
	PICTURE:		Name of dialog display: The individual displays are selected via the names of the dialog displays.
	MGUD.DEF:		User data definition file, which is addressed while reading/writing variables.
	PICTURE_3.AWB:		Graphics file
	TEST_1:		Display time or acknowledgement variable.
	A1:		Text variables...",
<acknowledgement mode>:	Parameters of the type CHAR		
	Value:	"N":	Without acknowledgement Program execution is continued when the command has been transmitted. The sender is not informed if the command cannot be executed successfully.
		"S":	Synchronous acknowledgement The program execution is paused until the receiving component acknowledges the command. If the acknowledgement is positive, the next command is executed. If the acknowledgement is negative, an error message is output.
		"A":	Asynchronous acknowledgement The program execution is continued after the command is issued. The acknowledgement is stored in an acknowledgement variable (pre-defined system variable) and must be explicitly queried from the program. The parameter following the acknowledgement mode is the number of the acknowledgement variable.
	If no <acknowledgement mode> has not been programmed, the synchronous acknowledgement is used as default setting.		

17.8 Program runtime/part counter

17.8.1 Program runtime/part counter (overview)

Information on the program runtime and workpiece counter are provided to support the machine tool operator.

This information can be processed as system variables in the NC and/or PLC program. This information is also available to be displayed on the operator interface.

17.8.2 Program runtime

The "program runtime" function provides internal NC timers to monitor technological processes, which can be read into the part program and into synchronized actions via the NC and channel-specific system variables.

The trigger for the runtime measurement ($AC_PROG_NET_TIME_TRIGGER) is the only system variable of the function that can be written to and is used to selectively measure program sections. This means that by writing the trigger into the NC program, the time measurement can be activated and deactivated.

System variable	Meaning	Activity
NC-specific		
$AN_SETUP_TIME	Time since the last control power up with default values ("cold restart") in minutes. Is automatically reset to "0" every time the control powers up with default values.	• Always active
$AN_POWERON_TIME	Time since the last normal control power up ("warm restart") in minutes. Is automatically reset to "0" every time the control powers up normally.	
Channel-specific		

System variable	Meaning	Activity
$AC_OPERATING_TIME	Total runtime of NC programs in automatic mode in seconds. The value is automatically reset to "0" every time the control powers up.	• Activated via MD27860 • Only AUTOMATIC mode
$AC_CYCLE_TIME	Runtime of the selected NC program in seconds. The value is automatically reset to "0" every time a new NC program starts up.	
$AC_CUTTING_TIME	Processing time in seconds The runtime of the path axes (at least one is active) is measured in all NC programs between NC start and end of program/NC reset without rapid traverse active. The measurement is interrupted when a dwell time is active. The value is automatically reset to "0" every time the control powers up with default values.	
$AC_ACT_PROG_NET_TIME	Actual net runtime of the current NC program in seconds. Is automatically reset to "0" when a new NC program starts.	• Always active • Only AUTOMATIC mode
$AC_OLD_PROG_NET_TIME	Net runtime in seconds of the program that has just be correctly ended with M30	
$AC_OLD_PROG_NET_TIME_COUNT	Changes to $AC_OLD_PROG_NET_TIME After POWER ON, $AC_OLD_PROG_NET_TIME_COUNT is at "0". $AC_OLD_PROG_NET_TIME_COUNT is always increased if the control has newly written to $AC_OLD_PROG_NET_TIME.	

System variable	Meaning		Activity
$AC_PROG_NET_TIME_TRIGGER	Trigger for the runtime measurement:		• Only AUTOMATIC mode
	0	Neutral state The trigger is not active.	
	1	Exit Ends the measurement and copies the value from $AC_ACT_PROG_NET_TIME into $AC_OLD_PROG_NET_TIME. $AC_ACT_PROG_NET_TIME is set to "0" and then continues to run.	
	2	Start Starts the measurement and in so doing sets $AC_ACT_PROG_NET_TIME to "0". $AC_OLD_PROG_NET_TIME is not changed.	
	3	Stop Stops the measurement. Does not change $AC_OLD_PROG_NET_TIME and keeps $AC_ACT_PROG_NET_TIME constant until it resumes	
	4	Resume The measurement is resumed, i.e. a measurement that was previously stopped is continued. $AC_ACT_PROG_NET_TIME continues to run. $AC_OLD_PROG_NET_TIME is not changed.	
All system variables are reset to 0 as a result of POWER ON!			
Reference: A detailed description of the listed system variables is provided in: Function Manual, Basic Functions; Mode Group, Channel, Program Operation, Reset Response (K1), Chapter: Program runtime			

Note

Machine manufacturer

Machine data MD27860 $MC_PROCESSTIMER_MODE is used to switch-in the timer that can be activated.

The behavior of active time measurements for certain functions (e.g. GOTOS, override = 0%, active test run feed, program test, ASUB, PROG_EVENT, …) is configured using machine data MD27850 $MC_PROG_NET_TIMER_MODE and MD27860 $MC_PROCESSTIMER_MODE.

References:
Function Manual, Basic Functions; BAG, Channel, Program Operation, Reset Response (K1), Chapter: Program runtime

Note

Residual time for a workpiece

If the same workpieces are produced one after the other, then the timer values:

- Processing time for the last workpiece produced (see $AC_OLD_PROG_NET_TIME)

and

- Current processing time (see $AC_ACT_PROG_NET_TIME)

can be used to determine the remaining residual time for a workpiece.

The residual time is displayed on the user interface in addition to the current processing time.

Note

Using STOPRE

The system variables $AC_OLD_PROG_NET_TIME and $AC_OLD_PROG_NET_TIME_COUNT do not generate any implicit preprocessing stop. This is uncritical when used in the part program if the value of the system variables comes from the previous program run. However, if the trigger for the runtime measurement ($AC_PROG_NET_TIME_TRIGGER) is written very frequently and as a result $AC_OLD_PROG_NET_TIME changes very frequently, then an explicit `STOPRE` should be used in the part program.

Supplementary conditions

- **Block search**
  No program runtimes are determined through block searches.
- **REPOS**
  The duration of a REPOS process is added to the current processing time ($AC_ACT_PROG_NET_TIME).

Examples

Example 1: Measuring the duration of "mySubProgrammA"

Program code

```
...
N50 DO $AC_PROG_NET_TIME_TRIGGER=2
N60 FOR ii= 0 TO 300
N70 mySubProgrammA
N80 DO $AC_PROG_NET_TIME_TRIGGER=1
N95 ENDFOR
N97 mySubProgrammB
N98 M30
```

After the program has processed line N80, the net runtime of "mySubProgrammA" is located in $AC_OLD_PROG_NET_TIME.

The value from $AC_OLD_PROG_NET_TIME:

- is kept beyond M30.
- is updated each time the loop is run through.

Example 2: Measuring the duration of "mySubProgrammA" and "mySubProgrammC"

Program code

```
...
N10 DO $AC_PROG_NET_TIME_TRIGGER=2
N20 mySubProgrammA
N30 DO $AC_PROG_NET_TIME_TRIGGER=3
N40 mySubProgrammB
N50 DO $AC_PROG_NET_TIME_TRIGGER=4
N60 mySubProgrammC
N70 DO $AC_PROG_NET_TIME_TRIGGER=1
N80 mySubProgrammD
N90 M30
```

17.8.3 Workpiece counter

The "Workpiece counter" function makes available various counters which can be used in particular internally in the control to count workpieces.

The counters exist as channel-specific system variables with read and write access in a range of values from 0 to 999 999 999.

System variable	Meaning
$AC_REQUIRED_PARTS	Number of workpieces to be produced (setpoint number of workpieces) In this counter the number of workpieces at which the actual workpiece count ($AC_ACTUAL_PARTS) will be reset to "0" can be defined.
$AC_TOTAL_PARTS	Total number of completed workpieces (actual workpiece total) This counter specifies the total number of all workpieces produced since the start time. The value is only automatically reset to "0" when the control powers up with default values.
$AC_ACTUAL_PARTS	Number of completed workpieces (actual workpiece total) This counter registers the total number of all workpieces produced since the start time. On condition that $AC_REQUIRED_PARTS > 0, the counter is automatically reset to "0" when the required number of workpieces ($AC_REQUIRED_PARTS) is reached.
$AC_SPECIAL_PARTS	Number of workpieces selected by the user This counter supports user-specific workpiece counts. An alarm can be defined to be output when the setpoint number of workpieces is reached ($AC_REQUIRED_PARTS). Users must reset the counter themselves.

Note

All workpiece counters are set to "0" when the control powers up with default values and can be read and written independent of their activation.

Note

Channel-specific machine data can be used to control counter activation, counter reset timing and the counting algorithm.

Note

Workpiece counting with user-defined M command

Machine data can be set so that the count pulses for the various workpiece counters are triggered using user-defined M commands rather than the end of the program (M2/M30).

References

For further information about the "Workpiece counter" function see:

- Function Manual, Basic Functions; Mode Group, Channel, Program Operation, Reset Response (K1), Section: Workpiece counter

17.9 Process DataShare - Output to an external device/file (EXTOPEN, WRITE, EXTCLOSE):

The writing of data from a part program to an external device/file is performed in three steps:

1. Open the external device/file
   The external device/file is opened for the channel for writing using the EXTOPEN command.
2. Writing data
   The output data can be processed using the string functions of the NC language, e.g. SPRINT. The WRITE command is used for writing.
3. Close the external device/file
   The external device/file assigned in the channel is released again using the EXTCLOSE command, when the end of the program is reached (M30) or for a channel reset.

Syntax

```
DEF INT <Result>
DEF STRING[<n>] <Output>
…
EXTOPEN(<Result>,<ExtDev>,<SyncMode>,<AccessMode>,<WriteMode>)
…
<Output>="data output"
WRITE(<Result>,<ExtDev>,<Output>)
```

```
...
EXTCLOSE(<Result>,<ExtDev>)
```

Meaning

EXTOPEN:	Pre-defined procedure to open an external device/file		
<Result>:	**Parameter 1:** Result variable By using the result variable value, it can be evaluated in the program as to whether the operation was successful and processing is then appropriately continued.		
	Type:	INT	
	Values:	0	No error
		1	External device cannot be opened
		2	External device is not configured
		3	External device with invalid path configured
		4	No access rights for external device
		5	Usage mode: External device already "exclusively" occupied
		6	Usage mode: External device already being "shared"
		7	File length longer than LOCAL_DRIVE_MAX_FILESIZE
		8	Maximum number of external devices has been exceeded
		9	Option for LOCAL_DRIVE not set
		11	V.24 interface has already been assigned with Easy-Message function (only 828D)
		12	Write mode: Data contradicts extdev.ini
		16	Invalid external path has been programmed
		22	External device not mounted

<ExtDev>:	**Parameter 2:** Symbolic identifier for the external device/file to be opened	
	Type: STRING	
	The symbolic identifier comprises: 1. the logical device name 2. where relevant, followed by a file path (attached using "/"). The following **logical device names** have been defined:	
	"LOCAL_DRIVE":	Local CF card (pre-defined)
	"CYC_DRIVE":	Reserved drive name for use in SIEMENS cycles (pre-defined)
	"/dev/ext/1", ... "/dev/ext/9":	Available network drives **Note:** It is necessary to configure in the extdev.ini file!
	"/dev/cyc/1", "/dev/cyc/2":	Reserved drive names for use in SIEMENS cycles **Note:** It is necessary to configure in the extdev.ini file!
	"/dev/v24":	V.24 interface **Note:** It is necessary to configure in the extdev.ini file!
	File path: • A file path must be specified for "LOCAL_DRIVE" and "CYC_DRIVE" e.g. "LOCAL_DRIVE/my_dir/my_file.txt" • The logical device names "/dev/ext/1...9" and "/dev/cyc/1...2" can be configured: – To already refer to a file, in which case only the logical device names may be specified, e.g.: "/dev/ext/4" – Or to a directory, in which case a file path must be specified, e.g.: "/dev/ext/5/my_dir/my_file.txt" • It is not permissible that a file path is attached to "/dev/v24".	
	Note: For the logical device names "/dev/ext/1...9", "/dev/v24" and "/dev/cyc/1...2" uppercase/lowercase is ignored; uppercase/lowercase is significant for specifying a path to a file. Only uppercase letters are permissible for "LOCAL_DRIVE" and "CYC_DRIVE".	

<SyncMode>:	**Parameter 3:** Processing mode for the WRITE commands to this device/file		
	Type:	STRING	
	Values:	"SYN":	Synchronous writing Program execution is stopped until the write operation has been completed. Successfully completing the synchronous write operation can be checked by evaluating the error variables of the WRITE command.
		"ASYN":	Asynchronous writing Program execution is not interrupted by the WRITE command. **Note.** In this mode, the result variable of the WRITE command does not provide any information and always has the value 0 (no error). In this particular mode, there is no certainty that the WRITE command was successful.
<AccessMode>:	**Parameter 4:** Usage mode for this device/file		
	Type:	STRING	
	Values:	"SHARED":	Device/file is requested in the "shared" mode. Other channels can also use the device, i.e. also open in this mode.
		"EXCL":	Device/file is exclusively used in the channel; no other channel can use the device.
<WriteMode>:	**Parameter 5:** Write mode for the WRITE commands to this file/device (optional)		
	Type:	STRING	
	Values:	"APP":	Attaching The file is always kept regarding its contents; write calls are attached at the end.
		"OVR":	Overwrite The contents of the file are deleted and re-generated using the subsequent write calls.
	Note: Using this parameter, the write mode configured in the extdev.ini file cannot be overwritten. In the case of a conflict, then the EXTOPEN call is acknowledged with error.		

WRITE:	Pre-defined procedure to write output data

EXTCLOSE:	Pre-defined procedure to close an external device/file that has been opened		
<Result>:	**Parameter 1:** Result variable		
	Type:	INT	
	Values:	0	No error
		16	Invalid external path has been programmed
		21	Error when closing the external device
<ExtDev>:	**Parameter 2:** Symbolic identifier for the external device/file description to be closed, see EXTOPEN! **Note:** The identifier must be identical to the identifier specified in the EXTOPEN call!		

Example

Program code

```
N10       DEF INT RESULT
N20       DEF BOOL EXTDEVICE
N30       DEF STRING[80] OUTPUT
N40       DEF INT PHASE
N50       EXTOPEN(RESULT,"LOCAL_DRIVE/my_file.txt","SYN","SHARED")
N60       IF RESULT > 0
N70          MSG("Error for EXTOPEN:" << RESULT)
N80       ELSE
N90          EXTDEVICE=TRUE
N100      ENDIF
…
N200      PHASE=4
N210      IF EXTDEVICE
N220         OUTPUT=SPRINT("End phase: %D",PHASE)
N230         WRITE(RESULT,"LOCAL_DRIVE/my_file.txt",OUTPUT)
N240      ENDIF
…
```

See also

String operations (Page 78)

Write file (WRITE) (Page 135)

17.10 Alarms (SETAL)

Alarms can be set in an NC program. Alarms are displayed in a separate field at the user interface. An alarm always goes hand in hand with a response from the control according to the alarm category.

References:
For further information on alarm responses, refer to the Commissioning Manual.

Syntax

```
SETAL(<alarm number>[,<character string>])
```

Meaning

SETAL:	Keyword to program an alarm. SETAL must be programmed in a separate NC block.	
<alarm number>:	Type INT variable. Contains the alarm number. The valid range for alarm numbers lies between 60000 and 69999, of which 60000 to 64999 are reserved for SIEMENS cycles and 65000 to 69999 are available to users.	
<character string>:	When programming user cycle alarms, in addition, a character string with up to four parameters can be specified. Variable user texts can be defined in these parameters. However, the following predefined parameters are available:	
	Parameter	**Meaning**
	%1	Channel number
	%2	Block number, label
	%3	Text index for cycle alarms
	%4	Additional alarm parameters

Note

Alarm texts must be configured in the user interface.

Note

If an alarm is to be output in the language active at the user interface, then the user requires information about the language that is currently set at the HMI. This information can be interrogated in the part program and in the synchronized actions using system variable $AN_LANGUAGE_ON_HMI (see "Currently set language in the HMI (Page 786)").

Example

```
Program code                          Comment
...
N100 SETAL (65000)                    ;Set alarm no. 65000
...
```

17.11 Extended stop and retract (ESR)

The extended stop and retract function - subsequently called ESR - offers the possibility of flexibly responding when a fault situation occurs as a function of the process:

- **Extended stop**
  Assuming that the specific fault situation permits it, all of the axes, enabled for extended stopping, are stopped in an orderly way.
- **Retraction**
  The tool currently in use is retracted from the workpiece as quickly as possible.
- **Generator operation (SINAMICS drive function "Vdc control")**
  If a parameterizable value of the DC-link voltage is fallen below, e.g. because the line voltage fails, the electrical energy required for retraction is generated by recovering the braking energy of the drive intended for this purpose (generator operation).

Trigger sources

General sources (NC-external/global or mode group-/channel-specific):

- Digital inputs (e.g. on NCU module or the control-internal digital output image that can be read back ($A_IN, $A_OUT)
- Channel state ($AC_STAT)
- VDI signals ($A_DBB)
- Group messages of a number of alarms ($AC_ALARM_STAT)

Axial sources

- Emergency retraction threshold of the following axis (synchronism of electronic coupling, $VA_EG_SYNCDIFF[<following axis>])
- Drive: DC-link warning threshold (imminent undervoltage), $AA_ESR_STAT[<axis>]
- Drive: Generator minimum speed threshold (no further regenerative rotation energy available), $AA_ESR_STAT[<axis>].

Gating logic of the static synchronized actions: Source/response link

The static synchronized actions' flexible gating possibilities are used to trigger specific reactions relatively quickly according to the sources.

Linking all relevant sources using static synchronized actions is the responsibility of the user. They can selectively evaluate the source system variables as a whole or by means of bit masks, and then make a logic operation with their desired reactions. The static synchronous actions are effective in all operating modes.

References:
Function Manual, Synchronized Actions

Activation

Function enable

The functions generator operation, shutdown, retraction are released by setting the corresponding control signal $AA_ESR_ENABLE. This control signal can be changed by synchronized actions.

Function triggering

ESR is triggered jointly for all enabled axes by setting the system variable $AC_ESR_TRIGGER.

Generator operation is "automatically" activated in the drive when an imminent DC-link undervoltage is detected.

Drive-independent stopping and/or retraction become active when a communication failure (between the NC and drive) is detected and when a DC-link undervoltage is detected in the drive (configuration and enable required).

Drive-independent stopping and/or retraction can also be triggered by the NC by setting the appropriate control signal $AN_ESR_TRIGGER (broadcast command to all drives).

References

For detailed information on ESR, see:

Function Manual, Special Functions; Extended Stop and Retract (R3)

17.11.1 NC-controlled ESR

17.11.1.1 NC-controlled retraction (POLF, POLFA, POLFMASK, POLFMLIN)

Certain initial conditions are required for NC-controlled retraction (see "NC-controlled retraction (POLF, POLFA, POLFMASK, POLFMLIN) (Page 577)"). When these requirements have been satisfied, then the rapid lift (LIFTFAST) configured for retraction axis(axes) in the channel is activated by setting the system variable $AC_ESR_TRIGGER (or $AA_ESR_TRIGGER for single axes).

Syntax

```
POLF(<axis>)=<position>
POLFA(<axis>,<type>,<position>)
POLFMASK(<axis_1>,<axis_2>,...)
POLFMLIN(<axis_1>,<axis_2>,...)
```

The following abbreviated forms are permitted for `POLFA`:

```
POLFA(<axis>,<type>) ; Abbreviated form for single axis retraction
POLFA(axis,0/1/2) ; Quick deactivation or activation
POLFA(axis,0,$AA_POLFA[axis]) ; Causes a preprocessing stop
POLFA(axis,0) ; Does not cause a preprocessing stop
```

Meaning

POLF:	Address for specifying the target position of the retraction axis POLF is modal.			
	<axis>:	Name of the geometry or channel/machine axis that retracts		
	<position>:	Retraction position		
		Type:	REAL	
		WCS is valid for geometry axes, otherwise MCS. With the same identifiers for geometry and channel/machine axes, retraction is in the WCS.		
POLFA:	Predefined subprogram call for the specification of the retraction position of single axes			
	<axis>:	Channel axis identifier		
	<type>:	Position specification mode		
		Type:	INT	
		Value:	0:	Mark position value as invalid
			1:	Position value is absolute
			2:	Position value is incremental (distance)
		Note: If an axis is not a single axis or if the type is missing or type=0, then a corresponding alarm is output.		
	<position>:	Retraction position (see above)		
		Note: The position value is also accepted with type=0. Only this value is marked as invalid and has to be reprogrammed for retraction.		
POLFMASK:	Predefined subprogram call for selection of the axes that are to be retracted after tripping of rapid lift **independently of one another**.			
	<axis_1>,…:	Names of the axes that are to be traversed to their positions defined with POLF during rapid lift. All the axes specified must be in the same coordinate system.		
	POLFMASK() without specification of an axis deactivates the rapid lift for all axes that have been retracted independently of one another.			
POLFMLIN:	Predefined subprogram call for selection of the axes that are to be retracted after tripping of rapid lift **in linear relation**.			
	<axis_1>,…:	See above.		
	POLFMLIN() without specification of an axis deactivates the rapid lift for all axes that have been retracted in linear relation.			

Note

Before rapid retraction to a fixed position can be enabled via POLFMASK or POLFMLIN, a position must have been programmed with POLF for the selected axes.

Note

If axes are enabled one after the other with POLFMASK, POLFMLIN or POLFMLIN, POLFMASK, then the last definition always applies for the particular axis.

Note

The positions programmed with POLF and the activation by POLFMASK or POLFMLIN are deleted when the part program is started. This means that the user must reprogram the values for POLF and the selected axes in POLFMASK or POLFMLIN in each part program.

Note

If, when using the abbreviated form POLFA only the type is changed, then the user must ensure that either the retraction position or the retraction path contains a practical and sensible value. In particular, the retraction position and the retraction path have to be set again after Power On.

Example

Retracting an individual axis:

Program code	Comment
MD37500 $MA_ESR_REACTION[AX1]=21	; NC-controlled retraction.
...	
$AA_ESR_ENABLE[AX1]=1	
POLFA(AX1,1,20.0)	; AX1 is assigned the axial retraction position 20.0 (absolute).
$AA_ESR_TRIGGER[AX1]=1	; Retraction starts from here.

Further information

Requirements for NC-controlled retraction

- A retraction axis is configured for the NC-controlled retraction in the channel:
  MD37500 $MA_ESR_REACTION = 21
- ESR must be must be enabled for this axis:
  $AA_ESR_ENABLE = 1
- Delay times are defined:
  MD21380 $MC_ESR_DELAY_TIME1
  MD21381 $MC_ESR_DELAY_TIME2
- The axis-specific retraction positions have been configured with POLF in the part program.
- The axes are selected with POLFMASK/POLFMLIN for the NC-controlled retraction.
- The activate signals must be set for the retraction movement and remain set.

Enable and start NC-controlled reactions

If system variable $AC_ESR_TRIGGER = 1 is set and if a retraction axis is configured in this channel (i.e. MD37500 $MA_ESR_REACTION = 21) and $AA_ESR_ENABLE = 1 is set for this axis, then rapid lift (LIFTFAST) is activated in this channel.

The lift movement configured with POLF (or POLF) for the axes selected with POLFMASK or POLFMLIN replaces the path motion defined for these axes in the part program.

The sum of the MD21380 $MC_ESR_DELAY_TIME1 and MD21381 $MC_ESR_DELAY_TIME2 times is the maximum available for the retraction. When this time has expired, rapid deceleration with follow-up is also initiated for the retraction axis.

Note

The extended retraction (i.e. LIFTFAST/LFPOS triggered by $AC_ESR_TRIGGER) **cannot be interrupted** and can only be terminated prematurely via an emergency stop.

Note

Retraction initiated via $AC_ESR_TRIGGER is locked, in order to prevent multiple retractions.

Single axis retraction

With single axis retraction, the retraction position of the single axis must have been programmed with `POLFA` and the following conditions must be satisfied:

- $AA_ESR_ENABLE = 1
- <Axis> must be a single axis at the time of triggering ($AAAA_ESR_TRIGGER = 1).
- <Type> must be either 1 or 2.

Retraction direction during rapid lift

The frame valid at the time when the lift fast is activated is taken into consideration.

Note

Frames with rotation also affect the direction of lift via `POLF`.

Axis replacement

Retraction axes must always be assigned to exactly one NC channel and may not be switched among the channels. When an attempt is made to exchange a retraction axis in another channel, an alarm is output. Only once this axis has been deactivated again using $AA_ESR_ENABLE[AX] = 0 can it be exchanged in a new channel. Once the axis has been exchanged, axes can be acted upon again with $AA_ESR_ENABLE[AX] = 1.

Neutral axes

Neutral axes cannot undertake NC-controlled ESR.

17.11.1.2 NC-controlled stopping

The NC-controlled stopping is activated for the stopping axes configured in the channel by setting system variable $AC_ESR_TRIGGER (or $AA_ESR_TRIGGER for single axes).

Requirements

- A stopping axis is configured for the NC-controlled stopping in the channel:
  MD37500 $MA_ESR_REACTION = 22
- ESR must be enabled for this axis:
  $AA_ESR_ENABLE = 1
- Delay times are defined:
  MD21380 $MC_ESR_DELAY_TIME1 (delay time, ESR axes)
  MD21381 $MC_ESR_DELAY_TIME2 (ESR time for interpolatory braking)

Execution

This axis continues interpolating as programmed for the time period set in MD21380: After the time delay specified in MD21380 has expired, controlled braking (ramp stop) is initiated: The time period in MD21381 is the maximum available for the interpolatory controlled braking. After this period expires, fast braking with subsequent tracking is initiated.

Example

Stopping a single axis:

```
Program code                                        Comment
MD37500 $MC_ESR_REACTION[AX1] = 22                  ; NC-controlled stopping.
MD21380 $MC_ESR_DELAY_TIME1[AX1] = 0.3
MD21381 $MC_ESR_DELAY_TIME2[AX1] = 0.06
...
$AA_ESR_ENABLE[AX1]=1
$AA_ESR_TRIGGER[AX1]=1                              ; Stopping starts from here.
```

17.11.2 Drive-integrated ESR

17.11.2.1 Configuring drive-integrated stopping (ESRS)

The drive parameters for "stopping" of the drive-integrated ESR function are configured using the `ESRS(...)` function.

Syntax

```
ESRS(<access_1>,<stopping time_1>[,...,<axis_n>,<stopping time_n>])
```

Meaning

`ESRS(...)`:	Function to write to the drive parameters for the ESR function "stopping" The function: • Must be alone in the block. • Triggers a preprocessing stop. • Cannot be used in synchronized actions.	

`<axis_1>,`
`...,`
`<axis_n>`:
	Axis for which drive-integrated stopping should be configured
	For this axis, drive parameter p0888 (configuration) is written to in the drive:
	p0888 = 1	

	Type:	AXIS
	Range of values:	Channel axis identifier

`<stopping time_1>,`
`...,`
`<stopping time_n>`:
	Time during which the drive continues to travel with the actual speed setpoint after a fault has occurred
	For the specified axis, drive parameter p0892 (timer) is written to in the drive:
	p0892 = <stopping time>	

	Unit:	s
	Type:	REAL
	Range of values:	0.00 - 20.00
A maximum of 5 axes can be programmed in a function call; n = 5		

17.11.2.2 Configuring drive-integrated retraction (ESRS)

The drive parameters for "retraction" of the drive-integrated ESR function are configured using the `ESRR(...)` function.

Syntax

```
ESRR(<axis_1>,<retraction distance_1>,<retraction
velocity_1>[,...,<axis_n>,<retraction distance_n>,<retraction
velocity_n>])
```

Meaning

ESRR(...):	Function to write to the drive parameters for the ESR function "retract" The function: • Must be alone in the block. • Triggers a preprocessing stop. • Cannot be used in synchronized actions.	
<axis_1>, ..., <axis_n>:	Axis for which drive-integrated retraction should be configured For this axis, drive parameter p0888 (configuration) is written to in the drive: p0888 = 2	
	Type:	AXIS
	Range of values:	Channel axis identifier
<retraction distance_1>, ..., <retraction distance_n>:	For the drive, the retraction distance is converted into a retraction speed. For the specified axis, the value is written to drive parameter p0893 (speed): p0893 = (<retraction distance _n> converted into retraction speed)	
	Unit:	mm/min, inch/min, degrees/min (depending on the unit of the axis)
	Type:	REAL
	Range of values:	MIN - MAX
<retraction velocity_1>, ..., <retraction velocity_n>:	For the drive, the retraction velocity is converted into a time. For the specified axis, the value is written to drive parameter p0892 (timer) [s]: p0892 = <retraction distance_n> / <retraction velocity _n>	
	Unit:	mm/min, inch/min, degrees/min (depending on the unit of the axis)
	Type:	REAL
	Range of values:	0.00 - MAX
A maximum of 5 axes can be programmed in a function call; n = 5		

User stock removal programs 18

18.1 Supporting functions for stock removal

Preprogrammed stock removal programs are provided for stock removal. Beyond this, you have the possibility of generating your own stock removal programs using the following listed functions:

- Generate contour table (CONTPRON)
- Generate coded contour table (CONTDCON)
- Deactivate contour preparation (EXECUTE)
- Determine point of intersection between two contour elements (INTERSEC)
  (Only for tables that were generated using CONTPRON)
- Execute contour elements of a table block-by-block (EXECTAB)
  (Only for tables that were generated using CONTPRON)
- Calculate circle data (CALCDAT)

Note

You can use these functions universally, not just for stock removal.

Requirements

The following must be done before calling the CONTPRON or CONTDCON functions:

- A starting point that permits collision-free machining must be approached.
- The cutting radius compensation must be deactivated with `G40`.

18.2 Generate contour table (CONTPRON)

`CONTPRON` switches on the contour preparation. The NC blocks that are subsequently called are not executed, but are split-up into individual movements and stored in the contour table. Each contour element corresponds to one row in the two-dimensional array of the contour table. The number of relief cuts is returned.

Syntax

Activate contour preparation:
```
CONTPRON(<contour table>,<machining type>,<relief cuts>,
<machining direction>)
```

Deactivate contour preparation and return to the normal execution mode:
```
EXECUTE(<ERROR>)
```

See "Deactivate contour preparation (EXECUTE) (Page 599)"

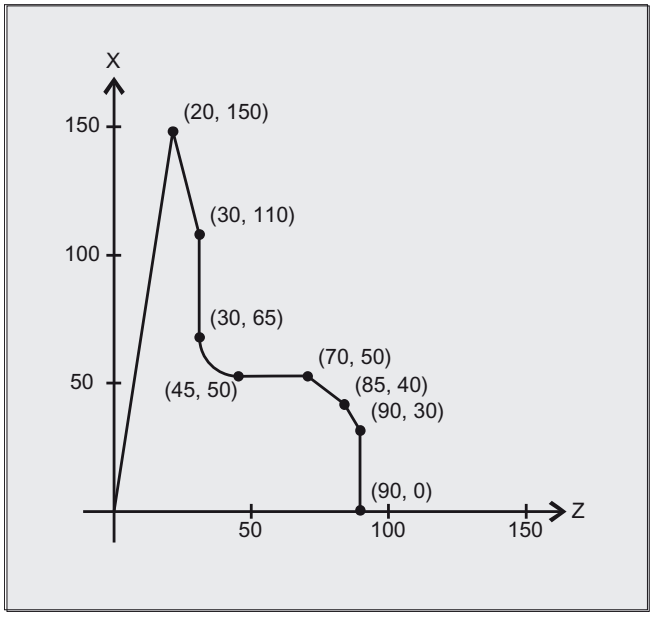
Meaning

CONTPRON:	Predefined procedure to activate the contour preparation to generate a contour table		
<contour table>:	Name of contour table		
<machining type>:	Parameter for the machining type		
	Type:	CHAR	
	Value:	"G":	Longitudinal turning: Internal machining
		"L":	Longitudinal turning: External machining
		"N":	Face turning: Internal machining
		"P":	Face turning: External machining
<relief cuts>:	Result variable for the number of relief cut elements that occur		
	Type:	INT	
<machining direction>:	Parameters for the machining direction		
	Type:	INT	
	Value:	0	Contour preparation, forward (default value)
		1	Contour preparation in both directions

Example 1

Generating a contour table with:

- Name "KTAB"
- Max. 30 contour elements (circles, straight lines)
- One variable for the number of relief cut elements that occur
- One variable for error messages



NC program:

Program code	Comment
N10 DEF REAL KTAB[30,11]	; Contour table with the name KTAB and max. 30 contour elements, parameter value 11 (number of table columns) is a fixed quantity.
N20 DEF INT ANZHINT	; Variable for the number of relief cut elements with the name ANZHINT.
N30 DEF INT ERROR	; Variable for error feedback signal (0=no error, 1=error).
N40 G18	
N50 CONTPRON(KTAB,"G",ANZHINT)	; Activate contour preparation.
N60 G1 X150 Z20	; N60 to N120: Contour description
N70 X110 Z30	
N80 X50 RND=15	
N90 Z70	
N100 X40 Z85	
N110 X30 Z90	
N120 X0	
N130 EXECUTE(ERROR)	; End filling the contour table, switch-over to normal program mode.
N140 …	; Continue to process the table.

Contour table KTAB:

Index Line	Column									
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
7	7	11	0	0	20	150	0	82.40535663	0	0

0	2	11	20	150	30	110	-1111	104.0362435	0	0
1	3	11	30	110	30	65	0	90	0	0
2	4	13	30	65	45	50	0	180	45	65
3	5	11	45	50	70	50	0	0	0	0
4	6	11	70	50	85	40	0	146.3099325	0	0
5	7	11	85	40	90	30	0	116.5650512	0	0
6	0	11	90	30	90	0	0	90	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Explanation of the column contents:

(0) Pointer to next contour element (to the row number of that column)

(1) Pointer to previous contour element

(2) Coding the contour mode for motion

Possible values for X = abc

a = 10^2 G90 = 0 G91 = 1

b = 10^1 G70 = 0 G71 = 1

c = 10^0 G0 = 0 G1 = 1 G2 = 2 G3 = 3

(3), (4) Starting point of contour elements

(3) = abscissa, (4) = ordinate of the current plane

(5), (6) Starting point of the contour elements

(5) = abscissa, (6) = ordinate of the current plane

(7) Max/min indicator: Identifies local maximum and minimum values on the contour

(8) Maximum value between contour element and abscissa (for longitudinal machining) or ordinate (for face cutting). The angle depends on the type of machining programmed.

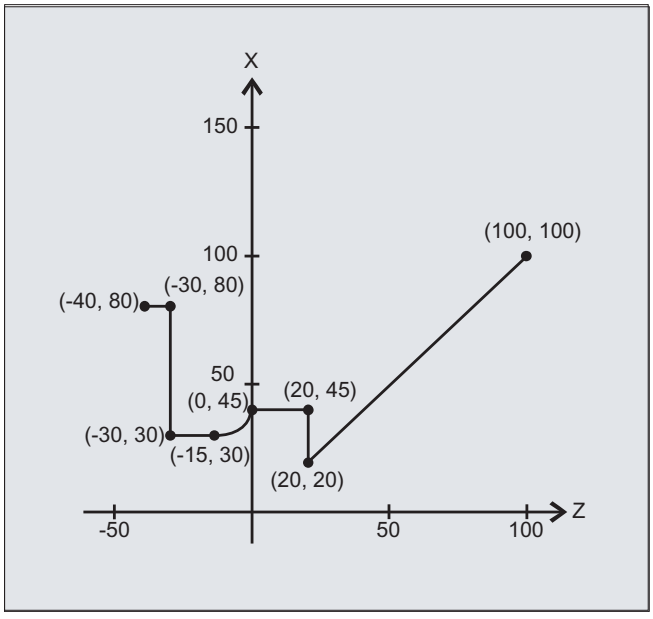
(9), (10) Center point coordinates of contour element, if it is a circle block.

(9) = abscissa, (10) = ordinate

Example 2

Generating a contour table with

- Name KTAB
- Max. 92 contour elements (circles, straight lines)
- Operating mode: Longitudinal turning, external machining
- Preparation, forward and backward



NC program:

Program code	Comment
N10 DEF REAL KTAB[92,11]	; Contour table with name KTAB and max. 92 contour elements, parameter value 11 is a fixed quantity.
N20 DEF CHAR BT="L"	; Mode for CONTPRON: Longitudinal turning, external machining
N30 DEF INT HE=0	;Number of relief cut elements=0
N40 DEF INT MODE=1	; Preparation, forward and backward
N50 DEF INT ERR=0	; Error feedback signal
...	
N100 G18 X100 Z100 F1000	
N105 CONTPRON(KTAB,BT,HE,MODE)	; Activate contour preparation.
N110 G1 G90 Z20 X20	
N120 X45	
N130 Z0	
N140 G2 Z-15 X30 K=AC(-15) I=AC(45)	
N150 G1 Z-30	
N160 X80	
N170 Z-40	
N180 EXECUTE(ERR)	; End filling the contour table, switchover to normal program mode.
...	

Contour table KTAB:

After contour preparation is finished, the contour is available in both directions.

Index	Column										
Line	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
0	6[1)]	7[2)]	11	100	100	20	20	0	45	0	0
1	0[3)]	2	11	20	20	20	45	-3	90	0	0
2	1	3	11	20	45	0	45	0	0	0	0
3	2	4	12	0	45	-15	30	5	90	-15	45
4	3	5	11	-15	30	-30	30	0	0	0	0
5	4	7	11	-30	30	-30	45	-1111	90	0	0
6	7	0[4)]	11	-30	80	-40	80	0	0	0	0
7	5	6	11	-30	45	-30	80	0	90	0	0
8	1[5)]	2[6)]	0	0	0	0	0	0	0	0	0
	...										
83	84	0[7)]	11	20	45	20	80	0	90	0	0
84	90	83	11	20	20	20	45	-1111	90	0	0
85	0[8)]	86	11	-40	80	-30	80	0	0	0	0
86	85	87	11	-30	80	-30	30	88	90	0	0
87	86	88	11	-30	30	-15	30	0	0	0	0
88	87	89	13	-15	30	0	45	-90	90	-15	45
89	88	90	11	0	45	20	45	0	0	0	0
90	89	84	11	20	45	20	20	84	90	0	0
91	83[9)]	85[10)]	11	20	20	100	100	0	45	0	0

Explanation of column contents and comments for lines 0, 1, 6, 8, 83, 85 and 91

The explanations of the column contents given in example 1 apply.

Always in table line 0:

1) Predecessor: Line n contains the contour end (forward)

2) Successor: Line n is the contour table end (forward)

Once each within the contour elements forward:

3) Predecessor: Contour start (forward)

4) Successor: Contour end (forward)

Always in line contour table end (forward) +1:

5) Predecessor: Number of relief cuts (forward)

6) Successor: Number of relief cuts (backward)

Once each within the contour elements backward:

7) Successor: Contour end (backward)

8) Predecessor: Contour start (backward)

Always in last line of table:

9) Predecessor: Line n is the contour table start (backward)

10) Successor: Line n contains the contour start (backward)

Further information

Permitted traversing commands, coordinate system

The following G commands can be used for the contour programming:

- G group 1: G0, G1, G2, G3

In addition, the following are possible:

- Rounding and chamfer
- Circle programming using CIP and CT

The spline, polynomial and thread functions result in errors.

Changes to the coordinate system by activating a frame are not permissible between CONTPRON and EXECUTE. The same applies for a change between G70 and G71 or G700 and G710.

Replacing the geometry axes with GEOAX while preparing the contour table produces an alarm.

Relief cut elements

The contour description for the individual relief cut elements can be performed either in a subprogram or in individual blocks.

Stock removal independent of the programmed contour direction

The contour preparation with CONTPRON was expanded so that after it has been called, the contour table is available independent of the programmed direction.

18.3 Generate coded contour table (CONTDCON)

With the contour preparation activated with CONTDCON, the following NC blocks that are called are saved in a coded form in a 6-column contour table to optimize memory use. Each contour element corresponds to one row in the contour table. When familiar with the coding rules specified below, e.g. you can combine DIN code programs for cycles from the table lines. The data of the output point is saved in the table line with the number 0.

Syntax

Activate contour preparation:
```
CONTDCON(<contour table>,<machining direction>)
```

Deactivate contour preparation and return to the normal execution mode:
```
EXECUTE(<ERROR>)
```

See "Deactivate contour preparation (EXECUTE) (Page 599)"

Meaning

CONTDCON:	Predefined procedure to activate the contour preparation to generate a coded contour table		
<contour table>:	Name of the contour table		
<machining direction>:	Parameter for machining direction		
	Type:	INT	
	Value:	0	Contour preparation according to the sequence of contour blocks (default value)
		1	Not permissible

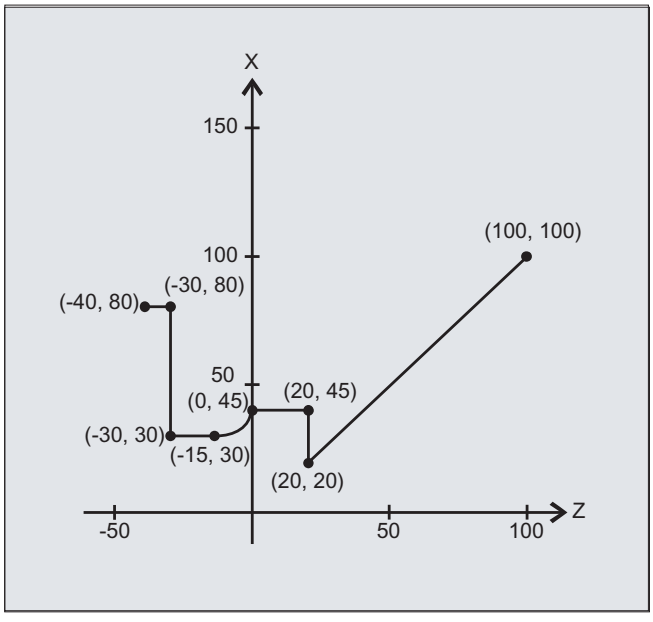
Note

The G codes permitted for CONTDCON in the program section to be included in the table are more comprehensive than for CONTPRON. Further, feedrates and feedrate type are saved for each contour section.

Example

Generating a contour table with:

- Name "KTAB"
- Contour elements (circles, straight lines)
- Operating mode: Turning
- Machining direction: Forward



NC program:

```
Program code                                    Comment
N10 DEF REAL KTAB[9,6]                          ;Contour table with name KTAB and 9
                                                table cells. These allow 8 contour
                                                sets. The parameter value 6 (column
                                                number in table) is a fixed size.
N20 DEF INT MODE = 0                            ; Variable for the machining direc-
                                                tion. Standard value 0: Only in the
                                                programmed direction of the contour.
N30 DEF INT ERROR = 0                           ; Variable for the error feedback sig-
                                                nal.
...
N100 G18 G64 G90 G94 G710
N101 G1 Z100 X100 F1000
N105 CONTDCON (KTAB, MODE)                      ; Contour preparation call (MODE can
                                                be omitted).
N110 G1 Z20 X20 F200                            ; Contour description.
N120 G9 X45 F300
N130 Z0 F400
N140 G2 Z-15 X30 K=AC(-15) I=AC(45)F100
N150 G64 Z-30 F600
N160 X80 F700
N170 Z-40 F800
N180 EXECUTE(ERROR)                             ; End filling the contour table,
                                                switchover to normal program mode.
...
```

Contour table KTAB:

	Column index					
	0	1	2	3	4	5
Line index	Contour mode	End point abscissa	End point ordinate	Center point abscissa	Center point ordinate	Feedrate
0	30	100	100	0	0	7
1	11031	20	20	0	0	200
2	111031	20	45	0	0	300
3	11031	0	45	0	0	400
4	11032	-15	30	-15	45	100
5	11031	-30	30	0	0	600
6	11031	-30	80	0	0	700
7	11031	-40	80	0	0	800
8	0	0	0	0	0	0

Explanation of the column contents:

Line 0 Coding for the **starting point:**

Column 0:	10^0 (units digit): G0 = 0
	10^1 (tens position): G70 = 0, G71 = 1, G700 = 2, G710 = 3
Column 1:	Starting point abscissa
Column 2:	Starting point ordinate
Column 3-4:	0
Column 5:	Line index of last contour piece in the table

Lines 1-n: Entries for **contour pieces**

Column 0:	10^0 (units digit): G0 = 0, G1 = 1, G2 = 2, G3 = 3
	10^1 (tens position): G70 = 0, G71 = 1, G700 = 2, G710 = 3
	10^2 (hundreds digit): G90 = 0, G91 = 1
	10^3 (thousands digit): G93 = 0, G94 = 1, G95 = 2, G96 = 3
	10^4 (ten thousands digit): G60 = 0, G44 = 1, G641 = 2, G642 = 3
	10^5 (hundred thousands digit): G9 = 1
Column 1:	End point abscissa
Column 2:	End point ordinate
Column 3:	Center point abscissa for circular interpolation
Column 4:	Center point ordinate for circular interpolation
Column 5:	Feedrate

Further information

Permitted traversing commands, coordinate system

The following G groups and G commands can be used for the contour programming:

G group 1:	G0, G1, G2, G3
G group 10:	G60, G64, G641, G642
G group 11:	G9
G group 13:	G70, G71, G700, G710
G group 14:	G90, G91
G group 15:	G93, G94, G95, G96, G961

In addition, the following are possible:

- Rounding and chamfer
- Circle programming using CIP and CT

The spline, polynomial and thread functions result in errors.

Changes to the coordinate system by activating a frame are not permissible between CONTDCON and EXECUTE. The same applies for a change between G70 and G71 or G700 and G710.

Replacing the geometry axes with GEOAX while preparing the contour table produces an alarm.

Machining direction

The contour table generated using CONTDCON is used for stock removal in the programmed direction of the contour.

18.4 Determine point of intersection between two contour elements (INTERSEC)

INTERSEC determines the point of intersection of two normalized contour elements from the contour tables generated using CONTPRON.

Syntax

```
<Status>=INTERSEC(<contour table_1>[<contour element_1>],
<contour table_2>[<contour element_2>],<intersection
point>,<machining type>)
```

Meaning

INTERSEC:	Predefined function to determine the point of intersection between two contour elements from the contour tables generated with CONTPRON		
<Status>:	Variable for the point of intersection status		
	Type:	BOOL	
	Value:	TRUE	Point of intersection found
		FALSE	No intersection found
<contour table_1>:	Name of the first contour table		
<contour element_1>:	Number of the contour element of the first contour table		
<contour table_2>:	Names of the second contour table		
<contour element_2>:	Number of the contour element of the second contour table		
<point of intersection>:	Intersection coordinates in the active plane (G17 / G18 / G19)		
	Type:	REAL	
<machining type>:	Parameter for the machining type		
	Type:	INT	
	Value:	0	Point of intersection calculation in the active plane with parameter 2 (standard value)
		1	Point of intersection calculation independent of the transferred plane

Note

Please note that the variables must be defined before they are used.

The values defined with CONTPRON must be observed when transferring the contours:

Parameter	Meaning
2	Coding of contour mode for the movement
3	Contour start point abscissa
4	Contour start point ordinate
5	Contour end point abscissa
6	Contour end point ordinate
9	Center point coordinates for abscissa (only for circle contour)
10	Center point coordinates for ordinate (only for circle contour)

Example

Calculate the intersection of contour element 3 in table TABNAME1 and contour element 7 in table TABNAME2. The intersection coordinates in the active plane are stored in the variables ISCOORD (1st element = abscissa, 2nd element = ordinate). If no intersection exists, the program jumps to NOCUT (no intersection found).

```
Program code                                          Comment
DEF REAL TABNAME1[12,11]                              ; Contour table 1
DEF REAL TABNAME2[10,11]                              ; Contour table 2
DEF REAL ISCOORD [2]                                  ; Variable for the intersection coor-
                                                      dinates.
DEF BOOL ISPOINT                                      ; Variable for the intersection sta-
                                                      tus.
DEF INT MODE                                          ; Variable for the machining type.
…
MODE=1                                                ; Calculation independent of the ac-
                                                      tive plane.
N10 ISPOINT=INTERSEC(TABNAME1[3],TABNAME2[7],         ; Intersection of the contour ele-
ISCOORD,MODE)                                         ments call.
N20 IF ISPOINT==FALSE GOTOF NOCUT                     ; Jump to NOCUT.
…
```

18.5 Execute the contour elements of a table block-by-block (EXECTAB)

Using EXECTAB, you can execute the contour elements of a table – that were generated, e.g. with CONTPRON – block-by-block.

Syntax

```
EXECTAB(<contour table>[<contour element>])
```

Meaning

EXECTAB:	Predefined procedure to execute a contour element
<contour table>:	Name of the contour table
<contour element>:	Number of the contour element

Example

Contour elements 0 to 2 in table KTAB should be executed block-by-block.

Program code	Comment
N10 EXECTAB(KTAB[0])	; Traverse element 0 of table KTAB.
N20 EXECTAB(KTAB[1])	; Traverse element 1 of table KTAB.
N30 EXECTAB(KTAB[2])	; Traverse element 2 of table KTAB.

18.6 Calculate circle data (CALCDAT)

With CALCDAT, you can calculate the radius and the circle center point coordinates from the three or four points known along the circle. The specified points must be different.

Where four points do not lie directly on the circle an average value is formed for the circle center point and the radius.

Note

Calculation regulation for the averaging

The arc calculation is performed four times:

1. With circle points 1, 2, 3
2. With circle points 1, 2, 4
3. With circle points 1, 3, 4
4. With circle points 2, 3, 4

The values of the circle center point coordinates abscissa and ordinate are calculated by adding the abscissa and ordinate values of the four arc calculations and dividing by four.

The radius is calculated by forming the root from the sum of the four radii from the arc calculations and multiplying the result with 0.5.

Syntax

```
<Status>=CALCDAT(<circle points>[<number>,<type>],<number>,<result>)
```

Meaning

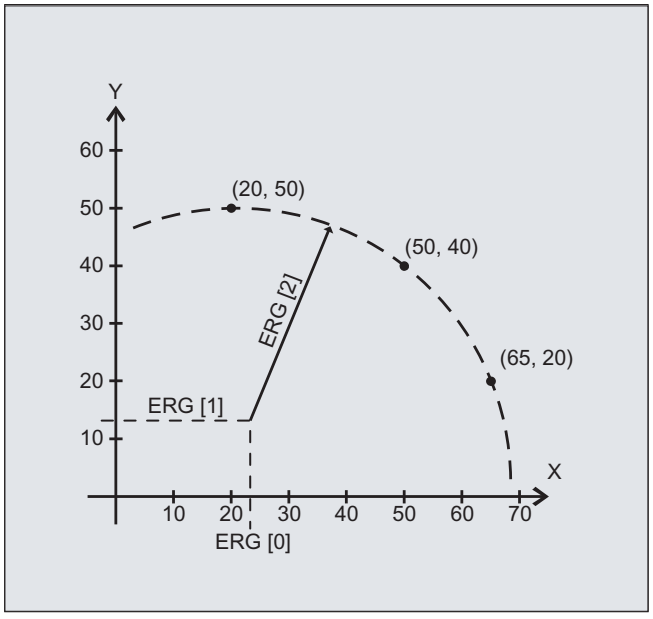
CALCDAT:	Predefined function to calculate the radius and center point coordinates of a circle from three or four points		
<Status>:	Variable for the circle calculation status		
	Type:	BOOL	
	Value:	TRUE	The specified points lie on a circle.
		FALSE	The specified points **do not** lie on a circle.
<circle points>[]:	Variable to specify the circle points using parameters		
	<number>:	Number of circle points (3 or 4)	
	<type>:	Type of coordinate data, e.g. 2 for 2-point coordinates	
<number>:	Parameter for the number of the points used for the calculation (3 or 4)		
<result>[3]:	Variable for result: Circle center point coordinates and radius		
	0	Circle center point coordinate: Abscissa value	
	1	Circle center point coordinate: Ordinate value	
	2	Radius	

Note

Please note that the variables must be defined before they are used.

Example

Using three points it should be determined as to whether they are located on a circle segment.



Program code	Comment
N10 DEF REAL PT[3,2]=(20,50,50,40,65,20)	; Variable to specify the points of a circle.
N20 DEF REAL RES[3]	; Variable for result.
N30 DEF BOOL STATUS	; Variable for status.
N40 STATUS=CALCDAT(PKT,3,ERG)	; Call of the determined circle data.
N50 IF STATUS == FALSE GOTOF ERROR	; Jump to error.

18.7 Deactivate contour preparation (EXECUTE)

EXECUTE deactivates the contour preparation and at the same time the system returns to the normal execution mode.

Syntax

```
EXECUTE(<ERROR>)
```

Meaning

EXECUTE:	Predefined procedure to terminate contour preparation	
<ERROR>:	Variable for the error feedback signal	
	Type:	INT
	The value of the variable indicates whether the contour was able to be prepared error-free:	
	0	Error
	1	No error

Example

Program code

```
...
N30 CONTPRON(...)
N40 G1 X... Z...
...
N100 EXECUTE(...)
...
```

Programming cycles externally 19

19.1 Technology cycles

19.1.1 Introduction

Contents

This section contains a description of the cycles for the turning, milling, and grinding technologies.

Structure

The description of a cycle is structured as follows:

- **Syntax**
  Cycle name and call sequence of the transfer parameters
- **Parameters**
  Tables to explain the individual parameters

Parameter description

The following data is specified in the table for a parameter: Name, description, value range, and dependencies on other parameters.

The column for reference to the parameter in the screen form is provided to more easily locate values programmed on the control when externally generated cycle calls are recompiled.

"For interface only" parameters

Certain parameters are marked "for interface only" in the table. These are not relevant to operation of the cycle. They are only needed in order to be able to recompile cycle calls completely. If they are not programmed the cycle can still be recompiled; the fields are then identified by color and must be completed in the mask.

"Reserved" parameters

Parameters that are described as "reserved" must be programmed with the value 0 or a comma so that the assignment of the following call parameters matches the internal cycle parameters. Exception: string parameters with the value "" or a comma.

Repeating cycles on a position pattern

Drilling and milling cycles can be repeated on the position pattern (modal calls). In such cases `MCALL` should be written in the same line before the cycle, e.g. `MCALL CYCLE83(...)`.

Note

If certain transfer parameters (e.g. `<_VARI>, <_GMODE>, <_DMODE>, <_AMODE>`) have been indirectly programmed as parameters, the screen form is opened on recompiling but it cannot be stored as there is no unambiguous assignment to defined selection fields.

19.1.2 Technology-specific overview

The following overview table lists all available externally programmable technology cycles and the technology assigned to each of them:

Technology	Technology cycle
Drilling	• CYCLE81 - drilling, centering (Page 641) • CYCLE82 - drilling, counterboring (Page 642) • CYCLE85 - reaming (Page 651) • CYCLE86 - boring (Page 652) • CYCLE83 - deep-hole drilling (Page 645) • CYCLE830 - deep-hole drilling 2 (Page 675) • CYCLE84 - tapping without compensating chuck (Page 648) • CYCLE840 - tapping with compensating chuck (Page 684) • CYCLE78 - Drill thread milling (Page 637) • CYCLE802 - arbitrary positions (Page 673) • HOLES1 - row of holes (Page 604) • CYCLE801 - grid or frame (Page 671) • HOLES2 - hole circle (Page 604)
Turning	• CYCLE951 - stock removal (Page 695) • CYCLE930 - groove (Page 690) • CYCLE940 - undercut forms (Page 692) • CYCLE99 - thread turning (Page 661) • CYCLE98 - thread chain (Page 657) • CYCLE92 - cut-off (Page 653)
Contour turning	• CYCLE62 - contour call (Page 623) • CYCLE952 - contour grooving (Page 698)

Technology	Technology cycle
Milling	• CYCLE61 - Face milling (Page 621) • POCKET3 - milling a rectangular pocket (Page 606) • POCKET4 - milling a circular pocket (Page 609) • CYCLE76 - rectangular spigot milling (Page 633) • CYCLE77 - circular spigot milling (Page 635) • CYCLE79 - multi-edge (Page 639) • SLOT1 - longitudinal slot (Page 611) • SLOT2 - circumferential slot (Page 614) • CYCLE899 - Milling open slot (Page 687) • LONGHOLE - elongated hole (Page 616) • CYCLE70 - thread milling (Page 627) • CYCLE60 - engraving cycle (Page 618)
Contour milling	• CYCLE62 - contour call (Page 623) • CYCLE72 - Path milling (Page 629) • CYCLE63 - Milling contour pocket (Page 624) • CYCLE64 - Predrilling contour pocket (Page 626)
Grinding	• CYCLE495 - form-truing (Page 666) • CYCLE435 - calculate dressing tool position (Page 666) • CYCLE4071 - longitudinal grinding with infeed at the reversal point (Page 704) • CYCLE4072 - longitudinal grinding with infeed at the reversal point and cancel signal (Page 705) • CYCLE4073 - longitudinal grinding with continuous infeed (Page 709) • CYCLE4074 - longitudinal grinding with continuous infeed and cancel signal (Page 710) • CYCLE4075 - surface grinding with infeed at the reversal point (Page 713) • CYCLE4077 - surface grinding with infeed at the reversal point and cancel signal (Page 716) • CYCLE4078 - surface grinding with continuous infeed (Page 719) • CYCLE4079 - surface grinding with intermittent infeed (Page 721)
Other	• CYCLE800 - swiveling (Page 668) • CYCLE832 - High-Speed Settings (Page 681)

19.1.3 HOLES1 - row of holes

Syntax

```
HOLES1(<SPCA>, <SPCO>, <STA1>, <FDIS>, <DBH>, <NUM>, <_VARI>,
<_UMODE>, <_HIDE>, <_NSP>, <_DMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	X0	<SPCA>	REAL	Reference point for row of holes along the 1st axis (abs)		
2	Y0	<SPCO>	REAL	Reference point for row of holes along the 2nd axis (abs)		
3	α0	<STA1>	REAL	Basic angle of rotation (angle to 1st axis)		
4	L0	<FDIS>	REAL	Distance from 1st hole to reference point		
5	L	<DBH>	REAL	Spacing between the holes		
6	N	<NUM>	INT	Number of holes		
7		<_VARI>	INT	Reserved		
8		<_UMODE>	INT	Reserved		
9		<_HIDE>	STRING [200]	Hidden positions • Max. 198 characters • Specification of consecutive position numbers, e.g. "1,3" (positions 1 and 3 are not executed)		
10		<_NSP>	INT	Reserved		
11		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)

19.1.4 HOLES2 - hole circle

Syntax

```
HOLES2(<CPA>, <CPO>, <RAD>, <STA1>, <INDA>, <NUM>, <_VARI>,
<_UMODE>, <_HIDE>, <_NSP>, <_DMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning			
1	X0	<CPA>	REAL	Center point for circle of holes along the 1st axis (abs) Reference point in the 1st axis			(for XY) (for XA, YB, ZC)
2	Y0	<CPO>	REAL	Center point for circle of holes along the 2nd axis (abs) Reference point in the 2nd axis			(for XY) (for XA, YB, ZC)
3	R	<RAD>	REAL	Radius of the circle of holes			(for XY)
4	α0	<STA1>	REAL	Starting angle or 1st rotary axis position			(for XY) (for XA, YB, ZC)
5	α1	<INDA>	REAL	Advance angle (for pitch circle only)			(for XY, XA, YB, ZC)
					< 0 =	Clockwise	
					> 0 =	Counter-clockwise	
6	N	<NUM>	INT	Number of positions			
7		<_VARI>	INT	Machining type			
				UNITS:	Reserved		
				TENS:	Positioning type		
					0 =	Approach position - linear	
					1 =	Approach position - circular path	
				HUNDREDS:	Reserved		
				THOUSANDS:	Circular pattern		
					0 =	Compatibility mode, if INDA = 0 then full circle, INDA <> 0 then pitch circle	
					1 =	Full circle	
					2 =	Pitch circle	
				TEN THOUSANDS:	Position pattern with rotary axis		
					0 =	XY (without rotary axis)	(for XY)
					1 =	XA (X axis and rotary axis around X)	(only for XA)
					2 =	YB (Y axis and rotary axis around Y)	(only for YB)
					3 =	ZC (Z axis and rotary axis around C)	(only for ZC)
				ONE MILLION + HUNDRED THOUSANDS:	Offset (for several rotary axes around the same axis; if index too large, then 1st axis)		
					00 =	1. A, B or C axis	
					01 =	2. A, B or C axis	
					...		
					10 =	20. A, B or C axis	
8		<_UMODE>	INT	Reserved			

No.	Parameter mask	Parameter internal	Data type	Meaning		
9		<_HIDE>	STRING [200]	Reserved		
10		<_NSP>	INT	Reserved		
11		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)

19.1.5 POCKET3 - milling a rectangular pocket

Syntax

```
POCKET3(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_LENG>, <_WID>, <_CRAD>,
<_PA>, <_PO>, <_STA>, <_MID>, <_FAL>, <_FALD>, <_FFP1>, <_FFD>,
<_CDIR>, <_VARI>, <_MIDA>, <_AP1>, <_AP2>, <_AD>, <_RAD1>, <_DP1>,
<_UMODE>, <_FS>, <_ZFS>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Pocket depth (abs/inc), see <_AMODE>		
5	L	<_LENG>	REAL	Pocket length (inc, to be entered with sign)		
6	W	<_WID>	REAL	Pocket width (inc, to be entered with sign)		
7	R	<_CRAD>	REAL	Corner radius of pocket		
8	X0	<_PA>	REAL	Reference point, 1st axis (abs)		
9	YO	<_PO>	REAL	Reference point, 2nd axis (abs)		
10	α0	<_STA>	REAL	Angle of rotation, angle between longitudinal axis (L) and 1st axis		
11	DZ	<_MID>	REAL	Maximum depth infeed		
12	UXY	<_FAL>	REAL	Finishing allowance, plane		
13	UZ	<_FALD>	REAL	Finishing allowance, depth		
14	F	<_FFP1>	REAL	Feedrate in the plane		
15	FZ	<_FFD>	REAL	Depth infeed rate		
16		<_CDIR>	INT	Milling direction:	0 =	Down-cut
					1 =	Up-cut

No.	Parameter mask	Parameter internal	Data type	Meaning		
17		<_VARI>	INT	Machining type		
				UNITS:		
					1 =	Roughing
					2 =	Finishing
					4 =	Edge finishing
					5 =	Chamfering
				TENS:		
					0 =	Predrilled, infeed with G0
					1 =	Vertically, infeed with G1
					2 =	Helical
					3 =	Oscillation on pocket longitudinal axis
				HUNDREDS:	Reserved	
18	DXY	<_MIDA>	REAL	Maximum plane infeed, for unit, see <_AMODE>		
19	L1	<_AP1>	REAL	Length of premachining (inc)		
20	W1	<_AP2>	REAL	Width of premachining (inc)		
21	AZ	<_AD>	REAL	Depth of premachining (inc)		
22	ER	<_RAD1>	REAL	Radius of helical path on helical insertion		
	EW			Maximum insertion angle for oscillation		
23	EP	<_DP1>	REAL	Helical pitch on helical insertion		
24		<_UMODE>	INT	Reserved		
25	FS	<_FS>	REAL	Chamfer width (inc)		
26	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		

No.	Parameter mask	Parameter internal	Data type	Meaning		
27		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining
				THOUSANDS:	Dimensioning via center/corner	
					0 =	Compatibility mode
					1 =	Dimensioning via center
					2 =	Dimensioning of corner point, pocket position +LENG/+WID
					3 =	Dimensioning of corner point, pocket position -LENG/+WID
					4 =	Dimensioning of corner point, pocket position +LENG/-WID
					5 =	Dimensioning of corner point, pocket position -LENG/-WID
				TEN THOUSANDS:	Complete machining/remachining	
					0 =	Compatibility mode (process <_AP1>, <_AP2> and <_AD> as before)
					1 =	Complete machining
					2 =	Post machining
28		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Type of feedrate: G group (G94/G95) for surface and depth feedrate	
					0 =	Compatibility mode
					1 =	G code as before cycle call. G94/G95 same for surface and depth feedrate
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
29		<_AMODE>	INT	Alternative mode		
				UNITS:	Pocket depth (Z1)	
					0 =	Absolute (compatibility mode)
					1 =	Incremental
				TENS:	Unit for plane infeed (DXY)	
					0 =	mm
					1 =	% of tool diameter
				HUNDREDS:	Insertion depth for chamfering (ZFS)	
					0 =	Absolute
					1 =	Incremental

19.1.6 POCKET4 - milling a circular pocket

Syntax

```
POCKET4(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_CDIAM>, <_PA>, <_PO>,
<_MID>, <_FAL>, <_FALD>, <_FFP1>, <_FFD>, <_CDIR>, <_VARI>, <_MIDA>,
<_AP1>, <_AD>, <_RAD1>, <_DP1>, <_UMODE>, <_FS>, <_ZFS>, <_GMODE>,
<_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Pocket depth (abs/inc), see <_AMODE>		
5	∅	<_CDIAM>	REAL	Pocket diameter or radius, see <_DMODE>		
6	X0	<_PA>	REAL	Reference point 1st axis (abs)		
7	Y0	<_PO>	REAL	Reference point 2nd axis (abs)		
8	DZ	<_MID>	REAL	Maximum depth setting, see <_VARI> = by planes Maximum helical setting, see <_VARI> = helically		
9	UXY	<_FAL>	REAL	Finishing allowance, plane		
10	UZ	<_FALD>	REAL	Finishing allowance, depth		
11	F	<_FFP1>	REAL	Feedrate for surface machining		
12	FZ	<_FFD>	REAL	Depth infeed rate		
13		<_CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut

No.	Parameter mask	Parameter internal	Data type	Meaning		
14		<_VARI>	INT	Machining type		
				UNITS:	Machining	
					1 =	Roughing
					2 =	Finishing
					4 =	Edge finishing
					5 =	Chamfering
				TENS:	Infeed type (roughing and finishing)	
					0 =	Predrilled, infeed with G0 (pocket is premachined)
					1 =	Vertically, infeed with G1
					2 =	Helical
				HUNDREDS:	Reserved	
				THOUSANDS:		
					0 =	Plane-by-plane
					1 =	Helical
15	DXY	<_MIDA>	REAL	Maximum plane infeed, see <_AMODE>, 0 = 0.8 x tool diameter		
16	∅	<_AP1>	REAL	Diameter/radius of premachining (inc)		
17	AZ	<_AD>	REAL	Depth of premachining (inc)		
18	ER	<_RAD1>	REAL	Radius of helical path on helical insertion		
19	EP	<_DP1>	REAL	Helical pitch on insertion on helical path		
20		<_UMODE>	INT	Reserved		
21	FS	<_FS>	REAL	Chamfer width (inc)		
22	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		
23		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Machining/calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Complete machining/remachining	
					0 =	Compatibility mode (process <_AP1> and <_AD> as before)
					1 =	Complete machining
					2 =	Post machining

No.	Parameter mask	Parameter internal	Data type	Meaning		
24		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Type of feedrate: G group (G94/G95) for surface and depth feedrate	
					0 =	Compatibility mode
					1 =	G code as before cycle call. G94/G95 same for surface and depth feedrate
				HUNDREDS:		
					0 =	Compatibility mode (enter <_CDIAM>/<_AP1> as radius)
					1 =	Enter <_CDIAM>/<_AP1> as diameter
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple
25		<_AMODE>	INT	Alternative mode		
				UNITS:	Pocket depth (Z1)	
					0 =	Absolute (compatibility mode)
					1 =	Incremental
				TENS:	Unit for infeed width (DXY)	
					0 =	mm
					1 =	% of tool diameter
				HUNDREDS:	Insertion depth for chamfering (ZFS)	
					0 =	Absolute
					1 =	Incremental

19.1.7 SLOT1 - longitudinal slot

Syntax

```
SLOT1 (<RTP>, <RFP>, <SDIS>, <_DP>, <_DPR>, <NUM>, <LENG>, <WID>,
<_CPA>, <_CPO>, <RAD>, <STA1>, <INDA>, <FFD>, <FFP1>, <_MID>,
<CDIR>, <_FAL>, <VARI>, <_MIDF>, <FFP2>, <SSF>, <_FALD>, <_STA2>,
<_DP1>, <_UMODE>, <_FS>, <_ZFS>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<RTP>	REAL	Retraction plane (abs)		
2	Z0	<RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Slot depth (abs)		
5		<_DPR>	REAL	Slot depth (inc) with respect to Z0 (enter without sign)		
6		<NUM>	INT	Number of slots = 1		
7	L	<LENG>	REAL	Slot length		
8	W	<WID>	REAL	Slot width		
9	X0	<_CPA>	REAL	Reference point in the 1st axis of the plane		
10	Y0	<_CPO>	REAL	Reference point in the 2nd axis of the plane		
11		<RAD>	REAL	Reserved		
12	α	<STA1>	REAL	Angle of rotation		
13		<INDA>	REAL	Reserved		
14	FZ	<FFD>	REAL	Depth infeed rate		
15	F	<FFP1>	REAL	Feedrate		
16	DZ	<_MID>	REAL	Maximum depth infeed		
17		<CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut
18	UXY	<_FAL>	REAL	Finishing allowance on plane or slot edge		
19		<VARI>	INT	Machining type		
				UNITS:		
					0 =	Reserved
					1 =	Roughing
					2 =	Finishing
					4 =	Edge finishing (only machine the edge)
					5 =	Chamfering
				TENS:	Approach	
					0 =	Predrilled, infeed with G0 (slot is premachined)
					1 =	Vertically, infeed with G1
					2 =	Helical
					3 =	Oscillation
				HUNDREDS:	Reserved	
20	DZF	<_MIDF>	REAL	Reserved		
21	FF	<FFP2>	REAL	Reserved		
22	SF	<SSF>	REAL	Reserved		
23	UZ	<_FALD>	REAL	Finishing allowance, depth		
24	ER	<_STA2>	REAL	Radius of helical path on helical insertion		
	EW			Maximum insertion angle for oscillation		
25	EP	<_DP1>	REAL	Insertion depth per rev for helix		

No.	Parameter mask	Parameter internal	Data type	Meaning		
26		<_UMODE>	INT	Reserved		
27	FS	<_FS>	REAL	Chamfer width (inc) for chamfering		
28	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		
29		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining or just calculation of start point	
					1 =	Normal machining
				THOUSANDS:	Dimensioning of reference point, slot length	
					0 =	Center
					1 =	Inner left-hand +L
					2 =	Inner right-hand -L
					3 =	Left-hand edge +L
					4 =	Right-hand edge -L
30		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Software version identification	
					1 =	Function extension SLOT1
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple
31		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth Z1 (abs/inc)	
					0 =	Compatibility
					1 =	Z1 (inc)
					2 =	Z1 (abs)
				TENS:	Reserved	
				HUNDREDS:	Insertion depth for chamfering ZFS	
					0 =	ZFS (abs)
					1 =	ZFS (inc)

Note

The cycle is provided with new functions that are not on earlier software versions. Consequently certain parameters in the screen form (<NUM>, <RAD>, <INDA>) are no longer displayed. Multiple slots on one position pattern can be programmed using "MCALL" and calling the desired position pattern, e.g. HOLES2.

19.1.8 SLOT2 - circumferential slot

Syntax

```
SLOT2(<RTP>, <RFP>, <SDIS>, <_DP>, <_DPR>, <NUM>, <AFSL>, <WID>,
<_CPA>, <_CPO>, <RAD>, <STA1>, <INDA>, <FFD>, <FFP1>, <_MID>,
<CDIR>, <_FAL>, <VARI>, <_MIDF>, <FFP2>, <SSF>, <_FFCP>, <_UMODE>,
<_FS>, <_ZFS>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<RTP>	REAL	Retraction plane (abs)		
2	Z0	<RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Slot depth (abs)		
5		<_DPR>	REAL	Slot depth (inc) with respect to Z0 (enter without sign)		
6	N	<NUM>	INT	Number of slots		
7	α1	<AFSL>	REAL	Opening angle of the slot		
8	W	<WID>	REAL	Slot width		
9	X0	<_CPA>	REAL	Reference point = Center point of circle, 1st axis of the plane		
10	Y0	<_CPO>	REAL	Reference point = Center point of circle, 2nd axis of the plane		
11	R	<RAD>	REAL	Radius of the circle		
12	α0	<STA1>	REAL	Starting angle		
13	α2	<INDA>	REAL	Incrementing angle		
14	FZ	<FFD>	REAL	Depth infeed rate		
15	F	<FFP1>	REAL	Feedrate		
16	DZ	<_MID>	REAL	Maximum depth infeed		
17		<CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut
18	UXY	<_FAL>	REAL	Finishing allowance on plane or slot edge		

No.	Parameter mask	Parameter internal	Data type	Meaning		
19		<VARI>	INT	Machining type		
				UNITS:		
					0 =	Complete machining
					1 =	Roughing
					2 =	Finishing
					3 =	Edge finishing
					5 =	Chamfering
				TENS:		
					0 =	Intermediate positioning with G0 line
					1 =	Intermediate positioning on circular path
				HUNDREDS:	Reserved	
				THOUSANDS:		
					0 =	Compatibility mode, if <INDA> = 0 then full circle, <INDA> <> 0 then pitch circle
					1 =	Full circle
					2 =	Pitch circle
20	DZF	<_MIDF>	REAL	Reserved		
21		<FFP2>	REAL	Reserved		
22		<SSF>	REAL	Reserved		
23	FF	<_FFCP>	REAL	Reserved		
24		<_UMODE>	INT	Reserved		
25	FS	<_FS>	REAL	Chamfer width (inc)		
26	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		
27		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining or just calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining

No.	Parameter mask	Parameter internal	Data type	Meaning		
28		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Software version identification	
					1 =	SLOT2 functions as of software version 2.5
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple
29		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth Z1 (abs/inc)	
					0 =	Compatibility
					1 =	Z1 (inc)
					2 =	Z1 (abs)
				TENS:	Reserved	
				HUNDREDS:	Insertion depth for chamfering ZFS	
					0 =	ZFS (abs)
					1 =	ZFS (inc)

19.1.9 LONGHOLE - elongated hole

Syntax

```
LONGHOLE(<RTP>, <RFP>, <SDIS>, <_DP>, <_DPR>, <NUM>, <LENG>, <_CPA>,
<_CPO>, <RAD>, <STA1>, <INDA>, <FFD>, <FFP1>, <MID>, <_VARI>,
<_UMODE>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	RP	<RTP>	REAL	Retraction plane (abs)
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)

No.	Parameter mask	Parameter internal	Data type	Meaning		
4	Z1	<_DP>	REAL	Long hole depth (abs)		
5		<_DPR>	REAL	Long hole depth (inc) with respect to Z0 (enter without sign)		
6		<NUM>	INT	Number of long holes = 1		
7	L	<LENG>	REAL	Length of long hole		
8	X0	<_CPA>	REAL	Reference point in the 1st axis of the plane		
9	Y0	<_CPO>	REAL	Reference point in the 2nd axis of the plane		
10		<RAD>	REAL	Reserved		
11	α0	<STA1>	REAL	Angle of rotation		
12		<INDA>	REAL	Reserved		
13	FZ	<FFD>	REAL	Depth infeed rate		
14	F	<FFP1>	REAL	Feedrate		
15	DZ	<MID>	REAL	Maximum depth infeed		
16		<_VARI>	INT	Machining type		
				UNITS:	Infeed type	
					1 =	Vertically with G1
					3 =	Oscillation
				HUNDREDS:	Reserved	
17		<_UMODE>	INT	Reserved		
18		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining or just calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining
				THOUSANDS:	Dimensioning of reference point, slot length	
					0 =	Center
					1 =	Inner left-hand +L
					2 =	Inner right-hand -L
					3 =	Left-hand edge +L
					4 =	Right-hand edge -L

No.	Parameter mask	Parameter internal	Data type	Meaning		
19		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Type of feedrate: G group (G94/G95) for surface and depth feedrate	
					0 =	Compatibility mode
					1 =	G code as before cycle call. G94/G95 same for surface and depth feedrate
				HUNDREDS:	Reserved	
				THOUSANDS:	Software version identification	
					1 =	Function extension LONGHOLE (dimensioning of reference point)
20		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth Z1 (abs/inc)	
					0 =	Compatibility
					1 =	Z1 (inc)
					2 =	Z1 (abs)

Note

The cycle is provided with new functions that are not on earlier software versions. Consequently certain parameters in the screen form (<NUM>, <RAD>, <INDA>) are no longer displayed. Multiple slots on one position pattern can be programmed using "MCALL" and calling the desired position pattern, e.g. HOLES2.

19.1.10 CYCLE60 - engraving cycle

Syntax

```
CYCLE60(<_TEXT>, <_RTP>, <_RFP>, <_SDIS>, <_DP>, <_DPR>, <_PA>,
<_PO>, <_STA>, <_CP1>, <_CP2>, <_WID>, <_DF>, <_FFD>, <_FFP1>,
<_VARI>, <_CODEP>, <_UMODE>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1		<_TEXT>	STRING [200]	Text to be engraved (up to 100 characters)
2	RP	<_RTP>	REAL	Retraction plane (abs)
3	Z0	<_RFP>	REAL	Reference point of tool axis (abs)
4	SC	<_SDIS>	REAL	Safety clearance (to be added to the reference plane, enter without sign)
5	Z1	<_DP>	REAL	Depth (abs), see <_AMODE>
6	Z1	<_DPR>	REAL	Depth (inc), see <_AMODE>
7	X0	<_PA>	REAL	Reference point in 1st axis of plane (abs) - right-angled, see <_VARI>
	R			Reference point, length (radius) - polar, see <_VARI>
8	Y0	<_PO>	REAL	Reference point in 2nd axis of plane (abs) - right-angled, see <_VARI>
	α0			Reference point, angle with respect to 1st axis - polar, see <_VARI>
9	α1	<_STA>	REAL	Text direction, angle of line of text with respect to 1st axis, see <_VARI>
10	XM	<_CP1>	REAL	Center of circle of text, 1st axis of plane (abs) - right-angled, see <_VARI>
	LM			Center of circle of text, length (radius) with respect to WNP - polar, see <_VARI>
11	YM	<_CP2>	REAL	Center of circle of text, 2nd axis of plane (abs) - right-angled, see <_VARI>
	αM			Center of circle of text, angle with respect to 1st axis - polar, see <_VARI>
12	W	<_WID>	REAL	Height of characters (enter without sign)
13	DX1 DX2	<_DF>	REAL	Distance between characters / overall width, see <_VARI>
	α2			Opening angle, see <_VARI>
14	FZ	<_FFD>	REAL	Depth infeed rate, see <_DMODE>
15	F	<_FFP1>	REAL	Feedrate for surface machining

No.	Parameter mask	Parameter internal	Data type	Meaning		
16		<_VARI>	INT	Machining (alignment and reference point for engraved text)		
				UNITS:	Reference point	
					0 =	Right-angled
					1 =	Polar
				TENS:	Text alignment	
					0 =	Text on one line
					1 =	Text in an upward pointing arc
					2 =	Text in a downward curving arc
				HUNDREDS:	Reserved	
				THOUSANDS:	Reference point of the text, horizontal	
					0 =	Left
					1 =	Center
					2 =	Right
				TEN THOUSANDS:	Reference point of the text, vertical	
					0 =	Bottom
					1 =	Center
					2 =	Top
				HUNDRED THOUSANDS:	Text length	
					0 =	Character spacing
					1 =	Overall text width (linear text only)
					2 =	Opening angle (only for circular text)
				ONE MILLION:	Circle center	
					0 =	Right-angled (Cartesian)
					1 =	Polar
				TEN MILLIONS:	Mirror writing	
					0 =	Compatibility
					1 =	Mirror writing ON
					2 =	Mirror writing OFF
17		<_CODEP>	INT	Code page number for writing (currently only 1252)		
18		<_UMODE>	INT	Reserved		
19		_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining

No.	Parameter mask	Parameter internal	Data type	Meaning		
20		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Type of feedrate: G group (G94/G95) for surface and depth feedrate	
					0 =	Compatibility mode
					1 =	G code as before cycle call. G94/G95 same for surface and depth feedrate
21		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth (<_DP>, <_DPR>)	
					0 =	Compatibility
					1 =	Incremental (<_DPR>)
					2 =	Absolute (<_DP>)

19.1.11 CYCLE61 - Face milling

Syntax

```
CYCLE61(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_PA>, <_PO>, <_LENG>,
<_WID>, <_MID>, <_MIDA>, <_FALD>, <_FFP1>, <_VARI>, <_LIM>,
<_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	RP	<_RTP>	REAL	Retraction plane (abs)
2	Z0	<_RFP>	REAL	Reference point of tool axis, height of blank (abs)
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)
4	Z1	<_DP>	REAL	Height of finished part (abs/inc), see <_AMODE>
5	X0	<_PA>	REAL	Corner point 1 in 1st axis (abs)
6	Y0	<_PO>	REAL	Corner point 1 in 2nd axis (abs)
7	X1	<_LENG>	REAL	Corner point 2 in 1st axis (abs/inc), see <_AMODE>
8	Y1	<_WID>	REAL	Corner point 2 in 2nd axis (abs/inc), see <_AMODE>
9	DZ	<_MID>	REAL	Maximum depth infeed
10	DXY	<_MIDA>	REAL	Maximum plane infeed (for unit, see <_AMODE>)

No.	Parameter mask	Parameter internal	Data type	Meaning		
11	UZ	<_FALD>	REAL	Finishing allowance, depth		
12	F	<_FFP1>	REAL	Machining feedrate		
13		<_VARI>	INT	Machining type		
				UNITS:	Machining	
					1 =	Roughing
					2 =	Finishing
				TENS:	Machining direction	
					1 =	Parallel to the 1st axis, in one direction
					2 =	Parallel to the 2nd axis, in one direction
					3 =	Parallel to the 1st axis, varying direction
					4 =	Parallel to the 2nd axis, varying direction
14		<_LIM>	INT	Limits		
				UNITS:	Limit 1st axis negative	
					0 =	No
					1 =	Yes
				TENS:	Limit 1st axis positive	
					0 =	No
					1 =	Yes
				HUNDREDS:	Limit 2nd axis negative	
					0 =	No
					1 =	Yes
				THOUSANDS:	Limit 2nd axis positive	
					0 =	No
					1 =	Yes
15		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)

No.	Parameter mask	Parameter internal	Data type	Meaning		
16		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth (<_DP>)	
					0 =	Absolute
					1 =	Incremental
				TENS:	Units for plane infeed (<_MIDA>)	
					0 =	mm
					1 =	% of tool diameter
				HUNDREDS:	Reserved	
				THOUSANDS:	Length of surface	
					0 =	Incremental
					1 =	Absolute
				TEN THOUSANDS:	Width of surface	
					0 =	Incremental
					1 =	Absolute

19.1.12 CYCLE62 - contour call

Syntax

```
CYCLE62(<_KNAME>,<_TYPE>, <_LAB1>, <_LAB2>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	PRG/CON	<_KNAME>	STRING [140]	Contour name or subprogram name does not have to be programmed in _TYPE = 2		
2		<_TYPE>	INT	Determination of contour input		
					0 =	Subprogram
					1 =	Contour name
					2 =	Labels
					3 =	Labels in the subprogram
3	LAB1	<_LAB1>	STRING[32]	Label 1, start of contour		
4	LAB2	<_LAB2>	STRING[32]	Label 2, end of contour		

19.1.13 CYCLE63 - Milling contour pocket

Syntax

```
CYCLE63(<_PRG>, <_VARI>, <_RP>, <_Z0>, <_SC>, <_Z1>, <_F>, <_FZ>,
<_DXY>, <_DZ>, <_UXY>, <_UZ>, <_CDIR>, <_XS>, <_YS>, <_ER>, <_EP>,
<_EW>, <_FS>, <_ZFS>, <_TR>, <_DR>, <_UMODE>, <_GMODE>, <_DMODE>,
<_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	PRG	<_PRG>	STRING [100]	Name of removal program		
2		<_VARI>	INT	Machining type		
				UNITS:	Machining process	
					1 =	Roughing
					3 =	Base finishing
					4 =	Edge finishing
					5 =	Chamfering
				TENS:	Infeed type	
					0 =	Central insertion
					1 =	Helical insertion
					2 =	Oscillating insertion
				HUNDREDS:	Reserved	
				THOUSANDS:	Lift mode	
					0 =	Lift off to retraction plane
					1 =	Lift off to reference point + safety clearance
				TEN THOUSANDS:	Start point for roughing and finishing base	
					0 =	Auto
					1 =	Manual
3	RP	<_RP>	REAL	Retraction plane (abs)		
4	Z0	<_Z0>	REAL	Reference point of tool axis (abs)		
5	SC	<_SC>	REAL	Safety clearance (to be added to reference point, enter without sign)		
6	Z1	<_Z1>	REAL	Final depth (see <_AMODE> UNITS)		
7	F	<_F>	REAL	Feedrate in the plane during roughing/finishing		
8	FZ	<_FZ>	REAL	Depth infeed rate		
9	DXY	<_DXY>	REAL	Infeed plane - unit (see <_AMODE> TENS)		
10	DZ	<_DZ>	REAL	Depth infeed		
11	UXY	<_UXY>	REAL	Finishing allowance, plane		
12	UZ	<_UZ>	REAL	Finishing allowance, depth		
13		<_CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut

No.	Parameter mask	Parameter internal	Data type	Meaning		
14	XS	<_XS>	REAL	Starting point X, absolute		
15	YS	<_YS>	REAL	Starting point Y, absolute		
16	ER	<_ER>	REAL	Helical insertion: Radius		
17	EP	<_EP>	REAL	Helical insertion: Pitch		
18	EW	<_EW>	REAL	Oscillating insertion: Maximum insertion angle		
19	FS	<_FS>	REAL	Chamfer width (inc) for chamfering		
20	ZFS	<_ZFS>	REAL	Insertion depth of tool tip when chamfering (see <_AMODE> HUNDREDS)		
21	TR	<_TR>	STRING[32]	Reference tool name when machining residual material		
22	DR	<_DR>	INT	Reference tool D number when machining residual material		
23		<_UMODE>	INT	Reserved		
24		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Normal machining (no compatibility mode needed)
					1 =	Normal machining
					2 =	Reserved
25		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Technology mode	
					1 =	Pocket
					2 =	Spigot
				THOUSANDS:	Machine residual material	
					0 =	No
					1 =	Yes
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
26		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth (Z1)	
					0 =	Absolute (compatibility mode)
					1 =	Incremental
				TENS:	Unit for plane infeed (DXY)	
					0 =	mm
					1 =	% of tool diameter
				HUNDREDS:	Insertion depth for chamfering (ZFS)	
					0 =	Absolute
					1 =	Incremental
				THOUSANDS:	---	Reserved

19.1.14 CYCLE64 - Predrilling contour pocket

Syntax

```
CYCLE64(<_PRG>, <_VARI>, <_RP>, <_Z0>, <_SC>, <_Z1>, <_F>, <_DXY>,
<_UXY>, <_UZ>, <_CDIR>, <_TR>, <_DR>, <_UMODE>, <_GMODE>, <_DMODE>,
<_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	PRG	<_PRG>	STRING [100]	Name of drilling/centering program		
2		<_VARI>	INT	Machining type		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Lift mode	
					0 =	Lift off to retraction plane
					1 =	Lift off to reference point + safety clearance
3	RP	<_RP>	REAL	Retraction plane (abs)		
4	Z0	<_Z0>	REAL	Reference point (abs)		
5	SC	<_SC>	REAL	Safety clearance (to be added to reference point, enter without sign)		
6	Z1	<_Z1>	REAL	Drilling/centering depth (see <_AMODE> UNITS)		
7	F	<_F>	REAL	Drilling/centering feedrate		
8	DXY	<_DXY>	REAL	Infeed plane - unit (see <_AMODE> TENS)		
9	UXY	<_UXY>	REAL	Finishing allowance, plane		
10	UZ	<_UZ>	REAL	Finishing allowance, depth		

No.	Parameter mask	Parameter internal	Data type	Meaning		
11		<_CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut
12	TR	<_TR>	STRING[20]	Reference tool name		
13	DR	<_DR>	INT	Reference tool D number		
14		<_UMODE>	INT	Reserved		
15		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Normal machining (no compatibility mode needed)
					1 =	Normal machining
					2 =	Reserved
25		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Technology mode	
					1 =	Predrilling
					2 =	Centering
26		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling/centering depth Z1	
					0 =	Absolute (compatibility mode)
					1 =	Incremental
				TENS:	Unit for plane infeed (DXY)	
					0 =	mm
					1 =	% of tool diameter

19.1.15 CYCLE70 - thread milling

Syntax

```
CYCLE70(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_DIATH>, <_H1>, <_FAL>,
<_PIT>, <_NT>, <_MID>, <_FFR>, <_TYPTH>, <_PA>, <_PO>, <_NSP>,
<_VARI>, <_PITA>, <_PITM>, <_PTAB>, <_PTABA>, <_GMODE>, <_DMODE>,
<_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Thread length (abs, inc), see <_AMODE> Take account of runout at base of hole (at least half pitch)		
5	∅	<_DIATH>	REAL	Nominal diameter of the thread		
6	H1	<_H1>	REAL	Thread depth		
7	U	<_FAL>	REAL	Finishing allowance		
8	P	<_PIT>	REAL	Pitch (select <_PITA>: mm, inch, MODULE, threads/inch)		
9	NT	<_NT>	INT	Number of teeth on the tool tip Tool length is always with respect to bottom tooth.		
10	DXY	<_MID>	REAL	Maximum infeed per cut <_MID> > <_H1>: All in one cut		
11	F	<_FFR>	REAL	Milling feed		
12		<_TYPTH>	INT	Thread type	0 =	Internal thread
					1 =	External thread
13	X0	<_PA>	REAL	Circle center 1st axis (abs)		
14	Y0	<_PO>	REAL	Circle center 2nd axis (abs)		
15	αS	<_NSP>	REAL	Start angle (multi-start thread)		
16		<_VARI>	INT	Machining type		
				UNITS:		
					1 =	Roughing
					2 =	Finishing
				TENS:		
					1 =	From top to bottom
					2 =	From bottom to top
				HUNDREDS:		
					0 =	Right-hand thread
					1 =	Left-hand thread
17		<_PITA>	INT	Evaluation of thread pitch		
					0 =	Compatibility mode
					1 =	Pitch in mm
					2 =	Pitch in threads per inch (TPI)
					3 =	Pitch in inches
					4 =	Pitch as MODULE
18		<_PITM>	STRING[15]	String as marker for pitch input (for the interface only)		
19		<_PTAB>	STRING[20]	String for thread table ("", "ISO", "BSW", "BSP", "UNC") (for the interface only)		
20		<_PTABA>	STRING[20]	String for selection from thread table (e.g. "M 10", "M 12", ...) (for the interface only)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
21		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Machining/calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining
22		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
23		<_AMODE>	INT	Alternative mode		
				UNITS:	Thread length (<_DP>)	
					0 =	Absolute
					1 =	Incremental

19.1.16 CYCLE72 - Path milling

Syntax

```
CYCLE72(<_KNAME>, <_RTP>, <_RFP>, <_SDIS>, <_DP>, <_MID>, <_FAL>,
<_FALD>, <_FFP1>, <_FFD>, <_VARI>, <_RL>, <_AS1>, <_LP1>, <_FF3>,
<_AS2>, <_LP2>,<_UMODE>, <_FS>, <_ZFS>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1		<_KNAME>	STRING [141]	Name of the contour subprogram
2	RP	<_RTP>	REAL	Retraction plane (abs)
3	Z0	<_RFP>	REAL	Reference point of tool axis (abs)
4	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)
5	Z1	<_DP>	REAL	End point, final depth (abs/inc), see <_AMODE>
6	DZ	<_MID>	REAL	Maximum depth infeed (inc; enter without sign)
7	UXY	<_FAL>	REAL	Finishing allowance, plane (inc), allowance at edge contour
8	UZ	<_FALD>	REAL	Finishing allowance depth (inc), allowance at base (enter without sign)
9	FX	<_FFP1>	REAL	Feedrate on contour
10	FZ	<_FFD>	REAL	Feedrate for depth infeed (or spatial infeed)

No.	Parameter mask	Parameter internal	Data type	Meaning		
11		<_VARI>	INT	Machining type		
				UNITS:	Machining	
					1 =	Roughing
					2 =	Finishing
					5 =	Chamfering
				TENS:		
					0 =	Intermediate paths with G0
					1 =	Intermediate paths with G1
				HUNDREDS:	Retraction at the end of contour	
					0 =	Retraction at the end of contour to reference point
					1 =	Retraction at the end of contour to reference point + <_SDIS>
					2 =	Retraction at the end of contour by <_SDIS>
					3 =	No retraction at the end of contour, approach next start point with contour feed
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Machine contour	
					0 =	Machine contour forward
					1 =	Machine contour backward Restrictions with backward machining: • Max 170 contour elements (including chamfers or rounding) • Only values in the (X/Y) and F planes are evaluated
12		<_RL>	INT	Machining direction		
					40 =	Center of contour (G40, approach and retract: straight line or vertical)
					41 =	Left of contour (G41, approach and retract: straight line or circle)
					42 =	Right of contour (G42, approach and retract: straight line or circle)

No.	Parameter mask	Parameter internal	Data type	Meaning		
13		<_AS1>	INT	Contour approach movement		
				UNITS:		
					1 =	Straight line
					2 =	Quadrant
					3 =	Semi-circle
					4 =	Approach and retraction vertically
				TENS:		
					0 =	Last movement, in the plane
					1 =	Last movement, spatial
14	L1	<_LP1>	REAL	Approach path or approach radius (inc; enter without sign)		
15	FZ	<_FF3>	REAL	Feedrate for intermediate paths (G94/G95 as to contour)		
16		<_AS2>	INT	Contour approach movement (not vertical approach/retract)		
				UNITS:		
					1 =	Straight line
					2 =	Quadrant
					3 =	Semi-circle
				TENS:		
					0 =	Last movement, in the plane
					1 =	Last movement, spatial
17	L2	<_LP2>	REAL	Retract path or retract radius (inc, to be entered without sign)		
18		<_UMODE>	INT	Reserved		
19	FS	<_FS>	REAL	Chamfer width (inc)		
20	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		
21		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining

No.	Parameter mask	Parameter internal	Data type	Meaning		
22		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Type of feedrate: G group (G94/G95) for surface and depth feedrate	
					0 =	Compatibility mode
					1 =	G code as before cycle call. G94/G95 same for surface and depth feedrate
				THOUSANDS:		
					0 =	Compatibility mode: Contour name is in <_KNAME>
					1 =	Contour name is programmed in CYCLE62 and transferred to _SC_CONT_NAME
23		<_AMODE>	INT	Alternative mode		
				UNITS:	End point Z1 (<_DP>)	
					0 =	Absolute (compatibility mode)
					1 =	Incremental
				TENS:	Units for plane infeed	
					0 =	mm, inch
					1 =	Reserved
				HUNDREDS:	Insertion depth for chamfering (<_ZFS>)	
					0 =	Absolute
					1 =	Incremental

Note

If the following transfer parameters are programmed indirectly (as parameters), the screen form is not reset:

<_VARI>, <_RL>, <_AS1>, <_AS2>, <_UMODE>, <_GMODE>, <_DMODE>, <_AMODE>

19.1.17 CYCLE76 - rectangular spigot milling

Syntax

```
CYCLE76(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_DPR>, <_LENG>, <_WID>,
<_CRAD>, <_PA>, <_PO>, <_STA>, <_MID>, <_FAL>, <_FALD>, <_FFP1>,
<_FFD>, <_CDIR>, <_VARI>, <_AP1>, <_AP2>, <_FS>, <_ZFS>, <_GMODE>,
<_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Spigot depth (abs)		
5		<_DPR>	REAL	Spigot depth (inc) with respect to Z0 (enter without sign)		
6	L	<_LENG>	REAL	Spigot length, see <_GMODE> (enter without sign)		
7	W	<_WID>	REAL	Spigot width, see <_GMODE> (enter without sign)		
8	R	<_CRAD>	REAL	Spigot corner radius (enter without sign)		
9	X0	<_PA>	REAL	Reference point for spigot in 1st axis of plane (abs)		
10	Y0	<_PO>	REAL	Reference point for spigot in 2nd axis of plane (abs)		
11	α0	<_STA>	REAL	Angle of rotation, angle between longitudinal axis (L) and 1st axis of plane		
12	DZ	<_MID>	REAL	Maximum depth infeed (inc; enter without sign)		
13	UXY	<_FAL>	REAL	Finishing allowance, plane (inc), allowance at edge contour		
14	UZ	<_FALD>	REAL	Finishing allowance depth (inc), allowance at base (enter without sign)		
15	FX	<_FFP1>	REAL	Feedrate on contour		
16	FZ	<_FFD>	REAL	Depth infeed rate		
17		<_CDIR>	INT	Milling direction (enter without sign)		
				UNITS:		
					0 =	Down-cut
					1 =	Up-cut
18		<_VARI>	INT	Machining		
				UNITS:		
					1 =	Roughing
					2 =	Finishing
					5 =	Chamfering
19	L1	<_AP1>	REAL	Length of blank spigot		
20	W1	<_AP2>	REAL	Width of blank spigot		
21	FS	<_FS>	REAL	Chamfer width (inc)		
22	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs, inc), see <_AMODE>		

No.	Parameter mask	Parameter internal	Data type	Meaning		
23		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining or just calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining
				THOUSANDS:	Dimensioning of spigot acc. to center or corner	
					0 =	Compatibility mode
					1 =	Dimensioning via center
					2 =	Dimensioning of corner point, spigot +L +W
					3 =	Dimensioning of corner point, spigot -L +W
					4 =	Dimensioning of corner point, spigot +L -W
					5 =	Dimensioning of corner point, spigot -L -W
				TEN THOUSANDS:	Complete machining or remachining	
					0 =	Compatibility mode
					1 =	Complete machining
					2 =	Post machining
24		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	---	Reserved
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
25		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth Z1 (DP)	
					0 =	Compatibility
					1 =	Incremental
					2 =	Absolute
				TENS:	Reserved	
				HUNDREDS:	Insertion depth for chamfering (ZFS)	
					0 =	Absolute
					1 =	Incremental

19.1.18 CYCLE77 - circular spigot milling

Syntax

```
CYCLE77(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_DPR>, <_CDIAM>, <_PA>,
<_PO>, <_MID>, <_FAL>, <_FALD>, <_FFP1>, <_FFD>, <_CDIR>, <_VARI>,
<_AP1>, <_FS>, <_ZFS>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Spigot depth (abs)		
5		<_DPR>	REAL	Spigot depth (inc) with respect to Z0 (enter without sign)		
6	∅	<_CDIAM>	REAL	Spigot diameter (enter without sign)		
7	X0	<_PA>	REAL	Reference point for spigot in 1st axis of plane (abs)		
8	Y0	<_PO>	REAL	Reference point for spigot in 2nd axis of plane (abs)		
9	DZ	<_MID>	REAL	Maximum depth infeed (inc; enter without sign)		
10	UXY	<_FAL>	REAL	Finishing allowance, plane (inc), allowance at edge contour		
11	UZ	<_FALD>	REAL	Finishing allowance depth (inc), allowance at base (enter without sign)		
12	FX	<_FFP1>	REAL	Feedrate on contour		
13	FZ	<_FFD>	REAL	Depth infeed rate		
14		<_CDIR>	INT	Milling direction (enter without sign)		
				UNITS:		
					0 =	Down-cut
					1 =	Up-cut

No.	Parameter mask	Parameter internal	Data type	Meaning		
15		<_VARI>	INT	Machining type		
				UNITS:	Machining	
					1 =	Roughing to final machining allowance
					2 =	Finishing (allowance X/Y/Z=0)
					5 =	Chamfering
16	∅1	<_AP1>	REAL	Diameter of blank spigot		
17	FS	<_FS>	REAL	Chamfer width (inc)		
18	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		
19		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Compatibility mode
					1 =	Normal machining
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Complete machining/remachining	
					0 =	Compatibility mode (process <_AP1> as before)
					1 =	Complete machining
					2 =	Post machining
20		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	---	Reserved
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
21		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth Z1 (DP)	
					0 =	Absolute (compatibility mode)
					1 =	Incremental
					2 =	Absolute
				TENS:	Reserved	
				HUNDREDS:	Insertion depth for chamfering (ZFS)	
					0 =	Absolute
					1 =	Incremental

19.1.19 CYCLE78 - Drill thread milling

Syntax

```
CYCLE78(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_ADPR>, <_FDPR>, <_LDPR>,
<_DIAM>, <_PIT>, <_PITA>, <_DAM>, <_MDEP>, <_VARI>, <_CDIR>, <_GE>,
<_FFD>, <_FRDP>, <_FFR>, <_FFP2>, <_FFA>, <_PITM>, <_PTAB>,
<_PTABA>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Final drilling depth (abs/inc), see <_AMODE>		
5		<_ADPR>	REAL	Predrilling depth with reduced drilling feedrate (inc) effective with <_VARI> TEN THOUSANDS		
6	D	<_FDPR>	REAL	Maximum depth infeed (inc) D ≥ Z1 ⇒ One infeed to the final drilling depth D < Z1 ⇒ Deep drilling cycle with multiple infeeds and chip removal		
7	ZR	<_LDPR>	REAL	Remaining drilling depth when through-drilling (inc) with FR feed		
8	∅	<_DIAM>	REAL	Nominal diameter of the thread		
9	P	<_PIT>	REAL	Pitch as a numerical value		
10		<_PITA>	INT	Evaluation of thread pitch P		
					1 =	Pitch in mm/rev
					2 =	Pitch in threads/inch
					3 =	Pitch in inch/rev
					4 =	Pitch as MODULE
11	DF	<_DAM>	REAL	Absolute value / percentage for each additional infeed (degression), see<_AMODE>		

No.	Parameter mask	Parameter internal	Data type	Meaning		
12	V1	<_MDEP>	REAL	Minimum infeed (inc), only active for degression		
13		<_VARI>	INT	Machining type		
				UNITS:	Reserved	
				TENS:	Swarf removal before thread milling	
					0 =	No chip removal before thread milling (only active at final drilling depth)
					1 =	Chip removal before thread milling (only active at final drilling depth)
				HUNDREDS:	Right-hand/left-hand threads	
					0 =	Right-hand thread
					1 =	Left-hand thread
				THOUSANDS:	Remaining drilling depth with drilling feedrate	
					0 =	No remaining drilling depth with drilling feedrate FR
					1 =	Remaining drilling depth with drilling feedrate FR
				TEN THOUSANDS:	Predrilling with reduced feedrate	
					0 =	No predrilling with reduced feedrate
					1 =	Predrilling with reduced feedrate Predrilling feedrate = 0.3 F1, if F1 < 0.15 mm/rev Predrilling feedrate = 0.1 mm/rev, if F1 ≥ 0.15 mm/rev
14		<_CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut
					4 =	Up-cut + down-cut (combined roughing + finishing)
15	Z2	<_GE>	REAL	Retraction distance before thread milling (inc)		
16	F1	<_FFD>	REAL	Drilling feedrate (mm/min or in/min or mm/rev)		
17	FR	<_FRDP>	REAL	Drilling feedrate for remaining drilling depth (mm/min or mm/rev)		
18	F2	<_FFR>	REAL	Feedrate for thread milling (mm/min or mm/tooth)		
19	FS	<_FFP2>	REAL	Finishing feedrate for <_CDIR> =4 (mm/min or mm/tooth)		
20		<_FFA>	INT	Evaluation of feedrates		
				UNITS:	Drilling feed F1	
				TENS:	Drilling feedrate for remaining drilling depth FR	
				HUNDREDS:	Feedrate for thread milling F2	
				THOUSANDS:	Finishing feedrate FS	
21		<_PITM>	STRING[15]	String as marker for pitch input (for the interface only)[1)]		
22		<_PTAB>	STRING[20]	String for thread table ("", "ISO", "BSW", "BSP", "UNC") (for the interface only)[1)]		

No.	Parameter mask	Parameter internal	Data type	Meaning		
23		<_PTABA>	STRING[20]	String for selection from thread table (e.g. "M 10", "M 12", ...) (for the interface only)[1)]		
24		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data), reserved		
25		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
26		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth = Final drilling depth Z1 abs/inc	
					0 =	Absolute
					1 =	Incremental
				TENS:	Absolute value / percentage DF for each additional infeed (degression)	
					0 =	Absolute value
					1 =	Percentage (0.001 to 100%)

Note

[1)] Parameters 21, 22 and 23 are only used for thread selection in the screen form thread tables. The thread tables cannot be accessed via cycle definition in the cycle run time.

19.1.20 CYCLE79 - multi-edge

Syntax

```
CYCLE79(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_NUM>, <_SWL>, <_PA>,
<_PO>, <_STA>, <_RC>, <_AP1>, <_MIDA>, <_MID>, <_FAL>, <_FALD>,
<_FFP1>, <_CDIR>, <_VARI>, <_FS>, <_ZFS>, <_GMODE>, <_DMODE>,
<_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	RP	<_RTP>	REAL	Retraction plane (abs)
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)

No.	Parameter mask	Parameter internal	Data type	Meaning		
4	Z1	<_DP>	REAL	Multiple-edge depth (abs/inc), see <_AMODE>		
5	N	<_NUM>	INT	Number of edges (1...n)		
6	SW/L	<_SWL>	REAL	Width across flats or edge length (depending on <_VARI>) ("SW" for width across flats, "L" for edge length) Width across flats only if even number of edges, and single edge		
7	X0	<_PA>	REAL	Spigot reference point, 1st axis (abs)		
8	Y0	<_PO>	REAL	Spigot reference point, 2nd axis (abs)		
9	α0	<_STA>	REAL	Angle of rotation, center of edge against 1st axis (X axis)		
10	R1/FS1	<_RC>	REAL	Corner rounding with <_NUM> > 2 (radius/chamfer, see <_AMODE>) (inc, to be entered without sign) ("R1" for radius, "FS1" for chamfer)		
11	∅	<_AP1>	REAL	Unmachined diameter of spigot		
12	DXY	<_MIDA>	REAL	Maximum infeed width (for unit, see <_AMODE>)		
13	DZ	<_MID>	REAL	Maximum depth infeed		
14	UXY	<_FAL>	REAL	Finishing allowance, plane		
15	UZ	<_FALD>	REAL	Finishing allowance, depth		
16	F	<_FFP1>	REAL	Machining feedrate		
17		<_CDIR>	INT	Milling direction	0 =	Down-cut
					1 =	Up-cut
18		<_VARI>	INT	Machining type		
				UNITS:	Machining	
					1 =	Roughing
					2 =	Finishing
					3 =	Edge finishing
					5 =	Chamfering
				TENS:	Width across flats or edge length	
					0 =	Width across flats
					1 =	Edge length
19	FS	<_FS>	REAL	Chamfer width (inc)		
20	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		
21		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining or just calculation of start point	
					1 =	Normal machining

No.	Parameter mask	Parameter internal	Data type	Meaning		
22		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/18/19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	---	Reserved
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple
23		<_AMODE>	INT	Alternative mode		
				UNITS:	Final depth (<_DP>)	
					0 =	Absolute
					1 =	Incremental
				TENS:	Units for plane infeed (<_MIDA>)	
					0 =	mm
					1 =	% of tool diameter
				HUNDREDS:	Insertion depth for chamfering (<_ZFS>)	
					0 =	Absolute
					1 =	Incremental
				THOUSANDS:	Corner rounding (<_RC>)	
					0 =	Radius
					1 =	Chamfer

19.1.21 CYCLE81 - drilling, centering

Syntax

```
CYCLE81(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <DTB>, <_GMODE>,
<_DMODE>, <_AMODE>)
```

Parameter

No.	Parameter mask	Parameter internal	Data type	Meaning
1	RP	<RTP>	REAL	Retraction plane (abs)
2	Z0	<RFP>	REAL	Reference point (abs)

No.	Parameter mask	Parameter internal	Data type	Meaning		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1/∅	<DP>	REAL	Drilling depth (abs) / centering diameter (abs), see <_GMODE>		
5	Z1	<DPR>	REAL	Drilling depth (inc)		
6	DT	<DTB>	REAL	Dwell time at final drilling depth, see <_AMODE>		
7		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Centering with respect to depth/diameter	
					0 =	Compatibility, depth
					1 =	Diameter
8		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
9		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from DP/DPR programming
					1 =	Incremental
					2 =	Absolute
				TENS:	Dwell time at final drilling depth DT in seconds/revolutions	
					0 =	Compatibility, from DTB sign (> 0 seconds or < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions

19.1.22 CYCLE82 - drilling, counterboring

Syntax

```
CYCLE82(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <DTB>, <_GMODE>,
<_DMODE>, <_AMODE>, <_VARI>, <S_ZA>, <S_FA>, <S_ZD>, <S_FD>)
```

Parameter

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<RTP>	REAL	Retraction plane (abs)		
2	Z0	<RFP>	REAL	Reference point (abs)		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<DP>	REAL	Drilling depth (abs), see <_AMODE>		
5	Z1	<DPR>	REAL	Drilling depth (inc), see <_AMODE>		
6	DT	<DTB>	REAL	Dwell time at final drilling depth, see <_AMODE>		
7		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Drilling depth with respect to tip/shank	
					0 =	Compatibility, tip
					1 =	Shank
8		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Basic

No.	Parameter mask	Parameter internal	Data type	Meaning		
9		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from DP/DPR programming
					1 =	Incremental
					2 =	Absolute
				TENS:	Dwell time DT at final drilling depth in seconds/revolutions	
					0 =	Compatibility, from DT sign (> 0 seconds / < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions
				HUNDREDS:	Drilling depth ZA abs/inc	
					0 =	Incremental
					1 =	Absolute
				THOUSANDS:	Evaluation of predrilling feedrate	
					0 =	As % of drilling feedrate
					1 =	F/min
					2 =	F/rev
				TEN THOUSANDS:	Remaining drilling depth ZD abs/inc	
					0 =	Incremental
					1 =	Absolute
				HUNDRED THOUSANDS:	Evaluation of remaining drilling feedrate	
					0 =	As % of drilling feedrate
					1 =	F/min
					2 =	F/rev
10		<_VARI>	INT	Predrilling/through-drilling machining type		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Through drilling	
					0 =	Through drilling "No"
					1 =	Through drilling "Yes"
				TEN THOUSANDS:	Predrilling	
					0 =	Predrilling "No"
					1 =	Predrilling "Yes"
11	ZA	<S_ZA>	REAL	Incremental predrilling depth in relation to reference point or absolute (see <_AMODE> HUNDREDS)		
12	FA	<S_FA>	REAL	Predrilling feedrate as value or in % (in conjunction with <_AMODE> THOUSANDS)		

No.	Parameter mask	Parameter internal	Data type	Meaning
13	ZD	<S_ZD>	REAL	Incremental remaining drilling depth in relation to final drilling depth or absolute (see <_AMODE> TEN THOUSANDS)
14	FD	<S_FD>	REAL	Remaining drilling feedrate as value or in % (in conjunction with <_AMODE> HUNDRED THOUSANDS)

19.1.23 CYCLE83 - deep-hole drilling

Syntax

```
CYCLE83(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <FDEP>, <FDPR>, <_DAM>,
<DTB>, <DTS>, <FRF>, <VARI>, <_AXN>, <_MDEP>, <_VRT>, <_DTD>,
<_DIS1>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameter

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<RTP>	REAL	Retraction plane (abs)		
2	Z0	<RFP>	REAL	Reference point (abs)		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<DP>	REAL	Final drilling depth (abs), see <_AMODE>		
5	Z1	<DPR>	REAL	Final drilling depth (inc), see <_AMODE>		
6	D	<FDEP>	REAL	1st drilling depth (abs), see <_AMODE>		
7	D	<FDPR>	REAL	1st drilling depth (inc), see <_AMODE>		
8	DF	<_DAM>	REAL	Degression value / percentage for each additional infeed, see <_AMODE>		
9	DTB	<DTB>	REAL	Dwell time at drilling depth, see <_AMODE>		
10	DTS	<DTS>	REAL	Dwell time at start point (for chip removal only), see <_AMODE>		
11	FD1	<FRF>	REAL	Percentage for the feedrate for the first infeed, see <_AMODE>		
12		<VARI>	INT	Machining type		
				UNITS:	Chip breaking/removal	
					0 =	Chip breaking
					1 =	Swarf removal
13		<_AXN>	INT	Tool axis		
					0 =	3rd geometry axis
					1 =	1st geometry axis
					2 =	2nd geometry axis
					> 2	3rd geometry axis
14	V1	<_MDEP>	REAL	Minimum infeed (only for degression percentage)		
15	V2	<_VRT>	REAL	Retraction distance after each machining step (for chip breaking only)		
					> 0	Variable retraction distance
					0 =	Default value 1 mm

No.	Parameter mask	Parameter internal	Data type	Meaning		
16	DT	<_DTD>	REAL	Dwell time at final drilling depth, see <_AMODE>		
17	V3	<_DIS1>	REAL	Limit distance (for chip removal only), see <_AMODE>		
18		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Drilling depth with respect to tip/shank	
					0 =	Tip
					1 =	Shank
19		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	---	Reserved
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
20		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth = Final drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from programming <DP>/<DPR>
					1 =	Incremental
					2 =	Absolute
				TENS:	Dwell time at drilling depth DTB in seconds/revolutions	
					0 =	Compatibility, from DTB sign (> 0 seconds or < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions
				HUNDREDS:	Dwell time at start point of DTS in seconds/revolutions	
					0 =	Compatibility, from DTS sign (> 0 seconds or < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions
				THOUSANDS:	Dwell time at final drilling depth DTD in seconds/revolutions	
					0 =	Compatibility, from DTD sign (> 0 seconds or < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions
				TEN THOUSANDS:	1st drilling depth D (abs/inc)	
					0 =	Compatibility, from programming <FDEPF>/<DPR>
					1 =	Incremental
					2 =	Absolute
				HUNDRED THOUSANDS:	Degression value / percentage <_DAM> for each additional infeed	
					0 =	Compatibility, from <_DAM> sign (> 0 degression value or < 0 factor 0.001 to 1.0)
					1 =	Degression value
					2 =	Percentage (0.001 to 100%)
				ONE MILLION:	Limit distance V3 automatic/manual	
					0 =	Compatibility from <_DIS1> sign (= 0 automatic or > 0 manual)
					1 =	Automatic (calculated in the cycle)
					2 =	Manual (programmed value)
				TEN MILLIONS:	Feedrate factor for first infeed <FRF> as factor/percentage	

No.	Parameter mask	Parameter internal	Data type	Meaning		
					0 =	Compatibility, as a factor (0.001 to 1.0, FRF = 0 means 100%)
					1 =	Percentage (0.001 to 999.999%)

19.1.24 CYCLE84 - tapping without compensating chuck

Syntax

```
CYCLE84(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <DTB>, <SDAC>, <MPIT>,
<PIT>, <POSS>, <SST>, <SST1>, <_AXN>, <_PITA>, <_TECHNO>, <_VARI>,
<_DAM>, <_VRT>, <_PITM>, <_PTAB>, <_PTABA>, <_GMODE>, <_DMODE>,
<_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<RTP>	REAL	Retraction plane (abs)		
2	Z0	<RFP>	REAL	Reference point (abs)		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<DP>	REAL	Drilling depth = final drilling depth (abs), see <_AMODE>		
5	Z1	<DPR>	REAL	Drilling depth = final drilling depth (inc), see <_AMODE>		
6	DT	<DTB>	REAL	Dwell time at drilling depth in seconds		
7	SDE	<SDAC>	INT	Direction of rotation after end of cycle		
8		<MPIT>	REAL	Thread size for "ISO metric" only (pitch is calculated internally during run time)		
9	P	<PIT>	REAL	Pitch as a value, for unit see <_PITA>		
10	αS[1)]	<POSS>	REAL	Spindle position for oriented spindle stop		
11	S	<SST>	REAL	Spindle speed for tapping		
12	SR	<SST1>	REAL	Spindle speed for retraction		
13		<_AXN>	INT	Drilling axis	0 =	3rd geometry axis
					1 =	1st geometry axis
					2 =	2nd geometry axis
					≥ 3 =	3rd geometry axis

No.	Parameter mask	Parameter internal	Data type	Meaning			
14		<_PITA>	INT	Pitch unit (evaluation of <PIT> and <MPIT>)			
					0 =	Pitch in mm	- evaluation<MPIT>/<PIT>
					1 =	Pitch in mm	- evaluation<PIT>
					2 =	Pitch in TPI	- evaluation of <PIT> (threads per inch)
					3 =	Pitch in inches	- evaluation<PIT>
					4 =	MODULUS	- evaluation<PIT>
15		<_TECHNO>	INT	Technology[1]			
				UNITS:	Exact stop response		
					0 =	Exact stop response active as before cycle call	
					1 =	Exact stop G601	
					2 =	Exact stop G602	
					3 =	Exact stop G603	
				TENS:	Feedforward control		
					0 =	With/without feedforward control active as before cycle call	
					1 =	With feedforward control FFWON	
					2 =	Without feedforward control FFWOF	
				HUNDREDS:	Acceleration		
					0 =	SOFT/BRISK/DRIVE active as before cycle call	
					1 =	With jerk limitation SOFT	
					2 =	Without jerk limitation BRISK	
					3 =	Reduced acceleration DRIVE	
				THOUSANDS:	MCALL spindle mode		
					0 =	Reactivate spindle operation for MCALL	
					1 =	For MCALL remain in position control	
16		<_VARI>	INT	Machining type			
				UNITS:	0 =	1 cut	
					1 =	Chip breaking (deep hole tapping)	
					2 =	Chip removal (deep hole tapping)	
				THOUSANDS:	ISO/SIEMENS mode not relevant for screen form		
					0 =	Call from ISO compatibility	
					1 =	Call from SIEMENS context	
17	D	<_DAM>	REAL	Maximum depth infeed (for chip removal/breaking only)			
18	V2	<_VRT>	REAL	Retraction distance after each machining step (for chip breaking only), see <_AMODE>			
19		<_PITM>	STRING[15]	String as marker for pitch input[2]			
20		<_PTAB>	STRING[5]	String for thread table ("", "ISO", "BSW", "BSP", "UNC")[2]			
21		<_PTABA>	STRING[20]	String for selection from thread table (e.g. "M 10", "M 12", ...)[2]			

No.	Parameter mask	Parameter internal	Data type	Meaning		
22		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
23		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Compatibility mode (for recompilation screen form only), if MD 52216 bit0 = 1[1)]	
					0 =	Technology parameters are displayed (compatibility): TECHNO parameters effective
					1 =	Technology parameters are not displayed: Technology active "as before cycle call"
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
24		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth = Final drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from programming <DP>/<DPR>
					1 =	Incremental
					2 =	Absolute
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Thread direction of rotation right/left	
					0 =	Compatibility, from PIT/MPTI sign
					1 =	Right
					2 =	Left
				TEN THOUSANDS:	Reserved	
				HUNDRED THOUSANDS:	Reserved	
				ONE MILLION:	Retraction distance after each machining step V2 manual/automatic	
					0 =	Compatibility, from <_VRT> programming (> 0 variable value or ≤ 0 standard value 1 mm / 0.0394 inch)
					1 =	Automatic (standard value 1 mm / 0.0394 inch)
					2 =	Manual (programmed as under V2)

[1] Technology fields may be hidden, depending on the setting date SD52216 $MCS_FUNCTION_MASK_DRILL

[2] Parameters 19, 20 and 21 are only used for thread selection in the screen form thread tables. The thread tables cannot be accessed via cycle definition in the cycle run time.

19.1.25 CYCLE85 - reaming

Syntax

```
CYCLE85(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <DTB>, <FFR>, <RFF>,
<_GMODE>, <_DMODE>, <_AMODE>)
```

Parameter

No.	Parameter mask	Parameter internal	Data type	Meaning
1	RP	<RTP>	REAL	Retraction plane (abs)
2	Z0	<RFP>	REAL	Reference point (abs)
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)
4	Z1	<DP>	REAL	Drilling depth (abs), see <_AMODE>
5	Z1	<DPR>	REAL	Drilling depth (inc), see <_AMODE>

No.	Parameter mask	Parameter internal	Data type	Meaning		
6	DT	<DTB>	REAL	Dwell time at final drilling depth, see <_AMODE>		
7	F	<FFR>	REAL	Feedrate		
8	FR	<RFF>	REAL	Feedrate during retraction		
9		<_GMODE>	INT	Reserved		
10		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
11		<_AMODE>	INT	Alternative mode (drilling)		
				UNITS:	Drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from DP/DPR programming
					1 =	Incremental
					2 =	Absolute
				TENS:	Dwell time DT at final drilling depth in seconds/revolutions	
					0 =	Compatibility, from DT sign (> 0 seconds or < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions

19.1.26 CYCLE86 - boring

Syntax

```
CYCLE86(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <DTB>, <SDIR>, <RPA>,
<RPO>, <RPAP>, <POSS>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	RP	<RTP>	REAL	Retraction plane (abs)
2	Z0	<RFP>	REAL	Reference point (abs)
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)
4	Z1	<DP>	REAL	Drilling depth (abs), see <_AMODE>
5	Z1	<DPR>	REAL	Drilling depth (inc), see <_AMODE>
6	DT	<DTB>	REAL	Dwell time at final drilling depth, see <_AMODE>

No.	Parameter mask	Parameter internal	Data type	Meaning		
7	DIR	<SDIR>	INT	Direction of spindle rotation	3 =	M3
					4 =	M4
8	DX	<RPA>	REAL	Lift-off distance in X direction		
9	DY	<RPO>	REAL	Lift-off distance in the Y direction		
10	DZ	<RPAP>	REAL	Lift-off distance in the Z direction		
11	SPOS	<POSS>	REAL	Spindle position for lift-off (for oriented spindle stop, in degrees)		
12		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Lift mode	
					0 =	Lift off, compatibility
					1 =	Do not lift off contour
13		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
14		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from programming<DP>/<DPR>
					1 =	Incremental
					2 =	Absolute
				TENS:	Dwell time at final drilling depth DT in seconds/revolutions	
					0 =	Compatibility, from DT sign (> 0 seconds or < 0 revolutions)
					1 =	In seconds
					2 =	In revolutions

19.1.27 CYCLE92 - cut-off

Syntax

```
CYCLE92(<_SPD>, <_SPL>, <_DIAG1>, <_DIAG2>, <_RC>, <_SDIS>, <_SV1>,
<_SV2>, <_SDAC>, <_FF1>, <_FF2>, <_SS2>, <_DIAGM>, <_VARI>, <_DN>,
<_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	X0	<_SPD>	REAL	Reference point (abs, always diameter)		
2	Y0	<_SPL>	REAL	Reference point (abs)		
3	X1	<_DIAG1>	REAL	Depth for speed reduction, see <_AMODE> (UNITS)		
4	X2	<_DIAG2>	REAL	Final depth, see <_AMODE> (TENS)		
5	R/FS	<_RC>	REAL	Rounding status or chamfer width, see <_AMODE> (THOUSANDS)		
6	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
7	S	<_SV1>	REAL	Constant spindle speed, see <_AMODE> (TEN THOUSANDS)		
	V			Constant cutting rate		
8	SV	<_SV2>	REAL	Maximum speed at constant cutting speed		
9	DIR	<_SDAC>	INT	Direction of spindle rotation	3 =	For M3
					4 =	For M4
10	F	<_FF1>	REAL	Infeed as far as depth for speed reduction		
11	FR	<_FF2>	REAL	Reduced infeed as far as final depth		
12	SR	<_SS2>	REAL	Reduced speed as far as final depth		
13	XM	<_DIAGM>	REAL	Depth to withdraw parts gripper (abs, always diameter)		
14		<_VARI>	INT	Machining type		
				UNITS:	Retraction	
					0 =	Retraction to <_SPD> + <_SDIS>
					1 =	No retraction at the end
				TENS:	Parts gripper	
					0 =	No, do not execute M command
					1 =	Yes, call from CUST_TECH-CYC(101)- open drawer, CUST_TECHCYC(102)- close drawer
15		<_DN>	INT	D number for 2nd edge of tool; if not programmed ⇒ D+1		
20		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)

No.	Parameter mask	Parameter internal	Data type	Meaning		
21		<_AMODE>	INT	Alternative mode		
				UNITS:	Depth for speed reduction (<_DIAG1>)	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
				TENS:	Final depth (<_DIAG2>)	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
				HUNDREDS:	Reserved	
				THOUSANDS:	Radius/chamfer (<_RC>)	
					0 =	Radius
					1 =	Chamfer
				TEN THOUSANDS:	Spindle speed / cutting rate (<_SV1>)	
					0 =	Constant spindle speed
					1 =	Constant cutting rate

19.1.28 CYCLE95 - contour cutting

Syntax

```
CYCLE95(<NPP>, <MID>, <FALZ>, <FALX>, <FAL>, <FF1>, <FF2>, <FF3>,
<_VARI>, <DT>, <DAM>, <_VRT>, <_GMODE>, <_DMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	CON	<NPP>	STRING [140]	Contour name
2	D	<MID>	REAL	Maximum depth infeed during roughing, see <_GMODE>
3	UZ	<FALZ>	REAL	Finishing allowance in Z
4	UX	<FALX>	REAL	Finishing allowance in X
5	U	<FAL>	REAL	Finishing allowance parallel to contour (effective in both axes)
6	F	<FF1>	REAL	Feedrate for roughing
7	FY	<FF2>	REAL	Insertion feedrate, relief cuts
8	FS	<FF3>	REAL	Finishing feedrate

No.	Parameter mask	Parameter internal	Data type	Meaning		
9		<_VARI>	INT	Machining type		
				UNITS and TENS:		
					1 =	Roughing, longitudinal, external
					2 =	Roughing, transverse, external
					3 =	Roughing, longitudinal, internal
					4 =	Roughing, transverse, internal
					5 =	Finishing, longitudinal, external
					6 =	Finishing, transverse, external
					7 =	Finishing, longitudinal, internal
					8 =	Finishing, transverse, internal
					9 =	Complete machining, longitudinal, external
					10 =	Complete machining, transverse, external
					11 =	Complete machining, longitudinal, internal
					12 =	Complete machining, transverse, internal
				HUNDREDS:		
					0 =	With rounding at the contour, without residual corners
					1 =	Without rounding at the contour
					2 =	Rounding only to previous intersection, residual corners can result
10	DT	<DT>	REAL	Dwell time at feed interruption		
11	DI	<DAM>	REAL	Distance for feed interruptions		
12	VRT	<_VRT>	REAL	Lift-off distance from the contour		
					0 =	A lift-off distance of 1 mm is used internally regardless of the active system (inch or metric)
					> 0 =	Lift-off distance
13		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Evaluation of the infeed depth	
					0 =	Infeed depth is calculated corresponding to the G group DIAMON/DIAMOF
					1 =	Infeed depth acts as radius value (independent of DIAMON/DIAMOF)

No.	Parameter mask	Parameter internal	Data type	Meaning		
14		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				THOUSANDS:		
					0 =	Compatibility mode: Contour name is in NPP
					1 =	Contour name is programmed in CYCLE62 and transferred to _SC_CONT_NAME

19.1.29 CYCLE98 - thread chain

Syntax

```
CYCLE98(<_PO1>, <_DM1>, <_PO2>, <_DM2>, <_PO3>, <_DM3>, <_PO4>,
<_DM4>, <APP>, <ROP>, <TDEP>, <FAL>, <_IANG>, <NSP>, <NRC>, <NID>,
<_PP1>, <_PP2>, <_PP3>, <_VARI>, <_NUMTH>, <_VRT>, <_MID>, <_GDEP>,
<_IFLANK>, <_PITA>, <_PITM1>, <_PITM2>, <_PITM3>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	Z0	<_PO1>	REAL	Reference point in Z (abs)
2	X0	<_DM1>	REAL	Reference point in X (abs), in diameter
3	Z1	<_PO2>	REAL	Intermediate point 1 in Z (abs/inc), see <_AMODE> (UNITS)
4	X1	<_DM2>	REAL	Intermediate point 1 in X (abs/inc), see <_AMODE> (TENS) or
	X1α			Thread inclination 1 (-90° to 90°) abs is always diameter, inc is always radius
5	Z2	<_PO3>	REAL	Intermediate point 2 in Z, (abs/inc), see <_AMODE> (HUNDREDS)
6	X2	<_DM3>	REAL	Intermediate point 2 in X (abs/inc), see <_AMODE> (THOUSANDS) or
	X2α			Thread inclination 2 (-90° to 90°) abs is always diameter, inc is always radius
7	Z3	<_PO4>	REAL	End point in Z, (abs/inc), see <_AMODE> (TEN THOUSANDS)
8	X3	<_DM4>	REAL	End point in X, (abs/inc), see <_AMODE> (HUNDRED THOUSANDS) or
	X3α			Thread inclination 3 (-90° to 90°) abs is always diameter, inc is always radius
9	LW	<APP>	REAL	Thread run-in (inc, to be entered without sign)

No.	Parameter mask	Parameter internal	Data type	Meaning		
10	LR	<ROP>	REAL	Thread run-out (inc, to be entered without sign)		
11	H1	<TDEP>	REAL	Thread depth (inc, to be entered without sign)		
12	U	<FAL>	REAL	Finishing allowance in X and Z		
13	DP	<_IANG>	REAL	Infeed slope as a distance or an angle, see <_AMODE> (ONE MILLION)		
	αP			The infeed slope is applied according to the setting of parameter <_VARI> (HUNDREDS).		
					Definition for <_VARI_HUNDREDS = 0 - Compatibility mode:	
					> 0 =	Side infeed on one side
					0 =	Infeed vertical in the thread
					< 0 =	Side infeed with alternating sides
					Definition for _VARI_HUNDREDS<>0:	
					> 0 =	Infeed on the positive side
					0 =	Center infeed
					< 0 =	Infeed on the negative side
14	α0	<NSP>	REAL	Starting angle offset for the 1st thread		
15		<NRC>	INT	Number of roughing cuts, see <_VARI> (TEN THOUSANDS)		
16	NN	<NID>	INT	Number of non-cuts		
17	P0	<_PP1>	REAL	Pitch for 1st section of thread, see <_PITA>		
18	P1	<_PP2>	REAL	Pitch for 2nd section of thread, see <_PITA>		
19	P2	<_PP3>	REAL	Pitch for 3rd section of thread, see <_PITA>		

No.	Parameter mask	Parameter internal	Data type	Meaning		
20		<_VARI>	INT	Machining		
				UNITS:	Technology	
					1 =	External thread with linear infeed
					2 =	Internal thread with linear infeed
					3 =	External thread with degressive infeed, cross-section of cut remains constant
					4 =	Internal thread with degressive infeed, cross-section of cut remains constant
				TENS:	Reserved	
				HUNDREDS:	Infeed type	
					0 =	Compatibility mode for <_IANG>
					1 =	Infeed on one side
					2 =	Infeed alternate sides
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Alternative depth infeed	
					0 =	Compatibility, preset number of roughing cuts (<_NRC>)
					1 =	Preset value for 1st infeed (<_MID>)
				HUNDRED THOUSANDS:	Machining type	
					0 =	Compatibility (roughing and finishing)
					1 =	Roughing
					2 =	Finishing
					3 =	Roughing and finishing
				ONE MILLION:	Machining sequence for multistart thread	
					0 =	In ascending order of threads
					1 =	In descending order of threads
21	N	<_NUMTH>	INT	Number of thread turns		
22		<_VRT>	REAL	Return distance (inc)		
					0 =	A lift-off distance of 1 mm is used internally regardless of the active system (inch or metric)
					> 0 =	Lift-off distance
23	D1	<_MID>	REAL	First infeed, see <_VARI> (TEN THOUSANDS)		
24	DA	<_GDEP>	REAL	Thread changeover depth (only effective with "multiple start")		
					0 =	Do not observe any thread changeover depth
					> 0 =	Observe thread changeover depth
25		<_IFLANK>	REAL	Infeed slope as width (for interface only)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
26		<_PITA>	INT	Evaluation of thread pitch		
					0 =	Compatibility mode for pitch, evaluation <_PP1> to <_PP3> as previously, according to active system (metric/inch)
					1 =	Pitch in mm
					2 =	Pitch in TPI (threads per inch)
					3 =	Pitch in inches
					4 =	MODULUS
27		<_PITM1>	STRING[15]	String as marker for pitch input (for the interface only)		
28		<_PITM2>	STRING[15]	String as marker for pitch input (for the interface only)		
29		<_PITM3>	STRING[15]	String as marker for pitch input (for the interface only)		
30		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	---	Reserved
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
31		<_AMODE>	INT	Alternative mode		
				UNITS:	1st intermediate point in Z (Z1)	
					0 =	Absolute
					1 =	Incremental
				TENS:	1st intermediate point in X (X1)	
					0 =	Absolute
					1 =	Incremental
					2 =	α
				HUNDREDS:	2nd intermediate point in Z (Z2)	
					0 =	Absolute
					1 =	Incremental
				THOUSANDS:	2nd intermediate point in X (X2)	
					0 =	Absolute
					1 =	Incremental
					2 =	α
				TEN THOUSANDS:	End point in Z (Z3)	
					0 =	Absolute
					1 =	Incremental
				HUNDRED THOUSANDS:	End point in X (X3)	
					0 =	Absolute
					1 =	Incremental
					2 =	α
				ONE MILLION:	Select infeed slope as angle or width	
					0 =	Infeed angle <_IANG>
					1 =	Infeed slope <_IFLANK>
				TEN MILLIONS:	Single/multiple thread	
					0 =	Compatibility mode (starting angle <_NSP> is evaluated)
					1 =	Single thread (with starting angle offset <_NSP>)
					2 =	Multiple

19.1.30 CYCLE99 - thread turning

Syntax

```
CYCLE99(<_SPL>, <_SPD>, <_FPL>, <_FPD>, <_APP>, <_ROP>, <_TDEP>,
<_FAL>, <_IANG>, <_NSP>, <_NRC>, <_NID>, <_PIT>, <_VARI>, <_NUMTH>,
<_SDIS>, <_MID>, <_GDEP>, <_PIT1>, <_FDEP>, <_GST>, <_GUD>,
<_IFLANK>, <_PITA>, <_PITM>, <_PTAB>, <_PTABA>, <_DMODE>, <_AMODE>,
<_S_XRS>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	Z0	<_SPL>	REAL	Reference point (abs)		
2	X0	<_SPD>	REAL	Reference point (abs, always diameter)		
3	Z1	<_FPL>	REAL	End point in conjunction with <_AMODE> (UNITS)		
4	X1	<_FPD>	REAL	End point in conjunction with <_AMODE> (TENS)		
5	LW/LW2	<_APP>	REAL	Thread run-in in conjunction with <_AMODE> (HUNDREDS) or Thread run-in = thread run-out in conjunction with <_AMODE> (HUNDREDS)		
6	LR	<_ROP>	REAL	Thread run-out		
7	H1	<_TDEP>	REAL	Thread depth		
8	U	<_FAL>	REAL	Finishing allowance in X and Z		
9	DP	<_IANG>	REAL	Infeed slope as a distance or an angle, in conjunction with <_AMODE> (THOUSANDS)		
	αP				> 0 =	Infeed on the positive side
					< 0 =	Infeed on the negative side
					0 =	Center infeed
10	α0	<_NSP>	REAL	Starting angle offset (only effective with "single start")		
11	ND	<_NRC>	INT	Number of roughing cuts, in combination with <_VARI> (TEN THOUSANDS)		
12	NN	<_NID>	INT	Number of non-cuts		
13	P	<_PIT>	REAL	Pitch as a value, in conjunction with <_PITA>		

No.	Parameter mask	Parameter internal	Data type	Meaning		
14		<_VARI>	INT	Machining type		
				UNITS:	Technology	
					1 =	External thread with linear infeed
					2 =	Internal thread with linear infeed
					3 =	External thread with degressive infeed, cross-section of cut remains constant
					4 =	Internal thread with degressive infeed, cross-section of cut remains constant
				TENS:	Reserved	
				HUNDREDS:	Infeed type	
					1 =	Infeed on one side
					2 =	Infeed alternate sides
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Alternative depth infeed	
					0 =	Preset number of roughing cuts (<_NRC>)
					1 =	Preset value for 1st infeed (<_MID>)
				HUNDRED THOUSANDS:	Machining type	
					1 =	Roughing
					2 =	Finishing
					3 =	Roughing and finishing
				ONE MILLION:	Machining sequence for multistart thread	
					0 =	In ascending order of threads
					1 =	In descending order of threads
15	N	<_NUMTH>	INT	Number of thread turns		
16	VR	<_SDIS>	REAL	Return distance, inc		
17	D1	<_MID>	REAL	First infeed depth, in conjunction with <_VARI> (TEN THOUSANDS)		
18	DA	<_GDEP>	REAL	Thread changeover depth (only effective with "multiple start")		
					0 =	Do not observe any thread changeover depth
					> 0 =	Observe thread changeover depth
19	G	<_PIT1>	REAL	Change of pitch per revolution		
					0 =	Pitch is constant (G33)
					> 0 =	Pitch increases (G34)
					< 0 =	Pitch decreases (G35)
20		<_FDEP>	REAL	Insertion depth (enter without sign)		
21	N1	<_GST>	INT	Starting thread N1 = 1...N, in conjunction with <_AMODE> (HUNDRED THOUSANDS)		
22		<_GUD>	INT	Reserved		
23		<_IFLANK>	REAL	Infeed slope as width (for interface only)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
24		<_PITA>	INT	Pitch unit (evaluation of PIT and/or MPIT)		
					0 =	Pitch in mm - MPIT/PIT evaluation
					1 =	Pitch in mm - PIT evaluation
					2 =	Pitch in TPI - PIT evaluation (threads per inch)
					3 =	Pitch in inches - PIT evaluation
					4 =	MODULE - PIT evaluation
25		<_PITM>	STRING[15]	String as marker for pitch input (for the interface only)[1)]		
26		<_PTAB>	STRING[20]	String for thread table (for the interface only)[1)]		
27		<_PTABA>	STRING[20]	String for selection in the thread table (for the interface only)[1)]		
28		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Type of thread	
					0 =	Longitudinal thread
					1 =	Face thread
					2 =	Tapered thread
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Basic

No.	Parameter mask	Parameter internal	Data type	Meaning		
29		<_AMODE>	INT	Alternative mode		
				UNITS:	Thread length in Z	
					0 =	Absolute
					1 =	Incremental
				TENS:	Thread length in X	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
					2 =	α
				HUNDREDS:	Calculation of approach/run-in path <_APP>	
					0 =	Thread run-in <_APP>
					1 =	Thread run-in = thread run-out <_APP> = -<_ROP>
					2 =	Specify thread run-in path <_APP> = -<_APP>
				THOUSANDS:	Select infeed slope as angle or width	
					0 =	Infeed angle <_IANG>
					1 =	Infeed slope <_IFLANK>
				TEN THOUSANDS:	Single/multiple thread	
					0 =	Single thread (with starting angle offset <_NSP>)
					1 =	Multiple
				HUNDRED THOUSANDS:	Starting thread <_GST>	
					0 =	Full machining
					1 =	Start machining from this thread
					2 =	Only machine this thread
				ONE MILLION:	Sag compensation for longitudinal thread	
					0 =	Segment height, crowned thread XS
					1 =	Radius, crowned thread RS
30	XS/RS	<_S_XRS>	REAL	Sag compensation for longitudinal thread in conjunction with <_AMODE>: ONE MILLION		

Note

[1] Parameters <_PITM>, <_PTAB> and <_PTABA> are only used for thread selection in the screen form thread tables.
The thread tables cannot be accessed via cycle definition in the cycle run time.

19.1.31 CYCLE435 - calculate dressing tool position

Syntax

```
CYCLE435(<_T>, <_DD>, <S_TA>, <S_DA>, <S_AD>, <S_AL>, <S_PVD>,
<S_PVL>, <S_PD>, <S_PL>, <_AMODE>)
```

Parameter

No.	Parameter mask	Parameter internal	Data type	Meaning		
1		<_T>	STRING[32]	Tool name of the grinding wheel		
2		<_DD>	INT	Cutting edge number of the grinding wheel		
3		<S_TA>	STRING[32]	Dressing tool reference point - dressing tool name		
4		<S_DA>	INT	Cutting edge number of the dressing tool		
5		<S_AD>	REAL	Dressing value, diameter		
6		<S_AL>	REAL	Dressing value, face		
7		<S_PVD>	REAL	Form-truing offset, diameter		
8		<S_PVL>	REAL	Form-truing offset, face		
9		<S_PD>	REAL	Form-truing allowance, diameter		
10		<S_PL>	REAL	Form-truing allowance, face		
11		<_AMODE>	INT	Alternative mode		
				UNITS:	active tool at the end of the cycle	
					0 =	dressing tool active
					1 =	wheel active

19.1.32 CYCLE495 - form-truing

Syntax

```
CYCLE495(<_T>, <_DD>, <_SC>, <_F>, <_VARI>, <_D>, <_DX>, <_DZ>,
<S_PA>, <S_N>, <_DMODE>, <_AMODE>, <S_FW>, <S_HW>)
```

Parameter

No.	Parameter mask	Parameter internal	Data type	Meaning
1		<_T>	STRING[20]	Tool name of the grinding wheel
2		<_DD>	INT	Cutting edge number of the grinding wheel
3		<_SC>	REAL	Lift-off distance for avoiding obstacles, incremental
4		<_F>	REAL	Form-truing feedrate

No.	Parameter mask	Parameter internal	Data type	Meaning		
5		<_VARI>	INT	Machining type		
				UNITS:	Form-truing type	
					1 =	Parallel to the axis
					2 =	Parallel to the contour
				TENS:	Machining direction	
					0 =	Pulling Possible with cutting edge positions 1 to 4
					1 =	Pushing Possible with cutting edge positions 1 to 4
					2 =	Alternating Possible with cutting edge positions 1 to 8
					3 =	Start → end Possible with cutting edge positions 1 to 8
					4 =	End → start Possible with cutting edge positions 1 to 8
				HUNDREDS:	Infeed direction	
					1 =	Infeed X for G18 or Y- for G19
					2 =	Infeed X+ for G18 or Y+ for G19
					3 =	Infeed Z- for G18 and for G19
					4 =	Infeed Z+ for G18 and for G19
6		<_D>	REAL	Dressing value for form-truing type parallel to the axis		
7		<_DX>	REAL	Dressing value X for G18 or Y for G19 for form-truing type parallel to the contour		
8		<_DZ>	REAL	Dressing value Z for G18 and G19 for form-truing type parallel to the contour		
9		<S_PA>	REAL	Form-truing allowance		
10		<S_N>	INT	Number of strokes in the form-truing program		
11		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)

No.	Parameter mask	Parameter internal	Data type	Meaning		
12		<_AMODE>	INT	Alternative mode		
				UNITS:	Form-truing selection, new/continue	
					1 =	New
					2 =	Continue
				TENS:	Select form-truing allowance	
					0 =	From the rough contour to the lowest point of the contour
					1 =	From the rough contour to the highest point of the contour
13		<S_FW>	REAL	Clear angle of the dressing tool		
14		<S_HW>	REAL	Holder angle of the dressing tool		

19.1.33 CYCLE800 - swiveling

Syntax

```
CYCLE800(<_FR>, <_TC>, <_ST>, <_MODE>, <_X0>, <_Y0>, <_Z0>, <_A>,
<_B>, <_C>, <_X1>, <_Y1>, <_Z1>, <_DIR>, <_FR_I>, <_DMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1		<_FR>	INT	Retraction mode:	0 =	No retraction
					1 =	Retraction machine axis Z
					2 =	Retraction machine axis Z and then XY
					3 =	Reserved
					4 =	Maximum retraction in tool direction
					5 =	Incremental retraction in tool direction
2		<_TC>	STRING[32]	Name of swivel data block:	""	"" (no name) if only one swivel data block exists
					"0"	Deselect swivel data block (delete the swivel frames)

No.	Parameter mask	Parameter internal	Data type	Meaning		
3		<_ST>	INT	Status transformations		
				UNITS:		
					0 =	New, swivel level is deleted and recalculated using the current parameters
					1 =	Additive, swivel level is added to active swivel level
				TENS:	Track tool tip yes/no (only active when the SWIVEL function is created in the commissionig)	
					0 =	Do not track tool tip
					1 =	Track tool tip (TRAORI)
				HUNDREDS:	Approach/align tool (function is shown in tool swivel screen form)	
					0 =	Do not approach tool
					1 =	Approach tool (preferably radial mill)
					2 =	Align **turning** tool (when B axis kinematic is set up for milling in commissioning swiveling)
					3 =	Align **milling** tool (when B axis kinematic is set up for milling in commissioning swiveling)
				THOUSANDS:	Internal "Swiveling in JOG" parameter	
				TEN THOUSANDS:	See direction parameter <_DIR>	
					0 =	Swivel "No"
					1 =	Swivel "No", "Minus" direction[3)]
					2 =	Swivel "No", "Plus" direction[3)]
				HUNDRED THOUSANDS:	See direction parameter <_DIR>	
					0 =	Compatibility
					1 =	Direction selection "Minus" optimized[4)]
					2 =	Direction selection "Plus" optimized[4)]

No.	Parameter mask	Parameter internal	Data type	Meaning		
4		<_MODE> [5]	INT	Swivel mode: Evaluation of swivel angle and swivel sequence (bit-coded)		
				Bit: 7 6	0 0:	Swivel angle axis-by-axis -> see parameters <_A>, <_B>, <_C>
					0 1:	Solid angle -> see parameters <_A>, <_B> [1]
					1 0:	Projection angle -> see parameters <_A>, <_B>, <_C> [1]
					1 1:	Direct rotary axis swivel mode -> see parameters <_A>, <_B> [1]
				Bit: 5 4 3 2 1 0 (these do not apply to solid angles)	x x x x 0 1	1st rotation _A around X
					x x x x 1 0	1st rotation _A around Y
					x x x x 1 1	1st rotation _A around Z
					x x 0 1 x x	2nd rotation _B around X
					x x 1 0 x x	2nd rotation _B around Y
					x x 1 1 x x	2nd rotation _B around Z
					0 1 x x x x	3rd rotation _C around X
					1 0 x x x x	3rd rotation _C around Y
					1 1 x x x x	3rd rotation _C around Z
5	X0	<_X0>	REAL	Reference point X prior to rotation		
6	Y0	<_Y0>	REAL	Reference point Y prior to rotation		
7	Z0	<_Z0>	REAL	Reference point Z prior to rotation		
8	X(A)	<_A>	REAL	1. rotation acc. to setting in <_MODE> parameter		
9	Y(B)	<_B>	REAL	2. rotation acc. to setting in <_MODE> parameter		
10	Z(C)	<_C>	REAL	3. rotation acc. to setting in <_MODE> parameter		
11	X1	<_X1>	REAL	Reference point X after rotation		
12	Y1	<_Y1>	REAL	Reference point Y after rotation		
13	Z1	<_Z1>	REAL	Reference point Z after rotation		
14	- or +	<_DIR>	INT	Initiate travel of rotary axes (default = -1!)		
					-1 =	Position at smaller value of rotary axis 1 or 2 [2]
					+1 =	Position at larger value of rotary axis 1 or 2 [2]
					0 =	Do not swivel (merely calculate swivel frame) [1] [3]
15	FR	<_FR_I>	REAL	Value (inc) of retraction in tool direction incremental		

No.	Parameter mask	Parameter internal	Data type	Meaning		
16		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Representation of the beta value during align tool	
					0 =	Value
					1 =	Arrow

Note

If the following transfer parameters are programmed indirectly (as parameters), the screen form is not reset: <_FR>, <_ST>, <_TC>, <_MODE>, <_DIR>

[1] Can be selected if the SWIVEL function is created in the commissioning

[2] Can be selected if direction reference to rotary axis 1 or 2 is set in IBN SWIVEL

If direction reference is "No" there is no selection field

[3] Swivel selection "No" can be grayed out SD 55221 Bit 0

Swivel "No", "Minus" direction corresponds to <_DIR> = 0 and _ST TEN THOUSANDS = 1

Swivel "No", "Plus" direction corresponds to <_DIR> = 0 and _ST TEN THOUSANDS = 2

[4] The direction selection for rotary axis 1 or 2 also occurs if the rotary axis with the direction reference is in the pole position (position value equals zero).

[5] Coding example: Axis-by-axis rotation, rotary sequence ZYX

Binary: 00011011 Decimal: 27

The axis identifiers XYZ correspond to the geometry axes of the NC channel. Individual rotations around the XYZ axes are permissible. Example: rotary sequence around ZXZ is not permitted in one call of CYCLE800.

19.1.34 CYCLE801 - grid or frame

Syntax

```
CYCLE801(<_SPCA>, <_SPCO>, <_STA>, <_DIS1>, <_DIS2>, <_NUM1>,
<_NUM2>, <_VARI>, <_UMODE>, <_ANG1>, <_ANG2>, <_HIDE>, <_NSP>,
<_DMODE>)
```

Parameters

No.	Parameter mask	Parameters internal	Data type	Meaning		
1	X0	<_SPCA>	REAL	Reference point for position pattern (grid/frame) along the 1st axis (abs)		
2	Y0	<_SPCO>	REAL	Reference point for position pattern (grid/frame) along the 2nd axis (abs)		
3	α0	<_STA>	REAL	Basic angle of rotation (angle to 1st axis)	< 0 =	Clockwise rotation
					> 0 =	Counterclockwise rotation
4	L1	<_DIS1>	REAL	Distance for columns (distance from the 1st axis, enter without sign)		
5	L2	<_DIS2>	REAL	Distance for rows (distance from the 2nd axis, enter without sign)		
6	N1	<_NUM1>	INT	Number of columns		
7	N2	<_NUM2>	INT	Number of rows		
8		<_VARI>	INT	Machining type		
				UNITS:	Position pattern	
					0 =	Grid
					1 =	Frame
				TENS:	Reserved	
				HUNDREDS:	Reserved	
9		<_UMODE>	INT	Reserved		
10	αX	<_ANG1>	REAL	Shear angle with 1st axis (lines arranged obliquely to the 1st axis)		
					< 0 =	Clockwise measurement (0 to -90 degrees)
					> 0 =	Counter-clockwise measurement (0 to 90 degrees)
11	αY	<_ANG2>	REAL	Shear angle with 2nd axis (lines arranged obliquely to the 2nd axis)		
					< 0 =	Clockwise measurement (0 to -90 degrees)
					> 0 =	Counter-clockwise measurement (0 to 90 degrees)
12		<_HIDE>	STRING [200]	Hidden positions • Max. 198 characters • Specification of consecutive position numbers, e.g. "1,3" (positions 1 and 3 are not executed)		
13		<_NSP>	INT	Reserved		
14		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)

19.1.35 CYCLE802 - arbitrary positions

Syntax

```
CYCLE802(<_XA>, <_YA>, <_X0>, <_Y0>, <_X1>, <_Y1>, <_X2>, <_Y2>,
<_X3>, <_Y3>, <_X4>, <_Y4>, <_X5>, <_Y5>, <_X6>, <_Y6>, <_X7>, <_Y7>,
<_X8>, <_Y8>, <_VARI>, <_UMODE>, <_DMODE>, <S_ABA>, <S_AB0>,
<S_AB1>, <S_AB2>, <S_AB3>, <S_AB4>, <S_AB5>, <S_AB6>, <S_AB7>,
<S_AB8>)
```

Parameters

No.	Parameter mask	Parameters internal	Data type	Meaning		
1		<_XA>	INT	Alternatives for all X positions (9-digit decimal value) Number of digits: 876543210 (digit position corresponds to drilling position Xn)		
				Position value:	1 =	Absolute (1st programmed position is always absolute)
					2 =	Incremental
2		<_YA>	INT	Alternatives for all Y positions (9-digit decimal value) Number of digits: 876543210 (digit position corresponds to drilling position Yn)		
				Position value:	1 =	Absolute (1st programmed position is always absolute)
					2 =	Incremental
3	X0	<_X0>	REAL	1. Position X		
4	Y0	<_Y0>	REAL	1. Position Y		
5	X1	<_X1>	REAL	2. Position X		
6	Y1	<_Y1>	REAL	2. Position Y		
7	X2	<_X2>	REAL	3. Position X		
8	Y2	<_Y2>	REAL	3. Position Y		
9	X3	<_X3>	REAL	4. Position X		
10	Y3	<_Y3>	REAL	4. Position Y		
11	X4	<_X4>	REAL	5. Position X		
12	Y4	<_Y4>	REAL	5. Position Y		
13	X5	<_X5>	REAL	6. Position X		
14	Y5	<_Y5>	REAL	6. Position Y		
15	X6	<_X6>	REAL	7. Position X		
16	Y6	<_Y6>	REAL	7. Position Y		
17	X7	<_X7>	REAL	8. Position X		
18	Y7	<_Y7>	REAL	8. Position Y		
19	X8	<_X8>	REAL	9. Position X		
20	Y8	<_Y8>	REAL	9. Position Y		

No.	Parameter mask	Parameters internal	Data type	Meaning		
21		<_VARI>	INT	Machining		
				HUNDREDS:	(Only for call from Jobshop) (At present only 0 and 2 evaluated)	
					0 =	Do not clamp spindle
					1 =	Only clamp spindle for vertical insertion with G00 or G01
					2 =	Clamp spindle during the entire machining operation
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Position pattern with/without rotary axis – axis combination (with <_VARI> HUNDRED THOUSANDS)	
					0 =	XY (only XY without rotary axis, compatibility)
					1 =	X,Y or Z and rotary axis: XA, YB, ZC (1 rotary axis with geometry axis around which the rotary axis rotates)
					2 =	XY and rotary axis: XYA, XYB, XYC (1 rotary axis with 1st and 2nd geometry axis, without TRACYL)
				HUNDRED THOUSANDS:	Rotary axis	
					0 =	Without rotary axis (only XY, compatibility)
					1 =	A axis (rotary axis around X)
					2 =	B axis (rotary axis around Y)
					3 =	C axis (rotary axis around Z)
				TEN MILLIONS + ONE MILLION:	Position pattern with rotary axis – offset (for several rotary axes around the same axis; if index too large, then 1st axis)	
					00 =	1st A, B or C axis or for compatibility
					01 =	2nd A, B or C axis
					...	
					19 =	20th A, B or C axis
22		<_UMODE>	INT	Selection of the spindle to be clamped: (Only for call from Jobshop) (Call of user cycle CUST_TECHCYC)		
					3 =	Clamp/release main spindle
					23 =	Clamp/release counterspindle

No.	Parameter mask	Parameters internal	Data type	Meaning		
23		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
24		<S_ABA>	INT	Alternatives for all AB positions (9-digit decimal value) Number of digits: 876543210 (digit position corresponds to position ABn)		
				Position value:	1 =	Absolute (1st programmed position is always absolute)
					2 =	Incremental
25	A0	<S_AB0>	REAL	1st rotary axis position for position pattern with rotary axis (in conjunction with <_VARI>))		
26	A1	<S_AB1>	REAL	2nd rotary axis position for position pattern with rotary axis		
27	A2	<S_AB2>	REAL	3rd rotary axis position for position pattern with rotary axis		
28	A3	<S_AB3>	REAL	4th rotary axis position for position pattern with rotary axis		
29	A4	<S_AB4>	REAL	5th rotary axis position for position pattern with rotary axis		
30	A5	<S_AB5>	REAL	6th rotary axis position for position pattern with rotary axis		
31	A6	<S_AB6>	REAL	7th rotary axis position for position pattern with rotary axis		
32	A7	<S_AB7>	REAL	8th rotary axis position for position pattern with rotary axis		
33	A8	<S_AB8>	REAL	9th rotary axis position for position pattern with rotary axis		

Note

Positions that are not required for parameters X1/Y1/A1 to X8/Y8/A8 can be ignored. The alternative values for <_XA>, <_YA> and <S_ABA>, however, must be provided in full for all 9 positions.

For position pattern XA, YB or ZC (a geometry axis and rotary axis), the axis of the machining plane that is not traversed via the position pattern (Y for G17 and XA) must be positioned before the cycle call.

19.1.36 CYCLE830 - deep-hole drilling 2

Syntax

```
CYCLE830(<RTP>, <RFP>, <SDIS>, <_DP>, <FDEP>, <_DAM>, <DTB>, <DTS>,
<FRF>, <VARI>, <_MDEP>, <_VRT>, <_DTD>, <_DIS1>, <S_FP>, <S_SDAC2>,
<S_SV2>, <S_FB>, <_SDAC>, <_SV1>, <S_SPOS>, <S_ZA>, <S_FA>, <S_ZP>,
<S_FS>, <S_ZS1>, <S_ZS2>, <S_N>, <S_ZD>, <S_FD>, <S_FR>, <S_SDAC3>,
```

```
<S_SV3>, <S_CON>, <S_COFF>, <_GMODE>, <_DMODE>, <_AMODE>,
<S_AMODE2>, <S_AMODE3>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<RTP>	REAL	Retraction plane (abs)		
2	Z0	<RFP>	REAL	Reference point (abs)		
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, without sign)		
4	Z1	<_DP>	REAL	Final drilling depth abs/inc (see <_AMODE>UNITS)		
5	D	<FDEP>	REAL	1. drilling depth for the absolute or incremental chip breaking/removal in relation to the reference point with/without predrilling or in relation to pilot hole depth (see <_AMODE> TEN THOUSANDS)		
6	DF	<_DAM>	REAL	Absolute value / percentage for each additional infeed, degression absolute value / percentage (see <_AMODE> HUNDRED THOUSANDS)		
7	DTB	<DTB>	REAL	Dwell time at each drilling depth (see <_AMODE> TENS)		
8	DTS	<DTS>	REAL	Dwell time during chip removal at starting point (see <_AMODE> HUNDREDS)		
9	FD1	<FRF>	REAL	Percentage for the feedrate for the first infeed (see <_AMODE> TEN MILLION)		
10		<VARI>	INT	Machining		
				UNITS:	Chip breaking / swarf removal	
					0 =	In one cut
					1 =	Chip breaking
					2 =	Swarf removal
					3 =	Chip breaking and swarf removal
				TENS:	Retraction during swarf removal	
					0 =	To pilot hole depth
					1 =	To safety clearance
				HUNDREDS:	Soft first cut	
					0 =	No
					1 =	Yes
				THOUSANDS:	Through drilling	
					0 =	No
					1 =	Yes
				TEN THOUSANDS:	Predrilling / pilot hole	
					0 =	Without predrilling
					1 =	With predrilling
					2 =	With pilot hole
				HUNDRED THOUSANDS:	Retraction	
					0 =	To pilot hole depth
					1 =	To retraction plane
11	V1	<_MDEP>	REAL	Minimum incremental infeed (only for degression percentage)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
12	V2	<_VRT>	REAL	Retraction distance after each incremental machining step (for chip breaking only)		
					0 =	Default value 1 mm
					> 0 =	Variable retraction distance
13	DT	<_DTD>	REAL	Dwell time at final drilling depth (see <_AMODE> THOUSANDS)		
14	V3	<_DIS1>	REAL	Incremental limit distance for chip removal only (see <_AMODE> ONE MILLION)		
15	FP	<S_FP>	REAL	Feedrate for travel into the pilot hole as value or in % (in conjunction with <S_AMODE2> HUNDREDS)		
16		<S_SDAC2>	INT	Direction of spindle rotation during approach		
					3 =	M3
					4 =	M4
					5 =	M5 (default)
17	SP	<S_SV2>	REAL	Approach with	constant spindle speed (see <S_AMODE2> TEN MILLION)	
	V4				constant cutting rate	
				Spindle speed in % of the drilling speed		
18	F	<S_FB>	REAL	Drilling feedrate (see <S_AMODE2> UNITS)		
19		<_SDAC>	REAL	Direction of spindle rotation during drilling		
					3 =	M3
					4 =	M4
20	S	<_SV1>	REAL	Drilling with	constant spindle speed (see <S_AMODE2> ONE MILLION)	
	V5				constant cutting rate	
21	SPOS	<S_SPOS>	REAL	Spindle position, only if approach with M5		
22	ZA	<S_ZA>	REAL	Incremental predrilling depth in relation to reference point or absolute (see <S_AMODE3> UNITS)		
23	FA	<S_FA>	REAL	Predrilling feedrate as value or in % (in conjunction with <S_AMODE2> TENS)		
24	ZP	<S_ZP>	REAL	Incremental pilot hole in relation to reference point or absolute or factor of the hole diameter (see <S_AMODE3> TENS)		
25	FS	<S_FS>	REAL	First cut feedrate as value or in % (in conjunction with <S_AMODE2> THOUSANDS)		
26	ZS1	<S_ZS1>	REAL	Depth of each first cut with constant feedrate (inc)		
27	ZS2	<S_ZS2>	REAL	Depth of each first cut for feedrate increase (inc)		
28	N	<S_N>	INT	Number of chip breaking strokes before each chip removal		
29	ZD	<S_ZD>	REAL	Incremental remaining drilling depth in relation to final drilling depth or absolute (see <S_AMODE3> HUNDREDS)		
30	FD	<S_FD>	REAL	Remaining drilling feedrate as value or in % (in conjunction with <S_AMODE2> TEN THOUSANDS)		
31	FR	<S_FR>	REAL	Retraction feedrate (in conjunction with <S_AMODE2> HUNDRED THOUSANDS)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
32		<S_SDAC3>	INT	Direction of spindle rotation during retraction		
					3 =	M3
					4 =	M4
					5 =	M5
33	SR	<S_SV3>	REAL	Retraction with	constant spindle speed (see <S_AMODE2> HUNDRED MILLION)	
	V6				constant cutting rate	
				Spindle speed in % of the drilling speed		
34	Coolant on	<S_CON>	STRING[10]	Coolant on, M command or subprogram call		
35	Coolant off	<S_COFF>	STRING[10]	Coolant off, M command or subprogram call		
36		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Drilling depth with respect to tip/shank	
					0 =	Tip
					1 =	Shank
37		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Basic

No.	Parameter mask	Parameter internal	Data type	Meaning		
38		<_AMODE>	INT	Alternative mode 1		
				UNITS:	Drilling depth = Final drilling depth Z1 abs/inc	
					0 =	Incremental
					1 =	Absolute
				TENS:	Dwell time at each drilling depth DTB in seconds/revolutions	
					0 =	In seconds
					1 =	In revolutions
				HUNDREDS:	Dwell time for chip removal DTS in seconds/revolutions	
					0 =	In seconds
					1 =	In revolutions
				THOUSANDS:	Dwell time at final drilling depth DT in seconds/revolutions	
					0 =	In seconds
					1 =	In revolutions
				TEN THOUSANDS:	1. drilling depth D abs/inc	
					0 =	Incremental
					1 =	Absolute
				HUNDRED THOUSANDS:	Absolute value / percentage DF for each additional infeed (degression)	
					0 =	Absolute value
					1 =	Percentage (0.001 to 100%)
				ONE MILLION:	Limit distance V3 automatic/manual	
					0 =	Automatic (calculated in the cycle)
					1 =	Manual (programmed value)

No.	Parameter mask	Parameter internal	Data type	Meaning		
39		<S_AMODE2>	INT	Alternative mode 2		
				UNITS:	UNITS: Drilling feedrate F	
					0 =	F/min
					1 =	F/rev
				TENS:	Evaluation of predrilling feedrate FA	
					0 =	As % of drilling feedrate
					1 =	F/min
					2 =	F/rev
				HUNDREDS:	Evaluation of feedrate for travel into pilot hole FP	
					0 =	As % of drilling feedrate
					1 =	F/min
					2 =	F/rev
				THOUSANDS:	Evaluation of first cut feedrate FS	
					0 =	As % of drilling feedrate
					1 =	F/min
					2 =	F/rev
				TEN THOUSANDS:	Evaluation of through-drilling feedrate FD	
					0 =	As % of drilling feedrate
					1 =	F/min
					2 =	F/rev
				HUNDRED THOUSANDS:	Retraction feedrate FR	
					0 =	F/min
					1 =	Rapid traverse
				ONE MILLION:	Drilling - spindle speed / cutting rate (S/V5)	
					0 =	Constant spindle speed
					1 =	Constant cutting rate
				TEN MILLIONS:	Approach with spindle speed / cutting rate (SP/V4)	
					0 =	Constant spindle speed
					1 =	Constant cutting rate
					2 =	Spindle speed in % of the drilling speed
				HUNDRED MILLIONS:	Retraction - spindle speed / cutting rate (SR/V6)	
					0 =	Constant spindle speed
					1 =	Constant cutting rate
					2 =	Spindle speed in % of the drilling speed

No.	Parameter mask	Parameter internal	Data type	Meaning		
40		<S_AMODE3>	INT	Alternative mode 3		
				UNITS:	Drilling depth ZA abs/inc	
					0 =	Incremental
					1 =	Absolute
				TENS:	Depth of the pilot hole ZP	
					0 =	Incremental
					1 =	Absolute
					2 =	Factor of the hole diameter
				HUNDREDS:	Remaining drilling depth ZD abs/inc	
					0 =	Incremental
					1 =	Absolute

19.1.37 CYCLE832 - High-Speed Settings

Syntax

```
CYCLE832(<S_TOL>, <S_TOLM>, <S_OTOL>)
```

Note

CYCLE832 does not relieve the machine manufacturer from optimization tasks that are necessary when commissioning the machine. This involves the optimization of the axes involved in the machining process and NCU settings (precontrol, jerk limiting, etc.).

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	Tolerance	<S_TOL>	REAL	Contour tolerance The contour tolerance corresponds to the axis tolerance of the geometry axes.		
2		<S_TOLM>	INT	Machining type (technology)		
				UNITS:		
					0 =	Deselection
					1 =	Finishing
					2 =	Semi-finishing
					3 =	Roughing
				TENS:		
					0 =	Compatibility[1] or no orientation tolerance
					1 =	Orientation tolerance in parameter <S_OTOL>
				HUNDREDS ... HUNDRED THOUSANDS	Assigned for reasons of compatibility	
				ONE MILLION:		
					0 =	Compatibility. The best available mold making function is automatically used: • Option Top Surface not active: ☐ Advanced Surface • Option Top Surface active: ⇒ Top Surface with smoothing
					1 =	Top Surface without smoothing
					2 =	Top Surface with smoothing
3	ORI tolerance	<S_OTOL>	REAL	Orientation tolerance or version identifier CYCLE832 Tolerance parameter for the orientation of the workpiece. Is required when executing a high-speed machining program on machines with dynamic orientation transformation (e.g. 5-axis machining). Parameter <S_OTOL> **must** be programmed. This also applies for applications on 3-axis machines for programs without orientation of the tool (<S_OTOL> = 1).		

[1] Orientation tolerance derived from the contour tolerance multiplied by the factor from the cycle setting data SD55441 to SD55443.
References:
Commissioning Manual, Base Software and Operator Software; SINUMERIK Operate (IM9), Section "Configuring the High-Speed Settings function (CYCLE832)"

Plain text entry

To improve the readability of the cycle call, parameter <S_TOLM> (machining type) can also be entered in the plain text. Plain texts are independent of any language. The following entries are permitted:

_OFF	for	0	Deselection
_FINISH	for	1	Finishing
_SEMIFIN	for	2	Rough-finishing
_ROUGH	for	3	Roughing
_ORI_FINISH	for	11	Finishing with input of an orientation tolerance
_ORI_SEMIFIN	for	12	Finishing with input of an orientation tolerance
_ORI_ROUGH	for	13	Roughing with input of an orientation tolerance
_TOP_SURFACE_SMOOTH_OFF	for	1000000	Top Surface without smoothing
_TOP_SURFACE_SMOOTH_ON	for	2000000	Top Surface with smoothing

For plain text input for Top Surface, plain texts are combined as shown in the following example:

```
CYCLE832(0.1, _TOP_SURFACE_SMOOTH_OFF+_ORI_FINISH, 1)
```

Note

The plain texts are based on the function names of the G group 59 (dynamic mode for path interpolation). With these plain texts, 3-axis machines and machines with multi-axis orientation transformation (TRAORI) are clearly separated in the application.

Deselecting CYCLE832

When CYCLE832 is deselected, parameter <S_TOL> must be transferred with zero.

Example: `CYCLE832(0,0,1)`

The syntax `CYCLE832()` is also permitted for deselecting CYCLE832.

Examples

Example 1: CYCLE832 on 3-axis machine without orientation transformation

a) Cycle call with plain text input

Program code	Comment
G710	; Dimension system is metric.
CYCLE832(0.004,_FINISH,1)	; CYCLE832 call with: Contour tolerance = 0.004 mm, machining type: Finishing
...	; Execution of a high-speed machining program

b) Cycle call without plain text input

Program code	Comment
G710	; See above
CYCLE832(0.004,1,1)	; See above
...	; See above

Example 2: CYCLE832 on 5-axis machine with orientation transformation

a) Cycle call and deselection with plain text input

Program code	Comment
G710	; Dimension system is metric.
TRAORI	; Activate orientation transformation.
CYCLE832(0.3,_ORI_ROUGH,0.8)	; CYCLE832 call with: Contour tolerance = 0.3 mm, machining type: Roughing with input of an orientation tolerance; orientation tolerance = 0.8 degrees
...	; Execution of a high-speed machining program
CYCLE832(0,_OFF,1)	; Contour tolerance = 0, machining type: Deselection of CYCLE832, orientation tolerance = 0 degrees

b) Cycle call and deselection without plain text input

Program code	Comment
G710	; See above
TRAORI	; See above
CYCLE832(0.3,13,0.8)	; See above
...	; See above
CYCLE832(0,0,1)	; See above

19.1.38 CYCLE840 - tapping with compensating chuck

Syntax

```
CYCLE840(<RTP>, <RFP>, <SDIS>, <DP>, <DPR>, <DTB>, <SDR>, <SDAC>,
<ENC>, <MPIT>, <PIT>, <_AXN>, <_PITA>, <_TECHNO>, <_PITM>, <_PTAB>,
<_PTABA>, <_GMODE>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning			
1	RP	<RTP>	REAL	Retraction plane (abs)			
2	Z0	<RFP>	REAL	Reference point (abs)			
3	SC	<SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)			
4	Z1	<DP>	REAL	Drilling depth (abs), see <_AMODE>			
5	Z1	<DPR>	REAL	Drilling depth (inc), see <_AMODE>			
6	DT	<DTB>	REAL	Dwell time in seconds at drilling depth / safety clearance after retraction, see <ENC>			
7		<SDR>	INT	Direction of rotation for retraction			
8	SDE	<SDAC>	INT	Direction of rotation after end of cycle			
9		<ENC>	INT	Tapping with spindle mounted encoder (G33)/tapping without spindle mounted encoder (G63)			
					0 =	With spindle mounted encoder	- Pitch from <MPIT>/<PIT> - without DT
					20 =	With spindle mounted encoder	- Pitch from <MPIT>/<PIT> - with DT after retraction to safety clearance
					11 =	Without spindle mounted encoder	- Pitch from <MPIT>/<PIT> - with DT at drilling depth
					1 =	Without spindle mounted encoder	- Pitch from programmed feedrate - with DT at drilling depth (feedrate = speed · pitch)
10		<MPIT>	REAL	Thread size for "ISO metric" only (pitch is calculated internally during run time) Range of values: 3 to 48 (for M3 to M48), alternative to <PIT>			
11		<PIT>	REAL	Pitch as a value, for unit see <_PITA> Range of values: > 0, alternative to MPIT			
12		<_AXN>	INT	Drilling axis	0 =	3rd geometry axis	
					1 =	1st geometry axis	
					2 =	2nd geometry axis	
					≥ 3 =	3rd geometry axis	

No.	Parameter mask	Parameter internal	Data type	Meaning			
13		<_PITA>	INT	Pitch unit (evaluation of <PIT> and <MPIT>)			
					0 =	Pitch in mm	- evaluation<MPIT>/<PIT>
					1 =	Pitch in mm	- evaluation<PIT>
					2 =	Pitch in TPI	- evaluation of <PIT> (threads per inch)
					3 =	Pitch in inches	- evaluation<PIT>
					4 =	MODULUS	- evaluation<PIT>
14		<_TECHNO>	INT	Technology[1]			
				UNITS:	Exact stop response		
					0 =	Exact stop response active as before cycle call	
					1 =	Exact stop G601	
					2 =	Exact stop G602	
					3 =	Exact stop G603	
				TENS:	Feedforward control		
					0 =	With/without feedforward control active as before cycle call	
					1 =	With feedforward control FFWON	
					2 =	Without feedforward control FFWOF	
15		<_PITM>	STRING[15]	String as marker for pitch input[2]			
16		<_PTAB>	STRING[5]	String for thread table ("", "ISO", "BSW", "BSP", "UNC")[2]			
17		<_PTABA>	STRING[20]	String for selection from thread table (e.g. "M 10", "M 12", ...)[2]			
18		<_GMODE>	INT	Reserved			

No.	Parameter mask	Parameter internal	Data type	Meaning		
19		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:	Compatibility mode (for recompilation screen form only), if MD 52216 bit0 = 1[1)]	
					0 =	Technology parameters are displayed (compatibility): TECHNO parameters effective
					1 =	Technology parameters are not displayed: Technology active "as before cycle call"
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple
20		<_AMODE>	INT	Alternative mode		
				UNITS:	Drilling depth Z1 (abs/inc)	
					0 =	Compatibility, from programming <DP>/<DPR>
					1 =	Incremental
					2 =	Absolute

[1)] Technology fields may be hidden, depending on the setting date SD52216 MCS_FUNCTION_MASK_DRILL

[2)] Parameters 15, 16 and 17 are only used for thread selection in the screen form thread tables. The thread tables cannot be accessed via cycle definition in cycle run time.

19.1.39 CYCLE899 - Milling open slot

Syntax

```
CYCLE899(<_RTP>, <_RFP>, <_SDIS>, <_DP>, <_LENG>, <_WID>, <_PA>,
<_PO>, <_STA>, <_MID>, <_MIDA>, <_FAL>, <_FALD>, <_FFP1>, <_CDIR>,
<_VARI>, <_GMODE>, <_DMODE>, <_AMODE>, <_UMODE>, <_FS>, <_ZFS>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	RP	<_RTP>	REAL	Retraction plane (abs)		
2	Z0	<_RFP>	REAL	Reference point of tool axis (abs)		
3	SC	<_SDIS>	REAL	Safety clearance (to be added to reference point, enter without sign)		
4	Z1	<_DP>	REAL	Slot depth (abs/inc), see <_AMODE>		
5	L	<_LENG>	REAL	Length of slot (inc)		
6	W	<_WID>	REAL	Width of slot (inc)		
7	X0	<_PA>	REAL	Reference/start point 1st axis (abs)		
8	Y0	<_PO>	REAL	Reference/start point 2nd axis (abs)		
9	α0	<_STA>	REAL	Angle of rotation with respect to 1st axis		
10	DZ	<_MID>	REAL	Maximum infeed depth (inc), for vortex milling only		
11	DXY	<_MIDA>	REAL	Maximum plane infeed, see <_AMODE>		
12	UXY	<_FAL>	REAL	Finishing allowance, plane		
13	UZ	<_FALD>	REAL	Finishing allowance, depth		
14	F	<_FFP1>	REAL	Feedrate		
15		<_CDIR>	INT	Milling direction		
				UNITS:		
					0 =	Down-cut
					1 =	Up-cut
					4 =	Alternating
16		<_VARI>	INT	Machining		
				UNITS:		
					1 =	Roughing
					2 =	Finishing
					3 =	Base finishing
					4 =	Edge finishing
					5 =	Rough-finishing
					6 =	Chamfering
				TENS:	Reserved	
				HUNDREDS:	Reserved	
				THOUSANDS:		
					1 =	Vortex milling
					2 =	Plunge cutting

No.	Parameter mask	Parameter internal	Data type	Meaning		
17		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					1 =	Normal machining
				THOUSANDS:	Dimensioning via center/edge	
					0 =	Dimensioning via center
					1 =	"Left-hand" dimensioning using edge ("-" direction of 1st axis)
					2 =	"Right-hand" dimensioning using edge ("+" direction of 1st axis)
18		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	---	Reserved
				HUNDREDS:	---	Reserved
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple
19		<_AMODE>	INT	Alternative mode		
				UNITS:	Slot depth Z1	
					0 =	Absolute
					1 =	Incremental
				TENS:	Unit for plane infeed (<_MIDA>) DXY	
					0 =	mm
					1 =	% of tool diameter
				HUNDREDS:	Insertion depth for chamfering ZFS	
					0 =	Absolute
					1 =	Incremental
20		<_UMODE>	INT	Reserved		
21	FS	<_FS>	REAL	Chamfer width (inc)		
22	ZFS	<_ZFS>	REAL	Insertion depth (tool tip) on chamfering (abs/inc), see <_AMODE>		

19.1.40 CYCLE930 - groove

Syntax

```
CYCLE930(<_SPD>, <_SPL>, <_WIDG>, <_WIDG2>, <_DIAG>, <_DIAG2>,
<_STA>, <_ANG1>, <_ANG2>, <_RCO1>, <_RCI1>, <_RCI2>, <_RCO2>,
<_FAL>, <_IDEP1>, <_SDIS>, <_VARI>, <_DN>, <_NUM>, <_DBH>, <_FF1>,
<_NR>, <_FALX>, <_FALZ>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning		
1	X0	<_SPD>	REAL	Reference point in the plane axis (always diameter)		
2	Z0	<_SPL>	REAL	Reference point along the longitudinal axis		
3	B1	<_WIDG>	REAL	Width at bottom of groove		
4	B2	<_WIDG2>	REAL	Width at top of groove (for interface only)		
5	T1	<_DIAG>	REAL	Depth of groove at the reference point for abs and longitudinal machining = diameter, otherwise inc		
6	T2	<_DIAG2>	REAL	Groove depth opposite the reference point (for interface only), for abs and longitudinal machining = diameter, otherwise inc		
7	α0	<_STA>	REAL	Angle of inclination (-180 ≤ <_STA> ≤ 180)		
8	α1	<_ANG1>	REAL	Side angle 1 (0 ≤ <_ANG1> < 90) at the side of the groove determined by the reference point		
9	α2	<_ANG2>	REAL	Side angle 2 (0 ≤ <_ANG2> < 90) opposite the reference point		
10	R1/FS1	<_RCO1>	REAL	Rounding radius or chamfer width 1, external at the reference point		
11	R2/FS2	<_RCI1>	REAL	Rounding radius or chamfer width 2, internal at the reference point		
12	R3/FS3	<_RCI2>	REAL	Rounding radius or chamfer width 3, internal opposite the reference point		
13	R4/FS4	<_RCO2>	REAL	Rounding radius or chamfer width 4, external opposite the reference point		
14	U	<_FAL>	REAL	Finishing allowance in X and Z, see <_VARI> (TEN THOUSANDS) (to be entered without sign)		
15	D	<_IDEP1>	REAL	Maximum depth infeed on insertion (enter without sign)		
					0 =	1. Cut directly to full depth
					> 0 =	1. Cut <_IDEP1>, 2nd cut 2 · <_IDEP1>, etc.
16	SC	<_SDIS>	REAL	Safety clearance (enter without sign)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
17		<_VARI>	INT	Machining type		
				UNITS:	Reserved	
				TENS:	Machining process	
					1 =	Roughing
					2 =	Finishing
					3 =	Roughing and finishing
				HUNDREDS:	Position longitudinal/transverse external/ internal +Z/+Z and +X/-X	
					1 =	Longitudinal/external +Z
					2 =	Transverse/internal -X
					3 =	Longitudinal/internal +Z
					4 =	Transverse/internal +X
					5 =	Longitudinal/external -Z
					6 =	Transverse/external -X
					7 =	Longitudinal/internal -Z
					8 =	Transverse/external +X
				THOUSANDS:	Position of reference point	
					0 =	Upper reference point
					1 =	Lower reference point
				TEN THOUSANDS:	Define effect of finishing allowances	
					0 =	Finishing allowance U parallel to the contour
					1 =	Separate UX and UZ finishing allowances
18		<_DN>	INT	D number for 2nd edge of tool		
					> 0 =	D number for tool offset of 2nd edge of grooving tool
					0 =	No 2nd edge programmed
19	N	<_NUM>	INT	Number of grooves (0 = 1 groove)		
20	DP	<_DBH>	REAL	Distance between grooves (only needed when <_NUM> > 1)		
21	F	<_FF1>	REAL	Feedrate		
22		<_NR>	INT	Identification for form of groove corresponds to vertical softkey for form selection		
					0 =	90° sides without chamfers/ rounding
					1 =	Inclined sides with chamfers/ rounding (without α0)
					2 =	As 1, but on taper (with α0)
23	UX	<_FALX>	REAL	Finishing allowance in X axis, see <_VARI> (TEN THOUSANDS) (to be entered without sign)		
24	UZ	<_FALZ>	REAL	Finishing allowance in Z axis, see <_VARI> (TEN THOUSANDS) (to be entered without sign)		

No.	Parameter mask	Parameter internal	Data type	Meaning		
25		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
26		<_AMODE>	INT	Alternative mode		
				UNITS:	Dimensioning for top of groove (for interface only)	
					0 =	At the reference point
					1 =	Opposite the reference point
				TENS:	Depth	
					0 =	Absolute
					1 =	Incremental
				HUNDREDS:	Dimensioning for width (for interface only)	
					0 =	At outer diameter (top)
					1 =	At inner diameter (bottom)
				THOUSANDS:	Radius/chamfer 1 (<_RCO1>)	
					0 =	Radius
					1 =	Chamfer
				TEN THOUSANDS:	Radius/chamfer 2 (<_RCI1>)	
					0 =	Radius
					1 =	Chamfer
				HUNDRED THOUSANDS:	Radius/chamfer 3 (<_RCI2>)	
					0 =	Radius
					1 =	Chamfer
				ONE MILLION:	Radius/chamfer 4 (<_RCO2>)	
					0 =	Radius
					1 =	Chamfer

19.1.41 CYCLE940 - undercut forms

Various undercuts can be programmed using the CYCLE940 cycle. In some cases, these differ significantly regarding the parameterization.

The additional columns in the table indicate which parameters are required for which undercut type. They correspond to the vertical selection softkeys in the cycle screen form:

- E: Undercut form E
- F: Undercut form F

- A-D: DIN thread undercut (forms A-D)
- T: Thread undercut (free definition of form)

Syntax

```
CYCLE940(<_SPD>, <_SPL>, <_FORM>, <_LAGE>, <_SDIS>, <_FFP>, <_VARI>,
<_EPD>, <_EPL>, <_R1>, <_R2>, <_STA>, <_VRT>, <_MID>, <_FAL>,
<_FALX>, <_FALZ>, <_PITI>, <_PTAB>, <_PTABA>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Prog. for form				Meaning		
				E	F	A-D	T			
1	X0	<_SPD>	REAL	x	x	x	x	Reference point in the plane axis (always diameter)		
2	Z0	<_SPL>	REAL	x	x	x	x	Reference point on longitudinal axis (abs)		
3	FORM	<_FORM>	CHAR	x	x	x	x	Form of undercut (capital letters, e.g. "T") Selection, table from which the undercut values should be taken		
									A =	External, reference DIN76, A = normal
									B =	External, reference DIN76, B = short
									C =	Internal, reference DIN76, C = normal
									D =	Internal, reference DIN76, D = short
									E =	Reference DIN509
									F =	Reference DIN509
									T=	Free-form
4	POSITION	<_LAGE>	INT	x	x	x	x	Position of undercut (parallel Z)	0 =	External +Z: ____|
									1 =	External -Z: |____/
									2 =	Internal +Z: /-----|
									3 =	Internal -Z: |-----\
5	SC	<_SDIS>		x	x	x	x	Safety clearance (inc)		
6	F	<_FFP>		x	x	x	x	Machining feedrate (mm/rev)		
7		<_VARI>	INT	-	-	x	x	Machining type		
								UNITS:	Machining	
									1 =	Roughing
									2 =	Finishing
									3 =	Roughing + finishing
								TENS:	Machining strategy	
									0 =	Parallel to the contour
									1 =	Longitudinal
				Undercut forms E and F are always machined in a single pass like finishing.						

No.	Parameter mask	Parameter internal	Data type	Prog. for form				Meaning			
8	X1	<_EPD>		x	x	-	-	Allowance X (abs/inc), see <_AMODE>			
				-	-	-	x	Undercut depth (abs/inc), see <_AMODE>			
9	Z1	<_EPL>		-	x	-	-	Allowance Z			
				-	-	-	x	Undercut width (abs/inc), see <_AMODE>			
10	R1	<_R1>		-	-	-	x	Rounding radius on slopes			
11	R2	<_R2>		-	-	-	x	Rounding radius in the corner			
12	α	<_STA>		-	-	x	x	Insertion angle			
13	VX	<_VRT>		x	x	-	-	Cross-feed X (abs/inc), see <_AMODE>			
				-	-	x	x	Cross-feed X when finishing, (abs/inc), see <_AMODE>			
14	D	<_MID>		-	-	x	x	Depth infeed			
15	U	<_FAL>		-	-	x	x	Finishing allowance parallel to contour, see <_AMODE>			
16	UX	<_FALX>		-	-	x	x	Finishing allowance X			
17	UZ	<_FALZ>		-	-	x	x	Finishing allowance Z			
18	P	<_PITI>	INT	-	-	x	-	Select pitch, form A-D, corresponds to M1 ... M68			
								0 = 0.20 1 = 0.25 2 = 0.30 3 = 0.35 4 = 0.40 5 = 0.45	6 = 0.50 7 = 0.60 8 = 0.70 9 = 0.75 10 = 0.80 11 = 1.00	12 = 1.25 13 = 1.50 14 = 1.75 15 = 2.00 16 = 2.50 17 = 3.00	18 = 3.50 19 = 4.00 20 = 4.50 21 = 5.00 22 = 5.50 23 = 6.00
				x	x	-	-	Select radius/depth, form E, F			
								0 = 0.6 x 0.3 1 = 1.0 x 0.4 2 = 1.0 x 0.2 3 = 1.6 x 0.3	4 = 2.5 x 0.4 5 = 4.0 x 0.5 6 = 0.4 x 0.2 7 = 0.6 x 0.2	8 = 0.1 x 0.1 9 = 0.2 x 0.1	
19		<_PTAB>	STRING [5]					String for thread table ("", "ISO", "BSW", "BSP", "UNC") (for the surface only)			
20		<_PTABA>	STRING [20]					String for selection from thread table (e.g. "M 10", "M 12", ...) (for the surface only)			
21		<_DMODE>	INT					Display mode			
				x	x	x	x	UNITS:	Machining plane G17/G18/G19		
									0 =	Compatibility, the plane effective before the cycle call remains active	
									1 =	G17 (only active in the cycle)	
									2 =	G18 (only active in the cycle)	
									3 =	G19 (only active in the cycle)	

No.	Parameter mask	Parameter internal	Data type	Prog. for form				Meaning		
22		<_AMODE>	INT					Alternative mode		
				x	x	-	x	UNITS:	Parameter <_EPD> allowance X or undercut depth	
									0 =	Absolute (always diameter)
									1 =	Incremental
				x	x	-	x	TENS:	Parameter <_EPL> allowance Z or undercut width	
									0 =	Absolute
									1 =	Incremental
				x	x	x	x	HUNDREDS:	Parameter <_VRT> cross-feed X	
									0 =	Absolute (always diameter)
									1 =	Incremental
				-	-	x	x	THOUSANDS:	Finishing allowance	
									0 =	Finishing allowance parallel to the contour (<_FAL>)
									=	Separate machining allowance (<_FALX>/<_FALZ>)

19.1.42 CYCLE951 - stock removal

Syntax

```
CYCLE951(<_SPD>, <_SPL>, <_EPD>, <_EPL>, <_ZPD>, <_ZPL>, <_LAGE>,
<_MID>, <_FALX>, <_FALZ>, <_VARI>, <_RF1>, <_RF2>, <_RF3>, <_SDIS>,
<_FF1>, <_NR>, <_DMODE>, <_AMODE>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	X0	<_SPD>	REAL	Reference point (abs, always diameter)
2	Z0	<_SPL>	REAL	Reference point (abs)
3	X1	<_EPD>	REAL	End point
4	Z1	<_EPL>	REAL	End point
5	XM α1 α2	<_ZPD>	REAL	Intermediate point, see <_DMODE> (TENS)
6	ZM α1 α2	<_ZPL>	REAL	Intermediate point, see <_DMODE> (TENS)

No.	Parameter mask	Parameter internal	Data type	Meaning		
7	Position	<_LAGE>	INT	Position of stock removal corner	0 =	External/rear
					1 =	External/front
					2 =	Internal/rear
					3 =	Internal/front
8	D	<_MID>	REAL	Maximum depth infeed on insertion		
9	UX	<_FALX>	REAL	Finishing allowance in X		
10	UZ	<_FALZ>	REAL	Finishing allowance in Z		
11		<_VARI>	INT	Machining type		
				UNITS:	Stock removal direction (longitudinal or transverse) in the coordinate system	
					1 =	Longitudinal
					2 =	Face
				TENS:		
					1 =	Roughing to final machining allowance
					2 =	Finishing
				HUNDREDS:	Reserved	
				THOUSANDS:	Reserved	
				TEN THOUSANDS:	Reserved	
12	R1/FS1	<_RF1>	REAL	Rounding radius or chamfer width 1, see <_AMODE> (TEN THOUSANDS)		
13	R2/FS2	<_RF2>	REAL	Rounding radius or chamfer width 2, see <_AMODE> (HUNDRED THOUSANDS)		
14	R3/FS3	<_RF3>	REAL	Rounding radius or chamfer width 3, see <_AMODE> (ONE MILLION)		
15	SC	<_SDIS>	REAL	Safety clearance		
16	F	<_FF1>	REAL	Feedrate for roughing/finishing		
17		<_NR>	INT	Identification of stock removal type (corresponds to vertical softkey for selecting form):		
					0 =	Stock removal 1, 90 degree corner without chamfers/rounding
					1 =	Stock removal 2, 90 degree corner with chamfers/rounding
					2 =	Stock removal 3, any corner with chamfers/rounding

No.	Parameter mask	Parameter internal	Data type	Meaning		
18		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Form of input <_ZPD>/<_ZPL>	
					0 =	Xm/Zm
					1 =	Xm/α1
					2 =	Xm/α2
					3 =	α1/Zm
					4 =	α2/Zm
					5 =	α1/α2
21		<_AMODE>	INT	Alternative mode		
				UNITS:	Intermediate point in X	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
				TENS:	Intermediate point in Z	
					0 =	Absolute
					1 =	Incremental
				HUNDREDS:	End point in X	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
				THOUSANDS:	End point in Z.	
					0 =	Absolute
					1 =	Incremental
				TEN THOUSANDS:	Radius/chamfer 1	
					0 =	Radius
					1 =	Chamfer
				HUNDRED THOUSANDS:	Radius/chamfer 2	
					0 =	Radius
					1 =	Chamfer
				ONE MILLION:	Radius/chamfer 3	
					0 =	Radius
					1 =	Chamfer

19.1.43 CYCLE952 - contour grooving

Syntax

```
CYCLE952(<_PRG>, <_CON>, <_CONR>, <_VARI>, <_F>, <_FR>, <_RP>, <_D>,
<_DX>, <_DZ>, <_UX>, <_UZ>, <_U>, <_U1>, <_BL>, <_XD>, <_ZD>, <_XA>,
<_ZA>, <_XB>, <_ZB>, <_XDA>, <_XDB>, <_N>, <_DP>, <_DI>, <_SC>,
<_DN>, <_GMODE>, <_DMODE>, <_AMODE>, <_PK>, <_DCH>, <_FS>)
```

Parameters

No.	Parameter mask	Parameter internal	Data type	Meaning
1	PRG	<_PRG>	STRING[100]	Name of the stock removal program
2	CON	<_CON>	STRING[100]	Name of the program from which the updated contour of the blank is read (for residual machining)
3	CONR	<_CONR>	STRING[100]	Name of the program into which the updated contour for the blank (see <_AMODE> TEN THOUSANDS) will be written

No.	Parameter mask	Parameter internal	Data type	Meaning		
4		<_VARI>	INT	Machining type		
				UNITS:	Type of stock removal	
					1 =	Longitudinal
					2 =	Face
					3 =	Parallel to the contour
				TENS:	Machining process (see <_GMODE> HUNDREDS)	
					1 =	Roughing
					2 =	Finishing
					3 =	Reserved
					4 =	Roughing, two-channel
					5 =	Finishing, two-channel
				HUNDREDS:	Machining direction	
					1 =	Machining direction X -
					2 =	Machining direction X +
					3 =	Machining direction Z -
					4 =	Machining direction Z +
				THOUSANDS:	Infeed direction	
					1 =	External X -
					2 =	Internal X +
					3 =	Front face Z -
					4 =	Rear face Z +
				TEN THOUSANDS:	Define effect of finishing allowances	
					0 =	Separate UX and UZ finishing allowances
					1 =	Finishing allowance U parallel to the contour
				HUNDRED THOUSANDS:	Rounding	
					0 =	Compatibility, automatic rounding
					1 =	With rounding at the contour
					2 =	Without rounding
					3 =	Automatic rounding
				ONE MILLION:	Relief cuts	
					0 =	Position is not evaluated during grooving, - residual and groove turning, - remainder
					1 =	Machine relief cuts
					2 =	No machining of relief cuts
				TEN MILLIONS:	Behind/in front of turning center	
					0 =	Machining in front of the turning center
					1 =	Reserved

No.	Parameter mask	Parameter internal	Data type	Meaning		
5	F	<_F>	REAL	Feedrate for roughing/finishing		
	FZ			Infeed abscissa groove turning		
6	FR	<_FR>	REAL	Feedrate for insertion into relief cuts, roughing		
	FX			Infeed ordinate groove turning		
7	RP	<_RP>	REAL	Retraction plane for internal machining (abs., always diameter)		
8	D	<_D>	REAL	Roughing infeed (see <_AMODE> UNITS)		
9	DX	<_DX>	REAL	X infeed (see <_AMODE> UNITS)		
10	DZ	<_DZ>	REAL	Z infeed (see <_AMODE> UNITS)		
11	UX	<_UX>	REAL	Finishing allowance X, (see <_VARI> TEN THOUSANDS)		
12	UZ	<_UZ>	REAL	Finishing allowance Z, (see <_VARI> TEN THOUSANDS)		
13	U	<_U>	REAL	Finishing allowance parallel to contour, (see <_VARI> TEN THOUSANDS)		
14	U1	<_U1>	REAL	Additional finishing allowance while finishing (see <_AMODE> THOUSANDS)		
15	BL	<_BL>	INT	Definition of blank	1 =	Cylinder with allowance
					2 =	Allowance at contour of finished part
					3 =	Contour of blank is specified
16	XD	<_XD>	REAL	Definition of blank X (see <_AMODE> HUNDRED THOUSANDS)		
17	ZD	<_ZD>	REAL	Definition of blank Z (see <_AMODE> ONE MILLION)		
18	XA	<_XA>	REAL	Limit 1 X (abs., always diameter)		
19	ZA	<_ZA>	REAL	Limit 1 Z (abs.)		
20	XB	<_XB>	REAL	Limit 2 X (see <_AMODE> TEN MILLIONS)		
21	ZB	<_ZB>	REAL	Limit 2 Z (see <_AMODE> HUNDRED MILLIONS)		
22	XDA	<_XDA>	REAL	Grooving limit 1 for the 1st groove position on the front face (abs., always diameter)		
23	XDB	<_XDB>	REAL	Grooving limit 2 for the 1st groove position on the front face (abs., always diameter)		
24	N	<_N>	INT	Number of grooves		
25	DP	<_DP>	REAL	Distance between grooves	Longitudinal groove: Parallel to Z axis	
					Transverse groove: Parallel to X axis	
26	DI	<_DI>	REAL	Distance for interruption of infeed	0 =	No interruption
					> 0 =	With interruption
27	SC	<_SC>	REAL	Safety clearance for avoiding obstacles, incremental		
28	D2	<_DN>	INT	D number for 2nd edge of tool; if not programmed ⇒ D+1		

No.	Parameter mask	Parameter internal	Data type	Meaning		
29		<_GMODE>	INT	Geometrical mode (evaluation of programmed geometrical data)		
				UNITS:	Reserved	
				TENS:	Reserved	
				HUNDREDS:	Select machining/only calculation of start point	
					0 =	Normal machining (no compatibility mode needed)
					1 =	Normal machining
					2 =	Calculate start point - no machining (only for call from ShopMill/ ShopTurn)
				THOUSANDS:	Limit	
					0 =	No
					1 =	Yes
				TEN THOUSANDS:	Enter limit 1 X	
					0 =	No
					1 =	Yes
				HUNDRED THOUSANDS:	Enter limit 2 X	
					0 =	No
					1 =	Yes
				ONE MILLION:	Enter limit 1 Z	
					0 =	No
					1 =	Yes
				TEN MILLIONS:	Enter limit 2 Z	
					0 =	No
					1 =	Yes

No.	Parameter mask	Parameter internal	Data type	Meaning		
30		<_DMODE>	INT	Display mode		
				UNITS:	Machining plane G17/G18/G19	
					0 =	Compatibility, the plane effective before the cycle call remains active
					1 =	G17 (only active in the cycle)
					2 =	G18 (only active in the cycle)
					3 =	G19 (only active in the cycle)
				TENS:	Technology mode	
					1 =	Stock removal along the contour
					2 =	Contour grooving
					3 =	Groove turning
				HUNDREDS:	Machine residual material	
					0 =	No
					1 =	Yes
				THOUSANDS:	---	Reserved
				TEN THOUSANDS:	Technology scaling in cycle screen forms (Page 723)	
					0 =	Input: Complete
					1 =	Input: Simple

No.	Parameter mask	Parameter internal	Data type	Meaning		
31		<_AMODE>	INT	Alternative mode		
				UNITS:	Select infeed	
					0 =	DX and DZ infeed for stock removal parallel to contour
					1 =	D infeed
				TENS:	Infeed strategy	
					0 =	Variable cutting depth (90 ... 100%)
					1 =	Constant cutting depth
				HUNDREDS:	Cut segmentation	
					0 =	Uniform
					1 =	Align to edges
				THOUSANDS:	Select contour allowance U1, double finishing	
					0 =	No
					1 =	Yes
				TEN THOUSANDS:	Update selection of blank	
					0 =	No
					1 =	Yes
				HUNDRED THOUSANDS:	Select allowance on blank XD	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
				ONE MILLION:	Select allowance on blank ZD	
					0 =	Absolute
					1 =	Incremental
				TEN MILLIONS:	Select limit 2 XB	
					0 =	Absolute, value of transverse axis in the diameter
					1 =	Incremental, value of transverse axis in the radius
				HUNDRED MILLION:	Select limit 2 ZB	
					0 =	Absolute
					1 =	Incremental
				ONE BILLION:		
					0 =	Leading channel
					1 =	Following channel
32		<_PK>	INT	Number of the partner channel if there are more than two channels available at the machine.		
33	DCH	<_DCH>	REAL	Channel offset		
34	FS	<_FS>	REAL	Finishing feedrate during complete machining		

19.1.44 CYCLE4071 - longitudinal grinding with infeed at the reversal point

Syntax

```
CYCLE4071(<S_A>, <S_B>, <S_W>, <S_U>, <S_I>, <S_K>, <S_H>, <S_A1>,
<S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_A>	REAL	Infeed depth at the start
2	<S_B>	REAL	Infeed depth at the end
3	<S_W>	REAL	Grinding width
4	<S_U>	REAL	Sparking-out time
5	<S_I>	REAL	Feedrate for infeed
6	<S_K>	REAL	Feedrate for transverse infeed
7	<S_H>	INT	Number of repetitions
8	<S_A1>	AXIS	Infeed axis (optional) or 1st geometry axis
9	<S_A2>	AXIS	Oscillating axis (optional) or 2nd geometry axis

Function

The cycle is used for the execution of <S_H> infeeds. The infeed depth at the start and at the end can be different. There is a tangential motion between the infeeds.

Sequence

1. Start of the cycle at the current position of the oscillating axis.
2. Traversing of the infeed axis to the infeed depth at the start P1 **<**S_A**>** with the feedrate for infeed P5 <S_I>.
3. Sparking out with the sparking-out time P4 **<**S_U**>**.
4. Traversing of the oscillating axis with the grinding width P3 **<**S_W**>** as travel path and the feedrate for transverse infeed P6 <S_K>.
5. Traversing of the infeed axis to the infeed depth at the end P2 **<**S_B**>** with the feedrate for infeed P5 <S_I>.
6. Sparking out with the sparking-out time P4 **<**S_U**>**.
7. Traversing of the oscillating axis with the grinding width P4 **<**S_A**>** as travel path to the starting point and the feedrate for transverse infeed P6 <S_K>.

The sequence cannot be interrupted with a single block.

The sequence is repeated according to the number of programmed repetitions P7 (<S_H>).



Example

Executing two oscillating motions with the following cycle parameters:

- Infeed depth at the start: 0.02 mm
- Infeed depth at the end: 0.01 mm
- Stroke: 100 mm
- Sparking-out time: 1 s
- Infeed feedrate: 1 mm/min
- Transverse feedrate: 1000 mm/min
- Repetitions: 2
- Oscillating and infeed axes: Standard geometry axes

Program code

```
N10 T1 D1
N20 CYCLE4071(0.02,0.01,100,1,1,1000,2)
N30 M30
```

19.1.45 CYCLE4072 - longitudinal grinding with infeed at the reversal point and cancel signal

Syntax

```
CYCLE4072(<S_GAUGE>, <S_A>, <S_B>, <S_W>, <S_U>, <S_I>, <S_K>,
<S_H>, <S_A1>, <S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_GAUGE>	STRING	Cancel conditions for infeed: 1. Number of a rapid input 2. Logical expression
2	<S_A>	REAL	Infeed depth at the start
3	<S_B>	REAL	Infeed depth at the end
4	<S_W>	REAL	Grinding width
5	<S_U>	REAL	Sparking-out time
6	<S_I>	REAL	Feedrate for infeed
7	<S_K>	REAL	Feedrate for transverse infeed
8	<S_H>	INT	Number of repetitions
9	<S_A1>	AXIS	Infeed axis (optional) or 1st geometry axis
10	<S_A2>	AXIS	Oscillating axis (optional) or 2nd geometry axis

Function

The cycle is used for the execution of <S_H> infeeds taking into account an external cancel signal. The infeed depth can be different at the start and at the end. There is a tangential motion between the infeeds. The depth infeed is cancelled when the cancel condition is satisfied. A complete stroke is always performed after the cancellation of the depth infeed.

Sequence

1. Start of the cycle at the current position of the oscillating axis.
2. Traversing of the infeed axis to the infeed depth at the start P2 **<**S_A**>** with the feedrate for infeed P6 <S_I>.
3. Sparking out with the sparking-out time P5 **<**S_U**>**.
4. Traversing of the oscillating axis with the grinding width P4 **<**S_W**>** as travel path and the feedrate for transverse infeed P7 <S_K>.
5. Traversing of the infeed axis to the infeed depth at the end P3 **<**S_B**>** with the feedrate for infeed P6 <S_I>.
6. Sparking out with the sparking-out time P5 **<**S_U**>**.
7. Traversing of the oscillating axis with the grinding width P4 **<**S_W**>** as travel path to the starting point and the feedrate for transverse infeed P7 <S_K>.
8. Without cancellation: The sequence described above is repeated until the number of programmed repetitions P7 (<S_H>) is reached.
   With cancellation: The machining is stopped when the next start point is reached.

The sequence cannot be interrupted by a single block.

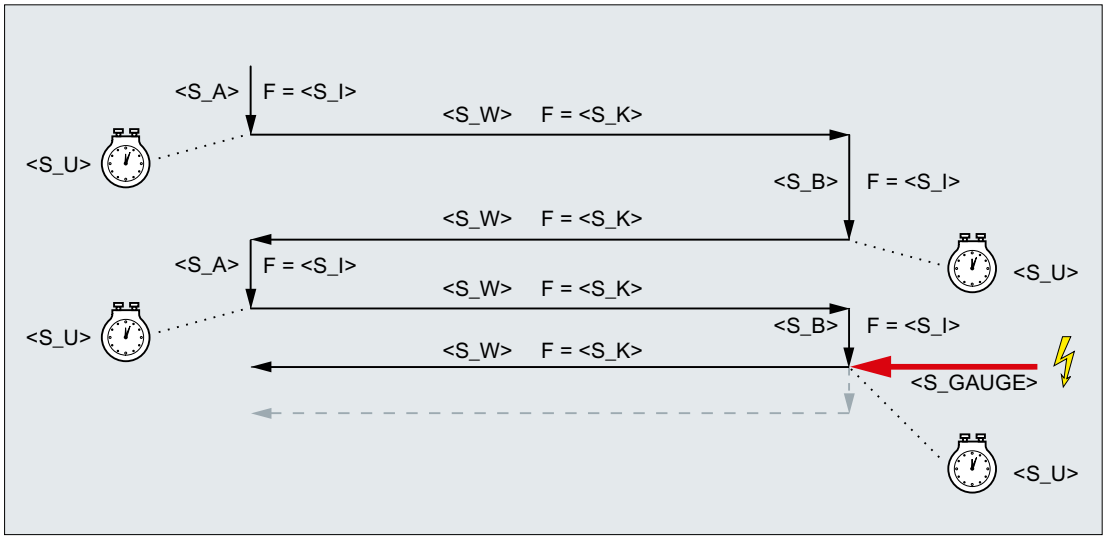


Figure 19-1 Cancellation of the infeed at the end

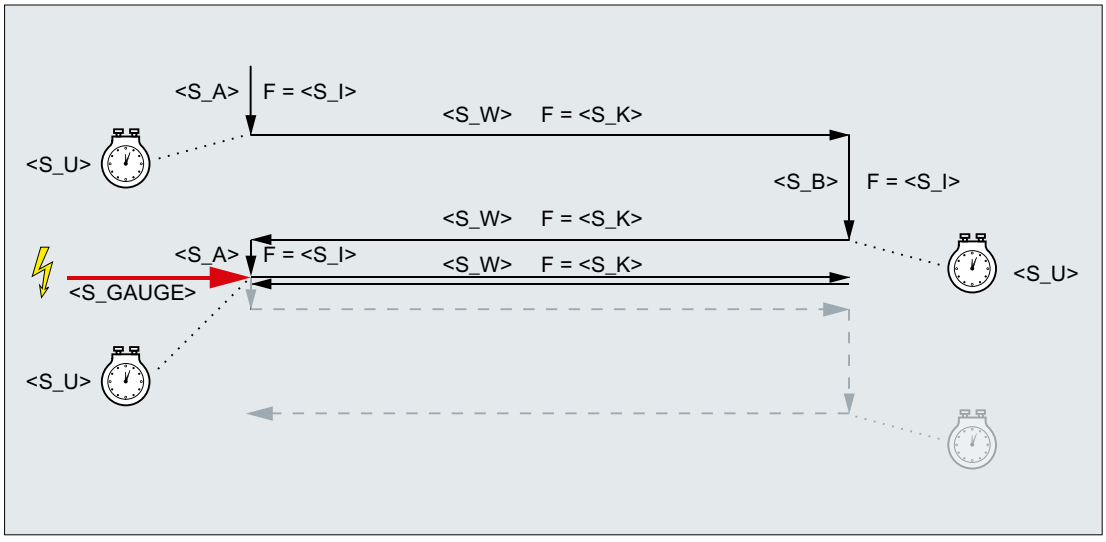


Figure 19-2 Cancellation of the infeed at the start

Resources

As resources, the cycle uses a block-wide synchronized action and a synchronized action variable. The synchronized action is determined dynamically from the free area of the synchronized action range (CUS.DIR - 1 ..., CMA.DIR - 1000 ..., CST.DIR – 1199 ...). SYG_IS[1] is used as the synchronized action variable.

Examples

Example 1: Oscillation with two strokes:

Cycle parameters

- Infeed depth at the start: 0.02 mm
- Infeed depth at the end: 0.01 mm
- Stroke: 100 mm
- Sparking-out time: 1 s
- Infeed feedrate: 1 mm/min
- Transverse feedrate: 1000 mm/min
- Repetitions: 2
- Oscillating and infeed axes: Standard geometry axes

Cancel signal Rapid input 1 ($A_IN[1])

Program code

```
N10 T1 D1
N20 CYCLE4072("1",0.02,0.01,100,1,1,1000,2)
N30 M30
```

Example 2: Oscillation with two strokes:

Cycle parameters

- Infeed depth at the start: 0.02 mm
- Infeed depth at the end: 0.01 mm
- Stroke: 100 mm
- Sparking-out time: 1 s
- Infeed feedrate: 1 mm/min
- Transverse feedrate: 1000 mm/min
- Repetitions: 2
- Oscillating and infeed axes: Standard geometry axes

Cancel signal Variable $A_DBR[20] < 0.01

Program code

```
N10 T1 D1
N20 CYCLE4072("($A_DBR[20]<0.01)",0.02,0.01,100,1,1,1000,2)
N30 M30
```

19.1.46 CYCLE4073 - longitudinal grinding with continuous infeed

Syntax

```
CYCLE4073(<S_A>, <S_B>, <S_W>, <S_U>, <S_K>, <S_H>, <S_A1>, <S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_A>	REAL	Infeed depth at the start
2	<S_B>	REAL	Infeed depth at the end
3	<S_W>	REAL	Grinding width
4	<S_U>	REAL	Sparking-out time
5	<S_K>	REAL	Feedrate for transverse infeed
6	<S_H>	INT	Number of repetitions
7	<S_A1>	AXIS	Infeed axis (optional) or 1st geometry axis
8	<S_A2>	AXIS	Oscillating axis (optional) or 2nd geometry axis

Function

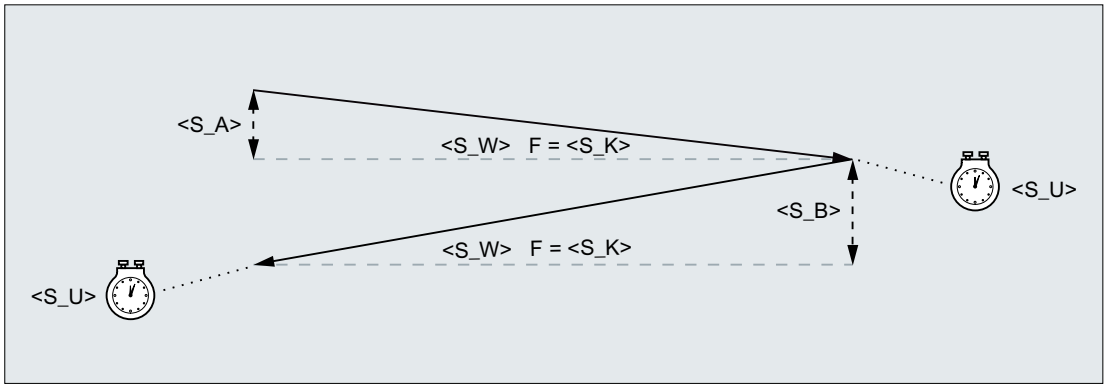
The cycle is used for the execution of <S_H> infeeds. The infeed from the start to the end and from the end to the start can be different.

Sequence

1. Start of the cycle at the current position of the oscillating axis with infeed depth 0
2. Traversing of the oscillating axis with the grinding width P3 <S_W> as travel path and feedrate for transverse infeed P5 <S_K> with continuous increase in the infeed depth up to the infeed depth at the start P1 <S_A>.
3. Sparking out with the sparking-out time P4 <S_U>.
4. Traversing of the oscillating axis with the grinding width P3 <S_W> as travel path to the starting point and feedrate for transverse infeed P5 <S_K> with continuous increase in the infeed depth up to the infeed depth at the end P2 <S_B>.
5. Sparking out with the sparking-out time P4 <S_U>.

The sequence cannot be interrupted by a single block.

The sequence is repeated according to the number of programmed repetitions P7 (<S_H>).



Example

Oscillation with two strokes:

Cycle parameters

- Infeed depth at the start: 0.02 mm
- Infeed depth at the end: 0.01 mm
- Stroke: 100 mm
- Sparking-out time: 1 s
- Transverse feedrate: 1000 mm/min
- Repetitions: 2
- Oscillating and infeed axes: Standard geometry axes

Program code

```
N10 T1 D1
N20 CYCLE4073(0.02,0.01,100,1,1000,2)
N30 M30
```

19.1.47 CYCLE4074 - longitudinal grinding with continuous infeed and cancel signal

Syntax

```
CYCLE4074(<S_GAUGE>, <S_A>, <S_B>, <S_W>, <S_U>, <S_K>, <S_H>,
<S_A1>, <S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_GAUGE>	STRING	Cancel conditions for infeed: 1. Number of a rapid input 2. Logical expression
2	<S_A>	REAL	Infeed depth at the start
3	<S_B>	REAL	Infeed depth at the end
4	<S_W>	REAL	Grinding width
5	<S_U>	REAL	Sparking-out time
6	<S_K>	REAL	Feedrate for transverse infeed
7	<S_H>	INT	Number of repetitions
8	<S_A1>	AXIS	Infeed axis (optional) or 1st geometry axis
9	<S_A2>	AXIS	Oscillating axis (optional) or 2nd geometry axis

Function

The cycle is used for the execution of <S_H> infeeds taking into account an external cancel signal, for example. The infeed depth can be different at the start and at the end. The depth infeed is cancelled when the cancel condition is satisfied. A complete stroke is always performed after the cancellation of the depth infeed.

Sequence

1. Start of the cycle at the current position of the oscillating axis with infeed depth 0
2. Traversing of the oscillating axis with the grinding width P4 <S_W> as travel path and feedrate for transverse infeed P6 <S_K> with continuous increase in the infeed depth up to the infeed depth at the start P2 <S_A>.
3. Sparking out with the sparking-out time P5 <S_U>.
4. Traversing of the oscillating axis with the grinding width P4 <S_W> as travel path to the starting point and feedrate for transverse infeed P6 <S_K> with continuous increase in the infeed depth up to the infeed depth at the end P3 <S_B>.
5. Sparking out with the sparking-out time P5 <S_U>.
6. Without cancellation: The sequence described above is repeated until the number of programmed repetitions P7 (<S_H>) is reached.
   With cancellation: The depth infeed is cancelled. The machining is stopped when the next start point is reached.

The sequence cannot be interrupted by a single block.

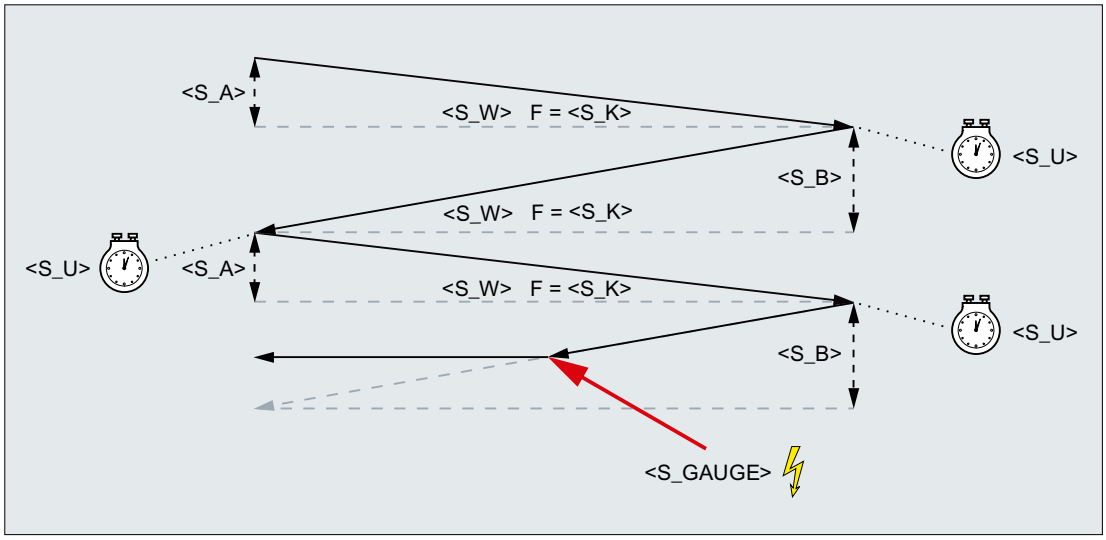


Figure 19-3 Cancellation of the infeed from the end to the start

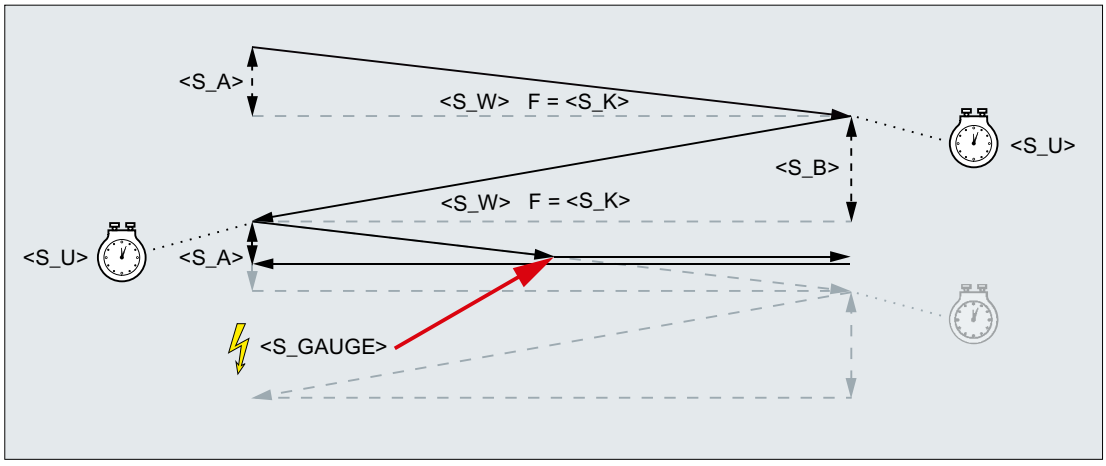


Figure 19-4 Cancellation of the infeed from the start to the end

Resources

As resources, the cycle uses a block-wide synchronized action and a synchronized action variable. The synchronized action is determined dynamically from the free area of the synchronized action range (CUS.DIR - 1 ..., CMA.DIR - 1000 ..., CST.DIR – 1199 ...). SYG_IS[1] is used as the synchronized action variable.

Examples

Example 1: Oscillation with two strokes:

Cycle parameters

- Infeed depth at the start: 0.02 mm
- Infeed depth at the end: 0.01 mm

- Stroke: 100 mm
- Sparking-out time: 1 s
- Transverse feedrate: 1000 mm/min
- Repetitions: 2
- Oscillating and infeed axes: Standard geometry axes

Cancel signal Rapid input 1 ($A_IN[1])

Program code

```
N10 T1 D1
N20 CYCLE4074("1",0.02,0.01,100,1,1000,2)
N30 M30
```

Example 2: Oscillation with two strokes:

Cycle parameters

- Infeed depth at the start: 0.02 mm
- Infeed depth at the end: 0.01 mm
- Stroke: 100 mm
- Sparking-out time: 1 s
- Transverse feedrate: 1000 mm/min
- Repetitions: 2
- Oscillating and infeed axes: Standard geometry axes

Cancel signal Variable $A_DBR[20] < 0.01

Program code

```
N10 T1 D1
N20 CYCLE4074("($A_DBR[20]<0.01)",0.02,0.01,100,1,1000,2)
N30 M30
```

19.1.48 CYCLE4075 - surface grinding with infeed at the reversal point

Syntax

```
CYCLE4075(<S_I>, <S_J>, <S_K>, <S_A>, <S_R>, <S_F>, <S_P>, <S_A1>,
<S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_I>	REAL	Infeed depth at the start
2	<S_J>	REAL	Infeed depth at the end

No.	Parameter	Data type	Meaning
3	<S_K>	REAL	Total infeed depth
4	<S_A>	REAL	Grinding width
5	<S_R>	REAL	Feedrate for infeed
6	<S_F>	REAL	Feedrate for transverse infeed
7	<S_P>	REAL	Sparking-out time
8	<S_A1>	AXIS	Infeed axis (optional)
9	<S_A2>	AXIS	Oscillating axis (optional)

Function

The cycle is used for machining with a total infeed depth P3 <S_K> in infeed steps. The infeed depths at the start P1 <S_I> and at the end P2 <S_J> can be different. There is a tangential motion between the infeeds.

The positional data P1 to P4 can be negative or positive.

The specification of the infeed axis P8 <S_A1> and/or oscillating axis P9 <S_A2> are optional. If one or both parameters are not specified, the cycle uses the first two geometry axes of the channel.

If the sum of the infeed depth at the start P1 <S_I> and at the end P2 <S_J> is 0 or the total infeed depth P3 <S_K> is 0, only a sparking-out stroke is performed.

Sequence

1. Start of the cycle at the current position of the oscillating axis.
2. Traversing of the infeed axis to the infeed depth at the start P1 <S_I> with the feedrate for infeed P5 <S_R>.
3. Sparking out with the sparking-out time P7 <S_P>.
4. Traversing of the oscillating axis with the grinding width P4 <S_A> as travel path and the feedrate for transverse infeed P6 <S_F>.
5. Traversing of the infeed axis to the infeed depth at the end P2 <S_J> with the feedrate for infeed P5 <S_R>.
6. Sparking out with the sparking-out time P7 <S_P>.
7. Traversing of the oscillating axis with the grinding width P4 <S_A> as travel path to the starting point and the feedrate for transverse infeed P6 <S_F>.

The sequence cannot be interrupted with a single block.

The sequence is repeated until the total infeed depth P3 <S_K> has been reached. The last stroke is then distributed unevenly.

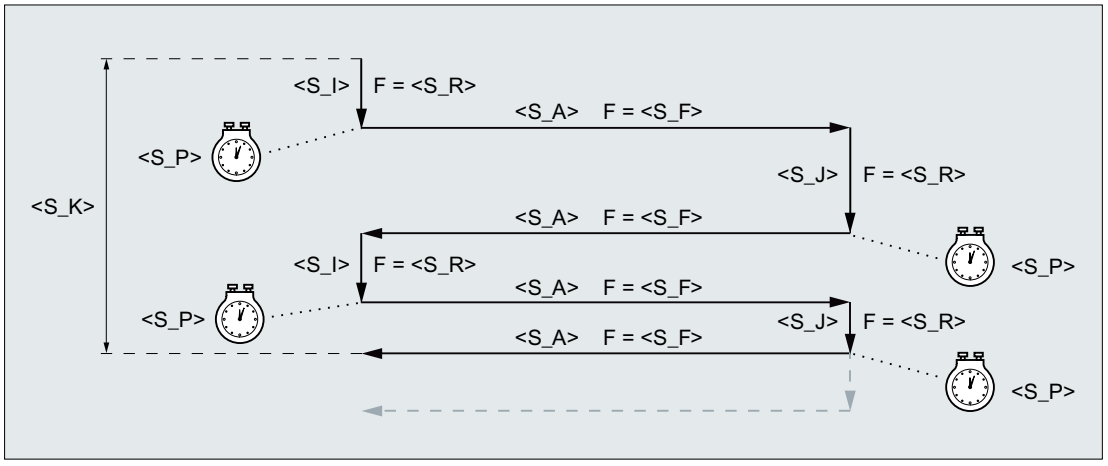


Figure 19-5 Total infeed depth reached with infeed at the second reversal point

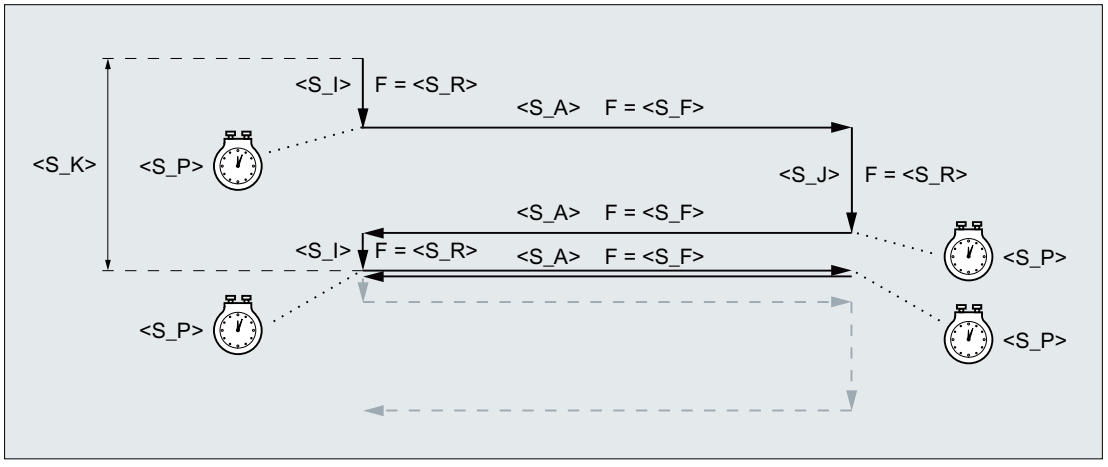


Figure 19-6 Total infeed depth reached with infeed at the first reversal point

Example

Oscillation with:

- 0.02 mm infeed depth at the start
- 0.01mm infeed depth at the end
- Total infeed depth 1 mm
- 100 mm stroke
- Infeed feedrate 1 mm/min
- Transverse feedrate 1000 mm/min
- 1 second sparking-out time
- Standard geometry axes

Program code
N10 T1 D1
N20 CYCLE4075(0.02,0.01,1,100,1,1000,1)
N30 M30

19.1.49 CYCLE4077 - surface grinding with infeed at the reversal point and cancel signal

Syntax

```
CYCLE4077(<S_GAUGE>, <S_I>, <S_J>, <S_K>, <S_A>, <S_R>, <S_F>,
<S_P>, <S_A1>, <S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_GAUGE>	STRING	Cancel condition for infeed: • Number of a rapid input • Logical expression
2	<S_I>	REAL	Infeed depth at the start
3	<S_J>	REAL	Infeed depth at the end
4	<S_K>	REAL	Total infeed depth
5	<S_A>	REAL	Grinding width
6	<S_R>	REAL	Feedrate for infeed
7	<S_F>	REAL	Feedrate for transverse infeed
8	<S_P>	REAL	Sparking-out time
9	<S_A1>	AXIS	Infeed axis (optional)
10	<S_A2>	AXIS	Oscillating axis (optional)

Function

The cycle is used for machining with a total infeed depth P4 <S_K> in infeed steps. The infeed depths at the start P2 <S_I> and at the end P3 <S_J> can be different. There is a tangential motion between the infeeds. The depth infeed is cancelled when the cancel signal of the rapid input is 1 or the cancel condition is satisfied. A complete stroke is performed after the cancellation.

The positional data P2 to P5 can be negative or positive.

The specification of the infeed axis P9 <S_A1> and/or oscillating axis P10 <S_A2> are optional. If one or both parameters are not specified, the cycle uses the first two geometry axes of the channel.

If the sum of the infeed depth at the start P2 <S_I> and at the end P3 <S_J> is 0 or the total infeed depth P4 <S_K> is 0, only a sparking-out stroke is performed.

Sequence

1. Start of the cycle at the current position of the oscillating axis.
2. Traversing of the infeed axis to the infeed depth at the start P2 <S_I> with the feedrate for infeed P6 <S_R>.
3. Sparking out with the sparking-out time P8 <S_P>.
4. Traversing of the oscillating axis with the grinding width P5 <S_A> as travel path and the feedrate for transverse infeed P7 <S_F>.
5. Traversing of the infeed axis to the infeed depth at the end P3 <S_J> with the feedrate for infeed P6 <S_R>.
6. Sparking out with the sparking-out time P8 <S_P>.
7. Traversing of the oscillating axis with the grinding width P5 <S_A> as travel path to the starting point and the feedrate for transverse infeed P7 <S_F>.
8. Without cancellation: The sequence described above is repeated until the total infeed depth P4 <S_K> has been reached. The last stroke is then distributed unevenly.
   With cancellation: The machining is stopped at the start point.

The sequence cannot be interrupted with a single block.

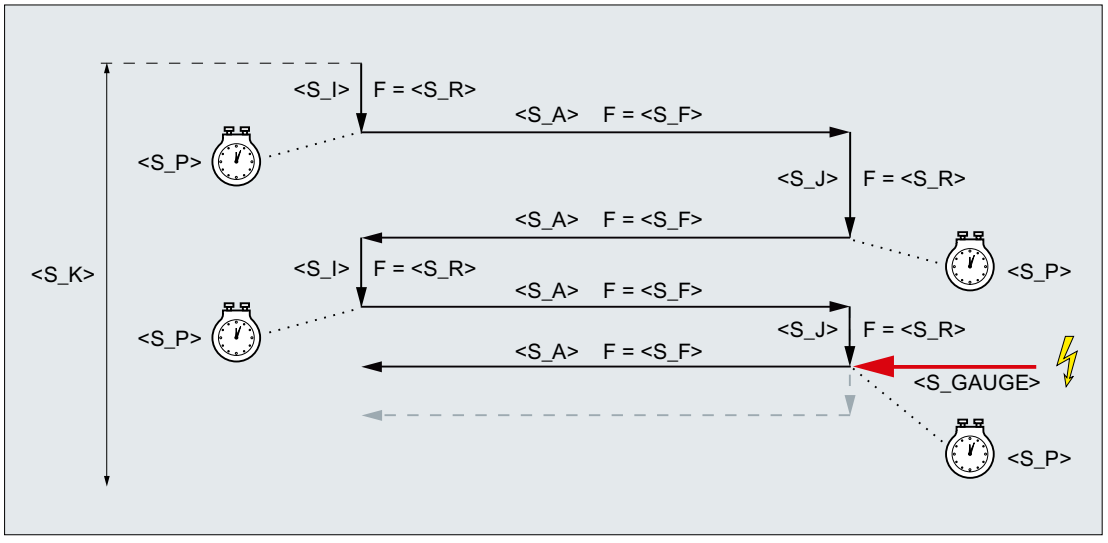


Figure 19-7 Cancellation of the infeed at the end

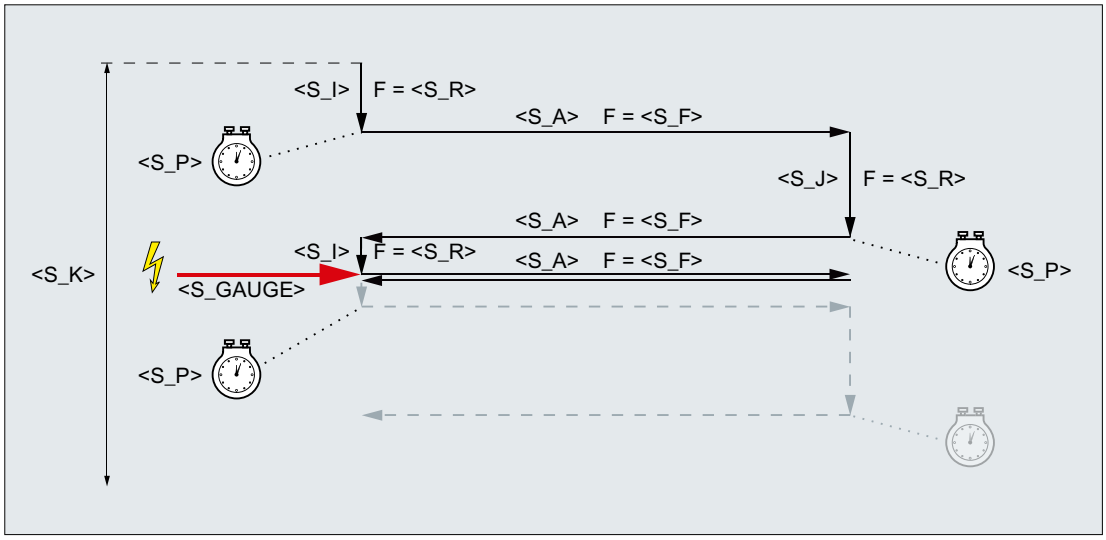


Figure 19-8 Cancellation of the infeed at the start

Resources

As resources, the cycle uses a block-wide synchronized action and a synchronized action variable. The synchronized action is determined dynamically from the free area of the synchronized action range (CUS.DIR - 1 ..., CMA.DIR - 1000 ..., CST.DIR – 1199 ...). SYG_IS[1] is used as the synchronized action variable.

Examples

Example 1

Oscillation with:

- 0.02 mm infeed depth at the start
- 0.01mm infeed depth at the end
- Total infeed depth 1 mm
- 100 mm stroke
- Infeed feedrate 1 mm/min
- Transverse feedrate 1000 mm/min
- 1 second sparking-out time
- Standard geometry axes

Cancel signal Rapid input 1 ($A_IN[1])

Program code

```
N10 T1 D1
N20 CYCLE4077("1",0.02,0.01,1,100,1,1000,1)
N30 M30
```

Example 2

Oscillation with:

- 0.02 mm infeed depth at the start
- 0.01mm infeed depth at the end
- Total infeed depth 1 mm
- 100 mm stroke
- Infeed feedrate 1 mm/min
- Transverse feedrate 1000 mm/min
- 1 second sparking-out time
- Standard geometry axes

Cancel signal Dual-port RAM variable 20 less than 0.01 ($A_DBR[20] < 0.01)

Program code

```
N10 T1 D1
N20 CYCLE4077("($A_DBR[20]<0.01)",0.02,0.01,1,100,1,1000,1)
N30 M30
```

19.1.50 CYCLE4078 - surface grinding with continuous infeed

Syntax

```
CYCLE4078(<S_I>, <S_J>, <S_K>, <S_A>, <S_F>, <S_P>, <S_A1>, <S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_I>	REAL	Infeed depth from the start to the end
2	<S_J>	REAL	Infeed depth from the end to the start
3	<S_K>	REAL	Total infeed depth
4	<S_A>	REAL	Grinding width
5	<S_F>	REAL	Feedrate
6	<S_P>	REAL	Sparking-out time
7	<S_A1>	AXIS	Infeed axis (optional)
8	<S_A2>	AXIS	Oscillating axis (optional)

Function

The cycle is used for machining with a total infeed depth P3 <S_K> by means of continuous infeed. The infeed depths from the start to the end P1 <S_I> and from the end to the start P2 <S_J> can be different.

The positional data P1 to P4 can be negative or positive.

The specification of the infeed axis P8 <S_A1> and/or oscillating axis P9 <S_A2> are optional. If one or both parameters are not specified, the cycle uses the first two geometry axes of the channel.

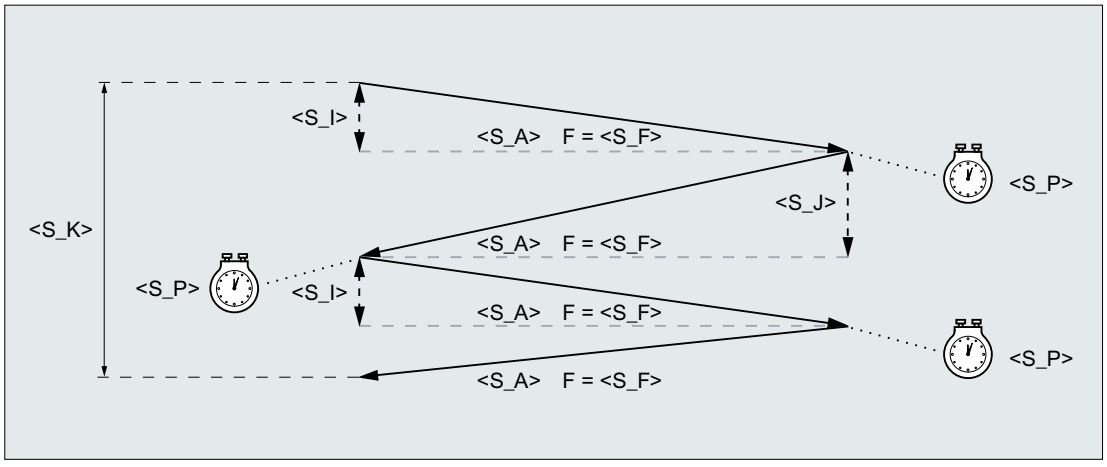
If the sum of the infeed depths P1 <S_I> and P2 <S_J> is 0 or the total infeed depth P3 <S_K> is 0, only a sparking-out stroke is performed.

Sequence

1. Start of the cycle at the current position of the oscillating axis with infeed depth 0
2. Traversing of the oscillating axis with the grinding width P4 <S_A> as travel path and feedrate P5 <S_F> with continuous increase in the infeed depth up to the infeed depth at the start P1 <S_I>.
3. Sparking out with the sparking-out time P7 <S_P>.
4. Traversing of the oscillating axis with the grinding width P4 <S_A> as travel path to the starting point and feedrate P5 <S_F> with continuous increase in the infeed depth up to the infeed depth at the end P2 <S_J>.
5. Sparking out with the sparking-out time P7 <S_P>.
6. Traversing of the oscillating axis with the grinding width P4 <S_A> as travel path to the starting point and feedrate P5 <S_F>.

The sequence cannot be interrupted with a single block.

The sequence is repeated until the total infeed depth P3 <S_K> has been reached. The last stroke is then distributed unevenly.



Example

Oscillation with:

- 20 mm infeed depth at the start
- 10 mm infeed depth at the end
- Total infeed depth 100 mm
- 100 mm stroke

- Feedrate 1000 mm/min
- 1 second sparking-out time
- Standard geometry axes

```
Program code
N10 T1 D1
N20 CYCLE4078(20,10,100,100,1000,1)
N30 M30
```

19.1.51 CYCLE4079 - surface grinding with intermittent infeed

Syntax

```
CYCLE4079(<S_I>, <S_J>, <S_K>, <S_A>, <S_R>, <S_F>, <S_P>, <S_A1>,
<S_A2>)
```

Parameters

No.	Parameter	Data type	Meaning
1	<S_I>	REAL	Infeed depth at the start
2	<S_J>	REAL	Infeed depth at the end
3	<S_K>	REAL	Total infeed depth
4	<S_A>	REAL	Grinding width
5	<S_R>	REAL	Feedrate for infeed
6	<S_F>	REAL	Feedrate for transverse infeed
7	<S_P>	REAL	Sparking-out time
8	<S_A1>	AXIS	Infeed axis (optional)
9	<S_A2>	AXIS	Oscillating axis (optional)

Function

The cycle is used for machining with a total infeed depth P3 <S_K> in infeed steps. The infeed depths at the start P1 <S_I> and at the end P2 <S_J> can be different. There is a tangential motion between the infeeds.

The positional data P1 to P4 can be negative or positive.

The specification of the infeed axis P8 <S_A1> and/or oscillating axis P9 <S_A2> are optional. If one or both parameters are not specified, the cycle uses the first two geometry axes of the channel.

If the sum of the infeed depth at the start P1 <S_I> and at the end P2 <S_J> is 0 or the total infeed depth P3 <S_K> is 0, only a sparking-out stroke is performed.

Sequence

1. Start of the cycle at the current position of the oscillating axis.
2. Traversing of the infeed axis to the infeed depth at the start P1 **<**S_I**>** with the feedrate for infeed P5 <S_R>.
3. Sparking out with the sparking-out time P7 **<**S_P**>**.
4. Traversing of the oscillating axis with the grinding width P4 **<**S_A**>** as travel path and the feedrate for transverse infeed P6 <S_F>.
5. Traversing of the infeed axis to the infeed depth at the end P2 **<**S_J**>** with the feedrate for infeed P5 <S_R>.
6. Sparking out with the sparking-out time P7 **<**S_P**>**.
7. Traversing of the oscillating axis with the grinding width P4 **<**S_A**>** as travel path to the starting point and the feedrate for transverse infeed P6 <S_F>.

The sequence cannot be interrupted with a single block.

The sequence is repeated until the total infeed depth P3 **<**S_K**>** has been reached. The last stroke is then distributed unevenly.

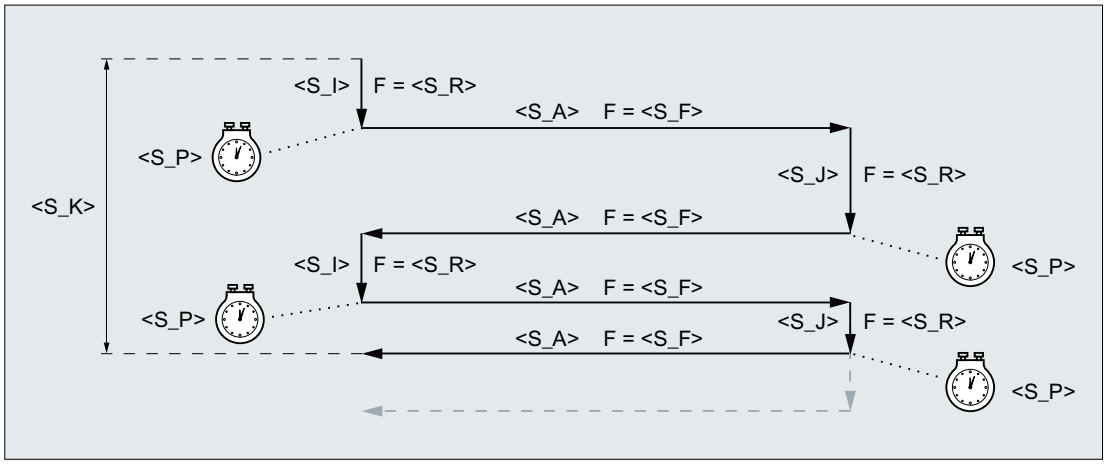


Figure 19-9 Total infeed depth reached with infeed at the second reversal point

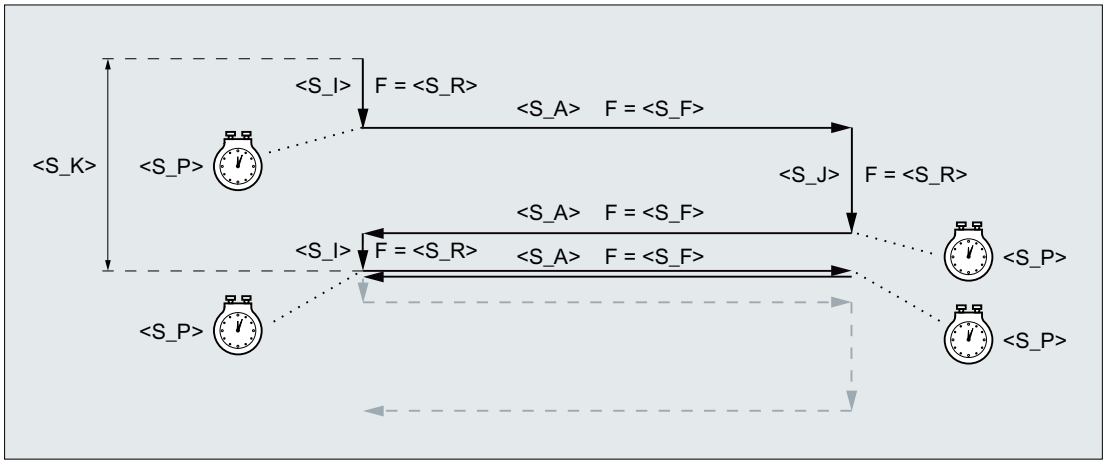


Figure 19-10 Total infeed depth reached with infeed at the first reversal point

Example

Oscillation with:

- 0.02 mm infeed depth at the start
- 0.01mm infeed depth at the end
- Total infeed depth 1 mm
- 100 mm stroke
- Infeed feedrate 1 mm/min
- Transverse feedrate 1000 mm/min
- 1 second sparking-out time
- Standard geometry axes

Program code

```
N10 T1 D1
N20 CYCLE4079(0.02,0.01,1,100,1,1000,1)
N30 M30
```

19.1.52 Supplementary conditions

19.1.52.1 Technology scaling in cycle screen forms

When the technology scaling is active, the simplified input can be selected for various cycle screen forms, in which only the most important cycle parameters are displayed

The simplified input can be selected for the following cycle screen forms:

Technology	Cycle screen form
Drilling	Deep-hole drilling
	Tapping
Milling	Rectangular pocket
	Contour milling: Pocket
Turning	Thread turning: Longitudinal
	Contour turning: Stock removal
	Contour turning: Grooving
	Contour turning: Groove turning

In the user interface of the relevant cycle screen forms, the options "Input: **Simple**" and "Input: **Complete**" are available.

Cycle parameters that are not displayed

The cycle parameters that are not displayed in the simplified input are pre-assigned fixed, technologically useful, but not variable values. Or the cycle parameters are assigned parameterizable values via the channel-specific cycle setting data. See the paragraph below "Commissioning" > "Channel-specific cycle setting data"

Switchover from "Input: Complete" > "Input: Simple"

If a cycle screen form is filled in with the setting "Input complete" and then switched to "Input simple", the default or setting data values are used for the parameters no longer displayed when generating the cycle call.

Commissioning

Channel-specific configuration machine data

The technology scaling in cycle screen forms can be activated with the machine data:

MD52210 $MCS_FUNCTION_MASK_DISP, Bit 9 = 1 (select display "Input simple")

Channel-specific cycle setting data

If the simplified input in cycle screen forms is active, the values for certain cycle parameters can be specified via the following setting data:

Number	Identifier	Meaning
SD55300	$SCS_EASY_SAFETY_CLEARANCE	Safety clearance
SD55301	$SCS_EASY_DWELL_TIME	Dwell time
SD55305	$SCS_EASY_DRILL_DEEP_FD1	Deep-hole drilling: Percentage: 1st feedrate
SD55306	$SCS_EASY_DRILL_DEEP_DF	Deep-hole drilling: Percentage: Infeed
SD55307	$SCS_EASY_DRILL_DEEP_V1	Deep-hole drilling: Minimum depth infeed
SD55308	$SCS_EASY_DRILL_DEEP_V2	Deep-hole drilling: Retraction distance
SD55309	$SCS_EASY_THREAD_RETURN_DIST	Thread turning: Return distance

19.2 Measuring cycles

Measuring cycles are special subprograms provided by Siemens to solve certain measuring tasks. As with cycles in general, measuring cycles can also be adapted to specific problems via parameter settings.

Measuring cycles are available in the following areas and technologies:

- Tool measurements, turning/milling
- Workpiece measurements, turning/milling

References

For a detailed description of the measuring cycles, refer to:

Measuring Cycles Programming Manual

Tables

20

20.1 Operations

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
:	O	NC main block number, jump label termination, concatenation operator		+		PGAsl
*	O	Operator for multiplication		+		PGAsl
+	O	Operator for addition		+		PGAsl
-	O	Operator for subtraction		+		PGAsl
<	O	Comparison operator, less than		+		PGAsl
<<	O	Concatenation operator for strings		+		PGAsl
<=	O	Comparison operator, less than or equal to		+		PGAsl
=	O	Assignment operator		+		PGAsl
>=	O	Comparison operator, greater than or equal to		+		PGAsl
/	O	Operator for division		+		PGAsl
/0 … … /7		block is skipped (1st skip level) ... ... block is skipped (8th skip level)		+		PGsl
O	O	Axis name	m/s	+		PGAsl
A2	O	Tool orientation: RPY or Euler angle	s	+		PGAsl
A3	O	Tool orientation: Direction/surface normal vector component	s	+		PGAsl
A4	O	Tool orientation: Surface normal vector for beginning of block	s	+		PGAsl
A5	O	Tool orientation: Surface normal vector for end of block	s	+		PGAsl
ABS	F	Absolute value (amount)		+	+	PGAsl
AC	K	Absolute dimensions of coordinates/positions	s	+		PGsl
ACC	K	Effect of current axial acceleration	m	+	+	PGsl
ACCLIMA	K	Effect of current maximum axial acceleration	m	+	+	PGAsl
ACN	K	Absolute dimensions for rotary axes, approach position in negative direction	s	+		PGsl
ACOS	F	Arc cosine (trigon. function)		+	+	PGAsl
ACP	K	Absolute dimensions for rotary axes, approach position in positive direction	s	+		PGsl
ACTBLOCNO	P	Output of current block number of an alarm block, even if "current block display suppressed" (DISPLOF) is active!		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
ADDFRAME	F	Inclusion and possible activation of a measured frame		+	-	PGAsl, FB1sl (K2)
ADIS	O	Rounding clearance for path functions G1, G2, G3, ...	m	+		PGsl
ADISPOS	O	Rounding clearance for rapid traverse G0	m	+		PGsl
ADISPOSA	P	Size of the tolerance window for IPOBRKA	m	+	+	PGAsl
ALF	O	LIFTFAST angle	m	+		PGAsl
AMIRROR	G	Programmable mirroring	s	+		PGsl
AND	K	Logical AND		+		PGAsl
ANG	O	Contour angle	s	+		PGsl
AP	O	Polar angle	m/s	+		PGsl
APR	K	Read/show access protection		+		PGAsl
APRB	K	Read access right, OPI		+		PGAsl
APRP	K	Read access right, part program		+		PGAsl
APW	K	Write access protection		+		PGAsl
APWB	K	Write access right, OPI		+		PGAsl
APWP	K	Write access right, part program		+		PGAsl
APX	K	Definition of the access right for executing the specified language element		+		PGAsl
AR	O	Opening angle	m/s	+		PGsl
AROT	G	Programmable rotation	s	+		PGsl
AROTS	G	Programmable frame rotations with solid angles	s	+		PGsl
AS	K	Macro definition		+		PGAsl
ASCALE	G	Programmable scaling	s	+		PGsl
ASIN	F	Arithmetic function, arc sine		+	+	PGAsl
ASPLINE	G	Akima spline	m	+		PGAsl
ATAN2	F	Arc tangent 2		+	+	PGAsl
ATOL	K	Axis-specific tolerance for compressor functions, orientation smoothing and smoothing types		+		PGAsl
ATRANS	G	Additive programmable translation	s	+		PGsl
AUXFUDEL	P	Delete auxiliary function channel-specifically from the global list		+	-	FB1sl (H2)
AUXFUDELG	P	Delete all auxiliary functions of an auxiliary function group channel-specifically from the global list		+	-	FB1sl (H2)
AUXFUMSEQ	P	Determine output sequence of M auxiliary functions		+	-	FB1sl (H2)
AUXFUSYNC	P	Generate a complete part program block for the channel-specific SERUPRO end ASUB as string from the global list of auxiliary functions		+	-	FB1sl (H2)
AX	K	Variable axis identifier	m/s	+		PGAsl
AXCTSWE	P	Rotate axis container		+	-	PGAsl
AXCTSWEC	P	Canceling enable for axis container rotation		+	+	PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
AXCTSWED	P	Rotating axis container (command variant for commissioning!)		+	-	PGAsl
AXIS	K	Axis identifier, axis address		+		PGAsl
AXNAME	F	Converts input string into axis identifier		+	-	PGAsl
AXSTRING	F	Converts string spindle number		+	-	PGAsl
AXTOCHAN	P	Request axis for a specific channel. Possible from NC program and synchronized action.		+	+	PGAsl
AXTOSPI	F	Converts axis identifier into a spindle index		+	-	PGAsl
B	O	Axis name	m/s	+		PGAsl
B2	O	Tool orientation: RPY or Euler angle	s	+		PGAsl
B3	O	Tool orientation: Direction/surface normal vector component	s	+		PGAsl
B4	O	Tool orientation: Surface normal vector for beginning of block	s	+		PGAsl
B5	O	Tool orientation: Surface normal vector for end of block	s	+		PGAsl
B_AND	O	Bit-by-bit AND		+		PGAsl
B_OR	O	Bit-by-bit OR		+		PGAsl
B_NOT	O	Bit-by-bit negation		+		PGAsl
B_XOR	O	Bit-by-bit exclusive OR		+		PGAsl
BAUTO	G	Definition of the first spline section by means of the next 3 points	m	+		PGAsl
BLOCK	K	Together with the keyword TO defines the program part to be processed in an indirect subprogram call		+		PGAsl
BLSYNC	K	Processing of interrupt routine is only to start with the next block change		+		PGAsl
BNAT [6]	G	Natural transition to first spline block	m	+		PGAsl
BOOL	K	Data type: Boolean value TRUE/FALSE or 1/0		+		PGAsl
BOUND	F	Tests whether the value falls within the defined value range. If the values are equal, the test value is returned.		+	+	PGAsl
BRISK [6]	G	Fast non-smoothed path acceleration	m	+		PGAsl
BRISKA	P	Switch on brisk path acceleration for the programmed axes		+	-	PGAsl
BSPLINE	G	B spline	m	+		PGAsl
BTAN	G	Tangential transition to first spline block	m	+		PGAsl
C	O	Axis name	m/s	+		PGAsl
C2	O	Tool orientation: RPY or Euler angle	s	+		PGAsl
C3	O	Tool orientation: Direction/surface normal vector component	s	+		PGAsl
C4	O	Tool orientation: Surface normal vector for beginning of block	s	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
C5	O	Tool orientation: Surface normal vector for end of block	s	+		PGAsl
CAC	K	Absolute position approach		+		PGAsl
CACN	K	Absolute approach of the value listed in the table in negative direction		+		PGAsl
CACP	K	Absolute approach of the value listed in the table in positive direction		+		PGAsl
CALCDAT	F	Calculates radius and center point of circle from 3 or 4 points		+	-	PGAsl
CALCPOSI	F	Checking for protection area violation, working area limitation and software limits		+	-	PGAsl
CALL	K	Indirect subprogram call		+		PGAsl
CALLPATH	P	Programmable search path for subprogram calls		+	-	PGAsl
CANCEL	P	Cancel modal synchronized action		+	-	FBSYsl
CASE	K	Conditional program branch		+		PGAsl
CDC	K	Direct approach of a position		+		PGAsl
CDOF [6]	G	Collision detection OFF	m	+		PGsl
CDOF2	G	Collision detection OFF, for 3D circumferential milling	m	+		PGsl
CDON	G	Collision detection ON	m	+		PGsl
CFC [6]	G	Constant feedrate on contour	m	+		PGsl
CFIN	G	Constant feedrate for internal radius only, not for external radius	m	+		PGsl
CFINE	F	Assignment of fine offset to a FRAME variable		+	-	PGAsl
CFTCP	G	Constant feedrate in tool center point (center point path)	m	+		PGsl
CHAN	K	Specify validity range for data		+		PGAsl
CHANDATA	P	Set channel number for channel data access		+	-	PGAsl
CHAR	K	Data type: ASCII character		+		PGAsl
CHF	O	Chamfer; value = length of chamfer	s	+		PGsl
CHKDM	F	Uniqueness check within a magazine		+	-	FBWsl
CHKDNO	F	Check for unique D numbers		+	-	PGAsl
CHR	O	Chamfer; value = length of chamfer in direction of movement		+		PGsl
CIC	K	Approach position by increments		+		PGAsl
CIP	G	Circular interpolation through intermediate point	m	+		PGsl
CLEARM	P	Reset one/several markers for channel coordination		+	+	PGAsl
CLRINT	P	Deselect interrupt		+	-	PGAsl
CMIRROR	F	Mirror on a coordinate axis		+	-	PGAsl
COARSEA	K	Motion end when "Exact stop coarse" reached	m	+		PGAsl
COLLPAIR	F	Check whether part of a collision pair		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
COMPCAD	G	Compressor ON: Optimum surface quality for CAD programs	m	+		PGAsl
COMPCURV	G	Compressor ON: Polynomials with constant curvature	m	+		PGAsl
COMPLETE		Control instruction for reading and writing data		+		PGAsl
COMPOF [6]	G	Compressor OFF	m	+		PGAsl
COMPON	G	Compressor ON		+		PGAsl
CONTDCON	P	Tabular contour decoding ON		+	-	PGAsl
CONTPRON	P	Activate reference preprocessing		+	-	PGAsl
CORROF	P	All active overlaid movements are deselected.		+	-	PGsl
COS	F	Cosine (trigon. function)		+	+	PGAsl
COUPDEF	P	Definition ELG group/synchronous spindle group		+	-	PGAsl
COUPDEL	P	Delete ELG group		+	-	PGAsl
COUPOF	P	ELG group / synchronous spindle pair OFF		+	-	PGAsl
COUPOFS	P	Deactivate ELG group/synchronous spindle pair with stop of following spindle		+	-	PGAsl
COUPON	P	ELG group / synchronous spindle pair ON		+	-	PGAsl
COUPONC	P	Transfer activation of ELG group/synchronous spindle pair with previous programming		+	-	PGAsl
COUPRES	P	Reset ELG group		+	-	PGAsl
CP [6]	G	Path motion	m	+		PGAsl
CPBC	K	Generic coupling: Block change criterion		+	+	FB3sl (M3)
CPDEF	K	Generic coupling: Creating a coupling module		+	+	FB3sl (M3)
CPDEL	K	Generic coupling: Deletion of a coupling module		+	+	FB3sl (M3)
CPFMOF	K	Generic coupling: Behavior of the following axis at complete switch-off		+	+	FB3sl (M3)
CPFMON	K	Generic coupling: Behavior of the following axis when switching on		+	+	FB3sl (M3)
CPFMSON	K	Generic coupling: Synchronization mode		+	+	FB3sl (M3)
CPFPOS	K	Generic coupling: Synchronized position of the following axis		+	+	FB3sl (M3)
CPFRS	K	Generic coupling: Coordinate reference system		+	+	FB3sl (M3)
CPLA	K	Generic coupling: Definition of a leading axis		+	-	FB3sl (M3)
CPLCTID	K	Generic coupling: Number of the curve table		+	+	FB3sl (M3)
CPLDEF	K	Generic coupling: Definition of a leading axis and creation of a coupling module		+	+	FB3sl (M3)
CPLDEL	K	Generic coupling: Deleting a leading axis of a coupling module		+	+	FB3sl (M3)
CPLDEN	K	Generic coupling: Denominator of the coupling factor		+	+	FB3sl (M3)
CPLINSC	K	Generic coupling: Scaling factor of the input value of a leading axis		+	+	FB3sl (M3)

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
CPLINTR	K	Generic coupling: Offset value of the input value of a leading axis		+	+	FB3sl (M3)
CPLNUM	K	Generic coupling: Numerator of the coupling factor		+	+	FB3sl (M3)
CPLOF	K	Generic coupling: Switching off a leading axis of a coupling module		+	+	FB3sl (M3)
CPLON	K	Generic coupling: Switching on a leading axis of a coupling module		+	+	FB3sl (M3)
CPLOUTSC	K	Generic coupling: Scaling factor for the output value of a coupling		+	+	FB3sl (M3)
CPLOUTTR	K	Generic coupling: Offset value for the output value of a coupling		+	+	FB3sl (M3)
CPLPOS	K	Generic coupling: Synchronized position of the leading axis		+	+	FB3sl (M3)
CPLSETVAL	K	Generic coupling: Coupling reference		+	+	FB3sl (M3)
CPMALARM	K	Generic coupling: Suppression of special coupling-related alarm outputs		+	+	FB3sl (M3)
CPMBRAKE	K	Generic coupling: Response of the following axis to certain stop signals and stop commands		+	-	FB3sl (M3)
CPMPRT	K	Generic coupling: Coupling response at part program start under search run via program test		+	+	FB3sl (M3)
CPMRESET	K	Generic coupling: Coupling behavior for RESET		+	+	FB3sl (M3)
CPMSTART	K	Generic coupling: Coupling behavior at part program start		+	+	FB3sl (M3)
CPMVDI	K	Generic coupling: Response of the following axis to certain NC/PLC interface signals		+	+	FB3sl (M3)
CPOF	K	Generic coupling: Switching off a coupling module		+	+	FB3sl (M3)
CPON	K	Generic coupling: Switching on a coupling module		+	+	FB3sl (M3)
CPRECOF [6]	G	Programmable contour accuracy OFF	m	+		PGAsl
CPRECON	G	Programmable contour accuracy ON	m	+		PGAsl
CPRES	K	Generic coupling: Activates the configured data of the synchronous spindle coupling		+	-	FB3sl (M3)
CPROT	P	Channel-specific protection area ON/OFF		+	-	PGAsl
CPROTDEF	P	Definition of a channel-specific protection area		+	-	PGAsl
CPSETTYPE	K	Generic coupling: Coupling type		+	+	FB3sl (M3)
CPSYNCOP	K	Generic coupling: Threshold value of position synchronism "Coarse"		+	+	FB3sl (M3)
CPSYNCOP2	K	Generic coupling: Threshold value of position synchronism "Coarse" 2		+	+	FB3sl (M3)
CPSYNCOV	K	Generic coupling: Threshold value of velocity synchronism "Coarse"		+	+	FB3sl (M3)
CPSYNFIP	K	Generic coupling: Threshold value of position synchronism "Fine"		+	+	FB3sl (M3)
CPSYNFIP2	K	Generic coupling: Threshold value of position synchronism "Fine" 2		+	+	FB3sl (M3)

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
CPSYNFIV	K	Generic coupling: Threshold value of velocity synchronism "Fine"		+	+	FB3sl (M3)
CR	O	Circle radius	s	+		PGsl
CROT	F	Rotation of the current coordinate system		+	-	PGAsl
CROTS	F	Programmable frame rotations with solid angles (rotation in the specified axes)	s	+	-	PGsl
CRPL	F	Frame rotation in any plane		+	-	FB1sl (K2)
CSCALE	F	Scale factor for multiple axes		+	-	PGAsl
CSPLINE	F	Cubic spline	m	+		PGAsl
CT	G	Circle with tangential transition	m	+		PGsl
CTAB	F	Define following axis position according to leading axis position from curve table		+	+	PGAsl
CTABDEF	P	Table definition ON		+	-	PGAsl
CTABDEL	P	Clear curve table		+	-	PGAsl
CTABEND	P	Table definition OFF		+	-	PGAsl
CTABEXISTS	F	Checks the curve table with number n		+	+	PGAsl
CTABFNO	F	Number of curve tables still possible in the memory		+	+	PGAsl
CTABFPOL	F	Number of polynomials still possible in the memory		+	+	PGAsl
CTABFSEG	F	Number of curve segments still possible in the memory		+	+	PGAsl
CTABID	F	Returns table number of the nth curve table		+	+	PGAsl
CTABINV	F	Define leading axis position according to following axis position from curve table		+	+	PGAsl
CTABISLOCK	F	Returns the lock state of the curve table with number n		+	+	PGAsl
CTABLOCK	P	Delete and overwrite, lock		+	+	PGAsl
CTABMEMTYP	F	Returns the memory in which curve table number n is created.		+	+	PGAsl
CTABMPOL	F	Max. number of polynomials still possible in the memory		+	+	PGAsl
CTABMSEG	F	Max. number of curve segments still possible in the memory		+	+	PGAsl
CTABNO	F	Number of defined curve tables in SRAM or DRAM		+	+	FB3sl (M3)
CTABNOMEM	F	Number of defined curve tables in SRAM or DRAM		+	+	PGAsl
CTABPERIOD	F	Returns the table periodicity of curve table number n		+	+	PGAsl
CTABPOL	F	Number of polynomials already used in the memory		+	+	PGAsl
CTABPOLID	F	Number of the curve polynomials used by the curve table with number n		+	+	PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
CTABSEG	F	Number of curve segments already used in the memory		+	+	PGAsl
CTABSEGID	F	Number of the curve segments used by the curve table with number n		+	+	PGAsl
CTABSEV	F	Returns the final value of the following axis of a segment of the curve table		+	+	PGAsl
CTABSSV	F	Returns the initial value of the following axis of a segment of the curve table		+	+	PGAsl
CTABTEP	F	Returns the value of the leading axis at curve table end		+	+	PGAsl
CTABTEV	F	Returns the value of the the following axis at curve table end		+	+	PGAsl
CTABTMAX	F	Returns the maximum value of the following axis of the curve table		+	+	PGAsl
CTABTMIN	F	Returns the minimum value of the following axis of the curve table		+	+	PGAsl
CTABTSP	F	Returns the value of the leading axis at curve table start		+	+	PGAsl
CTABTSV	F	Returns the value of the following axis at curve table start		+	+	PGAsl
CTABUNLOCK	P	Revoke delete and overwrite lock		+	+	PGAsl
CTOL	K	Contour tolerance for compressor functions, orientation smoothing and smoothing types		+		PGAsl
CTRANS	F	Zero offset for multiple axes		+	-	PGAsl
CUT2D [6]	G	2D tool offset	m	+		PGsl
CUT2DF	G	2D tool offset The tool offset is applied relative to the current frame (inclined plane).	m	+		PGsl
CUT3DC	G	3D tool offset circumferential milling	m	+		PGAsl
CUT3DCC	G	3D tool offset circumferential milling with limitation surfaces	m	+		PGAsl
CUT3DCCD	G	3D tool offset circumferential milling with limitation surfaces with differential tool	m	+		PGAsl
CUT3DF	G	3D tool offset face milling	m	+		PGAsl
CUT3DFF	G	3D tool offset face milling with constant tool orientation dependent on active frame	m	+		PGAsl
CUT3DFS	G	3D tool offset face milling with constant tool orientation independent of active frame	m	+		PGAsl
CUTCONOF [6]	G	Constant radius compensation OFF	m	+		PGsl
CUTCONON	G	Constant radius compensation ON	m	+		PGsl
CUTMOD	K	Activate "Modification of the offset data for rotatable tools"		+		PGAsl
CYCLE60	C	Engraving cycle		+		PGAsl
CYCLE61	C	Face milling		+		PGAsl
CYCLE62	C	Contour call		+		PGAsl
CYCLE63	C	Contour pocket milling		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
CYCLE64	C	Contour pocket predrilling		+		PGAsl
CYCLE70	C	Thread milling		+		PGAsl
CYCLE72	C	Path milling		+		PGAsl
CYCLE76	C	Milling the rectangular spigot		+		PGAsl
CYCLE77	C	Circular spigot milling		+		PGAsl
CYCLE78	C	Mill cutting thread		+		PGAsl
CYCLE79	C	Multiple edge		+		PGAsl
CYCLE81	C	Drilling, centering		+		PGAsl
CYCLE82	C	Drilling, counterboring		+		PGAsl
CYCLE83	C	Deep-hole drilling		+		PGAsl
CYCLE84	C	Tapping without compensating chuck		+		PGAsl
CYCLE85	C	Reaming		+		PGAsl
CYCLE86	C	Boring		+		PGAsl
CYCLE92	C	Parting		+		PGAsl
CYCLE95	C	Stock removal along the contour		+		PGAsl
CYCLE98	C	Thread chain		+		PGAsl
CYCLE99	C	Thread cutting		+		PGAsl
CYCLE495	C	Form-truing		+		PGAsl
CYCLE751	C	Open / perform / close an optimization session		+		FB3sl (T4)
CYCLE752	C	Add axis to an optimization session		+		FB3sl (T4)
CYCLE753	C	Select optimization mode		+		FB3sl (T4)
CYCLE754	C	Add / remove language block		+		FB3sl (T4)
CYCLE755	C	Backing up/restoring data block		+		FB3sl (T4)
CYCLE756	C	Activate optimization results		+		FB3sl (T4)
CYCLE757	C	Store optimization data		+		FB3sl (T4)
CYCLE758	C	Changing the parameter value		+		FB3sl (T4)
CYCLE759	C	Read parameter value		+		FB3sl (T4)
CYCLE800	C	Swiveling		+		PGAsl
CYCLE801	C	Grid or frame		+		PGAsl
CYCLE802	C	Arbitrary positions		+		PGAsl
CYCLE830	C	Deep-hole drilling 2		+		PGAsl
CYCLE832	C	High-Speed Settings		+		PGAsl
CYCLE840	C	Tapping with compensating chuck		+		PGAsl
CYCLE899	C	Open slot milling		+		PGAsl
CYCLE930	C	Groove		+		PGAsl
CYCLE940	C	Undercut forms		+		PGAsl
CYCLE951	C	Stock removal		+		PGAsl
CYCLE952	C	Contour grooving		+		PGAsl
CYCLE4071	C	Longitudinal grinding with infeed at the reversal point		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
CYCLE4072	C	Longitudinal grinding with infeed at the reversal point and cancel signal		+		PGAsl
CYCLE4073	C	Longitudinal grinding with continuous infeed		+		PGAsl
CYCLE4074	C	Longitudinal grinding with continuous infeed and cancel signal		+		PGAsl
CYCLE4075	C	Surface grinding with infeed at the reversal point		+		PGAsl
CYCLE4077	C	Surface grinding with infeed at the reversal point and cancel signal		+		PGAsl
CYCLE4078	C	Surface grinding with continuous infeed		+		PGAsl
CYCLE4079	C	Surface grinding with intermittent infeed		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
D	O	Tool offset number		+		PGsl
D0	O	With D0, offsets for the tool are ineffective		+		PGsl
DAC	K	Absolute non-modal axis-specific diameter programming	s	+		PGsl
DC	K	Absolute dimensions for rotary axes, approach position directly	s	+		PGsl
DCI	K	Assign data class I (= Individual) (only SINUMERIK 828D)		+		PGAsl
DCM	K	Assign data class M (= Manufacturer) (only SINUMERIK 828D)		+		PGAsl
DCU	K	Assign data class U (= User) (only SINUMERIK 828D)		+		PGAsl
DEF	K	Variable definition		+		PGAsl
DEFAULT	K	Branch in CASE branch		+		PGAsl
DEFINE	K	Keyword for macro definitions		+		PGAsl
DELAYFSTOF	P	Define the end of a stop delay section	m	+	-	PGAsl
DELAYFSTON	P	Define the start of a stop delay section	m	+	-	PGAsl
DELDL	F	Delete additive offsets		+	-	PGAsl
DELDTG	P	Delete distance-to-go		-	+	FBSYsl
DELETE	P	Delete the specified file. The file name can be specified with path and file identifier.		+	-	PGAsl
DELMLOWNER	F	Delete owner magazine location of the tool		+	-	FBWsl
DELMLRES	F	Delete magazine location reservation		+	-	FBWsl
DELMT	P	Delete multitool		+	-	FBWsl
DELOBJ	F	Deletion of elements from kinematic chains, protection areas, protection area elements, collision pairs and transformation data		+		PGAsl
DELT	P	Delete Tool		+	-	FBWsl
DELTC	P	Delete toolholder data record		+	-	FBWsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
DELTOOLENV	F	Delete data records describing tool environments		+	-	FB1sl (W1)
DIACYCOFA	K	Axis-specific modal diameter programming: OFF in cycles	m	+		FB1sl (P1)
DIAM90	G	Diameter programming for G90, radius programming for G91	m	+		PGAsl
DIAM90A	K	Axis-specific modal diameter programming for G90 and AC, radius programming for G91 and IC	m	+		PGsl
DIAMCHAN	K	Transfer of all axes from MD axis functions to diameter programming channel status		+		PGsl
DIAMCHANA	K	Transfer of the diameter programming channel status		+		PGsl
DIAMCYCOF	G	Channel-specific diameter programming: OFF in cycles	m	+		FB1sl (P1)
DIAMOF [6]	G	Diameter programming: OFF Normal position, see machine manufacturer	m	+		PGsl
DIAMOFA	K	Axis-specific modal diameter programming: OFF Normal position, see machine manufacturer	m	+		PGsl
DIAMON	G	Diameter programming: ON	m	+		PGsl
DIAMONA	K	Axis-specific modal diameter programming: ON Activation, see machine manufacturer	m	+		PGsl
DIC	K	Relative non-modal axis-specific diameter programming	s	+		PGsl
DILF	O	Retraction path (length)	m	+		PGsl
DISABLE	P	Interrupt OFF		+	-	PGAsl
DISC	O	Transition circle overshoot tool radius compensation	m	+		PGsl
DISCL	O	Clearance between the end point of the fast infeed motion and the machining plane		+		PGsl
DISPLOF	PA	Suppress current block display		+		PGAsl
DISPLON	PA	Revoke suppression of the current block display		+		PGAsl
DISPR	O	Path differential for repositioning	s	+		PGAsl
DISR	O	Distance for repositioning	s	+		PGAsl
DISRP	O	Distance between the retraction plane and the machining plane during smooth approach and retraction		+		PGsl
DITE	O	Thread run-out path	m	+		PGsl
DITS	O	Thread run-in path	m	+		PGsl
DIV	K	Integer division		+		PGAsl
DL	O	Select location-dependent additive tool offset (DL, total set-up offset)	m	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
DO	O	Keyword for synchronized action, triggers action when condition is fulfilled		-	+	FBSYsl
DRFOF	P	Deactivation of handwheel offsets (DRF)	m	+	-	PGsl
DRIVE	G	Velocity-dependent path acceleration	m	+		PGAsl
DRIVEA	P	Activate knee-shaped acceleration characteristic for the programmed axes		+	-	PGAsl
DYNFINISH	G	Dynamic response for smooth finishing	m	+		PGAsl
DYNNORM [6]	G	Standard dynamic response	m	+		PGAsl
DYNPOS	G	Dynamic response for positioning mode, tapping	m	+		PGAsl
DYNROUGH	G	Dynamic response for roughing	m	+		PGAsl
DYNSEMIFIN	G	Dynamic response for finishing	m	+		PGAsl
DZERO	P	Marks all D numbers of the TO unit as invalid		+	-	PGAsl
EAUTO	G	Definition of the last spline section by means of the last 3 points	m	+		PGAsl
EGDEF	P	Definition of an electronic gear		+	-	PGAsl
EGDEL	P	Delete coupling definition for the following axis		+	-	PGAsl
EGOFC	P	Turn off electronic gear continuously		+	-	PGAsl
EGOFS	P	Turn off electronic gear selectively		+	-	PGAsl
EGON	P	Turn on electronic gear		+	-	PGAsl
EGONSYN	P	Turn on electronic gear		+	-	PGAsl
EGONSYNE	P	Turn on electronic gear, with specification of approach mode		+	-	PGAsl
ELSE	K	Program branch, if IF condition not fulfilled		+		PGAsl
ENABLE	P	Interrupt ON		+	-	PGAsl
ENAT [6]	G	Natural transition to next traversing block	m	+		PGAsl
ENDFOR	K	End line of FOR counter loop		+		PGAsl
ENDIF	K	End line of IF branch		+		PGAsl
ENDLABEL	K	End label for part program repetitions with REPEAT		+		PGAsl, FB1sl (K1)
ENDLOOP	K	End line of endless program loop LOOP		+		PGAsl
ENDPROC	K	End line of program with start line PROC		+		
ENDWHILE	K	End line of WHILE loop		+		PGAsl
ESRR	P	Parameterizing drive-autonomous ESR retraction in the drive		+		PGAsl
ESRS	P	Parameterizing drive-autonomous ESR shutdown in the drive		+		PGAsl
ETAN	G	Tangential transition to next traversing block at spline begin	m	+		PGAsl
EVERY	K	Execute synchronized action on transition of condition from FALSE to TRUE		-	+	FBSYsl
EX	K	Keyword for value assignment in exponential notation		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
EXECSTRING	P	Transfer of a string variable with the executing part program line		+	-	PGAsl
EXECTAB	P	Execute an element from a motion table		+	-	PGAsl
EXECUTE	P	Program execution ON		+	-	PGAsl
EXP	F	Exponential function ex		+	+	PGAsl
EXTCALL	O	Execute external subprogram		+	+	PGAsl
EXTCLOSE	P	Closing external device / file that was opened for writing		+	-	PGAsl
EXTERN	K	Declaration of a subprogram with parameter transfer		+		PGAsl
EXTOPEN	P	Opening external device / file for the channel for writing		+	-	PGAsl
F	O	Feedrate value (in conjunction with G4 the dwell time is also programmed with F)		+	+	PGsl
FA	K	Axial feedrate	m	+	+	PGsl
FAD	O	Infeed rate for soft approach and retraction		+		PGsl
FALSE	K	Logical constant: Incorrect		+	+	PGAsl
FB	O	Non-modal feedrate		+		PGsl
FCTDEF	P	Define polynomial function		+	-	PGAsl
FCUB	G	Feedrate variable according to cubic spline	m	+		PGAsl
FD	O	Path feedrate for handwheel override	s	+		PGsl
FDA	K	Axis feedrate for handwheel override	s	+		PGsl
FENDNORM [6]	G	Corner deceleration OFF	m	+		PGAsl
FFWOF [6]	G	Feedforward control OFF	m	+		PGAsl
FFWON	G	Feedforward control ON	m	+		PGAsl
FGREF	K	Reference radius for rotary axes or path reference factors for orientation axes (vector interpolation)	m	+		PGsl
FGROUP	P	Definition of axis/axes with path feedrate		+	-	PGsl
FI	K	Parameter for access to frame data: Fine offset		+		PGAsl
FIFOCTRL	G	Control of preprocessing buffer	m	+		PGAsl
FILEDATE	P	Returns date of most recent write access to file		+	-	PGAsl
FILEINFO	P	Returns summary information listing FILEDATE, FILESIZE, FILESTAT, and FILETIME		+	-	PGAsl
FILESIZE	P	Returns current file size		+	-	PGAsl
FILESTAT	P	Returns file status of rights for read, write, execute, display, delete (rwxsd)		+	-	PGAsl
FILETIME	P	Returns time of most recent write access to file		+	-	PGAsl
FINEA	K	End of motion when "Exact stop fine" reached	m	+		PGAsl
FL	K	Limit velocity for synchronized axis	m	+		PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
FLIN	G	Feed linear variable	m	+		PGAsl
FMA	K	Multiple feedrates axial	m	+		PGsl
FNORM [6]	G	Feedrate normal to DIN 66025	m	+		PGAsl
FOC	K	Non-modal torque/force limitation	s	-	+	FBSYsl
FOCOF	K	Switch off modal torque/force limitation	m	-	+	FBSYsl
FOCON	K	Switch on modal torque/force limitation	m	-	+	FBSYsl
FOR	K	Counter loop with fixed number of passes		+		PGAsl
FP	O	Fixed point: Number of fixed point to be approached	s	+		PGsl
FPO	K	Feedrate characteristic programmed via a polynomial		+		PGAsl
FPR	P	Rotary axis identifier		+	-	PGsl
FPRAOF	P	Deactivate revolutional feedrate		+	-	PGsl
FPRAON	P	Activate revolutional feedrate		+	-	PGsl
FRAME	K	Data type for the definition of coordinate systems		+		PGAsl
FRC	O	Feedrate for radius and chamfer	s	+		PGsl
FRCM	O	Feedrate for radius and chamfer, modal	m	+		PGsl
FROM	K	The action is executed if the condition is fulfilled once and as long as the synchronized action is active		-	+	FBSYsl
FTOC	P	Change fine tool offset		-	+	FBSYsl
FTOCOF [6]	G	Online fine tool offset OFF	m	+		PGAsl
FTOCON	G	Online fine tool offset ON	m	+		PGAsl
FXS	K	Travel to fixed stop ON	m	+	+	PGsl
FXST	K	Torque limit for travel to fixed stop	m	+	+	PGsl
FXSW	K	Monitoring window for travel to fixed stop		+	+	PGsl
FZ	K	Tooth feedrate	m	+		PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
G0	G	Linear interpolation with rapid traverse (rapid traverse motion)	m	+		PGsl
G1 [6]	G	Linear interpolation with feedrate (linear interpolation)	m	+		PGsl
G2	G	Circular interpolation clockwise	m	+		PGsl
G3	G	Circular interpolation counter-clockwise	m	+		PGsl
G4	G	Dwell time, preset	s	+		PGsl
G5	G	Oblique plunge-cut grinding	s	+		PGAsl
G7	G	Compensatory motion during oblique plunge-cut grinding	s	+		PGAsl
G9	G	Exact stop - deceleration	s	+		PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
G17 [6]	G	Selection of working plane X/Y	m	+		PGsl
G18	G	Selection of working plane Z/X	m	+		PGsl
G19	G	Selection of working plane Y/Z	m	+		PGsl
G25	G	Lower working area limitation	s	+		PGsl
G26	G	Upper working area limitation	s	+		PGsl
G33	G	Thread cutting with constant lead	m	+		PGsl
G34	G	Thread cutting with linear increasing lead	m	+		PGsl
G35	G	Thread cutting with linear decreasing lead	m	+		PGsl
G40 [6]	G	Tool radius compensation OFF	m	+		PGsl
G41	G	Tool radius compensation left of contour	m	+		PGsl
G42	G	Tool radius compensation right of contour	m	+		PGsl
G53	G	Suppression of current zero offset (non-modal)	s	+		PGsl
G54	G	1st settable zero offset	m	+		PGsl
G55	G	2. Settable work offset	m	+		PGsl
G56	G	3. Settable work offset	m	+		PGsl
G57	G	4. Settable work offset	m	+		PGsl
G58 (840D sl)	G	Axial programmable zero offset, absolute, coarse offset	s	+		PGsl
G58 (828D)	G	5. Settable work offset	m	+		PGsl
G59 (840D sl)	G	Axial programmable zero offset, additive, fine offset	s	+		PGsl
G59 (828D)	G	6. Settable work offset	m	+		PGsl
G60 [6]	G	Exact stop - deceleration	m	+		PGsl
G62	G	Corner deceleration at inside corners when tool radius offset is active (G41, G42)	m	+		PGAsl
G63	G	Tapping with compensating chuck	s	+		PGsl
G64	G	Continuous-path mode	m	+		PGsl
G70	G	Inch dimensions for geometric specifications (lengths)	m	+	+	PGsl
G71 [6]	G	Metric dimensions for geometric specifications (lengths)	m	+	+	PGsl
G74	G	Search for reference	s	+		PGsl
G75	G	Fixed point approach	s	+		PGsl
G90 [6]	G	Absolute dimensions	m/s	+		PGsl
G91	G	Incremental dimensions	m/s	+		PGsl
G93	G	Inverse-time feedrate rpm	m	+		PGsl
G94 [6]	G	Linear feedrate F in mm/min or inch/min and degree/min	m	+		PGsl
G95	G	Revolutional feedrate F in mm/rev or inch/rev	m	+		PGsl
G96	G	Constant cutting rate (as for G95) ON	m	+		PGsl
G97	G	Constant cutting rate (as for G95) OFF	m	+		PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
G110	G	Pole programming relative to the last programmed setpoint position	s	+		PGsl
G111	G	Pole programming relative to zero of current workpiece coordinate system	s	+		PGsl
G112	G	Pole programming relative to the last valid pole	s	+		PGsl
G140 [6]	G	SAR approach direction defined by G41/G42	m	+		PGsl
G141	G	SAR approach direction to left of contour	m	+		PGsl
G142	G	SAR approach direction to right of contour	m	+		PGsl
G143	G	SAR approach direction tangent-dependent	m	+		PGsl
G147	G	Soft approach with straight line	s	+		PGsl
G148	G	Soft retraction with straight line	s	+		PGsl
G153	G	Suppression of current frames including basic frame	s	+		PGsl
G247	G	Soft approach with quadrant	s	+		PGsl
G248	G	Soft retraction with quadrant	s	+		PGsl
G290 [6]	G	Switch over to SINUMERIK mode ON	m	+		FBWsl
G291	G	Switch over to ISO2/3 mode ON	m	+		FBWsl
G331	G	Rigid tapping, positive lead, clockwise	m	+		PGsl
G332	G	Rigid tapping, negative lead, counter-clockwise	m	+		PGsl
G335	G	Turning a convex thread in clockwise direction	m	+		PGsl
G336	G	Turning a convex thread in counter-clockwise direction	m	+		PGsl
G340 [6]	G	Spatial approach block (depth and in plane at the same time (helix))	m	+		PGsl
G341	G	Initial infeed on perpendicular axis (z), then approach in plane	m	+		PGsl
G347	G	Soft approach with semicircle	s	+		PGsl
G348	G	Soft retraction with semicircle	s	+		PGsl
G450 [6]	G	Transition circle	m	+		PGsl
G451	G	Intersection of equidistances	m	+		PGsl
G460 [6]	G	Activation of collision detection for the approach and retraction block	m	+		PGsl
G461	G	Insertion of a circle into the TRC block	m	+		PGsl
G462	G	Insertion of a straight line into the TRC block	m	+		PGsl
G500 [6]	G	Deactivation of all adjustable frames, basic frames are active	m	+		PGsl
G505 ... G599	G	5 ... 99. Settable work offset	m	+		PGsl
G601 [6]	G	Block change at exact stop fine	m	+		PGsl
G602	G	Block change at exact stop coarse	m	+		PGsl
G603	G	Block change at IPO block end	m	+		PGsl
G621	G	Corner deceleration at all corners	m	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
G641	G	Continuous-path mode with smoothing as per distance criterion (= programmable rounding clearance)	m	+		PGsl
G642	G	Continuous-path mode with smoothing within the defined tolerances	m	+		PGsl
G643	G	Continuous-path mode with smoothing within the defined tolerances (block-internal)	m	+		PGsl
G644	G	Continuous-path mode with smoothing with maximum possible dynamic response	m	+		PGsl
G645	G	Continuous-path mode with smoothing and tangential block transitions within the defined tolerances	m	+		PGsl
G700	G	Inch dimensions for geometric and technological specifications (lengths, feedrate)	m	+	+	PGsl
G710 [6]	G	Metric dimensions for geometric and technological specifications (lengths, feedrate)	m	+	+	PGsl
G810 [6], ..., G819	G	G group reserved for the OEM user		+		PGAsl
G820 [6], ..., G829	G	G group reserved for the OEM user		+		PGAsl
G931	G	Feedrate specified by traversing time	m	+		
G942	G	Freeze linear feedrate and constant cutting rate or spindle speed	m	+		
G952	G	Freeze revolutional feedrate and constant cutting rate or spindle speed	m	+		
G961	G	Constant cutting rate and linear feedrate	m	+		PGsl
G962	G	Linear or revolutional feedrate and constant cutting rate	m	+		PGsl
G971	G	Freeze spindle speed and linear feedrate	m	+		PGsl
G972	G	Freeze linear or revolutional feedrate and constant spindle speed	m	+		PGsl
G973	G	Revolutional feedrate without spindle speed limitation	m	+		PGsl
GEOAX	P	Assign new channel axes to geometry axes 1 - 3		+	-	PGAsl
GET	P	Replace enabled axis between channels		+	+	PGAsl
GETACTT	F	Gets active tool from a group of tools with the same name		+	-	FBWsl
GETACTTD	F	Gets the T number associated with an absolute D number		+	-	PGAsl
GETD	P	Replace axis directly between channels		+	-	PGAsl
GETDNO	F	Returns the D number of a cutting edge (CE) of a tool (T)		+	-	PGAsl
GETEXET	P	Reading of the loaded T number		+	-	FBWsl
GETFREELOC	P	Find a free space in the magazine for a given tool		+	-	FBWsl
GETSELT	P	Return selected T number		+	-	FBWsl
GETT	F	Get T number for tool name		+	-	FBWsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
GETTCOR	F	Read out tool lengths and/or tool length components		+	-	FB1sl (W1)
GETTENV	F	Read T, D and DL numbers		+	-	FB1sl (W1)
GETVARAP	F	Read access rights to a system/user variable		+	-	PGAsl
GETVARDFT	F	Read default value of a system/user variable		+	-	PGAsl
GETVARLIM	F	Read limit values of a system/user variable		+	-	PGAsl
GETVARPHU	F	Read physical unit of a system/user variable		+	-	PGAsl
GETVARTYP	F	Read data type of a system/user variable		+	-	PGAsl
GOTO	K	Jump operation first forward then backward (direction initially to end of program and then to beginning of program)		+		PGAsl
GOTOB	K	Jump backward (toward the beginning of the program)		+		PGAsl
GOTOC	K	As GOTO, but suppress alarm 14080 "Jump destination not found"		+		PGAsl
GOTOF	K	Jump forward (toward the end of the program)		+		PGAsl
GOTOS	K	Jump back to beginning of program		+		PGAsl
GP	K	Keyword for the indirect programming of position attributes		+		PGAsl
GWPSOF	P	Deselect constant grinding wheel peripheral speed (GWPS)	s	+	-	PGsl
GWPSON	P	Select constant grinding wheel peripheral speed (GWPS)	s	+	-	PGsl
H...	O	Auxiliary function output to the PLC		+	+	PGsl/FB1sl (H2)
HOLES1	C	Row of holes		+		PGAsl
HOLES2	C	Circle of holes		+		PGAsl
I	O	Interpolation parameters	s	+		PGsl
I1	O	Intermediate point coordinate	s	+		PGsl
IC	K	Incremental dimensions	s	+		PGsl
ICYCOF	P	All blocks of a technology cycle are processed in one interpolation cycle following ICYCOF		+	+	FBSYsl
ICYCON	P	Each block of a technology cycle is processed in a separate interpolation cycle following ICYCON		+	+	FBSYsl
ID	K	Identifier for modal synchronized actions	m	-	+	FBSYsl
IDS	K	Identifier for modal static synchronized actions		-	+	FBSYsl
IF	K	Introduction of a conditional jump in the part program/technology cycle		+	+	PGAsl
INDEX	F	Define index of character in input string		+	-	PGAsl
INICF	K	Initialization of variables at NewConfig		+		PGAsl
INIPO	K	Initialization of variables at POWER ON		+		PGAsl
INIRE	K	Initialization of variables at reset		+		PGAsl
INIT	P	Selection of a particular NC program for execution in a particular channel		+	-	PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
INITIAL		Generation of an INI file across all areas		+		PGAsl
INT	K	Data type: Integer with sign		+		PGAsl
INTERSEC	F	Calculate intersection between two contour elements		+	-	PGAsl
INVCCW	G	Trace involute, counter-clockwise	m	+		PGsl
INVCW	G	Trace involute, clockwise	m	+		PGsl
INVFRAME	F	Calculate the inverse frame from a frame		+	-	FB1sl (K2)
IP	K	Variable interpolation parameter		+		PGAsl
IPOBRKA	P	Motion criterion from braking ramp activation	m	+	+	
IPOENDA	K	End of motion when “IPO stop” reached	m	+		PGAsl
IPTRLOCK	P	Freeze start of the untraceable program section at next machine function block.	m	+	-	PGAsl
IPTRUNLOCK	P	Set end of untraceable program section at current block at time of interruption.	m	+	-	PGAsl
IR	O	Center of circle coordinate (X axis) when turning a convex thread		+		PGsl
ISAXIS	F	Check if geometry axis 1 specified as parameter		+	-	PGAsl
ISD	O	Insertion depth	m	+		PGAsl
ISFILE	F	Check whether the file exists in the NCK application memory		+	-	PGAsl
ISNUMBER	F	Check whether the input string can be converted to a number		+	-	PGAsl
ISOCALL	K	Indirect call of a program programmed in an ISO language		+		PGAsl
ISVAR	F	Check whether the transfer parameter contains a variable declared in the NC		+	-	PGAsl
J	O	Interpolation parameters	s	+		PGsl
J1	O	Intermediate point coordinate	s	+		PGsl
JERKA	P	Activate acceleration response set via MD for programmed axes		+	-	
JERKLIM	K	Reduction or overshoot of maximum axial jerk	m	+		PGAsl
JERKLIMA	K	Reduction or overshoot of maximum axial jerk	m	+	+	PGAsl
JR	O	Center of circle coordinate (Y axis) when turning a convex thread		+		PGsl
K	O	Interpolation parameters	s	+		PGsl
K1	O	Intermediate point coordinate	s	+		PGsl
KONT	G	Travel around contour on tool offset	m	+		PGsl
KONTC	G	Approach/retract with continuous-curvature polynomial	m	+		PGsl
KONTT	G	Approach/retract with continuous-tangent polynomial	m	+		PGsl
KR	O	Center of circle coordinate (Z axis) when turning a convex thread		+		PGsl
L	O	Subprogram number	s	+	+	PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
LEAD	O	Lead angle 1. Tool orientation 2. Orientation polynomial	m	+		PGAsl
LEADOF	P	Axial master value coupling OFF		+	+	PGAsl
LEADON	P	Axial master value coupling on		+	+	PGAsl
LENTOAX	F	Provides information about the assignment of tool lengths L1, L2, and L3 of the active tool to the abscissa, ordinate and applicate		+	-	FB1sl (W1)
LFOF [6]	G	Fast retraction for thread cutting OFF	m	+		PGsl
LFON	G	Fast retraction for thread cutting ON	m	+		PGsl
LFPOS	G	Retraction of the axis declared with POLFMASK or POLFMLIN to the absolute axis position programmed with POLF	m	+		PGsl
LFTXT [6]	G	The plane of the retraction movement for fast retraction is determined from the path tangent and the current tool direction	m	+		PGsl
LFWP	G	The plane of the retraction movement for fast retraction is determined by the current working plane (G17/G18/G19)	m	+		PGsl
LIFTFAST	K	Fast retraction		+		PGsl
LIMS	K	Speed limitation for G96/G961 and G97	m	+		PGsl
LLI	K	Lower limit value of variables		+		PGAsl
LN	F	Natural logarithm		+	+	PGAsl
LOCK	P	Disable synchronized action with ID (stop technology cycle)		-	+	FBSYsl
LONGHOLE	C	Elongated hole		+		PGAsl
LOOP	K	Introduction of an endless loop		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
M0		Programmed stop		+	+	PGsl
M1		Optional stop		+	+	PGsl
M2		End of program, main program (as M30)		+	+	PGsl
M3		CW spindle rotation		+	+	PGsl
M4		CCW spindle rotation		+	+	PGsl
M5		Spindle stop		+	+	PGsl
M6		Tool change		+	+	PGsl
M17		End of subprogram		+	+	PGsl
M19		Spindle positioning to the position entered in SD43240		+	+	PGsl
M30		End of program, main program (as M2)		+	+	PGsl
M40		Automatic gear change		+	+	PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
M41 ... M45		Gear stage 1 ... 5		+	+	PGsl
M70		Transition to axis mode		+	+	PGsl
MASLDEF	P	Define master/slave axis grouping		+	+	PGAsl
MASLDEL	P	Uncouple master/slave axis grouping and clear grouping definition		+	+	PGAsl
MASLOF	P	Deactivation of a temporary coupling		+	+	PGAsl
MASLOFS	P	Deactivation of a temporary coupling with automatic slave axis stop		+	+	PGAsl
MASLON	P	Activation of a temporary coupling		+	+	PGAsl
MATCH	F	Search for string in string		+	-	PGAsl
MAXVAL	F	Larger value of two variables (arithm. function)		+	+	PGAsl
MCALL	K	Modal subprogram call		+		PGAsl
MEAC	K	Continuous axial measurement without delete distance-to-go	s	+	+	PGAsl
MEAFRAME	F	Frame calculation from measuring points		+	-	PGAsl
MEAS	O	Measurement with deletion of distance-to-go	s	+		PGAsl
MEASA	K	Axial measurement with delete distance-to-go	s	+	+	PGAsl
MEASURE	F	Calculation method for workpiece and tool measurement		+	-	FB1sl (M5)
MEAW	O	Measurement without delete distance-to-go	s	+		PGAsl
MEAWA	K	Axial measurement without delete distance-to-go	s	+	+	PGAsl
MI	K	Access to frame data: Mirroring		+		PGAsl
MINDEX	F	Define index of character in input string		+	-	PGAsl
MINVAL	F	Smaller value of two variables (arithm. function)		+	+	PGAsl
MIRROR	G	Programmable mirroring	s	+		PGAsl
MMC	P	Call the dialog window interactively from the part program on the HMI		+	-	PGAsl
MOD	K	Modulo division		+		PGAsl
MODAXVAL	F	Determine modulo position of a modulo rotary axis		+	-	PGAsl
MOV	K	Start positioning axis		-	+	FBSYsl
MOVT	O	Specify end point of a traversing motion in the tool direction				FB1(K2)
MSG	P	Programmable messages	m	+	-	PGsl
MVTOOL	P	Language command to move tool		+	-	FBWsl
N	O	NC auxiliary block number		+		PGsl
NAMETOINT	F	Determining the system variable index		+		PGAsl
NCK	K	Specify validity range for data		+		PGAsl
NEWCONF	P	Apply modified machine data (corresponds to "Activate machine data")		+	-	PGAsl
NEWMT	F	Create new multitool		+	-	FBWsl
NEWT	F	Create new tool		+	-	FBWsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
NORM [6]	G	Standard setting in starting point and end point with tool offset	m	+		PGsl
NOT	K	Logic NOT (negation)		+		PGAsl
NPROT	P	Machine-specific protection area ON/OFF		+	-	PGAsl
NPROTDEF	P	Definition of a machine-specific protection area		+	-	PGAsl
NUMBER	F	Convert input string to number		+	-	PGAsl
OEMIPO1	G	OEM interpolation 1	m	+		PGAsl
OEMIPO2	G	OEM interpolation 2	m	+		PGAsl
OF	K	Keyword in CASE branch		+		PGAsl
OFFN	O	Allowance on the programmed contour	m	+		PGsl
OMA1	O	OEM address 1	m	+		PGAsl
OMA2	O	OEM address 2	m	+		PGAsl
OMA3	O	OEM address 3	m	+		PGAsl
OMA4	O	OEM address 4	m	+		PGAsl
OMA5	O	OEM address 5	m	+		PGAsl
OR	K	Logic operator, OR operation		+		PGAsl
ORIAXES	G	Linear interpolation of machine axes or orientation axes	m	+		PGAsl
ORIAXPOS	G	Orientation angle via virtual orientation axes with rotary axis positions	m	+		PGAsl
ORIC [6]	G	Orientation changes at outside corners are superimposed on the circle block to be inserted	m	+		PGAsl
ORICONCCW	G	Interpolation on a circular peripheral surface in CCW direction	m	+		PGAsl/FB3sl (F3)
ORICONCW	G	Interpolation on a circular peripheral surface in CW direction	m	+		PGAsl/FB3sl (F4)
ORICONIO	G	Interpolation on a circular peripheral surface with intermediate orientation setting	m	+		PGAsl/FB3sl (F4)
ORICONTO	G	Interpolation on circular peripheral surface in tangential transition (final orientation)	m	+		PGAsl/FB3sl (F5)
ORICURVE	G	Interpolation of orientation with specification of motion of two contact points of tool	m	+		PGAsl/FB3sl (F6)
ORID	G	Orientation changes are performed before the circle block	m	+		PGAsl
ORIEULER [6]	G	Orientation angle via Euler angle	m	+		PGAsl
ORIMKS	G	Tool orientation in the machine coordinate system	m	+		PGAsl
ORIPATH	G	Tool orientation in relation to path	m	+		PGAsl
ORIPATHS	G	Tool orientation in relation to path, blips in the orientation characteristic are smoothed	m	+		PGAsl
ORIPLANE	G	Interpolation in a plane (corresponds to ORIVECT), large-radius circular interpolation	m	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
ORIRESET	P	Initial tool orientation with up to 3 orientation axes		+	-	PGAsl
ORIROTA [6]	G	Angle of rotation to an absolute direction of rotation	m	+		PGAsl
ORIROTC	G	Tangential rotational vector in relation to path tangent	m	+		PGAsl
ORIROTR	G	Angle of rotation relative to the plane between the start and end orientation	m	+		PGAsl
ORIROTT	G	Angle of rotation relative to the change in the orientation vector	m	+		PGAsl
ORIRPY	G	Orientation angle via RPY angle (XYZ)	m	+		PGAsl
ORIRPY2	G	Orientation angle via RPY angle (ZYX)	m	+		PGAsl
ORIS	O	Change in orientation	m	+		PGAsl
ORISOF [6]	G	Smoothing of the orientation characteristic OFF	m	+		PGAsl
ORISON	G	Smoothing of the orientation characteristic ON	m	+		PGAsl
ORIVECT [6]	G	Large-circle interpolation (identical to ORIPLANE)	m	+		PGAsl
ORIVIRT1	G	Orientation angle via virtual orientation axes (definition 1)	m	+		PGAsl
ORIVIRT2	G	Orientation angle via virtual orientation axes (definition 1)	m	+		PGAsl
ORIWKS [6]	G	Tool orientation in the workpiece coordinate system	m	+		PGAsl
OS	K	Oscillation on/off		+		PGAsl
OSB	K	Oscillating: Starting point	m	+		FB1sl (P5)
OSC	G	Continuous tool orientation smoothing	m	+		PGAsl
OSCILL	K	Axis: 1 - 3 infeed axes	m	+		PGAsl
OSCTRL	K	Oscillation options	m	+		PGAsl
OSD	G	Smoothing of tool orientation by specifying smoothing distance with SD	m	+		PGAsl
OSE	K	Oscillation end position	m	+		PGAsl
OSNSC	K	Oscillating: Number of spark-out cycles	m	+		PGAsl
OSOF [6]	G	Tool orientation smoothing OFF	m	+		PGAsl
OSP1	K	Oscillating: Left reversal point	m	+		PGAsl
OSP2	K	Oscillation right reversal point	m	+		PGAsl
OSS	G	Tool orientation smoothing at end of block	m	+		PGAsl
OSSE	G	Tool orientation smoothing at start and end of block	m	+		PGAsl
OST	G	Smoothing of tool orientation by specifying angular tolerance in degrees with SD (maximum deviation from programmed orientation characteristic)	m	+		PGAsl
OST1	K	Oscillating: Stopping point in left reversal point	m	+		PGAsl
OST2	K	Oscillating: Stopping point in right reversal point	m	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
OTOL	K	Orientation tolerance for compressor functions, orientation smoothing and smoothing types		+		PGAsl
OVR	K	Speed offset	m	+		PGAsl
OVRA	K	Axial speed offset	m	+	+	PGAsl
OVRRAP	K	Rapid traverse override	m	+		PGAsl
P	O	Number of subprogram repetitions		+		PGAsl
PAROT	G	Align workpiece coordinate system on workpiece	m	+		PGsl
PAROTOF [6]	G	Deactivate frame rotation in relation to workpiece	m	+		PGsl
PCALL	K	Call subprograms with absolute path and parameter transfer		+		PGAsl
PDELAYOF	G	Punching with delay OFF	m	+		PGAsl
PDELAYON [6]	G	Punching with delay ON	m	+		PGAsl
PHI	K	Angle of rotation of the orientation around the direction axis of the taper		+		PGAsl
PHU	K	Physical unit of a variable		+		PGAsl
PL	O	1. B spline: Node clearance 2. Polynomial interpolation: Length of the parameter interval for polynomial interpolation	s	+		PGAsl
PM	K	Per minute		+		PGsl
PO	K	Polynomial coefficient for polynomial interpolation	s	+		PGAsl
POCKET3	C	Milling the rectangular pocket		+		PGAsl
POCKET4	C	Milling the circular pocket		+		PGAsl
POLF	K	LIFTFAST retraction position	m	+		PGsl/PGAsl
POLFA	P	Start retraction position of single axes with $AA_ESR_TRIGGER	m	+	+	PGsl
POLFMASK	P	Enable axes for retraction without a connection between the axes	m	+	-	PGsl
POLFMLIN	P	Enable axes for retraction with a linear connection between the axes	m	+	-	PGsl
POLY	G	Polynomial interpolation	m	+		PGAsl
POLYPATH	P	Polynomial interpolation can be selected for the AXIS or VECT axis groups	m	+	-	PGAsl
PON	G	Punching ON	m	+		PGAsl
PONS	G	Punching ON in interpolation cycle	m	+		PGAsl
POS	K	Axis positioning		+	+	PGsl
POSA	K	Position axis across block boundary		+	+	PGsl
POSM	P	Position magazine		+	-	FBWsl
POSMT	P	Position multitool on toolholder at location number		+	-	FBWsl
POSP	K	Positioning axis in parts (oscillation)		+		PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
POSRANGE	F	Determine whether the currently interpolated position setpoint of an axis is located in a window at a predefined reference position		+	+	FBSYsl
POT	F	Square (arithmetic function)		+	+	PGAsl
PR	K	Per revolution		+		PGsl
PREPRO	PA	Identify subprograms with preparation		+		PGAsl
PRESETON	P	Actual value setting with loss of the referencing status		+	+	PGAsl
PRESETONS	P	Actual value setting with loss of the referencing status		+	+	PGAsl
PRIO	K	Keyword for setting the priority for interrupt processing		+		PGAsl
PRLOC	K	Initialization of variables at reset only after local change		+		PGAsl
PROC	K	First operation in a program		+		PGAsl
PROTA	P	Request for a recalculation of the collision model		+		PGAsl
PROTD	F	Calculating the distance between two protection areas		+		PGAsl
PROTS	P	Setting the protection area status		+		PGAsl
PSI	K	Opening angle of the taper		+		PGAsl
PTP	G	Point-to-point motion	m	+		PGAsl
PTPG0	G	Point-to-point motion only with G0, otherwise CP	m	+		PGAsl
PUNCHACC	P	Travel-dependent acceleration for nibbling		+	-	PGAsl
PUTFTOC	P	Tool fine offset for parallel dressing		+	-	PGAsl
PUTFTOCF	P	Tool fine offset dependent on a function for parallel dressing defined with FCTDEF		+	-	PGAsl
PW	O	B spline, point weight	s	+		PGAsl
QU	K	Fast additional (auxiliary) function output		+		PGsl
R...	O	Arithmetic parameter also as settable address identifier and with numerical extension		+		PGAsl
RAC	K	Absolute non-modal axis-specific radius programming	s	+		PGsl
RDISABLE	P	Read-in disable		-	+	FBSYsl
READ	P	Reads one or more lines in the specified file and stores the information read in the array		+	-	PGAsl
REAL	K	Data type: Floating-point variable with sign (real numbers)		+		PGAsl
REDEF	K	Redefinition of system variables, user variables, and NC language commands		+		PGAsl
RELEASE	P	Release machine axes for axis exchange		+	+	PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
REP	K	Keyword for initialization of all elements of an array with the same value		+		PGAsl
REPEAT	K	Repetition of a program loop		+		PGAsl
REPEATB	K	Repetition of a program line		+		PGAsl
REPOSA	G	Linear repositioning with all axes	s	+		PGAsl
REPOSH	G	Repositioning with semicircle	s	+		PGAsl
REPOSHA	G	Repositioning with all axes; geometry axes in semicircle	s	+		PGAsl
REPOSL	G	Linear repositioning	s	+		PGAsl
REPOSQ	G	Repositioning in a quadrant	s	+		PGAsl
REPOSQA	G	Linear repositioning with all axes, geometry axes in quadrant	s	+		PGAsl
RESET	P	Reset technology cycle		-	+	FBSYsl
RESETMON	P	Language command for setpoint activation		+	-	FBWsl
RET	P	End of subprogram		+	+	PGAsl
RETB	P	End of subprogram		+	+	PGAsl
RIC	K	Relative non-modal axis-specific radius programming	s	+		PGsl
RINDEX	F	Define index of character in input string		+	-	PGAsl
RMB	G	Repositioning to start of block	m	+		PGAsl
RMBBL	G	Repositioning to start of block	s	+		PGAsl
RME	G	Repositioning to end of block	m	+		PGAsl
RMEBL	G	Repositioning to end of block	s	+		PGAsl
RMI [6]	G	Repositioning to interrupt point	m	+		PGAsl
RMIBL [6]	G	Repositioning to interrupt point	s	+		PGAsl
RMN	G	Repositioning to the nearest path point	m	+		PGAsl
RMNBL	G	Repositioning to the nearest path point	s	+		PGAsl
RND	O	Round the contour corner	s	+		PGsl
RNDM	O	Modal rounding	m	+		PGsl
ROT	G	Programmable rotation	s	+		PGsl
ROTS	G	Programmable frame rotations with solid angles	s	+		PGsl
ROUND	F	Rounding of decimal places		+	+	PGAsl
ROUNDUP	F	Rounding up of an input value		+	+	PGAsl
RP	O	Polar radius	m/s	+		PGsl
RPL	O	Rotation in the plane	s	+		PGsl
RT	K	Parameter for access to frame data: Rotation		+		PGAsl
RTLIOF	G	G0 without linear interpolation (single-axis interpolation)	m	+		PGsl
RTLION [6]	G	G0 with linear interpolation	m	+		PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
S	O	Spindle speed (with G4, G96/G961 different meaning)	m/s	+	+	PGsl
SAVE	PA	Attribute for saving information when subprograms are called		+		PGAsl
SBLOF	P	Suppress single block		+	-	PGAsl
SBLON	P	Revoke suppression of single block		+	-	PGAsl
SC	K	Parameter for access to frame data: Scaling		+		PGAsl
SCALE	G	Programmable scaling	s	+		PGsl
SCC	K	Selective assignment of transverse axis to G96/G961/G962. Axis identifiers may take the form of geometry, channel or machine axes.		+		PGsl
SCPARA	K	Program servo parameter set		+	+	PGAsl
SD	O	Spline degree	s	+		PGAsl
SET	K	Keyword for initialization of all elements of an array with listed values		+		PGAsl
SETAL	P	Set alarm		+	+	PGAsl
SETDNO	F	Assign the D number of a cutting edge (CE) of a tool (T)		+	-	PGAsl
SETINT	K	Define which interrupt routine is to be activated when an NCK input is present		+		PGAsl
SETM	P	Setting of markers in dedicated channel		+	+	PGAsl
SETMS	P	Reset to the master spindle defined in machine data		+	-	PGsl
SETMS(n)	P	Set spindle n as master spindle		+		PGsl
SETMTH	P	Set master toolholder number		+	-	FBWsl
SETPIECE	P	Set piece number for all tools assigned to the spindle		+	-	FBWsl
SETTA	P	Activate tool from wear group		+	-	FBWsl
SETTCOR	F	Modification of tool components taking all supplementary conditions into account		+	-	FB1sl (W1)
SETTIA	P	Deactivate tool from wear group		+	-	FBWsl
SF	O	Starting point offset for thread cutting	m	+		PGsl
SIN	F	Sine (trigon. function)		+	+	PGAsl
SIRELAY	F	Activate the safety functions parameterized with SIRELIN, SIRELOUT, and SIRELTIME		-	+	FBSIsl
SIRELIN	P	Initialize input variables of function block		+	-	FBSIsl
SIRELOUT	P	Initialize output variables of function block		+	-	FBSIsl
SIRELTIME	P	Initialize timers of function block		+	-	FBSIsl
SLOT1	C	Longitudinal groove		+		PGAsl
SLOT2	C	Circumferential groove		+		PGAsl
SOFT	G	Soft path acceleration	m	+		PGsl
SOFTA	P	Activate jerk-limited axis acceleration for the programmed axes		+	-	PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
SON	G	Nibbling ON	m	+		PGAsl
SONS	G	Nibbling ON in interpolation cycle	m	+		PGAsl
SPATH [6]	G	Path reference for FGROUP axes is arc length	m	+		PGAsl
SPCOF	P	Switch master spindle or spindle(s) from position control to speed control	m	+	-	PGsl
SPCON	P	Switch master spindle or spindle(s) from speed control to position control	m	+	-	PGAsl
SPI	F	Converts spindle number into axis identifier		+	-	PGAsl
SPIF1 [6]	G	Fast NCK inputs/outputs for punching/nibbling byte 1	m	+		FB2sl (N4)
SPIF2	G	Fast NCK inputs/outputs for punching/nibbling byte 2	m	+		FB2sl (N4)
SPLINEPATH	P	Define spline grouping		+	-	PGAsl
SPN	O	Number of path sections per block	s	+		PGAsl
SPOF [6]	G	Stroke OFF, nibbling, punching OFF	m	+		PGAsl
SPOS	K	Spindle position	m	+	+	PGsl
SPOSA	K	Spindle position across block boundaries	m	+		PGsl
SPP	O	Length of a path section	m	+		PGAsl
SPRINT	F	Returns an input string formatted		+		PGAsl
SQRT	F	Square root (arithmetic function)		+	+	PGAsl
SR	O	Oscillation retraction path for synchronized action	s	+		PGsl
SRA	K	Oscillation retraction path with external input axial for synchronized action	m	+		PGsl
ST	O	Oscillation sparking-out time for synchronized action	s	+		PGsl
STA	K	Oscillation sparking-out time axial for synchronized action	m	+		PGsl
START	P	Start selected programs simultaneously in several channels from current program		+	-	PGAsl
STARTFIFO [6]	G	Execute; fill preprocessing memory simultaneously	m	+		PGAsl
STAT		Position of joints	s	+		PGAsl
STOLF	K	G0 tolerance factor	m	+		PGAsl
STOPFIFO	G	Stop machining; fill preprocessing memory until STARTFIFO is detected, preprocessing memory is full or end of program	m	+		PGAsl
STOPRE	P	Preprocessing stop until all prepared blocks in the main run are executed		+	-	PGAsl
STOPREOF	P	Revoke preprocessing stop		-	+	FBSYsl
STRING	K	Data type: Character string		+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
STRINGIS	F	Checks the present scope of NC language and the NC cycle names, user variables, macros, and label names belonging specifically to this command to establish whether these exist, are valid, defined or active.		+	-	PGAsl
STRLEN	F	Define string length		+	-	PGAsl
SUBSTR	F	Define index of character in input string		+	-	PGAsl
SUPA	G	Suppression of current zero offset, including programmed offsets, system frames, handwheel offsets (DRF), external zero offset, and overlaid movement	s	+		PGsl
SVC	K	Tool cutting rate	m	+		PGsl
SYNFCT	P	Evaluation of a polynomial as a function of a condition in the motion-synchronous action		-	+	FBSYsl
SYNR	K	The variable is read synchronously, i.e. at the time of execution		+		PGAsl
SYNRW	K	The variable is read and written synchronously, i.e. at the time of execution		+		PGAsl
SYNW	K	The variable is written synchronously, i.e. at the time of execution		+		PGAsl
T	O	Call tool (only change if specified in machine data; otherwise M6 command necessary)		+		PGsl
TAN	F	Tangent (trigon. function)		+	+	PGAsl
TANG	P	Definition of axis grouping tangential correction		+	-	PGAsl
TANGDEL	P	Deletion of definition of axis grouping tangential correction		+	-	PGAsl
TANGOF	P	Tangential correction OFF		+	-	PGAsl
TANGON	P	Tangential correction ON		+	-	PGAsl
TCA (828D: _TCA)	P	Tool selection/tool change irrespective of tool status		+	-	FBWsl
TCARR	O	Request toolholder (number "m")		+		PGAsl
TCI	P	Load tool from buffer into magazine		+	-	FBWsl
TCOABS [6]	G	Determine tool length components from the current tool orientation	m	+		PGAsl
TCOFR	G	Determine tool length components from the orientation of the active frame	m	+		PGAsl
TCOFRX	G	Determine tool orientation of an active frame on selection of tool, tool points in X direction	m	+		PGAsl
TCOFRY	G	Determine tool orientation of an active frame on selection of tool, tool points in Y direction	m	+		PGAsl
TCOFRZ	G	Determine tool orientation of an active frame on selection of tool, tool points in Z direction	m	+		PGAsl
THETA	O	Angle of rotation	s	+		PGAsl
TILT	O	Tilt angle	m	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
TLIFT	P	In tangential control insert intermediate block at contour corners		+	-	PGAsl
TML	P	Tool selection with magazine location number		+	-	FBWsl
TMOF	P	Deselect tool monitoring		+	-	PGAsl
TMON	P	Activate tool monitoring		+	-	PGAsl
TO	K	Designates the end value in a FOR counter loop		+		PGAsl
TOFF	K	Tool length offset in the direction of the tool length component that is effective parallel to the geometry axis specified in the index.	m	+		PGsl
TOFFL	K	Tool length offset in the direction of the tool length component L1, L2 or L3	m	+		PGsl
TOFFOF	P	Deactivate online tool offset		+	-	PGAsl
TOFFON	P	Activate online tool length offset		+	-	PGAsl
TOFFR	O	Tool radius offset	m	+		PGsl
TOFRAME	G	Align Z axis of the WCS by rotating the frame parallel to the tool orientation	m	+		PGsl
TOFRAMEX	G	Align X axis of the WCS by rotating the frame parallel to the tool orientation	m	+		PGsl
TOFRAMEY	G	Align Y axis of the WCS by rotating the frame parallel to the tool orientation	m	+		PGsl
TOFRAMEZ	G	As TOFRAME	m	+		PGsl
TOLOWER	F	Convert the letters of a string into lowercase		+	-	PGAsl
TOOLENV	F	Save current states which are of significance to the evaluation of the tool data stored in the memory		+	-	FB1sl (W1)
TOOLGNT	F	Determine number of tools of a tool group		+	-	FBWsl
TOOLGT	F	Determine T number of a tool from a tool group		+	-	FBWsl
TOROT	G	Align Z axis of the WCS by rotating the frame parallel to the tool orientation	m	+		PGsl
TOROTOF [6]	G	Frame rotations in tool direction OFF	m	+		PGsl
TOROTX	G	Align X axis of the WCS by rotating the frame parallel to the tool orientation	m	+		PGsl
TOROTY	G	Align Y axis of the WCS by rotating the frame parallel to the tool orientation	m	+		PGsl
TOROTZ	G	As TOROT	m	+		PGsl
TOUPPER	F	Convert the letters of a string into uppercase		+	-	PGAsl
TOWBCS	G	Wear values in the basic coordinate system (BCS)	m	+		PGAsl
TOWKCS	G	Wear values in the coordinate system of the tool head for kinetic transformation (differs from machine coordinate system through tool rotation)	m	+		PGAsl
TOWMCS	G	Wear values in machine coordinate system	m	+		PGAsl
TOWSTD [6]	G	Initial setting value for offsets in tool length	m	+		PGAsl
TOWTCS	G	Wear values in the tool coordinate system (toolholder ref. point T at the tool holder)	m	+		PGAsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
TOWWCS	G	Wear values in workpiece coordinate system	m	+		PGAsl
TR	K	Offset component of a frame variable		+		PGAsl
TRAANG	P	Transformation inclined axis		+	-	PGAsl
TRACON	P	Cascaded transformation		+	-	PGAsl
TRACYL	P	Cylinder: Peripheral surface transformation		+	-	PGAsl
TRAFOOF	P	Deactivate active transformations in the channel		+	-	PGAsl
TRAILOF	P	Asynchronous coupled motion OFF		+	+	PGAsl
TRAILON	P	Asynchronous coupled motion ON		+	+	PGAsl
TRANS	G	Programmable offset	s	+		PGsl
TRANSMIT	P	Pole transformation (face machining)		+	-	PGAsl
TRAORI	P	4-axis, 5-axis transformation, generic transformation		+	-	PGAsl
TRUE	K	Logical constant: True		+		PGAsl
TRUNC	F	Truncation of decimal places		+	+	PGAsl
TU		Axis angle	s	+		PGAsl
TURN	O	Number of turns for helix	s	+		PGsl
ULI	K	Upper limit value of variables		+		PGAsl
UNLOCK	P	Enable synchronized action with ID (continue technology cycle)		-	+	FBSYsl
UNTIL	K	Condition for end of REPEAT loop		+		PGAsl
UPATH	G	Path reference for FGROUP axes is curve parameter	m	+		PGAsl
VAR	K	Keyword: Type of parameter transfer		+		PGAsl
VELOLIM	K	Reduction of the maximum axial velocity	m	+		PGAsl
VELOLIMA	K	Reduction or increase of the maximum axial velocity of the following axis	m	+	+	PGAsl
WAITC	P	Wait for the coupling block change criterion to be fulfilled for the axes/spindles		+	-	PGAsl
WAITE	P	Wait for end of program in another channel.		+	-	PGAsl
WAITENC	P	Wait for synchronized or restored axis positions		+	-	PGAsl
WAITM	P	Wait for marker in specified channel; terminate previous block with exact stop.		+	-	PGAsl
WAITMC	P	Wait for marker in specified channel; exact stop only if the other channels have not yet reached the marker.		+	-	PGAsl
WAITP	P	Wait for end of travel of the positioning axis		+	-	PGsl
WAITS	P	Wait for spindle position to be reached		+	-	PGsl
WALCS0 [6]	G	Workpiece coordinate system working area limitation deselected	m	+	-	PGsl
WALCS1	G	WCS working area limitation group 1 active	m	+	-	PGsl
WALCS2	G	WCS working area limitation group 2 active	m	+	-	PGsl
WALCS3	G	WCS working area limitation group 3 active	m	+	-	PGsl
WALCS4	G	WCS working area limitation group 4 active	m	+	-	PGsl

Operation	Type [1]	Meaning	W [2]	TP [3]	SA [4]	Description see [5]
[1] [2] [3] [4] [5] for explanations, see legend (Page 759).						
WALCS5	G	WCS working area limitation group 5 active	m	+	-	PGsl
WALCS6	G	WCS working area limitation group 6 active	m	+	-	PGsl
WALCS7	G	WCS working area limitation group 7 active	m	+	-	PGsl
WALCS8	G	WCS working area limitation group 8 active	m	+	-	PGsl
WALCS9	G	WCS working area limitation group 9 active	m	+	-	PGsl
WALCS10	G	WCS working area limitation group 10 active	m	+	-	PGsl
WALIMOF	G	BCS working area limitation OFF	m	+	-	PGsl
WALIMON [6]	G	BCS working area limitation ON	m	+	-	PGsl
WHEN	K	The action is executed once whenever the condition is fulfilled.		-	+	FBSYsl
WHENEVER	K	The action is executed cyclically in each interpolator cycle when the condition is fulfilled.		-	+	FBSYsl
WHILE	K	Start of WHILE program loop		+		PGAsl
WRITE	P	Write text to file system. Appends a block to the end of the specified file.		+	-	PGAsl
WRTPR	P	Delays the machining job without interrupting continuous-path mode		+	-	PGAsl
X	O	Axis name	m/s	+	+	PGsl
XOR	O	Logic exclusive OR		+		PGAsl
Y	O	Axis name	m/s	+	+	PGsl
Z	O	Axis name	m/s	+	+	PGsl

Legend

[1] Type of operation:

- A Address
  Identifier to which a value is assigned (e.g. OVR=10). There are also some addresses that switch on or off a function without value assignment (e.g. CPLON and CPLOF).
- C Technological cycle
  Predefined part program in which a generally valid specific cycle (machining operation), such as tapping of a thread or milling a pocket, is programmed. The adaptation to a specific machine situation is realized via parameters that are transferred to the cycle during the call.
- F Predefined function (supplies a return value)
  The call of the predefined function can be an operand in an expression.
- G G function
  G functions are divided into function groups. Only one G function of a group can be programmed in a block. A G function can be either modal (until it is canceled by another function of the same group) or only effective for the block in which it is programmed (non-modal).
- K Keyword
  Identifier that defines the syntax of a block. No value is assigned to a keyword, and no NC function can be switched on/off with a keyword.
  Examples: Control structures (IF, ELSE, ENDIF, WHEN, ...), program execution (GOTOB, GOTO, RET ...)
- O Operator
  Operator for a mathematical, comparison or logical operation
- P Predefined procedure (does not supply a return value)
- PA Program attribute
  Program attributes are at the end of the definition line of a subprogram:
  `PROC <program name>(...) <program attribute>`
  They determine the behavior during execution of the subprogram.

[2] Effectiveness of the operation:

- m Modal
- s Non-modal

[3] Programmability in part program:

- \+ Programmable
- \- Not programmable

[4] Programmability in synchronized actions:

- \+ Programmable
- \- Not programmable
- T Programmable only in technology cycles

[5] Reference to the document containing the detailed description of the operation:
PGsl Programming Manual, Fundamentals
PGAsl Programming Manual, Job Planning

BNMs/ Programming Manual Measuring Cycles

BHDs/ Operating Manual, Turning

BHFs/ Operating Manual, Milling

FB1s/ ( ) Function Manual, Basic Functions (with the alphanumeric abbreviation of the corresponding function description in brackets)

FB2s/ ( ) Function Manual, Extended Functions (with the alphanumeric abbreviation of the corresponding function description in brackets)

FB3s/ ( ) Function Manual, Special Functions (with the alphanumeric abbreviation of the corresponding function description in brackets)

FBSs/ Function Manual, Safety Integrated

FBSYs/ Function Manual, Synchronized Actions

FBWs/ Function Manual, Tool Management

6) Default setting at beginning of program (factory settings of the control, if nothing else programmed).

20.2 Operations: Availability for SINUMERIK 828D

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
:	●	●	●	●	●	●
*	●	●	●	●	●	●
+	●	●	●	●	●	●
-	●	●	●	●	●	●
<	●	●	●	●	●	●
<<	●	●	●	●	●	●
<=	●	●	●	●	●	●
=	●	●	●	●	●	●
>=	●	●	●	●	●	●
/	●	●	●	●	●	●
/0 … … /7	● ○	● ○	● ○	● ○	● ○	● ○
A	●	●	●	●	●	●
A2	-	-	-	-	-	-
A3	-	-	-	-	-	-
A4	-	-	-	-	-	-
A5	-	-	-	-	-	-
ABS	●	●	●	●	●	●
AC	●	●	●	●	●	●
ACC	●	●	●	●	●	●
ACCLIMA	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
ACN	●	●	●	●	●	●
ACOS	●	●	●	●	●	●
ACP	●	●	●	●	●	●
ACTBLOCNO	●	●	●	●	●	●
ADDFRAME	●	●	●	●	●	●
ADIS	●	●	●	●	●	●
ADISPOS	●	●	●	●	●	●
ADISPOSA	●	●	●	●	●	●
ALF	●	●	●	●	●	●
AMIRROR	●	●	●	●	●	●
AND	●	●	●	●	●	●
ANG	●	●	●	●	●	●
AP	●	●	●	●	●	●
APR	●	●	●	●	●	●
APRB	●	●	●	●	●	●
APRP	●	●	●	●	●	●
APW	●	●	●	●	●	●
APWB	●	●	●	●	●	●
APWP	●	●	●	●	●	●
APX	●	●	●	●	●	●
AR	●	●	●	●	●	●
AROT	●	●	●	●	●	●
AROTS	●	●	●	●	●	●
AS	●	●	●	●	●	●
ASCALE	●	●	●	●	●	●
ASIN	●	●	●	●	●	●
ASPLINE	-	○	-	○	-	○
ATAN2	●	●	●	●	●	●
ATOL	-	●	-	●	-	●
ATRANS	●	●	●	●	●	●
AUXFUDEL	●	●	●	●	●	●
AUXFUDELG	●	●	●	●	●	●
AUXFUMSEQ	●	●	●	●	●	●
AUXFUSYNC	●	●	●	●	●	●
AX	●	●	●	●	●	●
AXCTSWE	-	-	-	-	-	-
AXCTSWEC	-	-	-	-	-	-
AXCTSWED	-	-	-	-	-	-
AXIS	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
AXNAME	●	●	●	●	●	●
AXSTRING	●	●	●	●	●	●
AXTOCHAN	●	●	●	●	●	●
AXTOSPI	●	●	●	●	●	●
B	●	●	●	●	●	●
B2	-	-	-	-	-	-
B3	-	-	-	-	-	-
B4	-	-	-	-	-	-
B5	-	-	-	-	-	-
B_AND	●	●	●	●	●	●
B_OR	●	●	●	●	●	●
B_NOT	●	●	●	●	●	●
B_XOR	●	●	●	●	●	●
BAUTO	-	○	-	○	-	○
BLOCK	●	●	●	●	●	●
BLSYNC	●	●	●	●	●	●
BNAT	-	○	-	○	-	○
BOOL	●	●	●	●	●	●
BOUND	●	●	●	●	●	●
BRISK	●	●	●	●	●	●
BRISKA	●	●	●	●	●	●
BSPLINE	-	○	-	○	-	○
BTAN	-	○	-	○	-	○
C	●	●	●	●	●	●
C2	-	-	-	-	-	-
C3	-	-	-	-	-	-
C4	-	-	-	-	-	-
C5	-	-	-	-	-	-
CAC	●	●	●	●	●	●
CACN	●	●	●	●	●	●
CACP	●	●	●	●	●	●
CALCDAT	●	●	●	●	●	●
CALCPOSI	●	●	●	●	●	●
CALL	●	●	●	●	●	●
CALLPATH	●	●	●	●	●	●
CANCEL	●	●	●	●	●	●
CASE	●	●	●	●	●	●
CDC	●	●	●	●	●	●
CDOF	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
CDOF2	●	●	●	●	●	●
CDON	●	●	●	●	●	●
CFC	●	●	●	●	●	●
CFIN	●	●	●	●	●	●
CFINE	●	●	●	●	●	●
CFTCP	●	●	●	●	●	●
CHAN	●	●	●	●	●	●
CHANDATA	●	●	●	●	●	●
CHAR	●	●	●	●	●	●
CHF	●	●	●	●	●	●
CHKDM	●	●	●	●	●	●
CHKDNO	●	●	●	●	●	●
CHR	●	●	●	●	●	●
CIC	●	●	●	●	●	●
CIP	●	●	●	●	●	●
CLEARM	-	-	-	-	-	-
CLRINT	●	●	●	●	●	●
CMIRROR	●	●	●	●	●	●
COARSEA	●	●	●	●	●	●
COLLPAIR	-	-	-	-	-	-
COMPCAD	-	○	-	○	-	○
COMPCURV	-	○	-	○	-	○
COMPLETE	●	●	●	●	●	●
COMPOF	-	○	-	○	-	○
COMPON	-	○	-	○	-	○
CONTDCON	●	●	●	●	●	●
CONTPRON	●	●	●	●	●	●
CORROF	●	●	●	●	●	●
COS	●	●	●	●	●	●
COUPDEF	○	-	○	-	○	-
COUPDEL	○	-	○	-	○	-
COUPOF	○	-	○	-	○	-
COUPOFS	○	-	○	-	○	-
COUPON	○	-	○	-	○	-
COUPONC	○	-	○	-	○	-
COUPRES	○	-	○	-	○	-
CP	●	●	●	●	●	●
CPBC	●	●	●	●	●	●
CPDEF	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
CPDEL	●	●	●	●	●	●
CPFMOF	●	●	●	●	●	●
CPFMON	●	●	●	●	●	●
CPFMSON	●	●	●	●	●	●
CPFPOS	●	●	●	●	●	●
CPFRS	●	●	●	●	●	●
CPLA	●	●	●	●	●	●
CPLCTID	●	●	●	●	●	●
CPLDEF	●	●	●	●	●	●
CPLDEL	●	●	●	●	●	●
CPLDEN	●	●	●	●	●	●
CPLINSC	●	●	●	●	●	●
CPLINTR	●	●	●	●	●	●
CPLNUM	●	●	●	●	●	●
CPLOF	●	●	●	●	●	●
CPLON	●	●	●	●	●	●
CPLOUTSC	●	●	●	●	●	●
CPLOUTTR	●	●	●	●	●	●
CPLPOS	●	●	●	●	●	●
CPLSETVAL	●	●	●	●	●	●
CPMALARM	●	●	●	●	●	●
CPMBRAKE	●	●	●	●	●	●
CPMPRT	●	●	●	●	●	●
CPMRESET	●	●	●	●	●	●
CPMSTART	●	●	●	●	●	●
CPMVDI	●	●	●	●	●	●
CPOF	●	●	●	●	●	●
CPON	●	●	●	●	●	●
CPRECOF	●	●	●	●	●	●
CPRECON	●	●	●	●	●	●
CPRES	●	●	●	●	●	●
CPROT	●	●	●	●	●	●
CPROTDEF	●	●	●	●	●	●
CPSETTYPE	●	●	●	●	●	●
CPSYNCOP	●	●	●	●	●	●
CPSYNCOP2	●	●	●	●	●	●
CPSYNCOV	●	●	●	●	●	●
CPSYNFIP	●	●	●	●	●	●
CPSYNFIP2	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
CPSYNFIV	●	●	●	●	●	●
CR	●	●	●	●	●	●
CROT	●	●	●	●	●	●
CROTS	●	●	●	●	●	●
CRPL	●	●	●	●	●	●
CSCALE	●	●	●	●	●	●
CSPLINE	-	○	-	○	-	○
CT	●	●	●	●	●	●
CTAB	-	-	-	-	-	-
CTABDEF	-	-	-	-	-	-
CTABDEL	-	-	-	-	-	-
CTABEND	-	-	-	-	-	-
CTABEXISTS	-	-	-	-	-	-
CTABFNO	-	-	-	-	-	-
CTABFPOL	-	-	-	-	-	-
CTABFSEG	-	-	-	-	-	-
CTABID	-	-	-	-	-	-
CTABINV	-	-	-	-	-	-
CTABISLOCK	-	-	-	-	-	-
CTABLOCK	-	-	-	-	-	-
CTABMEMTYP	-	-	-	-	-	-
CTABMPOL	-	-	-	-	-	-
CTABMSEG	-	-	-	-	-	-
CTABNO	-	-	-	-	-	-
CTABNOMEM	-	-	-	-	-	-
CTABPERIOD	-	-	-	-	-	-
CTABPOL	-	-	-	-	-	-
CTABPOLID	-	-	-	-	-	-
CTABSEG	-	-	-	-	-	-
CTABSEGID	-	-	-	-	-	-
CTABSEV	-	-	-	-	-	-
CTABSSV	-	-	-	-	-	-
CTABTEP	-	-	-	-	-	-
CTABTEV	-	-	-	-	-	-
CTABTMAX	-	-	-	-	-	-
CTABTMIN	-	-	-	-	-	-
CTABTSP	-	-	-	-	-	-
CTABTSV	-	-	-	-	-	-
CTABUNLOCK	-	-	-	-	-	-

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
CTOL	-	○	-	○	-	○
CTRANS	●	●	●	●	●	●
CUT2D	●	●	●	●	●	●
CUT2DF	●	●	●	●	●	●
CUT3DC	-	-	-	-	-	-
CUT3DCC	-	-	-	-	-	-
CUT3DCCD	-	-	-	-	-	-
CUT3DF	-	-	-	-	-	-
CUT3DFF	-	-	-	-	-	-
CUT3DFS	-	-	-	-	-	-
CUTCONOF	●	●	●	●	●	●
CUTCONON	●	●	●	●	●	●
CUTMOD	●	●	●	●	●	●
CYCLE60	●	●	●	●	●	●
CYCLE61	●	●	●	●	●	●
CYCLE62	●	●	●	●	●	●
CYCLE63	●	●	●	●	●	●
CYCLE64	●	●	●	●	●	●
CYCLE70	●	●	●	●	●	●
CYCLE72	●	●	●	●	●	●
CYCLE76	●	●	●	●	●	●
CYCLE77	●	●	●	●	●	●
CYCLE78	●	●	●	●	●	●
CYCLE79	●	●	●	●	●	●
CYCLE81	●	●	●	●	●	●
CYCLE82	●	●	●	●	●	●
CYCLE83	●	●	●	●	●	●
CYCLE84	●	●	●	●	●	●
CYCLE85	●	●	●	●	●	●
CYCLE86	●	●	●	●	●	●
CYCLE92	●	●	●	●	●	●
CYCLE95	●	●	●	●	●	●
CYCLE98	●	●	●	●	●	●
CYCLE99	●	●	●	●	●	●
CYCLE495	●	●	●	●	●	●
CYCLE751	-	-	-	-	-	-
CYCLE752	-	-	-	-	-	-
CYCLE753	-	-	-	-	-	-
CYCLE754	-	-	-	-	-	-

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
CYCLE755	-	-	-	-	-	-
CYCLE756	-	-	-	-	-	-
CYCLE757	-	-	-	-	-	-
CYCLE758	-	-	-	-	-	-
CYCLE759	-	-	-	-	-	-
CYCLE800	●	●	●	●	●	●
CYCLE801	●	●	●	●	●	●
CYCLE802	●	●	●	●	●	●
CYCLE830	●	●	●	●	●	●
CYCLE832	●	●	●	●	●	●
CYCLE840	●	●	●	●	●	●
CYCLE899	●	●	●	●	●	●
CYCLE930	●	●	●	●	●	●
CYCLE940	●	●	●	●	●	●
CYCLE951	●	●	●	●	●	●
CYCLE952	●	●	●	●	●	●
CYCLE4071	●	●	●	●	●	●
CYCLE4072	●	●	●	●	●	●
CYCLE4073	●	●	●	●	●	●
CYCLE4074	●	●	●	●	●	●
CYCLE4075	●	●	●	●	●	●
CYCLE4077	●	●	●	●	●	●
CYCLE4078	●	●	●	●	●	●
CYCLE4079	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
D	●	●	●	●	●	●
D0	●	●	●	●	●	●
DAC	●	●	●	●	●	●
DC	●	●	●	●	●	●
DCI	●	●	●	●	●	●
DCM	●	●	●	●	●	●
DCU	●	●	●	●	●	●
DEF	●	●	●	●	●	●
DEFINE	●	●	●	●	●	●
DEFAULT	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
DELAYFSTON	●	●	●	●	●	●
DELAYFSTOF	●	●	●	●	●	●
DELDL	●	●	●	●	●	●
DELDTG	●	●	●	●	●	●
DELETE	●	●	●	●	●	●
DELMLOWNER	●	●	●	●	●	●
DELMLRES	●	●	●	●	●	●
DELMT	●	●	●	●	●	●
DELOBJ	-	-	-	-	-	-
DELT	●	●	●	●	●	●
DELTC	●	●	●	●	●	●
DELTOOLENV	●	●	●	●	●	●
DIACYCOFA	●	●	●	●	●	●
DIAM90	●	●	●	●	●	●
DIAM90A	●	●	●	●	●	●
DIAMCHAN	●	●	●	●	●	●
DIAMCHANA	●	●	●	●	●	●
DIAMCYCOF	●	●	●	●	●	●
DIAMOF	●	●	●	●	●	●
DIAMOFA	●	●	●	●	●	●
DIAMON	●	●	●	●	●	●
DIAMONA	●	●	●	●	●	●
DIC	●	●	●	●	●	●
DILF	●	●	●	●	●	●
DISABLE	●	●	●	●	●	●
DISC	●	●	●	●	●	●
DISCL	●	●	●	●	●	●
DISPLOF	●	●	●	●	●	●
DISPLON	●	●	●	●	●	●
DISPR	●	●	●	●	●	●
DISR	●	●	●	●	●	●
DISRP	●	●	●	●	●	●
DITE	●	●	●	●	●	●
DITS	●	●	●	●	●	●
DIV	●	●	●	●	●	●
DL	-	-	-	-	-	-
DO	●	●	●	●	●	●
DRFOF	●	●	●	●	●	●
DRIVE	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
DRIVEA	●	●	●	●	●	●
DYNFINISH	●	●	●	●	●	●
DYNNORM	●	●	●	●	●	●
DYNPOS	●	●	●	●	●	●
DYNROUGH	●	●	●	●	●	●
DYNSEMIFIN	●	●	●	●	●	●
DZERO	●	●	●	●	●	●
EAUTO	-	○	-	○	-	○
EGDEF	○	-	○	-	○	-
EGDEL	○	-	○	-	○	-
EGOFC	○	-	○	-	○	-
EGOFS	○	-	○	-	○	-
EGON	○	-	○	-	○	-
EGONSYN	○	-	○	-	○	-
EGONSYNE	○	-	○	-	○	-
ELSE	●	●	●	●	●	●
ENABLE	●	●	●	●	●	●
ENAT	-	○	-	○	-	○
ENDFOR	●	●	●	●	●	●
ENDIF	●	●	●	●	●	●
ENDLABEL	●	●	●	●	●	●
ENDLOOP	●	●	●	●	●	●
ENDPROC	●	●	●	●	●	●
ENDWHILE	●	●	●	●	●	●
ESRR	●	●	●	●	●	●
ESRS	●	●	●	●	●	●
ETAN	-	○	-	○	-	○
EVERY	●	●	●	●	●	●
EX	●	●	●	●	●	●
EXECSTRING	●	●	●	●	●	●
EXECTAB	●	●	●	●	●	●
EXECUTE	●	●	●	●	●	●
EXP	●	●	●	●	●	●
EXTCALL	●	●	●	●	●	●
EXTCLOSE	●	●	●	●	●	●
EXTERN	●	●	●	●	●	●
EXTOPEN	●	●	●	●	●	●
F	●	●	●	●	●	●
FA	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
FAD	●	●	●	●	●	●
FALSE	●	●	●	●	●	●
FB	●	●	●	●	●	●
FCTDEF	-	-	-	-	-	-
FCUB	●	●	●	●	●	●
FD	●	●	●	●	●	●
FDA	●	●	●	●	●	●
FENDNORM	●	●	●	●	●	●
FFWOF	●	●	●	●	●	●
FFWON	●	●	●	●	●	●
FGREF	●	●	●	●	●	●
FGROUP	●	●	●	●	●	●
FI	●	●	●	●	●	●
FIFOCTRL	●	●	●	●	●	●
FILEDATE	●	●	●	●	●	●
FILEINFO	●	●	●	●	●	●
FILESIZE	●	●	●	●	●	●
FILESTAT	●	●	●	●	●	●
FILETIME	●	●	●	●	●	●
FINEA	●	●	●	●	●	●
FL	●	●	●	●	●	●
FLIN	●	●	●	●	●	●
FMA	-	-	-	-	-	-
FNORM	●	●	●	●	●	●
FOCOF	○	-	○	-	○	-
FOCON	○	-	○	-	○	-
FOR	●	●	●	●	●	●
FP	●	●	●	●	●	●
FPO	-	-	-	-	-	-
FPR	●	●	●	●	●	●
FPRAOF	●	●	●	●	●	●
FPRAON	●	●	●	●	●	●
FRAME	●	●	●	●	●	●
FRC	●	●	●	●	●	●
FRCM	●	●	●	●	●	●
FROM	●	●	●	●	●	●
FTOC	●	●	●	●	●	●
FTOCOF	●	●	●	●	●	●
FTOCON	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
FXS	●	●	●	●	●	●
FXST	●	●	●	●	●	●
FXSW	●	●	●	●	●	●
FZ	●	●	●	●	●	●
G0	●	●	●	●	●	●
G1	●	●	●	●	●	●
G2	●	●	●	●	●	●
G3	●	●	●	●	●	●
G4	●	●	●	●	●	●
G5	●	●	●	●	●	●
G7	●	●	●	●	●	●
G9	●	●	●	●	●	●
G17	●	●	●	●	●	●
G18	●	●	●	●	●	●
G19	●	●	●	●	●	●
G25	●	●	●	●	●	●
G26	●	●	●	●	●	●
G33	●	●	●	●	●	●
G34	●	●	●	●	●	●
G35	●	●	●	●	●	●
G40	●	●	●	●	●	●
G41	●	●	●	●	●	●
G42	●	●	●	●	●	●
G53	●	●	●	●	●	●
G54	●	●	●	●	●	●
G55	●	●	●	●	●	●
G56	●	●	●	●	●	●
G57	●	●	●	●	●	●
G58	●	●	●	●	●	●
G59	●	●	●	●	●	●
G60	●	●	●	●	●	●
G62	●	●	●	●	●	●
G63	●	●	●	●	●	●
G64	●	●	●	●	●	●
G70	●	●	●	●	●	●
G71	●	●	●	●	●	●
G74	●	●	●	●	●	●
G75	●	●	●	●	●	●
G90	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
G91	●	●	●	●	●	●
G93	●	●	●	●	●	●
G94	●	●	●	●	●	●
G95	●	●	●	●	●	●
G96	●	●	●	●	●	●
G97	●	●	●	●	●	●
G110	●	●	●	●	●	●
G111	●	●	●	●	●	●
G112	●	●	●	●	●	●
G140	●	●	●	●	●	●
G141	●	●	●	●	●	●
G142	●	●	●	●	●	●
G143	●	●	●	●	●	●
G147	●	●	●	●	●	●
G148	●	●	●	●	●	●
G153	●	●	●	●	●	●
G247	●	●	●	●	●	●
G248	●	●	●	●	●	●
G290	●	●	●	●	●	●
G291	●	●	●	●	●	●
G331	●	●	●	●	●	●
G332	●	●	●	●	●	●
G335	●	●	●	●	●	●
G336	●	●	●	●	●	●
G340	●	●	●	●	●	●
G341	●	●	●	●	●	●
G347	●	●	●	●	●	●
G348	●	●	●	●	●	●
G450	●	●	●	●	●	●
G451	●	●	●	●	●	●
G460	●	●	●	●	●	●
G461	●	●	●	●	●	●
G462	●	●	●	●	●	●
G500	●	●	●	●	●	●
G505 ... G599	●	●	●	●	●	●
G601	●	●	●	●	●	●
G602	●	●	●	●	●	●
G603	●	●	●	●	●	●
G621	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
G641	●	●	●	●	●	●
G642	●	●	●	●	●	●
G643	●	●	●	●	●	●
G644	●	●	●	●	●	●
G645	●	●	●	●	●	●
G700	●	●	●	●	●	●
G710	●	●	●	●	●	●
G810 ... G819	-	-	-	-	-	-
G820 ... G829	-	-	-	-	-	-
G931	●	●	●	●	●	●
G942	●	●	●	●	●	●
G952	●	●	●	●	●	●
G961	●	●	●	●	●	●
G962	●	●	●	●	●	●
G971	●	●	●	●	●	●
G972	●	●	●	●	●	●
G973	●	●	●	●	●	●
GEOAX	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
GET	●	●	●	●	●	●
GETACTT	●	●	●	●	●	●
GETACTTD	●	●	●	●	●	●
GETD	●	●	●	●	●	●
GETDNO	●	●	●	●	●	●
GETEXET	●	●	●	●	●	●
GETFREELOC	●	●	●	●	●	●
GETSELT	●	●	●	●	●	●
GETT	●	●	●	●	●	●
GETTCOR	●	●	●	●	●	●
GETTENV	●	●	●	●	●	●
GETVARAP	●	●	●	●	●	●
GETVARDFT	●	●	●	●	●	●
GETVARLIM	●	●	●	●	●	●
GETVARPHU	●	●	●	●	●	●
GETVARTYP	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
GOTO	●	●	●	●	●	●
GOTOB	●	●	●	●	●	●
GOTOC	●	●	●	●	●	●
GOTOF	●	●	●	●	●	●
GOTOS	●	●	●	●	●	●
GP	●	●	●	●	●	●
GWPSOF	●	●	●	●	●	●
GWPSON	●	●	●	●	●	●
H...	●	●	●	●	●	●
HOLES1	●	●	●	●	●	●
HOLES2	●	●	●	●	●	●
I	●	●	●	●	●	●
I1	●	●	●	●	●	●
IC	●	●	●	●	●	●
ICYCOF	●	●	●	●	●	●
ICYCON	●	●	●	●	●	●
ID	●	●	●	●	●	●
IDS	●	●	●	●	●	●
IF	●	●	●	●	●	●
INDEX	●	●	●	●	●	●
INIPO	●	●	●	●	●	●
INIRE	●	●	●	●	●	●
INICF	●	●	●	●	●	●
INIT	-	-	-	-	-	-
INITIAL	●	●	●	●	●	●
INT	●	●	●	●	●	●
INTERSEC	●	●	●	●	●	●
INVCCW	-	-	-	-	-	-
INVCW	-	-	-	-	-	-
INVFRAME	●	●	●	●	●	●
IP	●	●	●	●	●	●
IPOBRKA	●	●	●	●	●	●
IPOENDA	●	●	●	●	●	●
IPTRLOCK	●	●	●	●	●	●
IPTRUNLOCK	●	●	●	●	●	●
IR	●	●	●	●	●	●
ISAXIS	●	●	●	●	●	●
ISD	-	-	-	-	-	-
ISFILE	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
ISNUMBER	●	●	●	●	●	●
ISOCALL	●	●	●	●	●	●
ISVAR	●	●	●	●	●	●
J	●	●	●	●	●	●
J1	●	●	●	●	●	●
JERKA	●	●	●	●	●	●
JERKLIM	●	●	●	●	●	●
JERKLIMA	●	●	●	●	●	●
JR	●	●	●	●	●	●
K	●	●	●	●	●	●
K1	●	●	●	●	●	●
KONT	●	●	●	●	●	●
KONTC	●	●	●	●	●	●
KONTT	●	●	●	●	●	●
KR	●	●	●	●	●	●
L	●	●	●	●	●	●
LEAD Tool orientation Orientation polynomial	 - -	 - -	 - -	 - -	 - -	 - -
LEADOF	-	-	-	-	-	-
LEADON	-	-	-	-	-	-
LENTOAX	●	●	●	●	●	●
LFOF	●	●	●	●	●	●
LFON	●	●	●	●	●	●
LFPOS	●	●	●	●	●	●
LFTXT	●	●	●	●	●	●
LFWP	●	●	●	●	●	●
LIFTFAST	●	●	●	●	●	●
LIMS	●	●	●	●	●	●
LLI	●	●	●	●	●	●
LN	●	●	●	●	●	●
LOCK	●	●	●	●	●	●
LONGHOLE	-	-	-	-	-	-
LOOP	●	●	●	●	●	●
M0	●	●	●	●	●	●
M1	●	●	●	●	●	●
M2	●	●	●	●	●	●
M3	●	●	●	●	●	●
M4	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
M5	●	●	●	●	●	●
M6	●	●	●	●	●	●
M17	●	●	●	●	●	●
M19	●	●	●	●	●	●
M30	●	●	●	●	●	●
M40	●	●	●	●	●	●
M41 ... M45	●	●	●	●	●	●
M70	●	●	●	●	●	●
MASLDEF	●	●	●	●	●	●
MASLDEL	●	●	●	●	●	●
MASLOF	●	●	●	●	●	●
MASLOFS	●	●	●	●	●	●
MASLON	●	●	●	●	●	●
MATCH	●	●	●	●	●	●
MAXVAL	●	●	●	●	●	●
MCALL	●	●	●	●	●	●
MEAC	-	-	-	-	-	-
MEAFRAME	●	●	●	●	●	●
MEAS	●	●	●	●	●	●
MEASA	-	-	-	-	-	-
MEASURE	●	●	●	●	●	●
MEAW	●	●	●	●	●	●
MEAWA	-	-	-	-	-	-
MI	●	●	●	●	●	●
MINDEX	●	●	●	●	●	●
MINVAL	●	●	●	●	●	●
MIRROR	●	●	●	●	●	●
MMC	●	●	●	●	●	●
MOD	●	●	●	●	●	●
MODAXVAL	●	●	●	●	●	●
MOV	●	●	●	●	●	●
MOVT	●	●	●	●	●	●
MSG	●	●	●	●	●	●
MVTOOL	●	●	●	●	●	●
N	●	●	●	●	●	●
NAMETOINT	-	-	-	-	-	-
NCK	●	●	●	●	●	●
NEWCONF	●	●	●	●	●	●
NEWMT	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
NEWT	●	●	●	●	●	●
NORM	●	●	●	●	●	●
NOT	●	●	●	●	●	●
NPROT	●	●	●	●	●	●
NPROTDEF	●	●	●	●	●	●
NUMBER	●	●	●	●	●	●
OEMIPO1	-	-	-	-	-	-
OEMIPO2	-	-	-	-	-	-
OF	●	●	●	●	●	●
OFFN	●	●	●	●	●	●
OMA1	-	-	-	-	-	-
OMA2	-	-	-	-	-	-
OMA3	-	-	-	-	-	-
OMA4	-	-	-	-	-	-
OMA5	-	-	-	-	-	-
OR	●	●	●	●	●	●
ORIAXES	-	-	-	-	-	-
ORIAXPOS	-	-	-	-	-	-
ORIC	-	-	-	-	-	-
ORICONCCW	-	-	-	-	-	-
ORICONCW	-	-	-	-	-	-
ORICONIO	-	-	-	-	-	-
ORICONTO	-	-	-	-	-	-
ORICURVE	-	-	-	-	-	-
ORID	-	-	-	-	-	-
ORIEULER	-	-	-	-	-	-
ORIMKS	-	-	-	-	-	-
ORIPATH	-	-	-	-	-	-
ORIPATHS	-	-	-	-	-	-
ORIPLANE	-	-	-	-	-	-
ORIRESET	-	-	-	-	-	-
ORIROTA	-	-	-	-	-	-
ORIROTC	-	-	-	-	-	-
ORIROTR	-	-	-	-	-	-
ORIROTT	-	-	-	-	-	-
ORIRPY	-	-	-	-	-	-
ORIRPY2	-	-	-	-	-	-
ORIS	-	-	-	-	-	-
ORISOF	-	-	-	-	-	-

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
ORISON	-	-	-	-	-	-
ORIVECT	-	-	-	-	-	-
ORIVIRT1	-	-	-	-	-	-
ORIVIRT2	-	-	-	-	-	-
ORIWKS	-	-	-	-	-	-
OS	-	-	-	-	-	-
OSB	-	-	-	-	-	-
OSC	-	-	-	-	-	-
OSCILL	-	-	-	-	-	-
OSCTRL	-	-	-	-	-	-
OSD	-	-	-	-	-	-
OSE	-	-	-	-	-	-
OSNSC	-	-	-	-	-	-
OSOF	-	-	-	-	-	-
OSP1	-	-	-	-	-	-
OSP2	-	-	-	-	-	-
OSS	-	-	-	-	-	-
OSSE	-	-	-	-	-	-
OST	-	-	-	-	-	-
OST1	-	-	-	-	-	-
OST2	-	-	-	-	-	-
OTOL	-	●	-	●	-	●
OVR	●	●	●	●	●	●
OVRA	●	●	●	●	●	●
OVRRAP	●	●	●	●	●	●
P	●	●	●	●	●	●
PAROT	●	●	●	●	●	●
PAROTOF	●	●	●	●	●	●
PCALL	●	●	●	●	●	●
PDELAYOF	-	-	-	-	-	-
PDELAYON	-	-	-	-	-	-
PHI	-	-	-	-	-	-
PHU	●	●	●	●	●	●
PL	- -	○ -	- -	○ -	- -	○ -
PM	●	●	●	●	●	●
PO	-	-	-	-	-	-
POCKET3	●	●	●	●	●	●
POCKET4	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
POLF	●	●	●	●	●	●
POLFA	●	●	●	●	●	●
POLFMASK	●	●	●	●	●	●
POLFMLIN	●	●	●	●	●	●
POLY	-	-	-	-	-	-
POLYPATH	-	-	-	-	-	-
PON	-	-	-	-	-	-
PONS	-	-	-	-	-	-
POS	●	●	●	●	●	●
POSA	●	●	●	●	●	●
POSM	●	●	●	●	●	●
POSMT	●	●	●	●	●	●
POSP	●	●	●	●	●	●
POSRANGE	●	●	●	●	●	●
POT	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
PR	●	●	●	●	●	●
PREPRO	●	●	●	●	●	●
PRESETON	●	●	●	●	●	●
PRESETONS	●	●	●	●	●	●
PRIO	●	●	●	●	●	●
PRLOC	●	●	●	●	●	●
PROC	●	●	●	●	●	●
PROTA	-	-	-	-	-	-
PROTD	-	-	-	-	-	-
PROTS	-	-	-	-	-	-
PSI	-	-	-	-	-	-
PTP	●	●	●	●	●	●
PTPG0	●	●	●	●	●	●
PUNCHACC	-	-	-	-	-	-
PUTFTOC	●	●	●	●	●	●
PUTFTOCF	●	●	●	●	●	●
PW	-	○	-	○	-	○
QU	●	●	●	●	●	●
R...	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
RAC	●	●	●	●	●	●
RDISABLE	●	●	●	●	●	●
READ	●	●	●	●	●	●
REAL	●	●	●	●	●	●
REDEF	●	●	●	●	●	●
RELEASE	●	●	●	●	●	●
REP	●	●	●	●	●	●
REPEAT	●	●	●	●	●	●
REPEATB	●	●	●	●	●	●
REPOSA	●	●	●	●	●	●
REPOSH	●	●	●	●	●	●
REPOSHA	●	●	●	●	●	●
REPOSL	●	●	●	●	●	●
REPOSQ	●	●	●	●	●	●
REPOSQA	●	●	●	●	●	●
RESET	●	●	●	●	●	●
RESETMON	●	●	●	●	●	●
RET	●	●	●	●	●	●
RETB	●	●	●	●	●	●
RIC	●	●	●	●	●	●
RINDEX	●	●	●	●	●	●
RMB	●	●	●	●	●	●
RME	●	●	●	●	●	●
RMI	●	●	●	●	●	●
RMN	●	●	●	●	●	●
RND	●	●	●	●	●	●
RNDM	●	●	●	●	●	●
ROT	●	●	●	●	●	●
ROTS	●	●	●	●	●	●
ROUND	●	●	●	●	●	●
ROUNDUP	●	●	●	●	●	●
RP	●	●	●	●	●	●
RPL	●	●	●	●	●	●
RT	●	●	●	●	●	●
RTLIOF	●	●	●	●	●	●
RTLION	●	●	●	●	●	●
S	●	●	●	●	●	●
SAVE	●	●	●	●	●	●
SBLOF	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
SBLON	●	●	●	●	●	●
SC	●	●	●	●	●	●
SCALE	●	●	●	●	●	●
SCC	●	●	●	●	●	●
SCPARA	●	●	●	●	●	●
SD	-	○	-	○	-	○
SET	●	●	●	●	●	●
SETAL	●	●	●	●	●	●
SETDNO	●	●	●	●	●	●
SETINT	●	●	●	●	●	●
SETM	-	-	-	-	-	-
SETMS	●	●	●	●	●	●
SETMS(n)	●	●	●	●	●	●
SETMTH	●	●	●	●	●	●
SETPIECE	●	●	●	●	●	●
SETTA	●	●	●	●	●	●
SETTCOR	●	●	●	●	●	●
SETTIA	●	●	●	●	●	●
SF	●	●	●	●	●	●
SIN	●	●	●	●	●	●
SIRELAY	-	-	-	-	-	-
SIRELIN	-	-	-	-	-	-
SIRELOUT	-	-	-	-	-	-
SIRELTIME	-	-	-	-	-	-
SLOT1	●	●	●	●	●	●
SLOT2	●	●	●	●	●	●
SOFT	●	●	●	●	●	●
SOFTA	●	●	●	●	●	●
SON	-	-	-	-	-	-
SONS	-	-	-	-	-	-
SPATH	●	●	●	●	●	●
SPCOF	●	●	●	●	●	●
SPCON	●	●	●	●	●	●
SPI	●	●	●	●	●	●
SPIF1	-	-	-	-	-	-
SPIF2	-	-	-	-	-	-
SPLINEPATH	-	○	-	○	-	○
SPN	-	-	-	-	-	-
SPOF	-	-	-	-	-	-

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
SPOS	●	●	●	●	●	●
SPOSA	●	●	●	●	●	●
SPP	-	-	-	-	-	-
SPRINT	●	●	●	●	●	●
SQRT	●	●	●	●	●	●
SR	-	-	-	-	-	-
SRA	-	-	-	-	-	-
ST	-	-	-	-	-	-
STA	-	-	-	-	-	-
START	-	-	-	-	-	-
STARTFIFO	●	●	●	●	●	●
STAT	●	●	●	●	●	●
STOLF	-	-	-	-	-	-
STOPFIFO	●	●	●	●	●	●
STOPRE	●	●	●	●	●	●
STOPREOF	●	●	●	●	●	●
STRING	●	●	●	●	●	●
STRINGFELD	●	●	●	●	●	●
STRINGIS	●	●	●	●	●	●
STRINGVAR	-	-	-	-	-	-
STRLEN	●	●	●	●	●	●
SUBSTR	●	●	●	●	●	●
SUPA	●	●	●	●	●	●
SVC	●	●	●	●	●	●
SYNFCT	●	●	●	●	●	●
SYNR	●	●	●	●	●	●
SYNRW	●	●	●	●	●	●
SYNW	●	●	●	●	●	●
T	●	●	●	●	●	●
TAN	●	●	●	●	●	●
TANG	-	-	-	-	-	-
TANGDEL	-	-	-	-	-	-
TANGOF	-	-	-	-	-	-
TANGON	-	-	-	-	-	-
TCA (828D: _TCA)	●	●	●	●	●	●
TCARR	-	●	-	●	-	●
TCI	●	●	●	●	●	●
TCOABS	-	●	-	●	-	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
TCOFR	-	●	-	●	-	●
TCOFRX	-	●	-	●	-	●
TCOFRY	-	●	-	●	-	●
TCOFRZ	-	●	-	●	-	●
THETA	-	-	-	-	-	-
TILT	-	-	-	-	-	-
TLIFT	-	-	-	-	-	-
TML	●	●	●	●	●	●
TMOF	●	●	●	●	●	●
TMON	●	●	●	●	●	●
TO	●	●	●	●	●	●
TOFF	●	●	●	●	●	●
TOFFL	●	●	●	●	●	●
TOFFOF	●	●	●	●	●	●
TOFFON	●	●	●	●	●	●
TOFFR	●	●	●	●	●	●
TOFRAME	●	●	●	●	●	●
TOFRAMEX	●	●	●	●	●	●
TOFRAMEY	●	●	●	●	●	●
TOFRAMEZ	●	●	●	●	●	●
TOLOWER	●	●	●	●	●	●
TOOLENV	●	●	●	●	●	●
TOOLGNT	●	●	●	●	●	●
TOOLGT	●	●	●	●	●	●
TOROT	●	●	●	●	●	●
TOROTOF	●	●	●	●	●	●
TOROTX	●	●	●	●	●	●
TOROTY	●	●	●	●	●	●
TOROTZ	●	●	●	●	●	●
TOUPPER	●	●	●	●	●	●
TOWBCS	-	●	-	●	-	●
TOWKCS	-	●	-	●	-	●
TOWMCS	-	●	-	●	-	●
TOWSTD	-	●	-	●	-	●
TOWTCS	-	●	-	●	-	●
TOWWCS	-	●	-	●	-	●
TR	●	●	●	●	●	●
TRAANG	-	-	-	-	○	-
TRACON	-	-	-	-	○	-

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
TRACYL	○	○	○	○	○	○
TRAFOOF	●	●	●	●	●	●
TRAILOF	●	●	●	●	●	●
TRAILON	●	●	●	●	●	●
TRANS	●	●	●	●	●	●
TRANSMIT	○	○	○	○	○	○
TRAORI	-	-	-	-	-	-
TRUE	●	●	●	●	●	●
TRUNC	●	●	●	●	●	●
TU	●	●	●	●	●	●
TURN	●	●	●	●	●	●
ULI	●	●	●	●	●	●
UNLOCK	●	●	●	●	●	●
UNTIL	●	●	●	●	●	●
UPATH	●	●	●	●	●	●
VAR	●	●	●	●	●	●
VELOLIM	●	●	●	●	●	●
VELOLIMA	●	●	●	●	●	●
WAITC	-	-	-	-	○	-
WAITE	-	-	-	-	-	-
WAITENC	-	-	-	-	-	-
WAITM	-	-	-	-	-	-
WAITMC	-	-	-	-	-	-
WAITP	●	●	●	●	●	●
WAITS	●	●	●	●	●	●
WALCS0	●	●	●	●	●	●
WALCS1	●	●	●	●	●	●
WALCS2	●	●	●	●	●	●
WALCS3	●	●	●	●	●	●
WALCS4	●	●	●	●	●	●
WALCS5	●	●	●	●	●	●
WALCS6	●	●	●	●	●	●
WALCS7	●	●	●	●	●	●
WALCS8	●	●	●	●	●	●
WALCS9	●	●	●	●	●	●
WALCS10	●	●	●	●	●	●
WALIMOF	●	●	●	●	●	●
WALIMON	●	●	●	●	●	●
WHEN	●	●	●	●	●	●

Operation	828D control version					
● Standard	PPU240.3 / 241.3		PPU260.3 / 261.3		PPU280.3 / 281.3	
○ Option - not available	Turning	Milling	Turning	Milling	Turning	Milling
WHENEVER	●	●	●	●	●	●
WHILE	●	●	●	●	●	●
WRITE	●	●	●	●	●	●
WRTPR	●	●	●	●	●	●
X	●	●	●	●	●	●
XOR	●	●	●	●	●	●
Y	●	●	●	●	●	●
Z	●	●	●	●	●	●

20.3 Currently set language in the HMI

The table below lists all of the languages available at the user interface.

The currently set language can be queried in the part program and in the synchronized actions using the following system variable:

$AN_LANGUAGE_ON_HMI = <value>

<value>	Language	Language code
1	German (Germany)	GER
2	French	FRA
3	English (Great Britain)	ENG
4	Spanish	ESP
6	Italian	ITA
7	Dutch	NLD
8	Simplified Chinese	CHS
9	Swedish	SVE
18	Hungarian	HUN
19	Finnish	FIN
28	Czech	CSY
50	Portuguese (Brazil)	PTB
53	Polish	PLK
55	Danish	DAN
57	Russian	RUS
68	Slovakian	SKY
72	Rumanian	ROM
80	Traditional Chinese	CHT
85	Korean	KOR
87	Japanese	JPN
89	Turkish	TRK

Note

$AN_LANGUAGE_ON_HMI is updated:

- after the system boots.
- after NCK and/or PLC reset.
- after switching over to another NCK within the scope of M2N.
- after changing over the language on the HMI.

Appendix

A

A.1 List of abbreviations

A	
O	Output
ADI4	(Analog drive interface for 4 axes)
AC	Adaptive Control
ALM	Active Line Module
ARM	Rotating induction motor
AS	Automation system
ASCII	American Standard Code for Information Interchange: American coding standard for the exchange of information
ASIC	Application-Specific Integrated Circuit: User switching circuit
ASUB	Asynchronous subprogram
AUXFU	Auxiliary function: Auxiliary function
STL	Statement List
UP	User Program

B	
OP	Operating Mode
BAG	Mode group
BCD	Binary Coded Decimals: Decimal numbers encoded in binary code
BERO	Contact-less proximity switch
BI	Binector Input
BICO	Binector Connector
BIN	BINary files: Binary files
BIOS	Basic Input Output System
BCS	Basic Coordinate System
BO	Binector Output
OPI	Operator Panel Interface

C	
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CC	Compile Cycle: Compile cycles
CI	Connector Input
CF Card	Compact Flash Card
CNC	Computerized Numerical Control: Computer-Supported Numerical Control

C	
CO	Connector Output
CoL	Certificate of License
COM	Communication
CPA	Compiler Projecting Data: Configuring data of the compiler
CRT	Cathode Ray Tube: picture tube
CSB	Central Service Board: PLC module
CU	Control Unit
CP	Communication Processor
CPU	Central Processing Unit: Central processing unit
CR	Carriage Return
CTS	Clear To Send: Ready to send signal for serial data interfaces
CUTCOM	Cutter radius Compensation: Tool radius compensation

D	
DAC	Digital-to-Analog Converter
DB	Data Block (PLC)
DBB	Data Block Byte (PLC)
DBD	Data Block Double word (PLC)
DBW	Data Block Word (PLC)
DBX	Data block bit (PLC)
DDE	Dynamic Data Exchange
DDS	Drive Data Set: Drive data set
DIN	Deutsche Industrie Norm
DIO	Data Input/Output: Data transfer display
DIR	Directory: Directory
DLL	Dynamic Link Library
DO	Drive Object
DPM	Dual Port Memory
DPR	Dual Port RAM
DRAM	Dynamic memory (non-buffered)
DRF	Differential Resolver Function: Differential revolver function (handwheel)
DRIVE-CLiQ	Drive Component Link with IQ
DRY	Dry Run: Dry run feedrate
DSB	Decoding Single Block: Decoding single block
DSC	Dynamic Servo Control / Dynamic Stiffness Control
DW	Data Word
DWORD	Double Word (currently 32 bits)

E	
I	Input
EES	Execution from External Storage

E	
I/O	Input/Output
ENC	Encoder: Actual value encoder
EFP	Compact I/O module (PLC I/O module)
ESD	Electrostatic Sensitive Devices
EMC	ElectroMagnetic Compatibility
EN	European standard
ENC	Encoder: Actual value encoder
EnDat	Encoder interface
EPROM	Erasable Programmable Read Only Memory: Erasable, electrically programmable read-only memory
ePS Network Services	Services for Internet-based remote machine maintenance
EQN	Designation for an absolute encoder with 2048 sine signals per revolution
ES	Engineering System
ESR	Extended Stop and Retract
ETC	ETC key ">"; softkey bar extension in the same menu

F	
FB	Function Block (PLC)
FC	Function Call: Function Block (PLC)
FEPROM	Flash EPROM: Read and write memory
FIFO	First In First Out: Memory that works without address specification and whose data is read in the same order in which they was stored
FIPO	Fine interpolator
FPU	Floating Point Unit: Floating Point Unit
CRC	Cutter Radius Compensation
FST	Feed Stop: Feedrate stop
FBD	Function Block Diagram (PLC programming method)
FW	Firmware

G	
GC	Global Control (PROFIBUS: Broadcast telegram)
GDIR	Global part program memory
GEO	Geometry, e.g. geometry axis
GIA	Gear Interpolation dAta: Gear interpolation data
GND	Signal Ground
GP	Basic program (PLC)
GS	Gear Stage
GSD	Device master file for describing a PROFIBUS slave
GSDML	Generic Station Description Markup Language: XML-based description language for creating a GSD file
GUD	Global User Data: Global user data

H	
HEX	Abbreviation for hexadecimal number
AuxF	Auxiliary function
HLA	Hydraulic linear drive
HMI	Human Machine Interface: SINUMERIK user interface
MSD	Main Spindle Drive
HW	Hardware

I	
IBN	Commissioning
ICA	Interpolatory compensation
IM	Interface Module: Interconnection module
IMR	Interface Module Receive: Interface module for receiving data
IMS	Interface Module Send: Interface module for sending data
INC	Increment: Increment
INI	Initializing Data: Initializing data
IPO	Interpolator
ISA	Industry Standard Architecture
ISO	International Standardization Organization

J	
JOG	Jogging: Setup mode

K	
K_V	Gain factor of control loop
K_p	Proportional gain
$K_Ü$	Transformation ratio
LAD	Ladder Diagram (PLC programming method)

L	
LAI	Logic Machine Axis Image: Logical machine axes image
LAN	Local Area Network
LCD	Liquid Crystal Display: Liquid crystal display
LED	Light Emitting Diode: Light-emitting diode
LF	Line Feed
PMS	Position Measuring System
LR	Position controller
LSB	Least Significant Bit: Least significant bit
LUD	Local User Data: User data (local)

M	
MAC	Media Access Control
MAIN	Main program: Main program (OB1, PLC)
MB	Megabyte
MCI	Motion Control Interface
MCIS	Motion Control Information System
MCP	Machine Control Panel: Machine control panel
MD	Machine Data
MDA	Manual Data Automatic: Manual input
MDS	Motor Data Set: Motor data set
MSGW	Message Word
MCS	Machine Coordinate System
MM	Motor Module
MPF	Main Program File: Main program (NC)
MCP	Machine control panel

N	
NC	Numerical Control: Numerical Control
NCK	Numerical Control Kernel: NC kernel with block preparation, traversing range, etc.
NCU	Numerical Control Unit: NCK hardware unit
NRK	Name for the operating system of the NCK
IS	Interface Signal
NURBS	Non-Uniform Rational B-Spline
WO	Work Offset
NX	Numerical Extension: Axis expansion board

O	
OB	Organization block in the PLC
OEM	Original Equipment Manufacturer
OP	Operator Panel: Operating equipment
OPI	Operator Panel Interface: Interface for connection to the operator panel
OPT	Options: Options
OLP	Optical Link Plug: Fiber optic bus connector
OSI	Open Systems Interconnection: Standard for computer communications

P	
PIQ	Process Image Output
PII	Process Image Input
PC	Personal Computer
PCIN	Name of the SW for data exchange with the control

P	
PCMCIA	Personal Computer Memory Card International Association: Plug-in memory card standardization
PCU	PC Unit: PC box (computer unit)
PG	Programming device
PKE	Parameter identification: Part of a PIV
PIV	Parameter identification: Value (parameterizing part of a PPO)
PLC	Programmable Logic Control: Adaptation control
PN	PROFINET
PNO	PROFIBUS user organization
PO	POWER ON
POU	Program Organization Unit
POS	Position/positioning
POSMO A	Positioning Motor Actuator: Positioning motor
POSMO CA	Positioning Motor Compact AC: Complete drive unit with integrated power and control module as well as positioning unit and program memory; AC infeed
POSMO CD	Positioning Motor Compact DC: Like CA but with DC infeed
POSMO SI	Positioning Motor Servo Integrated: Positioning motor, DC infeed
PPO	Parameter Process data Object: Cyclic data telegram for PROFIBUS DP transmission and "Variable speed drives" profile
PPU	Panel Processing Unit (central hardware for a panel-based CNC, e.g SINUMERIK 828D)
PROFIBUS	Process Field Bus: Serial data bus
PRT	Program Test
PSW	Program control word
PTP	Point-To-Point: Point-To-Point
PUD	Program global User Data: Program-global user variables
PZD	Process data: Process data part of a PPO

Q	
QEC	Quadrant Error Compensation

R	
RAM	Random Access Memory: Read/write memory
REF	REFerence point approach function
REPOS	REPOSition function
RISC	Reduced Instruction Set Computer: Type of processor with small instruction set and ability to process instructions at high speed
ROV	Rapid Override: Input correction
RP	R Parameter, arithmetic parameter, predefined user variable
RPA	R Parameter Active: Memory area on the NCK for R parameter numbers
RPY	Roll Pitch Yaw: Rotation type of a coordinate system
RTLI	Rapid Traverse Linear Interpolation: Linear interpolation during rapid traverse motion

R	
RTS	Request To Send: Control signal of serial data interfaces
RTCP	Real Time Control Protocol

S	
SA	Synchronized Action
SBC	Safe Brake Control: Safe Brake Control
SBL	Single Block: Single block
SBR	Subroutine: Subprogram (PLC)
SD	Setting Data
SDB	System Data Block
SEA	Setting Data Active: Identifier (file type) for setting data
SERUPRO	SEarch RUn by PROgram test: Search run by program test
SFB	System Function Block
SFC	System Function Call
SGE	Safety-related input
SGA	Safety-related output
SH	Safe standstill
SIM	Single in Line Module
SK	Softkey
SKP	Skip: Function for skipping a part program block
SLM	Synchronous Linear Motor
SM	Stepper Motor
SMC	Sensor Module Cabinet Mounted
SME	Sensor Module Externally Mounted
SMI	Sensor Module Integrated
SPF	Sub Routine File: Subprogram (NC)
PLC	Programmable Logic Controller
SRAM	Static RAM (non-volatile)
TNRC	Tool Nose Radius Compensation
SRM	Synchronous Rotary Motor
LEC	Leadscrew Error Compensation
SSI	Serial Synchronous Interface: Synchronous serial interface
SSL	Block search
STW	Control word
GWPS	Grinding Wheel Peripheral Speed
SW	Software
SYF	System Files: System files
SYNACT	SYNchronized ACTion: Synchronized Action

T	
TB	Terminal Board (SINAMICS)
TCP	Tool Center Point: Tool tip
TCP/IP	Transport Control Protocol / Internet Protocol
TCU	Thin Client Unit
TEA	Testing Data Active: Identifier for machine data
TIA	Totally Integrated Automation
TM	Terminal Module (SINAMICS)
TO	Tool Offset: Tool offset
TOA	Tool Offset Active: Identifier (file type) for tool offsets
TRANSMIT	Transform Milling Into Turning: Coordination transformation for milling operations on a lathe
TTL	Transistor-Transistor Logic (interface type)
TZ	Technology cycle

U	
UFR	User Frame: Work offset
SR	Subprogram
USB	Universal Serial Bus
UPS	Uninterruptible Power Supply

V	
VDI	Internal communication interface between NCK and PLC
VDI	Verein Deutscher Ingenieure [Association of German Engineers]
VDE	Verband Deutscher Elektrotechniker [Association of German Electrical Engineers]
VI	Voltage Input
VO	Voltage Output
FDD	Feed Drive

W	
SAR	Smooth Approach and Retraction
WCS	Workpiece Coordinate System
T	Tool
TLC	Tool Length Compensation
WOP	Workshop-Oriented Programming
WPD	Workpiece Directory: Workpiece directory
TRC	Tool Radius Compensation
T	Tool
TO	Tool Offset
TM	Tool Management
TC	Tool change

X	
XML	Extensible Markup Language

Z	
WOA	Work Offset Active: Identifier for work offsets
ZSW	Status word (of drive)

A.2 Documentation overview

General documentation



Sales brochure
– SINUMERIK 840D sl
– SINUMERIK 828D
– SINUMERIK 828D BASIC T



Catalog NC 62 SINUMERIK 840D sl type 1B



Catalog NC 82 SINUMERIK 828D BASIC T/BASIC M SINAMICS S120 combi motors 1FK7 and 1PH8



Catalog PM 21 SIMOTION, SINAMICS S120 and motors for production machines



Configuration Manual EMC design guidelines



System Manual Ctrl-Energy

User documentation



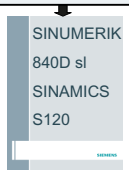
Operating Manual
– Universal
– Turning
– Milling



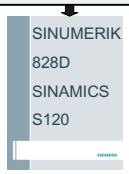
Programming Manual
– Basics
– Job planning
– Measuring cycles



Programming Manual
– ISO turning
– ISO milling



Diagnostics Manual



Diagnostics Manual

Manufacturer / service documentation



Manual
– NCU
– Operation components and networking



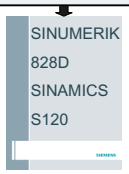
Manual
Commissioning Manual
Service Manual



Commissioning Manual
– CNC: NCK, PLC, drive
– Basic software and operating software



List Manual
– Machine data
– Interface signals
– Variables



List Manual
– Machine data
– Interface signals
– Parameters
– Variables



System Manual Guidelines for machine configuration

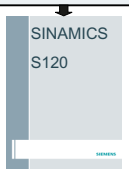
Manufacturer / service documentation



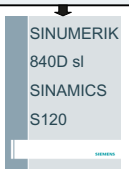
Function Manual
– Basic functions
– Extended functions
– Special functions
– Synchronized actions
– ISO dialects



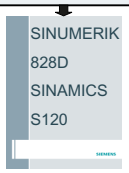
Function Manual
Tool management



Function Manual
Drive functions



Function Manual
Safety Integrated



Function Manual
Safety Integrated

Info / training



Training documents
– Easy milling with ShopMill
– Turning made easy with ShopTurn



Manual
Tool and mold making

Electronic documentation



DOConCD



My Documentation Manager



Industry Mall

Glossary

Absolute dimensions

A destination for an axis motion is defined by a dimension that refers to the origin of the currently valid coordinate system. See → Incremental dimension

Acceleration with jerk limitation

In order to optimize the acceleration response of the machine whilst simultaneously protecting the mechanical components, it is possible to switch over in the machining program between abrupt acceleration and continuous (jerk-free) acceleration.

Address

An address is the identifier for a certain operand or operand range, e.g. input, output, etc.

Alarms

All → messages and alarms are displayed on the operator panel in plain text with date and time and the corresponding symbol for the deletion criterion. Alarms and messages are displayed separately.

1. Alarms and messages in the part program:
   Alarms and messages can be displayed in plain text directly from the part program.
2. Alarms and messages from the PLC:
   Alarms and messages for the machine can be displayed in plain text from the PLC program. No additional function block packages are required for this purpose.

Archiving

Reading out of files and/or directories on an **external** memory device.

Asynchronous subprogram

Part program that can be started asynchronously to (independently of) the current program status using an interrupt signal (e.g. "Rapid NC input" signal).

Automatic

Operating mode of the controller (block sequence operation according to DIN): Operating mode for NC systems in which a → subprogram is selected and executed continuously.

Auxiliary functions

Auxiliary functions enable → part programs to transfer → parameters to the → PLC, which then trigger reactions defined by the machine manufacturer.

Axes

In accordance with their functional scope, the CNC axes are subdivided into:

- Axes: Interpolating path axes
- Auxiliary axes: Non-interpolating feed and positioning axes with an axis-specific feedrate. Auxiliary axes are not involved in actual machining, e.g. tool feeder, tool magazine.

Axis address

See → Axis name

Axis name

To ensure clear identification, all channel and → machine axes of the control system must be designated with unique names in the channel and control system. The → geometry axes are called X, Y, Z. The rotary axes rotating around the geometry axes → are called A, B, C.

Backlash compensation

Compensation for a mechanical machine backlash, e.g. backlash on reversal for ball screws. Backlash compensation can be entered separately for each axis.

Backup battery

The backup battery ensures that the → user program in the → CPU is stored so that it is safe from power failure and so that specified data areas and bit memory, timers and counters are stored retentively.

Basic axis

Axis whose setpoint or actual value position forms the basis of the calculation of a compensation value.

Basic Coordinate System

Cartesian coordinate system which is mapped by transformation onto the machine coordinate system.

The programmer uses axis names of the basic coordinate system in the → part program. The basic coordinate system exists parallel to the → machine coordinate system if no → transformation is active. The difference lies in the → axis names.

Baud rate

Rate of data transfer (bits/s).

Blank

Workpiece as it is before it is machined.

Block

"Block" is the term given to any files required for creating and processing programs.

Block search

For debugging purposes or following a program abort, the "Block search" function can be used to select any location in the part program at which the program is to be started or resumed.

Booting

Loading the system program after power ON.

C axis

Axis around which the tool spindle describes a controlled rotational and positioning motion.

C spline

The C spline is the most well-known and widely used spline. The transitions at the interpolation points are continuous, both tangentially and in terms of curvature. Polynomials of the 3rd degree are used.

Channel

A channel is characterized by the fact that it can process a → part program independently of other channels. A channel exclusively controls the axes and spindles assigned to it. Part program runs of different channels can be coordinated through → synchronization.

Circular interpolation

The → tool moves on a circle between specified points on the contour at a given feedrate, and the workpiece is thereby machined.

CNC

See → NC

Computerized Numerical Control: includes the components → NCK, → PLC, HMI, → COM.

CNC

See → NC

Computerized Numerical Control: includes the components → NCK, → PLC, HMI, → COM.

COM

Component of the NC for the implementation and coordination of communication.

Compensation axis

Axis with a setpoint or actual value modified by the compensation value

Compensation table

Table containing interpolation points. It provides the compensation values of the compensation axis for selected positions on the basic axis.

Compensation value

Difference between the axis position measured by the encoder and the desired, programmed axis position.

Continuous-path mode

The objective of continuous-path mode is to avoid substantial deceleration of the → path axes at the part program block boundaries and to change to the next block at as close to the same path velocity as possible.

Contour

Contour of the → workpiece

Contour monitoring

The following error is monitored within a definable tolerance band as a measure of contour accuracy. An unacceptably high following error can cause the drive to become overloaded, for example. In such cases, an alarm is output and the axes are stopped.

Coordinate system

See → Machine coordinate system, → Workpiece coordinate system

CPU

Central processing unit, see → PLC

CU

Transformation ratio

Curvature

The curvature k of a contour is the inverse of radius r of the nestling circle in a contour point (k = 1/r).

Cycles

Protected subprograms for execution of repetitive machining operations on the → workpiece.

Data block

1. Data unit of the → PLC that → HIGHSTEP programs can access.
2. Data unit of the → NC: Data blocks contain data definitions for global user data. This data can be initialized directly when it is defined.

Data word

Two-byte data unit within a → data block.

Diagnostics

1. Operating area of the control.
2. The control has a self-diagnostics program as well as test functions for servicing purposes: status, alarm, and service displays

Dimensions specification, metric and inches

Position and pitch values can be programmed in inches in the machining program. Irrespective of the programmable dimensions (G70/G71), the control is set to a basic system.

DRF

Differential Resolver Function: NC function which generates an incremental work offset in Automatic mode in conjunction with an electronic handwheel.

Drive

The drive is the unit of the CNC that performs the speed and torque control based on the settings of the NC.

Dynamic feedforward control

Inaccuracies in the → contour due to following errors can be practically eliminated using dynamic, acceleration-dependent feedforward control. This results in excellent machining accuracy even at high → path velocities. Feedforward control can be selected and deselected on an axis-specific basis via the → part program.

Editor

The editor makes it possible to create, edit, extend, join, and import programs / texts / program blocks.

Exact stop

When an exact stop statement is programmed, the position specified in a block is approached exactly and, if necessary, very slowly. To reduce the approach time, → exact stop limits are defined for rapid traverse and feed.

Exact stop limit

When all path axes reach their exact stop limits, the control responds as if it had reached its precise destination point. A block advance of the → part program occurs.

External work offset

Work offset specified by the → PLC.

Fast retraction from the contour

When an interrupt occurs, a motion can be initiated via the CNC machining program, enabling the tool to be quickly retracted from the workpiece contour that is currently being machined. The retraction angle and the distance retracted can also be parameterized. An interrupt routine can also be executed following the fast retraction.

Feed override

The programmed velocity is overriden by the current velocity setting made via the → machine control panel or from the → PLC (0 to 200%). The feedrate can also be corrected by a programmable percentage factor (1 to 200%) in the machining program.

Finished-part contour

Contour of the finished workpiece. See → Raw part.

Fixed machine point

Point that is uniquely defined by the machine tool, e.g. machine reference point.

Fixed-point approach

Machine tools can approach fixed points such as a tool change point, loading point, pallet change point, etc. in a defined way. The coordinates of these points are stored in the control. The control moves the relevant axes in → rapid traverse, whenever possible.

Frame

A frame is an arithmetic rule that transforms one Cartesian coordinate system into another Cartesian coordinate system. A frame contains the following components: → work offset, → rotation, → scaling, → mirroring.

Geometry

Description of a → workpiece in the → workpiece coordinate system.

Geometry axis

The geometry axes form the 2 or 3-dimensional → workpiece coordinate system in which, in → part programs, the geometry of the workpiece is programmed.

Ground

Ground is taken as the total of all linked inactive parts of a device which will not become live with a dangerous contact voltage even in the event of a malfunction.

Helical interpolation

The helical interpolation function is ideal for machining internal and external threads using form milling cutters and for milling lubrication grooves.

The helix comprises two motions:

- Circular motion in one plane
- A linear motion perpendicular to this plane

High-level CNC language

The high-level language is used to write NC programs, → synchronized actions, and → cycles. It provides: control structures → user-defined variables, → system variables, → macro programming.

High-speed digital inputs/outputs

The digital inputs can be used for example to start fast CNC program routines (interrupt routines). High-speed, program-driven switching functions can be initiated via the digital CNC outputs

HIGHSTEP

Summary of programming options for → PLCs of the AS300/AS400 system.

HW Config

SIMATIC S7 tool for the configuration and parameterization of hardware components within an S7 project

Identifier

In accordance with DIN 66025, words are supplemented using identifiers (names) for variables (arithmetic variables, system variables, user variables), subprograms, key words, and words with multiple address letters. These supplements have the same meaning as the words with respect to block format. Identifiers must be unique. It is not permissible to use the same identifier for different objects.

Inch measuring system

Measuring system which defines distances in inches and fractions of inches.

Inclined surface machining

Drilling and milling operations on workpiece surfaces that do not lie in the coordinate planes of the machine can be performed easily using the function "inclined-surface machining".

Increment

Travel path length specification based on number of increments. The number of increments can be stored as → setting data or be selected by means of a suitably labeled key (i.e. 10, 100, 1000, 10000).

Incremental dimension

Incremental dimension: A destination for axis traversal is defined by a distance to be covered and a direction referenced to a point already reached. See → Absolute dimension.

Intermediate blocks

Motions with selected → tool offset (G41/G42) may be interrupted by a limited number of intermediate blocks (blocks without axis motions in the offset plane), whereby the tool offset can still be correctly compensated for. The permissible number of intermediate blocks which the controller reads ahead can be set in system parameters.

Interpolator

Logic unit of the → NCK that defines intermediate values for the motions to be carried out in individual axes based on information on the end positions specified in the part program.

Interpolatory compensation

Mechanical deviations of the machine are compensated for by means of interpolatory compensation functions, such as → leadscrew error, sag, angularity, and temperature compensation.

Interrupt routine

Interrupt routines are special → subprograms that can be started by events (external signals) in the machining process. A part program block which is currently being worked through is interrupted and the position of the axes at the point of interruption is automatically saved.

Inverse-time feedrate

The time required for the path of a block to be traversed can also be programmed for the axis motion instead of the feed velocity (G93).

JOG

Operating mode of the control (setup mode): The machine can be set up in JOG mode. Individual axes and spindles can be traversed in JOG mode by means of the direction keys. Additional functions in JOG mode include: → Reference point approach, → Repos, and → Preset (set actual value).

Key switch

The key switch on the → machine control panel has four positions that are assigned functions by the operating system of the controller. The key switch has three different colored keys that can be removed in the specified positions.

Keywords

Words with specified notation that have a defined meaning in the programming language for → part programs.

KV

Servo gain factor, a control variable in a control loop.

Leading axis

The leading axis is the → gantry axis that exists from the point of view of the operator and programmer and, thus, can be influenced like a standard NC axis.

Leadscrew error compensation

Compensation for the mechanical inaccuracies of a leadscrew participating in the feed. The controller uses stored deviation values for the compensation.

Limit speed

Maximum/minimum (spindle) speed: The maximum speed of a spindle can be limited by specifying machine data, the → PLC or → setting data.

Linear axis

In contrast to a rotary axis, a linear axis describes a straight line.

Linear interpolation

The tool travels along a straight line to the destination point while machining the workpiece.

Load memory

The load memory is the same as the → working memory for the CPU 314 of the → PLC.

Look Ahead

The **Look Ahead** function is used to achieve an optimal machining speed by looking ahead over an assignable number of traversing blocks.

Machine axes

Physically existent axes on the machine tool.

Machine control panel

An operator panel on a machine tool with operating elements such as keys, rotary switches, etc., and simple indicators such as LEDs. It is used to directly influence the machine tool via the PLC.

Machine coordinate system

A coordinate system, which is related to the axes of the machine tool.

Machine zero

Fixed point of the machine tool to which all (derived) measuring systems can be traced back.

Machining channel

A channel structure can be used to shorten idle times by means of parallel motion sequences, e.g. moving a loading gantry simultaneously with machining. Here, a CNC channel must be regarded as a separate CNC control system with decoding, block preparation and interpolation.

Macro techniques

Grouping of a set of statements under a single identifier. The identifier represents the set of consolidated statements in the program.

Main block

A block preceded with ":" that contains all information to start the operating sequence in a → part program.

Main program

The term "main program" has its origins during the time when part programs were split strictly into main and → subprograms. This strict division no longer exists with today's SINUMERIK NC language. In principle, any part program in the channel can be selected and started. It then runs through in → program level 0 (main program level). Further part programs or → cycles as subprograms can be called up in the main program.

MDI

Operating mode of the control: Manual Data Input. In the MDI mode, individual program blocks or block sequences with no reference to a main program or subprogram can be input and executed immediately afterwards through actuation of the NC start key.

Messages

All messages programmed in the part program and → alarms detected by the system are displayed on the operator panel in plain text with date and time and the corresponding symbol for the deletion criterion. Alarms and messages are displayed separately.

Metric measuring system

Standardized system of units: For length, e.g. mm (millimeters), m (meters).

Mirroring

Mirroring reverses the signs of the coordinate values of a contour, with respect to an axis. It is possible to mirror with respect to more than one axis at a time.

Mode

An operating concept on a SINUMERIK control The following modes are defined: → Jog, → MDI, → Automatic.

Mode group

Axes and spindles that are technologically related can be combined into one mode group. Axes/spindles of a mode group can be controlled by one or more → channels. The same → mode type is always assigned to the channels of the mode group.

NC

Numerical Control component of the → CNC that executes the → part programs and coordinates the movements of the machine tool.

Network

A network is the connection of multiple S7-300 and other end devices, e.g. a programming device via a → connecting cable. A data exchange takes place over the network between the connected devices.

NRK

Numeric robotic kernel (operating system of → NCK)

NURBS

The motion control and path interpolation that occurs within the control is performed based on NURBS (**N**on **U**niform **R**ational **B**-**S**plines). This provides a uniform procedure for all internal interpolations.

OEM

The scope for implementing individual solutions (OEM applications) has been provided for machine manufacturers, who wish to create their own user interface or integrate technology-specific functions in the control.

Offset memory

Data range in the control, in which the tool offset data is stored.

Oriented spindle stop

Stops the workpiece spindle in a specified angular position, e.g. in order to perform additional machining at a particular location.

Overall reset

In the event of an overall reset, the following memories of the → CPU are deleted:

- → Working memory
- Read/write area of → load memory
- → System memory
- → Backup memory

Override

Manual or programmable possibility of intervention that enables the user to override programmed feedrates or speeds in order to adapt them to a specific workpiece or material.

Part program

Series of statements to the NC that act in concert to produce a particular → workpiece. Likewise, this term applies to execution of a particular machining operation on a given → raw part.

Part program block

Part of a → part program that is demarcated by a line feed. There are two types: → main blocks and → subblocks.

Part program management

Part program management can be organized by → workpieces. The size of the user memory determines the number of programs and the amount of data that can be managed. Each file (programs and data) can be given a name consisting of a maximum of 24 alphanumeric characters.

Path axis

Path axes include all machining axes of the → channel that are controlled by the → interpolator in such a way that they start, accelerate, stop, and reach their end point simultaneously.

Path feedrate

Path feedrate affects → path axes. It represents the geometric sum of the feedrates of the → geometry axes involved.

Path velocity

The maximum programmable path velocity depends on the input resolution. For example, with a resolution of 0.1 mm the maximum programmable path velocity is 1000 m/min.

PCIN data transfer program

PCIN is a utility program for sending and receiving CNC user data (e.g. part programs, tool offsets) via the serial interface. The PCIN program can run under MS-DOS on standard industrial PCs.

Peripheral module

I/O modules represent the link between the CPU and the process.

I/O modules are:

- → Digital input/output modules
- → Analog input/output modules
- → Simulator modules

PLC

Programmable **L**ogic **C**ontroller: → Programmable logic controller. Component of → NC: Programmable control for processing the control logic of the machine tool.

PLC program memory

SINUMERIK 840D sl: The PLC user program, the user data and the basic PLC program are stored together in the PLC user memory.

PLC programming

The PLC is programmed using the **STEP 7** software. The STEP 7 programming software is based on the **WINDOWS** standard operating system and contains the STEP 5 programming functions with innovative enhancements.

Polar coordinates

A coordinate system which defines the position of a point on a plane in terms of its distance from the origin and the angle formed by the radius vector with a defined axis.

Polynomial interpolation

Polynomial interpolation enables a wide variety of curve characteristics to be generated, such as **straight line, parabolic, exponential functions** (SINUMERIK 840D sl).

Positioning axis

Axis that performs an auxiliary motion on a machine tool (e.g. tool magazine, pallet transport). Positioning axes are axes that do not interpolate with → path axes.

Pre-coincidence

Block change occurs already when the path distance approaches an amount equal to a specifiable delta of the end position.

Program block

Program blocks contain the main program and subprograms of → part programs.

Program level

A part program started in the channel runs as a → main program on program level 0 (main program level). Any part program called up in the main program runs as a → subprogram on a program level 1 ... n of its own.

Programmable frames

Programmable → frames enable dynamic definition of new coordinate system output points while the part program is being executed. A distinction is made between absolute definition using a new frame and additive definition with reference to an existing starting point.

Programmable logic controller

Programmable logic controllers (PLCs) are electronic controllers, the function of which is stored as a program in the control unit. This means that the layout and wiring of the device do not depend on the function of the controller. The programmable logic control has the same structure as a computer; it consists of a CPU (central module) with memory, input/output modules and an internal bus system. The peripherals and the programming language are matched to the requirements of the control technology.

Programmable working area limitation

Limitation of the motion space of the tool to a space defined by programmed limitations.

Programming key

Characters and character strings that have a defined meaning in the programming language for → part programs.

Protection zone

Three-dimensional zone within the → working area into which the tool tip must not pass.

Quadrant error compensation

Contour errors at quadrant transitions, which arise as a result of changing friction conditions on the guideways, can be virtually entirely eliminated with the quadrant error compensation. Parameterization of the quadrant error compensation is performed by means of a circuit test.

R parameters

Arithmetic parameter that can be set or queried by the programmer of the → part program for any purpose in the program.

Rapid traverse

The highest traverse velocity of an axis. It is used, for example, when the tool approaches the → workpiece contour from a resting position or when the tool is retracted from the workpiece

contour. The rapid traverse velocity is set on a machine-specific basis using a machine data item.

Reference point

Machine tool position that the measuring system of the → machine axes references.

Rotary axis

Rotary axes apply a workpiece or tool rotation to a defined angular position.

Rotation

Component of a → frame that defines a rotation of the coordinate system around a particular angle.

Rounding axis

Rounding axes rotate a workpiece or tool to an angular position corresponding to an indexing grid. When a grid index is reached, the rounding axis is "in position".

RS-232-C

Serial interface for data input/output. Machining programs as well as manufacturer and user data can be loaded and saved via this interface.

Safety functions

The controller is equipped with permanently active monitoring functions that detect faults in the → CNC, the → PLC, and the machine in a timely manner so that damage to the workpiece, tool, or machine is largely prevented. In the event of a fault, the machining operation is interrupted and the drives stopped. The cause of the malfunction is logged and output as an alarm. At the same time, the PLC is notified that a CNC alarm has been triggered.

Scaling

Component of a → frame that implements axis-specific scale modifications.

Setting data

Data which communicates the properties of the machine tool to the NC as defined by the system software.

Softkey

A key, whose name appears on an area of the screen. The choice of softkeys displayed is dynamically adapted to the operating situation. The freely assignable function keys (softkeys) are assigned defined functions in the software.

Software limit switch

Software limit switches limit the traversing range of an axis and prevent an abrupt stop of the slide at the hardware limit switch. Two value pairs can be specified for each axis and activated separately by means of the → PLC.

Spline interpolation

With spline interpolation, the controller can generate a smooth curve characteristic from only a few specified interpolation points of a set contour.

Standard cycles

Standard cycles are provided for machining operations which are frequently repeated:

- For the drilling/milling technology
- For turning technology

The available cycles are listed in the "Cycle support" menu in the "Program" operating area. Once the desired machining cycle has been selected, the parameters required for assigning values are displayed in plain text.

Subblock

Block preceded by "N" containing information for a sequence, e.g. positional data.

Subprogram

The term "subprogram" has its origins during the time when part programs were split strictly into →main and subprograms. This strict division no longer exists with today's SINUMERIK NC language. In principle, any part program or any → cycle can be called up as a subprogram within another part program. It then runs through in the next → program level (x+1) (subprogram level (x+1)).

Synchronization

Statements in → part programs for coordination of sequences in different → channels at certain machining points.

Synchronized actions

1. Auxiliary function output
   During workpiece machining, technological functions (→ auxiliary functions) can be output from the CNC program to the PLC. For example, these auxiliary functions are used to control additional equipment for the machine tool, such as quills, grabbers, clamping chucks, etc.
2. Fast auxiliary function output
   For time-critical switching functions, the acknowledgement times for the → auxiliary functions can be minimized and unnecessary hold points in the machining process can be avoided.

Synchronized axes

Synchronized axes take the same time to traverse their path as the geometry axes take for their path.

Synchronized axis

A synchronized axis is the → gantry axis whose set position is continuously derived from the motion of the → leading axis and is, thus, moved synchronously with the leading axis. From the point of view of the programmer and operator, the synchronized axis "does not exist".

System memory

The system memory is a memory in the CPU in which the following data is stored:

- Data required by the operating system
- The operands timers, counters, markers

System variable

A variable that exists without any input from the programmer of a → part program. It is defined by a data type and the variable name preceded by the character **$**. See → User-defined variable.

Tapping without compensating chuck

This function allows threads to be tapped without a compensating chuck. By using the interpolating method of the spindle as a rotary axis and the drilling axis, threads can be cut to a precise final drilling depth, e.g. for blind hole threads (requirement: spindles in axis operation).

Text editor

See → Editor

TOA area

The TOA area includes all tool and magazine data. By default, this area coincides with the → channel area with regard to the access of the data. However, machine data can be used to specify that multiple channels share one → TOA unit so that common tool management data is then available to these channels.

TOA unit

Each → TOA area can have more than one TOA unit. The number of possible TOA units is limited by the maximum number of active → channels. A TOA unit includes exactly one tool data block and one magazine data block. In addition, a TOA unit can also contain a toolholder data block (optional).

Tool

Active part on the machine tool that implements machining (e.g. turning tool, milling tool, drill, LASER beam, etc.).

Tool nose radius compensation

Contour programming assumes that the tool is pointed. Because this is not actually the case in practice, the curvature radius of the tool used must be communicated to the controller which then takes it into account. The curvature center is maintained equidistantly around the contour, offset by the curvature radius.

Tool offset

Consideration of the tool dimensions in calculating the path.

Tool radius compensation

To directly program a desired → workpiece contour, the control must traverse an equistant path to the programmed contour taking into account the radius of the tool that is being used (G41/ G42).

Transformation

Additive or absolute zero offset of an axis.

Travel range

The maximum permissible travel range for linear axes is ± 9 decades. The absolute value depends on the selected input and position control resolution and the unit of measurement (inch or metric).

User interface

The user interface (UI) is the display medium for a CNC in the form of a screen. It features horizontal and vertical softkeys.

User memory

All programs and data, such as part programs, subprograms, comments, tool offsets, and work offsets / frames, as well as channel and program user data, can be stored in the shared CNC user memory.

User program

User programs for the S7-300 automation systems are created using the programming language STEP 7. The user program has a modular layout and consists of individual blocks.

The basic block types are:

- Code blocks
  These blocks contain the STEP 7 commands.
- Data blocks
  These blocks contain constants and variables for the STEP 7 program.

User-defined variable

Users can declare their own variables for any purpose in the → part program or data block (global user data). A definition contains a data type specification and the variable name. See → System variable.

Variable definition

A variable definition includes the specification of a data type and a variable name. The variable names can be used to access the value of the variables.

Velocity control

In order to achieve an acceptable traverse rate in the case of very slight motions per block, an anticipatory evaluation over several blocks (→ Look Ahead) can be specified.

WinSCP

WinSCP is a freely available open source program for Windows for the transfer of files.

Work offset

Specifies a new reference point for a coordinate system through reference to an existing zero point and a → frame.

1. Settable
   A configurable number of settable work offsets are available for each CNC axis. The offsets - which are selected by means of G commands - take effect alternatively.
2. External
   In addition to all the offsets which define the position of the workpiece zero, an external work offset can be overridden by means of the handwheel (DRF offset) or from the PLC.
3. Programmable
   Work offsets can be programmed for all path and positioning axes using the `TRANS` statement.

Working area

Three-dimensional zone into which the tool tip can be moved on account of the physical design of the machine tool. See → Protection zone.

Working area limitation

With the aid of the working area limitation, the traversing range of the axes can be further restricted in addition to the limit switches. One value pair per axis may be used to describe the protected working area.

Working memory

The working memory is a RAM in the → CPU that the processor accesses when processing the application program.

Workpiece

Part to be made/machined by the machine tool.

Workpiece contour

Set contour of the → workpiece to be created or machined.

Workpiece coordinate system

The workpiece coordinate system has its starting point in the → workpiece zero-point. In machining operations programmed in the workpiece coordinate system, the dimensions and directions refer to this system.

Workpiece zero

The workpiece zero is the starting point for the → workpiece coordinate system. It is defined in terms of distances to the → machine zero.

Index

$

$AA_ATOL, 463
$AA_COUP_ACT
  during coupled motion, 470
  for axial master value coupling, 492
$AA_ESR_ENABLE, 577
$AA_LEAD_SP, 492
$AA_LEAD_SV, 492
$AC_ACT_PROG_NET_TIME, 566
$AC_ACTUAL_PARTS, 569
$AC_AXCTSWA, 556
$AC_AXCTSWE, 557
$AC_CTOL, 463
$AC_CUT_INV, 421
$AC_CUTMOD, 421
$AC_CUTMOD_ANG, 421
$AC_CUTTING_TIME, 566
$AC_CYCLE_TIME, 566
$AC_DELAYFST, 445
$AC_ESR_TRIGGER, 577
$AC_OLD_PROG_NET_TIME, 566
$AC_OLD_PROG_NET_TIME_COUNT, 566
$AC_OPERATING_TIME, 566
$AC_OTOL, 463
$AC_PROG_NET_TIME_TRIGGER, 567
$AC_REPOS_PATH_MODE, 455
$AC_REQUIRED_PARTS, 569
$AC_SMAXVELO, 459
$AC_SMAXVELO_INFO, 459
$AC_SPECIAL_PARTS, 569
$AC_STOLF, 465
$AC_TOTAL_PARTS, 569
$AN_AXCTAS, 557
$AN_AXCTSWA, 556
$AN_ESR_TRIGGER, 577
$AN_LANGUAGE_ON_HMI, 786
$AN_POWERON_TIME, 565
$AN_SETUP_TIME, 565
$P_ACTBFRAME, 289
$P_AD, 421
$P_BFRAME, 289
$P_CHBFRAME, 289
$P_CHBFRMASK, 290
$P_CTOL, 464
$P_CUT_INV, 421
$P_CUTMOD, 421
$P_CUTMOD_ANG, 421
$P_DELAYFST, 445
$P_IFRAME, 290
$P_IS_EES_PATH, 211
$P_NCBFRAME, 289
$P_NCBFRMASK, 290
$P_OTOL, 464
$P_PATH, 210
$P_PFRAME, 291
$P_PROG, 210
$P_PROGPATH, 211
$P_SIM, 262
$P_STACK, 210
$P_STOLF, 465
$P_SUBPAR, 151
$PA_ATOL, 464
$SA_LEAD_TYPE, 492
$SC_CONTPREC, 440
$SC_MINFEED, 441
$SC_PA_ACTIV_IMMED, 221
$SN_PA_ACTIV_IMMED, 221
$TC_CARR1...14, 406
$TC_CARR18...23, 406
$TC_CARR18[m], 410
$TC_DP1 ... 25, 367
$TC_ECPxy, 371
$TC_SCPxy, 371

*

* (arithmetic function), 71

/

/ (arithmetic function), 71

+

+ (arithmetic function), 71

<

< (comparison operator), 72
<< (concatenation operator), 81
<= (relational operator), 72
<> (comparison operator), 72

=

== (comparison operator), 72

>

> (comparison operator), 72
>= (relational operator), 72

0

0 character, 79

3

3D tool offset, 388
  Insertion depth, 390
  Intersection procedure, 392
  Offset on the path, 389
  Path curvature, 390
3D tool radius compensation, 384
  3D intersection point of the equidistance, 392
  Circumferential milling, 386
  Face milling, 387
  Inside corners/outside corners, 391
  Transition circle, 392

A

A spline, 234
ABS, 71
Acceleration mode, 434
ACCLIMA, 436
ACOS, 71
Acquiring and finding untraceable sections, 447
ACTBLOCNO, 163
ACTFRAME, 269
Actual value coupling, 502
Addressing, 205
ADISPOSA, 264
Alarms
  set in the NC program, 574
ALF
  for fast retraction from contour, 125
AND, 72
APR, 43
APRB, 43
APRP, 43
APW, 43
APWB, 43
APWP, 43
Arbitrary positions - CYCLE802
  External programming, 673
Arithmetic parameters
  Number n, 22
Arithmetic parameters (R), 22
Array, 50
  definition, 50
  element, 50
Array index, 52
AS, 197
ASIN, 71
ASPLINE, 228
Asynchronous oscillation, 523
ATAN2, 71
ATOL, 461
Automatic interrupt pointer, 448
Automatic path segmentation, 540
AV, 500
Availability
  System-dependent, 5
AX, 549
AXCTSWE, 555
AXCTSWEC, 555
AXCTSWED, 555
Axes
  Coupled-motion, 469
Axial master value coupling, 487
Axis
  replacement, 129
AXIS, 27
AXNAME, 80
AXSTRING, 549
AXTOCHAN, 133
AXTOSPI, 549

B

B spline, 235
B_AND, 72
B_NOT, 72
B_OR, 72
B_XOR, 72
BAUTO, 228
BFRAME, 269
BLOCK, 187
Block display, 188
  suppress, 163
BLSYNC, 120
BNAT, 228
BOOL, 27

Boring - CYCLE86
External programming, 652
BOUND, 75
BRISK, 434
BRISKA, 434
BSPLINE, 228
BTAN, 228

C

C spline, 236
CAC, 227
CACN, 227
CACP, 227
Calculating the dressing tool position - CYCLE435
External programming, 666
CALL, 186
CALLPATH, 189
Cartesian PTP travel, 297
CASE, 98
Case-insensitive, 204
CDC, 227
Centering - CYCLE81
External programming, 641
CFINE, 278
CHAN, 27
CHANDATA, 211
CHAR, 27
Check
structures, 105
CHKDNO, 403
CIC, 227
Circle data
calculating, 597
Circular pocket - POCKET4
External programming, 609
Circular position pattern - HOLES2
External programming, 604
Circular spigot - CYCLE77
External programming, 635
Circumferential milling
with limitation surfaces, 393
Circumferential slot - SLOT2
External programming, 614
CLEARM, 112
CLRINT, 122
COARSE, 500
COARSEA, 264
COLLPAIR, 361
Comparison operators, 72
COMPCAD, 239
COMPCURV, 239
COMPLETE, 211
COMPOF, 239
COMPON, 239
COMPSURF, 239
Concatenation
of strings, 81
Constraints for transformations, 353
CONTDCON, 591
Contour
-coding, 591
-preparation, 585
reposition, 448
table, 585
-table, 591
Contour accuracy
Programmable, 440
Contour call - CYCLE62
External programming, 623
Contour cutting - CYCLE95
External programming, 655
Contour element
travel, 596
Contour preparation
Error feedback signal, 599
CONTPRON, 585
Corner deceleration at all corners, 263
Corner deceleration at inside corners, 263
COS, 71
Count loop, 109
COUPDEF, 500
COUPDEL, 500
Coupled motion, 467
Coupled-axis combinations, 467
Coupled-motion axes, 469
coupling
Generic, 509
Coupling factor, 467
Coupling status
during coupled motion, 470
for axial master value coupling, 492
COUPOF, 500
COUPOFS, 500
COUPON, 500
COUPONC, 500
COUPRES, 500
CPBC, 511
CPDEF, 510
CPDEL, 510
CPFMOF, 513
CPFMON, 513
CPFMSON, 512
CPFPOS, 511, 513

CPFRS, 510
CPLA, 510
CPLCTID, 510
CPLDEF, 510
CPLDEL, 510
CPLDEN, 510
CPLINSC, 515
CPLINTR, 515
CPLNUM, 510
CPLOF, 510
CPLON, 510
CPLOUTSC, 515
CPLOUTTR, 515
CPLPOS, 511
CPLSETVAL, 510
CPMALARM, 516
CPMBRAKE, 516
CPMPRT, 515
CPMRESET, 514
CPMSTART, 515
CPMVDI, 516
CPOF, 510
CPON, 510
CPRECOF, 440
CPRECON, 440
CPROT, 218
CPROTDEF, 215
CPSETTYPE, 516
CPSYNCOP, 515
CPSYNCOP2, 515
CPSYNCOV, 515
CPSYNFIP, 515
CPSYNFIP2, 515
CPSYNFIV, 515
CSPLINE, 228
CTAB, 481
CTABDEF, 471
CTABDEL, 478
CTABEND, 471
CTABEXISTS, 477
CTABFNO, 486
CTABFPOL, 486
CTABFSEG, 486
CTABID, 480
CTABINV, 481
CTABISLOCK, 480
CTABLOCK, 479
CTABMEMTYP, 480
CTABMPOL, 486
CTABMSEG, 486
CTABNO, 486
CTABNOMEM, 486
CTABPERIOD, 480
CTABPOL, 486
CTABPOLID, 486
CTABSEG, 486
CTABSEGID, 486
CTABSEV, 481
CTABSSV, 481
CTABTEP, 481
CTABTEV, 481
CTABTMAX, 481
CTABTMIN, 481
CTABTSP, 481
CTABTSV, 481
CTABUNLOCK, 479
CTOL, 461
CTRANS, 278
CUT3DC, 384, 390
CUT3DCC, 393
CUT3DCCD, 393
CUT3DCD, 384, 390
CUT3DF, 384
CUT3DFF, 384
CUT3DFS, 384
CUTMOD, 417
Cut-off - CYCLE92
  External programming, 653
Cutting edge number, 403
Cycle alarms, 575
CYCLE4071
  External programming, 704
CYCLE4072
  External programming, 705
CYCLE4073
  External programming, 709
CYCLE4074
  External programming, 710
CYCLE4075
  External programming, 713
CYCLE4077
  External programming, 716
CYCLE4078
  External programming, 719
CYCLE4079
  External programming, 721
CYCLE435 - calculate dressing tool position
  External programming, 666
CYCLE495 - form-truing
  External programming, 666
CYCLE60 - Engraving
  External programming, 618
CYCLE61 - Face milling
  External programming, 621

CYCLE62- contour call
External programming, 623
CYCLE63 - Milling contour pocket
External programming, 624
CYCLE64 - Predrilling contour pocket
External programming, 626
CYCLE70 - thread milling
External programming, 627
CYCLE72 - Path milling
External programming, 629
CYCLE76 - rectangular spigot
External programming, 633
CYCLE77 - circular spigot
External programming, 635
CYCLE78 - Drill thread milling
External programming, 637
CYCLE79 - multi-edge
External programming, 639
CYCLE800 - swiveling
External programming, 668
CYCLE801 - grid/frame position pattern
External programming, 671
CYCLE802 - arbitrary positions
External programming, 673
CYCLE81 - centering
External programming, 641
CYCLE82 - drilling
External programming, 642
CYCLE83 - deep-hole drilling
External programming, 645
CYCLE830 - deep-hole drilling 2
External programming, 675
CYCLE832 - High-Speed Settings
External programming, 681
CYCLE84 - tapping without compensating chuck
External programming, 648
CYCLE840 - tapping with compensating chuck
External programming, 684
CYCLE85 - reaming
External programming, 651
CYCLE86 - boring
External programming, 652
CYCLE899 - Milling open slot
External programming, 687
CYCLE92 - cut-off
External programming, 653
CYCLE930 - groove
External programming, 690
CYCLE940 - Undercut
External programming, 693
CYCLE95 - contour cutting
External programming, 655
CYCLE951 - stock removal
External programming, 695
CYCLE952 - Grooving
External programming, 698
CYCLE98 - thread chain
External programming, 657
CYCLE99 - thread turning
External programming, 661
Cylinder surface transformation, 297

D

D number
Freely assigned, 403
D numbers
Check, 403
Renaming, 404
Data class, 47
DCI, 47
DCM, 47
DCU, 47
Deep-hole drilling - CYCLE83
External programming, 645
Deep-hole drilling 2 - CYCLE830
External programming, 675
DEF, 27
DEFAULT, 98
DEFINE ... AS, 197
DELAYFSTOF, 444
DELAYFSTON, 444
DELDL, 372
DELETE, 139
Delete distance-to-go, 257
DELOBJ, 357
Denominator polynomial, 244
DIN subprogram name, 209
Direction vector, 311
Directory path, 207
DISABLE, 122
DISPLOF, 163
DISPLON, 163
DISPR, 448
DIV, 71
DL, 370
DO, 521
Drill thread milling - CYCLE78
External programming, 637
Drilling - CYCLE82
External programming, 642
DRIVE, 434
Drive name, 206
DRIVEA, 434

DV, 500
DYNFINISH, 438
DYNNORM, 438
DYNPOS, 438
DYNROUGH, 438
DYNSEMIFIN, 438

E

EAUTO, 228
EES, 203
EES notation, 205
EG
    Electronic gear, 493
EGDEF, 493
EGDEL, 498
EGOFC, 497
EGOFS, 497
EGON, 494
EGONSYN, 494
EGONSYNE, 494
Electronic gear, 493
Elongated hole - LONGHOLE
    External programming, 616
ELSE, 107
ENABLE, 122
ENAT, 228
ENDFOR, 109
ENDIF, 107
ENDLABEL, 100
Endless loop, 108
ENDLOOP, 108
End-of-motion criterion
    Programmable, 264
ENDWHILE, 110
Engraving - CYCLE60
    External programming, 618
ESR, 576
ESRR, 582
ESRS, 581
ETAN, 228
Euler angles, 309
EVERY, 521
EXECSTRING, 69
EXECTAB, 596
EXECUTE, 599
EXP, 71
EXTCALL, 191, 194
EXTCLOSE, 570
EXTERN, 181
External zero offset, 279
EXTOPEN, 570

F

F10, 215
Face milling, 313
Face milling - CYCLE61
    External programming, 621
FALSE, 27
Fast retraction from the contour, 123
FCTDEF, 380
FCUB, 429
Feed
    Axis, 529
FENDNORM, 263
FFWOF, 439
FFWON, 439
FIFOCTRL, 442
File
    -information, 143
File name, 208
FILEDATE, 143
FILEINFO, 143
FILESIZE, 143
FILESTAT, 143
FILETIME, 143
FINE, 500
FINEA, 264
FLIN, 429
FNORM, 429
Following axis
    for axial master value coupling, 487
FOR, 109
Form-truing - CYCLE495
    External programming, 666
FPO, 429
Frame
    Call, 275
    -chaining, 292
FRAME, 27
Frame component
    FI, 274
    MI, 274
    RT, 274
    SC, 274
    TR, 274
Frame variable
    Assigning values, 272
    Calling coordinate transformations, 267
    Predefined frame variable, 269, 275
Frames
    Assign, 276
    Channel-specific, 288

Frame chains, 277
NCU global, 287
System, 288
FROM, 521
FTOCOF, 383
FTOCON, 383

G

G code
Indirect programming, 65
G group
Technology, 438
G0 tolerance factor, 464
G5, 342
G62, 263
G621, 263
G7, 342
G810 ... G819, 262
G820 ... G829, 262
GEOAX, 551
Geometry axis
Switching, 551
GET, 129
GETACTTD, 405
GETD, 129
GETDNO, 404
GETVARAP, 59
GETVARDFT, 61
GETVARLIM, 60
GETVARPHU, 58
GETVARTYP, 62
Global part program memory (GDIR), 203
Global R parameter, 23
GOTO, 95
GOTOB, 95
GOTOC, 95
GOTOF, 95
GOTOS, 94
GP, 66
Grid/frame position pattern - CYCLE801
External programming, 671
Groove - CYCLE930
External programming, 690
Grooving - CYCLE952
External programming, 698
GUD, 27

H

Help functions, 540
High-speed settings – CYCLE832
External programming, 681
Hold block, 447
HOLES1 - line position pattern
External programming, 604
HOLES2 - circle position pattern
External programming, 604

I

ID, 521
IDS, 521
IF, 107
IFRAME, 269
Inclined axis (TRAANG), 341
INDEX, 83
Indirect programming
of addresses, 63
of G codes, 65
of part program lines, 69
of position attributes, 66
INICF, 27
INIPO, 27
INIRE, 27
INIT, 112
INITIAL, 211
Initial tool orientation setting ORIRESET, 306
INITIAL_INI, 211
Initialization
of arrays, 50
Initialization program, 212
Insertion depth, 390
INT, 27
Interpolation of the rotation vector, 325
Interrupt routine
Deactivating/activating, 122
Delete, 122
Fast retraction from the contour, 123
Newly assign, 121
Programmable traverse direction, 125
Retraction movement, 125
INTERSEC, 595
IPOBRKA, 264
IPOENDA, 264
IPOSTOP, 500
IPTRLOCK, 446
IPTRUNLOCK, 446
ISAXIS, 549
ISD, 384, 390
ISFILE, 142
ISNUMBER, 80
ISOCALL, 188

ISVAR, 57

J

Jerk
Limitation, 434
offset, 457
JERKLIM, 457
JERKLIMA, 436
Jump
to beginning of program, 94
to jump labels, 95
Jump label
for program jumps, 96
Jump marker
For program section repetitions, 100

K

Kinematic type, 410
Kinematics
Resolved, 410

L

L..., 179
Label, 100
LEAD, 306
Lead angle, 308
Leading axis
for axial master value coupling, 487
LEADOF, 487
LEADON, 487
LIFTFAST, 123
Line position pattern - HOLES1
External programming, 604
Link
variables, 25
LLI, 39
LN, 71
Logic operators, 72
LONGHOLE - elongated hole
External programming, 616
Longitudinal slot - SLOT1
External programming, 611
LOOP, 108
LUD, 27

M

M17, 166
M30, 166
Macro, 197
MASLDEF, 516
MASLDEL, 516
MASLOF, 516
MASLOFS, 516
MASLON, 516
Master value coupling
Actual value and setpoint coupling, 491
Synchronization of leading and following axis, 490
Master value simulation, 492
MATCH, 83
MAXVAL, 75
MCALL, 184
MD15800, 24
MD18156, 24
MEAC, 251
MEAFRAME, 283
MEAS, 248
MEASA, 251
Measuring task status, 260
MEAW, 248
MEAWA, 251
Memory
Preprocessing, 442
Program, 201
Working, 211
Milling contour pocket – CYCLE63
External programming, 624
Milling open slot - CYCLE899
External programming, 687
Milling tool
reference point (FH), 391
tip (FS), 391
Milling tool shapes, 388
MINDEX, 83
MINVAL, 75
MMC, 563
MOD, 71
MODAXVAL, 549
MPF, 202
Multi-edge - CYCLE79
External programming, 639

N

NAMETOINT, 360
NCK, 27
NCK notation, 205
Nesting depth
of check structures, 106
NEWCONF, 134
Nibbling
Activate/deactivate, 535
automatic path segmentation, 540
NOC, 500
NOT, 72
NPROT, 218
NPROTDEF, 215
NUMBER, 80
NUT, 318

O

Oblique plunge-cut grinding, 342
OEM addresses, 262
OEM functions, 262
OEMIPO1/2, 262
OFFN, 340
Offset memory, 367
OMA1 ... OMA5, 262
Online tool length offset, 413
Operating mode
During measurement, 257
OR, 72
ORIAXES, 316
ORIC, 397
ORICONCCW, 318
ORICONCW, 318
ORICONIO, 318
ORICONTO, 318
ORICURVE, 321
ORID, 397
Orientation axes, 316
Orientation programming, 316
orientation transformation TRAORI
Orientation programming, 305
Variants of orientation programming, 305
Orientation transformation TRAORI
Generic 5/6-axis transformation, 297
Machine kinematics, 296
Travel movements and orientation movements, 295
Orientation vector THETA, 325
ORIEULER, 316
ORIMKS, 314
ORIPATH, 328
ORIPATHS, 328
ORIPLANE, 318
ORIRESET(A, B, C), 305
ORIROTA, 325
ORIROTC
during interpolation the tool rotation, 330
for rotation of the tool orientation, 325
ORIROTR, 325
ORIROTT, 325
ORIRPY, 316
ORIRPY2, 316
ORIS, 397
ORISOF, 336
ORISON, 336
ORIVECT, 316
ORIVIRT1, 316
ORIVIRT2, 316
ORIWKS, 314
OS, 523
OSB, 523
OSC, 397
OSCILL, 527
Oscillating motion
Infeed at reversal point, 532
Reversal point, 530
Reversal range, 530
Oscillation
Asynchronous, 523
Asynchronous oscillation, 523
Control via synchronized action, 527
Partial infeed, 530
Synchronous oscillation, 527
OSCTRL, 523
OSD, 397
OSE, 523
OSNSC, 523
OSOF, 397
OSP1, 523
OSP2, 523
OSS, 397
OSSE, 397
OST, 397
OST1, 523
OST2, 523
OTOL, 461
Output
to external device/file, 570

P

P..., 183
P_ACTFRAME, 291
Parameter
  Actual, 149
  Formal, 149
  transfer for subprogram call, 181
  Transfer on subprogram call, 150
Parameters
  Machine, 367
Path milling - CYCLE72
  External programming, 629
Path segmentation, 544
Path segmentation for path axes, 542
Path segments - automatic path segmentation, 540
Path specification, 206
PCALL, 189
PDELAYOF, 535
PDELAYON, 535
PFRAME, 269
PHI
  For orientation along the peripheral surface of a taper, 318
  Orientation polynomials, 324
PHU, 40
PL
  for polynomial interpolation, 240
  for spline interpolation, 228
PO, 240
PO[PHI]
  For orientation along the peripheral surface of a taper, 318
  for rotation of the tool orientation, 328
  Orientation polynomials, 324
PO[PSI]
  For orientation along the peripheral surface of a taper, 318
  for rotation of the tool orientation, 328
  Orientation polynomials, 324
PO[THT]
  for rotation of the tool orientation, 328
  Orientation polynomials, 324
PO[XH]
  for orientation specification of two contact points, 321
  Orientation polynomials, 324
PO[YH]
  for orientation specification of two contact points, 321
  Orientation polynomials, 324
PO[ZH]
  for orientation specification of two contact points, 321
  Orientation polynomials, 324
POCKET3 - rectangular pocket
  External programming, 606
POCKET4 - circular pocket
  External programming, 609
Polar transformation, 297
POLF
  for NC-controlled retraction, 577
POLFA, 577
POLFMASK
  for NC-controlled retraction, 577
POLFMLIN
  for NC-controlled retraction, 577
POLY, 240
Polynomial coefficient, 241
Polynomial interpolation, 240
POLYPATH, 240
PON, 543
PONS, 535
POSFS, 500
Position attributes
  Indirect programming, 66
Position synchronism, 500
Position synchronism with angular offset, 500
POT, 71
Predrilling a contour pocket – CYCLE64
  External programming, 626
PREPRO, 166
Preprocessing
  -memory, 442
PRESETON, 280
PRESETONS, 282
PRIO, 120
PRLOC, 27
Process DataShare, 570
Processing time, 566
Program
  addressing, 205
  Branch, 98
  -coordination, 112
  Initialization, 212
  Jumps, 95
  memory, 202
  repetition, 183
  Runtimes, 565
Program loop
  Count loop, 109
  End of loop, 108
  IF loop, 107

REPEAT loop, 111
WHILE loop, 110
Program memory
File types, 202
Standard directories, 202
Program section
-repetition, 100
Program section repetition
with indirect programming CALL, 187
PROTA, 362
PROTD, 364
Protection
zones, 215
PROTS, 363
PSI
For orientation along the peripheral surface of a taper, 318
Orientation polynomials, 324
PUD, 27
PUNCHACC, 535
Punching
Activate/deactivate, 535
automatic path segmentation, 540
PUTFTOC, 382
PUTFTOCF, 381
PW, 228

R

R parameters
Global, 23
R..., 22
READ, 140
REAL, 27
Reaming - CYCLE85
External programming, 651
Rectangular pocket - POCKET3
External programming, 606
Rectangular spigot - CYCLE76
External programming, 633
REDEF, 33
RELEASE, 129
REP, 50
REPEAT, 100
REPEATB, 100
REPOSA, 448
REPOSH, 448
REPOSHA, 448
REPOSL, 448
REPOSQ, 448
REPOSQA, 448
Residual time
for a workpiece, 568
RET, 167
RET (parameterizable), 168
RETB (parameterizable), 175
Retraction
drive-autonomous, 582
NC-controlled, 577
Reversal
Point, 527
RG, 23
RINDEX, 83
RMBBL, 448
RMEBL, 448
RMIBL, 448
RMNBL, 448
Rotary axes
Angle of rotation, 406
Direction vectors, 406
Distance vectors, 406
Rotation
of the orientation vector, 325
ROUND, 71
Round up, 144
ROUNDUP, 144
RPY, 310
Runtime
Response of check structures, 106

S

SAVE, 156
SBLOF, 158
SBLON, 158
SCPARA, 558
SD, 228
SD42475, 334
SD42476, 334
SD42477, 334
SD42678, 336
SD42680, 336
SD42900, 375
SD42910, 375
SD42920, 375
SD42930, 376
SD42935, 378
SD42940, 379
SD42984, 418
Search path
for subprogram call, 209
Programmable search path, 189
SET, 50

SETAL, 574
SETDNO, 404
SETINT, 120
SETM, 112
Setpoint value coupling, 502
Setup value, 371
SIN, 71
Single axis motion, 544
Single-block
suppression, 158
Singular positions, 315
SLOT1 - longitudinal slot
External programming, 611
SLOT2 - circumferential slot
External programming, 614
Smoothing
the orientation characteristic, 336
SOFT, 434
SOFTA, 434
SON, 535
SONS, 535
Speed coupling, 502
Speed synchronism, 500
SPF, 202
SPI, 549
SPIF1, 535
SPIF2, 535
Spindle
replacement, 129
Spline
interpolation, 228
types, 234
Spline group, 238
SPLINEPATH, 238
SPN, 540
SPOF, 535
SPP, 540
SPRINT, 87
SQRT, 71
START, 112
STARTFIFO, 442
STAT, 346
Stock removal
supporting functions, 585
Stock removal - CYCLE951
External programming, 695
STOLF, 464
STOPFIFO, 442
Stopping
drive-autonomous, 581
NC-controlled, 580
STOPRE, 442
String
concatenation, 81
formatting, 87
length, 83
-operations, 78
STRING, 27
STRINGIS, 559
STRLEN, 83
Stroke initiation, 538
Subprogram
Application, 146
call with parameter transfer, 181
call without parameter transfer, 179
call, indirect, 186
call, modal, 184
Name, 147
Programmable search path, 189
repetition, 183
return, parameterizable, 168, 175
Subprogram with path specification and parameters, 189
SUBSTR, 84
Switchable geometry axes, 551
Swiveling - CYCLE800
External programming, 668
Synchronism
coarse, 502
Fine, 502
Synchronous oscillation
Assignment of oscillating and infeed axes, 531
Define infeeds, 531
Evaluation, IPO cycle, 533
Infeed in reversal point range, 532
Infeed movement, 532
Next partial infeed, 534
Synchronized actions, 531
Synchronous spindle
Coupling, 499
pair definition, 505
SYNR, 27
SYNRW, 27
SYNW, 27
System
-dependent availability, 5
System frames, 288
System variables
Probe limitation, 260
Probe status, 259

T

TAN, 71

TANG, 423
TANGDEL, 427
TANGOF, 427
TANGON, 425
Tapping with compensating chuck - CYCLE840
External programming, 684
Tapping without compensating chuck - CYCLE84
External programming, 648
TCARR, 411
TCOABS, 411
TCOFR, 411
TCOFRX, 411
TCOFRY, 411
TCOFRZ, 411
THETA
during interpolation the tool rotation, 330
for rotation of the tool orientation, 325
Thread chain - CYCLE98
External programming, 657
Thread milling - CYCLE70
External programming, 627
Thread turning - CYCLE99
External programming, 661
TILT, 306
Tilt angle, 308
TLIFT, 424
TMOF, 547
TMON, 547
TOFFOF, 414
TOFFON, 414
Tolerance";" for G0, 464
TOLOWER, 82
Tool
-length compensation, 411
-orientation, 306, 397
-orientation for frame change, 413
orientation, smoothing, 336
-parameters, 367
-radius compensation, 373
Tool offset
Coordinate system for wear values, 376
Offset memory, 367
Tool offsets
additive, 370
Tool orientation
relative to the path, 327
Tool orientation relative to the path, 327
Tool radius compensation
Corner deceleration, 263
Toolholder
kinematics, 406
-orientable, 411
Toolholder with orientation capability, 406
TOUPPER, 82
TOWBCS, 376
TOWKCS, 376
TOWMCS, 376
TOWSTD, 376
TOWTCS, 376
TOWWCS, 376
TRAANG, 341
TRACON, 343
TRACYL, 338
TRAFOOF, 354
TRAILOF, 467
TRAILON, 467
Transformation types
General function, 293
Transformation with a swiveling linear axis, 303
Transformations
Chained transformations, 295
Concatenated, 343
Initial tool orientation setting regardless of kinematics, 294
Kinematic transformations, 294
Orientation transformation, 293
Three-, four- and five-axis transformation, 304
TRANSMIT, 337
TRAORI, 304
Trigger event
During measurement, 255
TRUE, 27
TRUNC, 71
TU, 349
Type of coupling, 502

U

ULI, 39
Undercut - CYCLE940
External programming, 693
UNTIL, 111

V

Variable
Type conversion, 77
Variables
Type conversion, 79
User-defined, 27
VELOLIM, 458
VELOLIMA, 436

W

WAITC, 500
WAITE, 112
WAITENC, 557
WAITM, 112
Wear value, 371
WHEN, 521
WHEN-DO, 531
WHENEVER, 521
WHENEVER-DO, 531
WHILE, 110
Working memory, 211
Workpiece
    counter, 569
    directories, 202
    main directory, 202
WRITE, 135

X

xe ye ze, 321
XH YH ZH, 321
xi yi zi, 321
XOR, 72

Z

Zero offset
    External zero offset, 279