

## SINUMERIK 840D sl

### Job planning

#### Programming Manual

Valid for

SINUMERIK 840D sl/840DE sl control system

Software Version  
NCU system software for 840D sl/840DE sl 1.5/2.5

01/2008  
6FC5398-2BP10-3BA0

#### Preface

Flexible NC programming

1

File and Program Management

2

Protection zones

3

Special Motion Commands

4

Coordinate transformation (FRAMES)

5

Transformations

6

Tool offsets

7

Path traversing behavior

8

Axis couplings

9

Motion synchronous actions

10

Oscillation

11

Punching and nibbling

12

Grinding

13

Additional functions

14

User stock removal programs

15

Tables

16

Appendix

A

## Safety Guidelines

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

with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

### CAUTION

without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

### NOTICE

indicates that an unintended result or situation can occur if the corresponding information is not taken into account.

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 device/system may only be set up and used in conjunction with this documentation. Commissioning and operation of a device/system may only be performed by **qualified personnel**. Within the context of the safety notes in this documentation qualified persons are defined as persons who are authorized to commission, ground and label devices, systems and circuits in accordance with established safety practices and standards.

## Prescribed Usage

Note the following:

### WARNING

This device may only be used for the applications described in the catalog or the technical description and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens. Correct, reliable operation of the product requires proper transport, storage, positioning and assembly as well as careful operation and maintenance.

## Trademarks

All names identified by ® are registered trademarks of the 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.

# Preface

## Foreword

### SINUMERIK® Documentation

The SINUMERIK documentation is organized in three parts:

- General Documentation
- User Documentation
- Manufacturer/service documentation

An overview of publications (updated monthly) indicating the language versions available can be found on the Internet at:

<http://www.siemens.com/motioncontrol>

Select the menu items "Support" → "Technical Documentation" → "Overview of Publications".

The Internet version of DOConCD (DOConWEB) is available at:

<http://www.automation.siemens.com/doconweb>

Information about training courses and FAQs (Frequently Asked Questions) can be found at the following website:

<http://www.siemens.com/motioncontrol> under "Support".

## 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 Guide describes the functionality afforded by standard functions. 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.

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

## Technical Support

If you have any technical questions, please contact our hotline:

	Europe/Africa
Phone	+49 180 5050 222
Fax	+49 180 5050 223
Internet	<a href="http://www.siemens.com/automation/support-request">http://www.siemens.com/automation/support-request</a>

	America
Phone	+1 423 262 2522
Fax	+1 423 262 2200
E-Mail	<a href="mailto:techsupport.sea@siemens.com">techsupport.sea@siemens.com</a>

	Asia/Pacific
Phone	+86 1064 719 990
Fax	+86 1064 747 474
E-mail	<a href="mailto:adsupport.asia@siemens.com">mailto:adsupport.asia@siemens.com</a>

---

### Note

Country telephone numbers for technical support are provided under the following Internet address:

Enter <http://www.siemens.com/automation/service&support>

---

## Questions about the manual

If you have any queries (suggestions, corrections) in relation to this documentation, please fax or e-mail us:

Fax: +49 9131- 98 63315

E-mail: mailto:docu.motioncontrol@siemens.com

A fax form is available in the appendix of this document.

## SINUMERIK Internet address

<http://www.siemens.com/sinumerik>

## Export version

The following functions are not available in the export version:

Function	840DE sl
Helical interpolation 2D+6 (Basic version, no options)	-
Milling machining package	-
Five axis machining package	-
Handling transformation package	-
Multi-axis interpolation (> 4 interpolating axes)	-
OA NCK compile cycles	-
Clearance control 1D/3D in position-control cycle <sup>1)</sup>	-
Synchronized actions <sup>1)</sup> (Basic version, no options)	#
Master-value coupling and curve-table interpolation	#
Sag compensation, multi-dimensional	#
Synchronized actions, stage 2 <sup>1)</sup>	-
Electronic gear <sup>1)</sup>	-
Electronic transfer	-

# Restricted functionality  
- Function not available

<sup>1)</sup> In the case of the SINUMERIK 840DE sl export version, the restricted functions are limited to "max. 4 interpolating axes"

## "Fundamentals" and "Job Planning" Programming Manuals

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 "Job Planning" Programming Manual is intended for use by technicians with in-depth, comprehensive programming knowledge. By virtue of a special programming language, the SINUMERIK 840D sl control enables the user to program complex workpiece programs (e.g. for free-form surfaces, channel coordination, ...) and greatly facilitates the programming of complicated operations.

# Table of contents

Preface .....	3
<b>1 Flexible NC programming .....</b>	<b>15</b>
1.1 Variables .....	15
1.1.1 General information on the variables .....	15
1.1.2 System variables .....	17
1.1.3 Arithmetic parameter (R) .....	18
1.1.4 User-defined variables (DEF) .....	22
1.2 Array definitions (DEF, SET, REP) .....	28
1.3 Indirect programming .....	36
1.3.1 Indirectly programming G codes .....	39
1.3.2 Indirectly programming position attributes (BP) .....	41
1.3.3 Indirectly programming part program lines (EXECSTRING) .....	44
1.4 Assignments .....	45
1.5 Arithmetic functions .....	46
1.6 Comparison and logic operations .....	48
1.7 Precision correction on comparison errors (TRUNC) .....	50
1.8 Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND) .....	53
1.9 Priority of the operations .....	55
1.10 Possible type conversions .....	56
1.11 String operations .....	57
1.11.1 Type conversion to STRING (AXSTRING) .....	58
1.11.2 Type conversion from STRING (NUMBER, ISNUMBER, AXNAME) .....	59
1.11.3 Concatenation of strings (<< ) .....	60
1.11.4 Conversion to lower/upper case letters (TOLOWER, TOUPPER) .....	62
1.11.5 Determine length of string (STRLEN) .....	63
1.11.6 Search for character/string in the string (INDEX, RINDEX, MINDEX, MATCH) .....	64
1.11.7 Selection of a substring (SUBSTR) .....	66
1.11.8 Selection of a single character (STRINGVAR, STRINGFELD) .....	67
1.12 Program jumps and branches .....	69
1.12.1 Return jump to the start of the program (GOTOS) .....	69
1.12.2 Program jumps to jump markers (GOTOB, GOTOF, GOTO, GOTOC) .....	70
1.12.3 Program branch (CASE, DEFAULT) .....	73
1.13 Repeating program sections (REPEAT, REPEATB) .....	76
1.14 Check structures .....	83
1.14.1 Program loop with alternative (IF, ELSE, ENDIF) .....	85
1.14.2 Continuous program loop (LOOP, ENDLOOP) .....	87
1.14.3 Count loop (FOR ... TO ..., ENDFOR) .....	88
1.14.4 Program loop with condition at start of loop (WHILE, ENDWHILE) .....	90
1.14.5 Program loop with condition at the end of the loop (REPEAT, UNTIL) .....	91

1.14.6	Program example with nested check structures .....	92
1.15	Program coordination (INIT, START, WAITM, WAITMC, WAITE, SETM, CLEARM) .....	93
1.16	Interrupt routine (ASUB).....	100
1.16.1	Function of an interrupt routine .....	100
1.16.2	Creating an interrupt routine .....	101
1.16.3	Assign and start interrupt routine (SETINT, PRIO) .....	102
1.16.4	Deactivating/reactivating the assignment of an interrupt routine (DISABLE, ENABLE) .....	104
1.16.5	Delete assignment of interrupt routine (CLRINT) .....	105
1.16.6	Fast retraction from the contour (SETINT LIFTFAST, ALF) .....	106
1.16.7	Traversing direction for fast retraction from the contour .....	108
1.16.8	Motion sequence for interrupt routines .....	112
1.17	Axis replacement, spindle replacement (RELEASE, GET, GETD).....	113
1.18	Transfer axis to another channel (AXTOCHAN).....	118
1.19	Activate machine data (NEWCONF).....	120
1.20	Write file (WRITE) .....	121
1.21	Delete file (DELETE).....	124
1.22	Read lines in the file (READ) .....	125
1.23	File present in the NCK user memory (ISFILE) .....	128
1.24	File information (FILEDATE, FILETIME, FILESIZE, FILESTAT, FILEINFO).....	129
1.25	Form the checksum over an array (CHECKSUM) .....	132
1.26	Roundup (ROUNDUP) .....	134
1.27	Subprogram technique.....	135
1.27.1	Subprograms.....	135
1.27.2	Subprograms with SAVE attribute (SAVE) .....	138
1.27.3	Subroutines with parameter transfer (PROC, VAR) .....	140
1.27.4	Call subroutines (L or EXTERN) .....	145
1.27.5	Parameterized subroutine return (RET) .....	152
1.27.6	Subroutine with program repetition (P) .....	159
1.27.7	Modal subprogram call (MCALL) .....	161
1.27.8	Indirect subroutine call (CALL).....	163
1.27.9	Executing a program part in an indirectly called subprogram (CALL BLOCK ... TO ...).....	164
1.27.10	Indirect call of a program programmed in ISO language (ISOCALL) .....	166
1.27.11	Calling subroutine with path specification and parameters (PCALL).....	167
1.27.12	Extend search path for subprogram calls (CALLPATH) .....	168
1.27.13	Execute external subroutine (EXTCALL).....	170
1.27.14	SUPPRESS individual block (SBLOF, SBLON).....	174
1.27.15	SUPPRESS current block display (DISPLOF) .....	180
1.27.16	Identifying subprograms with preparation (PREPRO) .....	182
1.27.17	Cycles: Setting parameters for user cycles.....	183
1.28	Macro technique (DEFINE ... AS) .....	188
<b>2</b>	<b>File and Program Management .....</b>	<b>191</b>
2.1	Program memory .....	191
2.2	Working memory (CHANDATA, COMPLETE, INITIAL) .....	197
2.3	Define user data (DEF) .....	200

2.4	Protection levels for user data, MD, SD and NC commands.....	204
2.4.1	Defining protection levels for user data (APR, APW) .....	204
2.4.2	Automatic activation of GUDs and MACs .....	207
2.4.3	Changing the protection levels for machine and setting data (REDEF, APR, APW) .....	208
2.4.4	Changing protection levels for NC language elements and system variables (REDEF, APX, APW).....	209
2.5	Changing attributes of NC language elements (REDEF) .....	212
2.6	Structuring instruction in step editor (SEFORM).....	220
<b>3</b>	<b>Protection zones .....</b>	<b>221</b>
3.1	Definition of the protection zones (CPROTDEF, NPROTDEF) .....	221
3.2	Activating/deactivating protection zones (CPROT, NPROT).....	224
3.3	Checking for protection zone violation, working area limitation and software limits.....	228
<b>4</b>	<b>Special Motion Commands.....</b>	<b>237</b>
4.1	Approaching coded positions (CAC, CIC, CDC, CACP, CACN) .....	237
4.2	Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL) .....	238
4.3	Spline grouping (SPLINEPATH) .....	250
4.4	NC block compression (COMPON, COMPCURV, COMPCAD, COMPOF).....	252
4.5	Polynomial interpolation (POLY, POLYPATH, PO, PL).....	255
4.6	Settable path reference (SPATH, UPATH).....	262
4.7	Measurements with touch trigger probe (MEAS, MEAW) .....	265
4.8	Extended measuring function (MEASA, MEAWA, MEAC, TE) (option) .....	268
4.9	Special functions for OEM users (OEMIPO1, OEMIPO2, G810 to G829) .....	278
4.10	Feed reduction with corner deceleration (FENDNORM, G62, G621).....	279
4.11	Programmed end-of-motion criterion (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA).....	280
4.12	Programmable servo parameter set (SCPARA) .....	284
<b>5</b>	<b>Coordinate transformation (FRAMES) .....</b>	<b>285</b>
5.1	Coordinate transformation via frame variables .....	285
5.1.1	Predefined frame variable (\$P_BFRAME, \$P_IFRAME, \$P_PFRAME, \$P_ACTFRAME).....	287
5.2	Frame variables / assigning values to frames .....	293
5.2.1	Assigning direct values (axis value, angle, scale) .....	293
5.2.2	Reading and changing frame components (TR, FI, RT, SC, MI).....	296
5.2.3	Linking complete frames .....	298
5.2.4	Defining new frames (DEF FRAME) .....	299
5.3	Coarse and fine offsets (CFINE, CTRANS).....	300
5.4	External zero offset .....	302
5.5	Preset offset (PRESETON).....	303
5.6	Frame calculation from three measuring points in space (MEAFRAME) .....	304
5.7	NCU global frames.....	308
5.7.1	Channel-specific frames (\$P_CHBFR, \$P_UBFR) .....	310
5.7.2	Frames active in the channel.....	311

<b>6</b>	<b>Transformations.....</b>	<b>317</b>
6.1	General programming of transformation types .....	317
6.1.1	Orientation movements for transformations.....	320
6.1.2	Overview of orientation transformation TRAORI.....	324
6.2	Three, four and five axis transformation (TRAORI) .....	326
6.2.1	General relationships of universal tool head.....	326
6.2.2	Three, four and five axis transformation (TRAORI) .....	330
6.2.3	Variants of orientation programming and initial setting (OTIRESET) .....	332
6.2.4	Programming of the tool orientation (A..., B..., C..., LEAD, TILT) .....	334
6.2.5	Face milling (3D-milling A4, B4, C4, A5, B5, C5).....	342
6.2.6	Orientation axis reference (ORIWKS, ORIMKS).....	343
6.2.7	Programming the orientation axes (ORIAxes, ORIVECT, ORIEULER, ORIRPY) .....	346
6.2.8	Orientation programming along the peripheral surface of a taper (ORIPLANE, ORICONxx) .....	349
6.2.9	Specification of orientation for two contact points (ORICURVE, PO[XH]=, PO[YH]=, PO[ZH]=) .....	354
6.3	Orientation polynomials (PO[angle], PO[coordinate]) .....	356
6.4	Rotations of the tool orientation (ORIROTA, ORIROTR/TT, ORIROTC, THETA).....	358
6.5	Orientations relative to the path.....	360
6.5.1	Orientation types relative to the path .....	360
6.5.2	Rotation of the tool orientation relative to the path (ORIPATH, ORIPATHS, angle of rotation) .....	362
6.5.3	Interpolation of the tool rotation relative to the path (ORIROTC, THETA).....	364
6.5.4	Smoothing of orientation characteristic (ORIPATHS A8=, B8=, C8=) .....	367
6.6	Compression of the orientation (COMPON, COMPCURV, COMPCAD).....	369
6.7	Kinematic transformation .....	373
6.7.1	Milling on turned parts (TRANSMIT).....	373
6.7.2	Cylinder surface transformation (TRACYL) .....	376
6.7.3	Inclined axis (TRAANG) .....	386
6.7.4	Inclined axis programming (G05, G07) .....	389
6.8	Cartesian PTP travel.....	391
6.8.1	PTP for TRANSMIT.....	398
6.9	Constraints when selecting a transformation .....	402
6.10	Deselect transformation (TRAFOOF) .....	403
6.11	Chained transformations (TRACON, TRAFOOF) .....	404
<b>7</b>	<b>Tool offsets.....</b>	<b>407</b>
7.1	Offset memory.....	407
7.2	Additive offsets .....	410
7.2.1	Selecting additive offsets (DL) .....	410
7.2.2	Specify wear and setup values (\$TC SCPxy[t,d], \$TC ECPxy[t,d]) .....	412
7.2.3	Delete additive offsets (DELDL).....	413
7.3	Special handling of tool offsets .....	414
7.3.1	Mirroring of tool lengths.....	416
7.3.2	Wear sign evaluation.....	417
7.3.3	Coordinate system of the active machining operation (TOWSTD, TOWMCS, TOWWCS, TOWBCS, TOWTCS, TOWKCS).....	418
7.3.4	Tool length and plane change.....	421
7.4	Language commands for tool management (T, NEWT, DELT, GETT, SETPIECE, GETSELT) (option) .....	423

7.5	Online tool offset (PUTFTOCF, FCTDEF, PUTFTOC, FTOCON, FTOCOF).....	426
7.6	Activate 3D tool offsets (CUT3DC..., CUT3DF...).....	432
7.6.1	Activating 3D tool offsets (CUT3DC, CUT3DF, CUT3DFS, CUT3DFF, ISD).....	432
7.6.2	3D tool offset peripheral milling, face milling .....	434
7.6.3	3D tool offset Tool shapes and tool data for face milling.....	436
7.6.4	3D tool offset Compensation on the path, path curvature, insertion depth (CUT3DC, ISD) ....	438
7.6.5	3D tool offset Inside/outside corners and intersection procedure (G450/G451) .....	440
7.6.6	3D tool offset 3D circumferential milling with limitation surfaces .....	442
7.6.7	3D tool offset Taking into consideration a limitation surface (CUT3DCC, CUT3DCCD).....	443
7.7	Tool orientation (ORIC, ORID, OSOF, OSC, OSS, OSSE, OSD, OST).....	447
7.8	Free assignment of D numbers, cutting edge numbers.....	454
7.8.1	Free assignment of D numbers, cutting edge numbers (CE address) .....	454
7.8.2	Free assignment of D numbers: Checking D numbers (CHKDNO).....	455
7.8.3	Free assignment of D numbers: Rename D numbers (GETDNO, SETDNO) .....	455
7.8.4	Free assignment of D numbers: Determine T number to the specified D number (GETACTTD) .....	457
7.8.5	Free assignment of D numbers: Invalidate D numbers (DZERO) .....	458
7.9	Tool holder kinematics .....	458
7.10	Tool length compensation for orientable toolholders (TCARR, TCOABS, TCOFR).....	464
7.11	Online tool length compensation (TOFFON, TOFFOF).....	468
7.12	Cutting data modification for tools that can be rotated (CUTMOD).....	471
<b>8</b>	<b>Path traversing behavior.....</b>	<b>477</b>
8.1	Tangential control (TANG, TANGON, TANGOF, TLIFT, TANGDEL).....	477
8.2	Feedrate response (FNORM, FLIN, FCUB, FPO) .....	485
8.3	Program run with preprocessing memory (STARTFIFO, STOPFIFO, STOPRE) .....	491
8.4	Conditionally interruptible program sections (DELAYFSTON, DELAYFSTOF) .....	493
8.5	Preventing program position for SERUPRO (IPTRLOCK, IPTRUNLOCK) .....	499
8.6	Repositioning to a contour (REPOSA, REPOSL, REPOSQ, REPOSQA, REPOSH, REPOSHA, DISR, DISPR, RMI, RMB, RME, RMN) .....	502
8.7	Influencing the motion control.....	512
8.7.1	Percentage jerk correction (JERKLIM) .....	512
8.7.2	Percentage velocity correction (VELOLIM).....	513
8.7.3	Program example for JERKLIM and VELOLIM .....	514
<b>9</b>	<b>Axis couplings.....</b>	<b>515</b>
9.1	Coupled motion (TRAILON, TRAILOF) .....	515
9.2	Curve tables (CTAB) .....	519
9.2.1	Curve tables: general relationships .....	519
9.2.2	Curve tables: Principal functions (CTABDEF, CATBEND, CTABDEL) .....	520
9.2.3	Curve tables: Forms (CTABNOMEM, CTABFNO, CTABID, CTABLOCK, CTABDEL, CTABUNLOCK, CTABISLOCK, CTABEXISTS, CTABMEMTYP, CTABPERIOD, CTABSEGID, CTABSEG, CTABFSEG, CTABMSEG, CTABPOLID, CTABFPOL, CTABMPOL) .....	527

9.2.4	Curve tables: Behavior at the edges (CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABTMIN, CTABTMAX) .....	532
9.2.5	Curve tables: Access to table positions and table segments (CTAB, CTABINV, CTABSSV, CATBSEV) .....	536
9.3	Axial leading value coupling (LEADON, LEADOF) .....	540
9.4	Electronic gear (EG).....	546
9.4.1	Defining an electronic gear (EGDEF) .....	547
9.4.2	Switch-in the electronic gearbox (EGON, EGONSYN, EGONSYNE) .....	548
9.4.3	Switching-in the electronic gearbox (EGOFS, EGOFC) .....	552
9.4.4	Deleting the definition of an electronic gear (EGDEL) .....	553
9.4.5	Rotational feedrate (G95) / electronic gear (FPR) .....	553
9.5	Synchronous spindle.....	554
9.5.1	Synchronous spindle: Programming (COUPDEF, COUPDEL, COUPON, COUPONC, COUPOF, COUPOFS, COUPRES, WAITC) .....	555
9.6	Master/slave group (MASLDEF, MASLDEL, MASLON, MASLOF, MASLOFS).....	568
<b>10</b>	<b>Motion synchronous actions .....</b>	<b>573</b>
10.1	Basics.....	573
10.1.1	Area of validity and machining sequence (ID, IDS) .....	576
10.1.2	Cyclically checking the condition (WHEN, WHENEVER, FROM, EVERY) .....	578
10.1.3	Actions (DO).....	581
10.2	Operators for conditions and actions .....	582
10.3	Main run variables for synchronized actions.....	583
10.3.1	System variables.....	583
10.3.2	Implicit type conversion.....	586
10.3.3	GUD variables .....	587
10.3.4	Default axis identifier (NO_AXIS) .....	590
10.3.5	Synchronized action marker (\$AC_MARKER[n]) .....	591
10.3.6	Synchronized action parameters (\$AC_PARAM[n]) .....	592
10.3.7	Arithmetic parameter (\$R[n]) .....	593
10.3.8	Read and write NC machine and NC setting data .....	594
10.3.9	Timer variable (\$AC_Timer[n]) .....	595
10.3.10	FIFO variables (\$AC_FIFO1[n] ... \$AC_FIFO10[n]) .....	596
10.3.11	Information about block types in the interpolator (\$AC_BLOCKTYPE, \$AC_BLOCKTYPEINFO, \$AC_SPLITBLOCK).....	599
10.4	Actions in synchronized actions .....	602
10.4.1	Overview of possible actions in synchronized actions .....	602
10.4.2	Output of auxiliary functions.....	605
10.4.3	Set read-in disable (RDISABLE) .....	606
10.4.4	Cancel preprocessing stop (STOPREOF) .....	606
10.4.5	Delete distance-to-go (DELDTG) .....	607
10.4.6	Polynomial definition (FCTDEF) .....	609
10.4.7	Synchronized function (SYNFCT) .....	612
10.4.8	Closed-loop clearance control with limited correction (\$AA_OFF_MODE) .....	615
10.4.9	Online tool offset (FTOC) .....	618
10.4.10	Online tool length compensation (\$AA_TOFF) .....	620
10.4.11	Positioning movements .....	622
10.4.12	Position axis (POS) .....	622
10.4.13	Position in specified reference range (POS RANGE) .....	624
10.4.14	Start/stop axis (MOV).....	626
10.4.15	Axis replacement (RELEASE, GET) .....	627
10.4.16	Axial feed (FA) .....	632

10.4.17	Software limit switch .....	632
10.4.18	Axis coordination.....	633
10.4.19	Set actual values (PRESETON) .....	634
10.4.20	Spindle motions .....	635
10.4.21	Coupled motion (TRAILON, TRAILOF) .....	636
10.4.22	Leading value coupling (LEADON, LEADOF) .....	638
10.4.23	Measuring (MEAWA, MEAC).....	641
10.4.24	Initialization of array variables (SET, REP).....	642
10.4.25	Set/delete wait markers (SETM, CLEARM).....	643
10.4.26	Fault responses (SETAL).....	644
10.4.27	Travel to fixed stop (FXS, FXST, FXSW, FCON, FCOF).....	645
10.4.28	Determining the path tangent in synchronized actions.....	647
10.4.29	Determining the current override .....	648
10.4.30	Time use evaluation of synchronized actions .....	649
10.5	Technology cycles .....	651
10.5.1	Context variable (\$P_TECCYCLE) .....	655
10.5.2	Call by value parameters .....	656
10.5.3	Default parameter initialization.....	656
10.5.4	Control processing of technology cycles (ICYCOF, ICYCON) .....	657
10.5.5	Cascading technology cycles.....	658
10.5.6	Technology cycles in non-modal synchronized actions.....	659
10.5.7	Check structures (IF) .....	659
10.5.8	Jump instructions (GOTO, GOTOF, GOTOB).....	660
10.5.9	Lock, unlock, reset (LOCK, UNLOCK, RESET).....	661
10.6	Delete synchronized action (CANCEL).....	663
10.7	Restrictions .....	664
<b>11</b>	<b>Oscillation.....</b>	<b>669</b>
11.1	Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE) .....	669
11.2	Control oscillation via synchronized actions .....	675
<b>12</b>	<b>Punching and nibbling .....</b>	<b>685</b>
12.1	Activation, deactivation .....	685
12.1.1	Punching and nibbling on or off (SPOF, SON, PON, SONS, PONS, PDELAYON, PDELAYOF).....	685
12.2	Automatic path segmentation .....	689
12.2.1	Path segmentation for path axes .....	692
12.2.2	Path segmentation for single axes.....	694
<b>13</b>	<b>Grinding.....</b>	<b>697</b>
13.1	Grinding-specific tool monitoring in the part program (TMON, TMOF).....	697
<b>14</b>	<b>Additional functions.....</b>	<b>699</b>
14.1	Axis functions (AXNAME, AX, SPI, AXTOSPI, ISAXIS, AXSTRING, MODAXVAL) .....	699
14.2	Replaceable geometry axes (GEOAX) .....	702
14.3	Link communication .....	707
14.3.1	Access to a global NCU memory area.....	709
14.4	Axis container (AXCTSWE, AXCTSWED).....	710
14.5	Extended stop and retract.....	715
14.5.1	Drive-independent responses to ESR .....	717

14.5.2	NC-controlled responses to retraction (POLF, POLFA, POLFMASK, POLFMLIN) .....	720
14.5.3	NC-controlled reactions to stoppage.....	725
14.5.4	Generator operation/DC link backup.....	726
14.5.5	Drive-independent stopping .....	726
14.5.6	Drive-independent retraction.....	727
14.6	Check scope of NC language present (STRINGIS).....	728
14.7	Function call ISVAR and read machine data array index .....	734
14.8	Learn compensation characteristics (QECLRNON, QECLRNOF) .....	737
14.9	Interactively call the window from the part program (MMC) .....	739
14.10	Program runtime/part counter .....	740
14.10.1	Program runtime/part counter (overview) .....	740
14.10.2	Program runtime .....	740
14.10.3	Workpiece counter .....	744
14.11	Alarms (SETAL) .....	746
<b>15</b>	<b>User stock removal programs .....</b>	<b>749</b>
15.1	Supporting functions for stock removal.....	749
15.2	Generate contour table (CONTPRON) .....	750
15.3	Generate coded contour table (CONTDCON) .....	757
15.4	Determine point of intersection between two contour elements (INTERSEC) .....	762
15.5	Execute the contour elements of a table block-by-block (EXECTAB) .....	764
15.6	Calculate circle data (CALCDAT) .....	765
15.7	Deactivate contour preparation (EXECUTE) .....	767
<b>16</b>	<b>Tables.....</b>	<b>769</b>
16.1	Statements A - G .....	769
<b>A</b>	<b>Appendix.....</b>	<b>821</b>
A.1	List of abbreviations .....	821
A.2	Feedback on the documentation.....	827
A.3	Overview .....	829
<b>Glossary .....</b>	<b>831</b>	
<b>Index.....</b>	<b>859</b>	

# Flexible NC programming

## 1.1 Variables

### 1.1.1 General information on the variables

#### Function

Using variables in place of constant values makes a program more flexible. You can respond to signals such as measured values or, by storing setpoints in the variables, you can use the same program for different geometries.

With variable calculation and jump instructions a skilled programmer is able to create a very flexible program archive and save a lot of programming work.

#### Variable types

The control uses 3 classes of variable:

- System variables

Variables provided by the control that can be processed in the program (write, read).  
System variables supply machine and control states.

- Arithmetic parameters

Special, predefined arithmetic variables whose address is R plus a number.

- User-defined variables

Variables where the user defines the name and variable type.

## Variable types

The data type permitted for the variable is determined when the variable is defined. The data type for system variables and predefined variables is fixed.

Elementary variable types/data types are:

Type	Significance	Value Range
INT	Integer values with signs	-2147483646 ... +2147483647
REAL	Real numbers (fractions with decimal point, LONG REAL in acc. with IEEE)	$\pm (2,2 \cdot 10^{-308} \dots 1,8 \cdot 10^{+308})$
BOOL	Boolean values: TRUE (1) and FALSE (0)	1, 0
CHAR	ASCII characters	corresponding to code 0 ... 255
STRING	Character string, No. of characters in [...]	maximum 200 characters (no special characters)
AXIS	Axis identifier (axis addresses)	Any axis identifiers in the channel
FRAME	Geometrical parameters for translation, rotation, scaling, and mirroring	

## 1.1.2 System variables

### Function

The system variables made available from the control offer access to machine and control state, e.g. zero offsets, tool offsets, actual values, measured values of the axes etc.

This means that they can be read and also partially written to in all of the programs being executed.

### System variable names

Special identifiers of system variables always begin with a "\$" sign. The specific names then follow:

1st letter	Significance
\$M	Machine data
\$S	Setting data
\$T	Tool management data
\$P	Programmed values
\$A	Current values
\$V	Service data

2nd letter	Significance
N	NCK global
C	Channel-specific
A	Axis-specific

Example:

\$AA\_IM means "axis-specific actual value in the machine coordinate system".

### 1.1.3 Arithmetic parameter (R)

#### Function

Arithmetic parameters (R parameters) can be used if an NC program is not only to be applicable for values that have been defined once or the values must first be calculated.

The required values can be set or calculated by the control during program execution. Another possibility consists of setting the arithmetic parameter values by operator inputs.

If values have been assigned to the arithmetic parameters, they can be assigned to other variable-setting NC addresses in the program.

#### Syntax

R<n>=<value>

#### Significance

R	Arithmetic parameter address
<n>	Number of the arithmetic parameter
Type:	INT
Range of values:	0 - Maximum The maximum value is set using a machine data and should be taken from the data of the machinery construction OEM.
	Initial setting: 0 - 99

<value> Value of the arithmetic parameter

The value must be assigned a dedicated block.

Possible data include:

<Real value> Type: REAL

Range of values: For a non-exponential notation:  
 $\pm (0.000\ 0001 \dots 9999\ 9999)$

The following rules apply for non-exponential notation:

- A maximum of 8 decimal places are permitted.
- The decimal point can be omitted for integer values.
- A plus sign can always be omitted.

For an exponential notation:

$\pm (10^{-300} \dots 10^{+300})$

The following rules apply for an exponential notation:

- The value of the exponent is written after the character combination "EX".
- A maximum of 10 characters are permitted (including sign and decimal point).
- A plus sign can always be omitted.

<Arithmetic function> When using arithmetic functions/operators, then the usual mathematical notation must be used.

Machining priorities are set using round brackets.  
Otherwise, the rule "multiplication or division takes precedence over addition or subtraction" applies.

Degrees are used for the trigonometrical functions.

R parameters are frequently part of arithmetic functions.

### **Assignment to other addresses**

The flexibility of an NC program comes down to being able to assign these arithmetic parameters or functions with arithmetic parameters to other NC addresses.

R parameters can be assigned at almost all addresses (exceptions: N, G and L).

Syntax:

<address>=R<n>  
<address>=<arithmetic function with R parameters>

It is also possible to have an assignment with a minus sign.

A separate block is required for assignments to axis addresses (traversing instructions).

### **Examples**

**Assigning values in a non-exponential notation:**

R0=3.5678  
R1=-37.3  
R3=-7  
R4=-45678.1234

**Assigning values in a exponential notation:**

<b>Word</b>	<b>Significance:</b>
R0=-0.1EX-5	R0 = -0.000 001
R1=1.874EX8	R1 = 187 400 000

**Assigning an arithmetic function:**

<b>Word</b>	<b>Significance:</b>
R5=SIN(25.3)	R13 corresponds to the sine of 25.3 degrees.

**Assigning arithmetic functions with R parameters:**

<b>Word</b>	<b>Significance:</b>
R1=R1+1	The new R1 is calculated from the old R1 plus 1.
R1=R2+R3	R1 is obtained by adding R2 and R3.
R4=R5-R6	R4 is obtained by subtracting R6 from R5.
R7=R8*R9	R7 is obtained by multiplying R8 and R9.
R10=R11/R12	R10 is obtained by dividing R11 (numerator) by R12 (denominator).
R14=R1*R2+R3	The calculation follows the rule "multiplication or division takes precedence over addition or subtraction" $\Rightarrow$ $R14=(R1*R2)+R3$
R15=SQRT (R1*R1+R2*R2)	R15 corresponds to the square root from the following sum: $(R1)^2+(R2)^2$

**Assigning R parameters to axis addresses:**

<b>Program code</b>
N10 G1 G91 X=R1 F300 N20 Z=R3 N30 X=-R4 N40 Z=-R5 ...

### 1.1.4 User-defined variables (DEF)

#### Function

The user can define his own variables and assign values. A demarcation to the system variables, these are called user-defined variables.

User-defined variables can be include:

- Local user variables (LUD; Local User Data)

LUD are user variables, defined in a part program that is not used as main program. They are created with part program start and deleted with part program end or reset. It is only possible to access LUD within the part program (reading/writing) in which they are defined.

- Program global user variables (PUD; Program Global User Data)

PUD are user variables, defined in a part program used as main program. They are created with part program start and deleted with part program end or reset. It is possible to access PUD in the main program and in all subprograms of the main program.

Prerequisite:

MD11120 \$MN\_LUD\_EXTENDED\_SCOPE is set to "1".

(if MD11120 is set to "0", then user variables defined in the main program are only effective in the main program).

- Global user variables (GUD; Global User Data)

GUD are NC and channel-global user variables that are also kept after a power on. GUD can be accessed in all programs.

The DEF command is used to define user-defined variables.

#### Syntax

```
DEF <variable type> <variable name> = <value>
```

#### Significance

DEF

Instruction to define user variables

User variables must be defined at the beginning of the program before they are used. The following rules must be met:

- The definition must be made a dedicated block.
- Only one variable type may be defined per block.
- Several variables of **one type** can be defined in block.

<variable type>	Specification of the variable type:	
INT	Value:	Integer value with sign
	Range of values:	-2147483646 ... +2147483647
REAL	Value:	Real number (fraction with decimal point, LONG REAL to IEEE)
	Range of values:	$\pm (2,2 \cdot 10^{-308} \dots 1,8 \cdot 10^{+308})$
BOOL	Value:	Truth value TRUE (1) / FALSE (0)
	Range of values:	1, 0
CHAR	Value:	ASCII character
	Range of values:	ASCII code 0 ... 255
<b>Note:</b>		
Instead of the ASCII code value, it is also possible to directly assign the ASCII characters. It should be noted that the ASCII character is set in quotation marks "...".		
STRING	Value:	Character string (<string>):
		<ul style="list-style-type: none"> <li>• maximum 200 characters</li> <li>• no special characters</li> </ul>
	Range of values:	ASCII code 0 ... 255
	Parameters:	[<string length>]
		Maximum number of characters; is enclosed in square brackets [...] after the variable type.
<b>Note:</b>		
The character string (<string>) must be set in quotation marks "...": "<string>"		
AXIS	Value:	<ul style="list-style-type: none"> <li>• Axis identifier (axis address)</li> </ul>
	Range of values:	All of the axis/spindle identifiers available in the channel
<b>Note:</b>		
Axis names with an extended address must be enclosed in parentheses.		
FRAME	Value:	<ul style="list-style-type: none"> <li>• Geometrical parameters for translation, rotation, scaling, and mirroring</li> </ul>

<variable name>	<b>Names of variables</b>
	The following rules must be observed when assigning names:
	<ul style="list-style-type: none"> <li>• A variable name consists of up to 31 characters.</li> <li>• The first two characters must be a letter or an underscore.</li> <li>• The "\$" sign can not be used for user-defined variables because it is used for system variables.</li> </ul>
<value>	<b>Value of variable</b>
	The type of value assignment and the value range depend on the variable type (refer above).
	<b>Note:</b>
	If a variable is not assigned a value on definition, the system sets "0" as the default.

## Examples

### Example 1: Defining variables of different types

Block	Significance
DEF INT NUMBER	This creates a variable of type INTEGER with the name "NUMBER". System initializes with "0".
DEF INT NUMBER=7	This creates a variable of type INTEGER with the name "NUMBER". The variable has the initial value "7".
DEF REAL DEPTH	This creates a variable of type REAL with the name "DEPTH". System initializes with "0.0".
DEF REAL DEPTH=6.25	This creates a variable of type REAL with the name "DEPTH". The variable has the initial value "6.25".
DEF REAL DEPTH=3.1, LENGTH=2, NUMBER	Defining several variables, type REAL in a block.

```
DEF BOOL IF_TOO MUCH
```

This creates a variable of type BOOL with the name "IF\_TOO MUCH".

System initializes with "0" (FALSE).

```
DEF BOOL IF_TOO MUCH=1
```

This creates a variable of type BOOL with the name "IF\_TOO MUCH".

The variable has the initial value "1" (TRUE).

```
DEF BOOL IF_TOO MUCH=TRUE
```

This creates a variable of type BOOL with the name "IF\_TOO MUCH".

The variable has the initial value "TRUE".

```
DEF BOOL IF_TOO MUCH=FALSE
```

This creates a variable of type BOOL with the name "IF\_TOO MUCH".

The variable has the initial value "FALSE".

```
DEF CHAR GUSTAV_1=65
```

This creates a variable of type CHAR with the name "GUSTAV\_1".

The variable has the ASCII code "65" as initial value. This corresponds to the letter "A".

```
DEF CHAR GUSTAV_1="A"
```

This creates a variable of type CHAR with the name "GUSTAV\_1".

The variable has the ASCII character "A" as initial value.

```
DEF STRING[6] MUSTER_1="BEGIN"
```

This creates a variable of type STRING with the name "MUSTER\_1".

The maximum number of characters in the character string is limited to 6.

The variable is assigned the character string "ANFANG".

```
DEF AXIS AXIS_NAME=(X1)
```

This creates a variable of type AXIS with the name "AXIS NAME".

The variable is assigned the axis identifier X1.

```
DEF FRAME BEVEL_1
```

This creates a variable of type FRAME with the name "BEVEL\_1".

**Example 2: Local and program-global user variables (LUD / PUD)**

<b>Programming</b>	<b>Comments</b>
PROC MAIN	; Main program
DEF INT VAR1	; PUD definition
...	
SUB2	; Subprogram call
...	
M30	

<b>Program code</b>	<b>Comments</b>
PROC SUB2	; Subprogram SUB2
DEF INT VAR2	; LUD DEFINITION
...	
IF (VAR1==1)	; Read PUD
VAR1=VAR1+1	; Read and write PUD
VAR2=1	; Write LUD
ENDIF	
SUB3	; Subprogram call
...	
M17	

<b>Program code</b>	<b>Comments</b>
PROC SUB3	; Subprogram SUB3
...	
IF (VAR1==1)	; Read PUD
VAR1=VAR1+1	; Read and write PUD
VAR2=1	; Error: LUD from SUB2 not known
ENDIF	
...	
M17	

**Example 3: Program local variables**

Program code	Comments
DEF INT COUNTER LOOP: G0 X... COUNT=COUNT+1 IF COUNT<50 GOTOB LOOP M30	; Loop

**Example 4: Querying existing geometry axes**

Program code	Comments
DEF AXIS ABSCISSA; IF ISAXIS(1) == FALSE GOTOF CONTINUE ABSCISSA = \$P_AXN1 CONTINUE:	; 1. Geometry axis

**Example 5: Indirect spindle programming**

Program code	Comments
DEF AXIS SPINDLE SPINDLE=(S1) OVRA[SPINDLE]=80 SPINDLE=(S3) ...	; Spindle override = 80%

## 1.2 Array definitions (DEF, SET, REP)

### Function

System variables as well as GUD or LUD variables can be defined as 1 to 3-dimensional arrays:

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

#### Note

STRING variables can only be set-up in 2-dimensions.

---

### Data types

The following data types are supported:

- for system variables:                   BOOL, CHAR, INT, REAL, STRING, AXIS
- for GUD or LUD variables:           BOOL, CHAR, INT, REAL, STRING, AXIS, FRAME

### Initialization of arrays

Initialization values can be assigned to array elements while the programs is being executed or already when defining the array.

There are two ways of initializing:

- Initializing with a value list (SET)
  - Initializing with the same values (REP)
- 

#### Note

Variables of type FRAME cannot be initialized.

---

### Sequence when initializing

For 2 and 3-dimensional arrays, the right array index is incremented first.

## Syntax

### Array definition (DEF):

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

### Initializing with a value list (SET):

- when defining the array:

```
DEF <data type> <variable name>[<n>,<m>,<o>]=SET(<VALUE1>,<value2>,...)
DEF <data type> <variable name>[<n>,<m>,<o>]=(<VALUE1>,<value2>,...)
```

#### Note

When defining the array, the use of the SET command is optional.

- when the program is being executed:

```
<variable name>[<n>,<m>,<o>]=SET(<VALUE1>,<value2>,...)
```

### Initializing with the same values (REP):

- when defining the array:

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

- when the program is being executed:

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

## Significance

DEF	Command to define variables
<data type>	Data type of variables
	Range of values:
	<ul style="list-style-type: none"> <li>for system variables: BOOL, CHAR, INT, REAL, STRING, AXIS</li> <li>for GUD or LUD variables: BOOL, CHAR, INT, REAL, STRING, AXIS, FRAME</li> </ul>
<string length>	Maximum number of characters for a STRING data type

<variable name>	Name of the system variables, GUD or LUD variables
[<n>, <m>, <o>]	Array index
<n>	Array size for 1st dimension Type: INT (for system variables, also AXIS) Value Range 0 < n ≤ 32767
<m>	Array size for 2nd dimension Type: INT (for system variables, also AXIS) Value Range 0 < m ≤ 32767
<o>	Array size for 3rd dimension Type: INT (for system variables, also AXIS) Value Range 0 < o ≤ 32767
SET	Command to initialize the array elements with the values of the specified value list
(<VALLUE1>, <value2>, ...)	Value list
REP	Command to initialize array elements with the specified <value>
<VALUE>	Value, which the array elements should be written when initializing with REP.
<number_array_elements>	Number of array elements that should be written to with the specified <value> when initializing with REP. The remaining array elements are pre-assigned zero (initializing for the array definition) or remain unchanged (initialization while the program is being executed).  If this parameter is not programmed, then all array elements are initialized with the specified <value>. If the value of this parameter is equal to zero, then the result depends on the initialization instant: <ul style="list-style-type: none"><li>• initializing when defining the array: → All elements are pre-assigned zero</li><li>• Initialization while the program is being executed: → The actual values of the array elements remain unchanged.</li></ul>

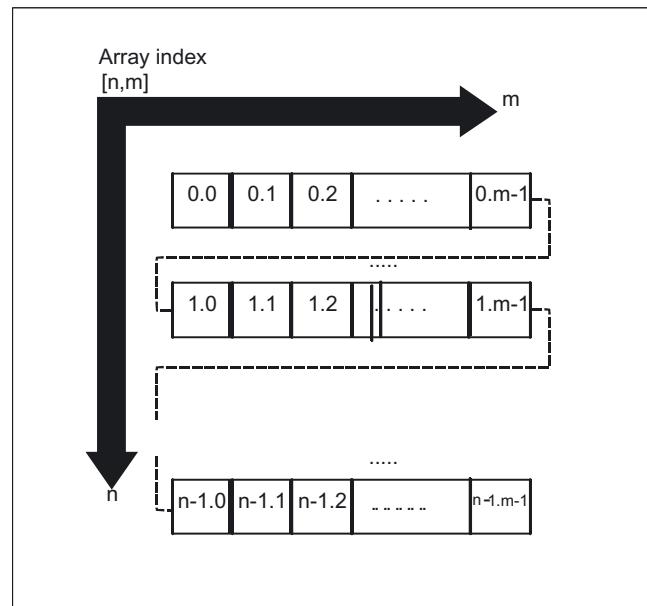
### Array index [..., ..., ...]

Elements of an array are accessed via the array index. The array elements can either be read or assigned values using this array index.

The first array element starts:

- for a 1-dimensional array with the index [0]
- for a 2-dimensional array with the index [0,0]
- for a 3-dimensional array with the index [0,0,0]

The array indices for a 2-dimensional array are shown in the following diagram:



Example:

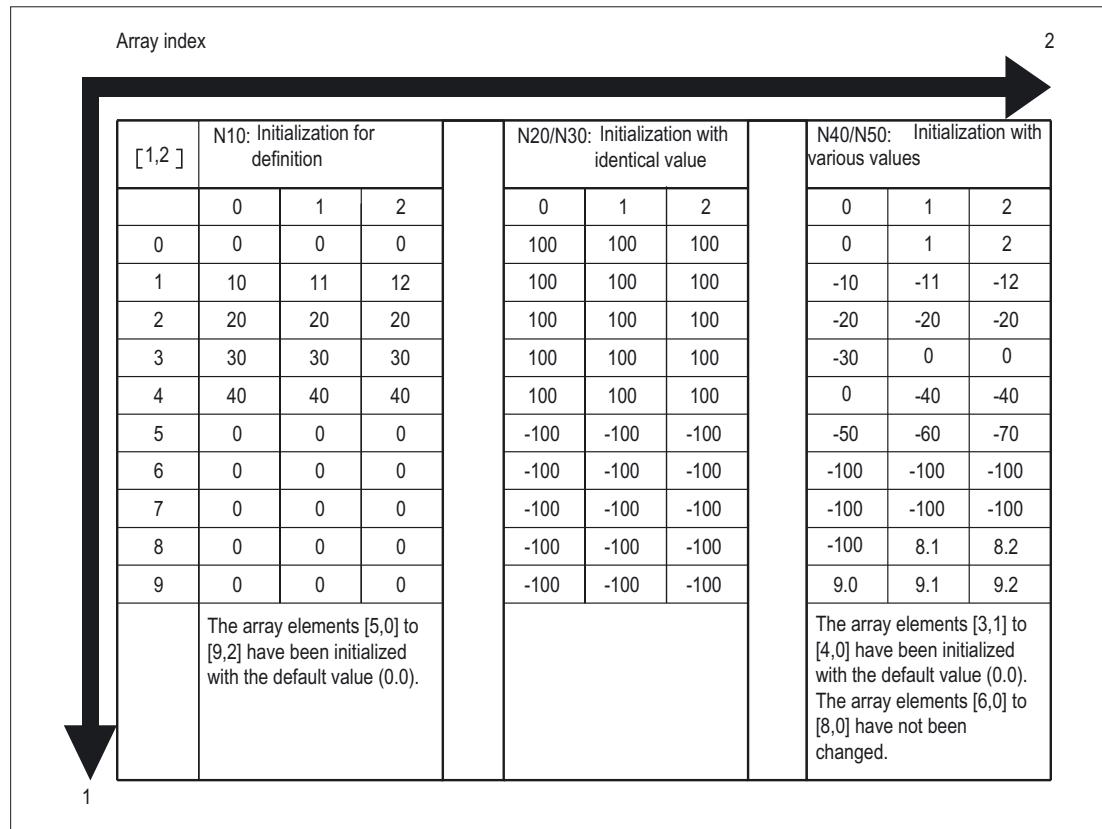
The maximum possible array index for a 2-dimensional array and an array size of [3,4] is [2,3].

**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,-30, , , ,-40,-40,-50,-60,-70)
N50 FELD1[8,1]=SET(8.1,8.2,9.0,9.1,9.2)
```



## Further Information

### Initialization with value lists (SET) when defining the array

- As many array elements are assigned as initialization values are programmed.
- Array elements without values (gaps in the value list) are automatically initialized to 0.
- For variables of type AXIS, gaps in the value list are not permitted.
- Programming more values than exist in the remaining array elements triggers an alarm.

### Initializing with value lists (SET) while the program is being executed

In principle, initialization when the program is being executed functions in precisely the same way as initialization when defining the array. However, initializing while the program is being executed offers the following additional possibilities:

- Expressions are also permitted as values.
- Initialization starts at the programmed array indexes. 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)	; Assigning the programmed value list from array index [0,0].
ARRAY[2,3]=SET(VARIABLE,4*5.6)	; Assigning the programmed expressions from array index [2,3].

For axis variables, the axis index is not run through:

Example:

```
$MA_AX_VELO_LIMIT[1,AX1]=SET(1.1,2.2,3.3)
```

Is equivalent to:

```
$MA_AX_VELO_LIMIT[1,AX1] = 1.1  
$MA_AX_VELO_LIMIT[2,AX1] = 2.2  
$MA_AX_VELO_LIMIT[3,AX1] = 3.3
```

**Initializing with the same values (REP) when defining the array**

- All array elements are assigned the same value (constant).
- Variables of type FRAME cannot be initialized.

Example 1: Setting array variables to a specific value range by range

Program code	Comments
DEF REAL varName[30]=REP(3.5,4)	; Define variable array varName and initialize range varName[0] to varName[3] with the value 3.5.
varName[5]=REP(4.5,10)	; Set range varName[5] to varName[14] to the value 4.

**Setting R parameters range by range to a specific value**

Program code	Comments
R10=REP(2.4,15)	; Set R10 to R24 to the value 2.4.

**Initializing with the same values (REP) while the program is being executed**

- Expressions are possible values in this case too.
- All array elements are initialized to the same value.
- Initialization starts at the programmed array indexes. Values can be assigned selectively to subarrays.

Example:

Program code	Comments
DEF FRAME FRM[10]	; Array definition
FRM[5] = REP(CTRANS (X,5))	;

### Memory requirements

Variable type	Memory requirement per element
BOOL	1 byte
CHAR	1 byte
INT	4 bytes
REAL	8 bytes
STRING	String length + 1
FRAME	~ 400 bytes, depending on the number of axes
AXIS	4 bytes

---

#### Note

The maximum array size determines the size of the memory areas in which the variable memory is managed. This memory should not be set larger than is actually required.

Default: 812 bytes

Recommended setting if no large arrays are defined: 256 bytes

---

## 1.3 Indirect programming

### Function

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>]

### Significance

<ADDRESS> [ . . . ]	Fixed address with extension (index)
<Index>	Index variable, e.g., spindle number, axis, ....

### Examples

#### Example 1: Indirectly programming a spindle number

Direct programming:

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

Indirect programming:

Program code	Comments
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	Comments
FA[U]=300	; Feed rate 300 for axis "U".

Indirect programming:

Program code	Comments
DEF AXIS AXVAR2=U	; Defining a variable, type AXIS and value assignment.
FA[AXVAR2]=300	; Feed rate 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:

Programming	Comments
\$AA_MM[X]	; Read probe measured value (MCS) of axis "X".

Indirect programming:

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

Indirect programming:

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

### 1.3.1 Indirectly programming G codes

#### Function

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

#### Syntax

G [<group>] = <number>

#### Significance

G [ . . . ]	G command with extension (index)
<group>	Index parameter: G function group Type: INT
	<b>Note:</b> Only modal G function groups can be indirectly programmed. G function groups that are effective blockwise are rejected with an alarm.
<number>	Variable for the G code number Type: INT or REAL
	<b>Note:</b> 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	Comments
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	Comments
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, refer to:  
Programming Manual, Fundamentals; Chapter "List of G functions/preparatory functions"

### 1.3.2 Indirectly programming position attributes (BP)

#### Function

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 BP.

#### 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 instruction is required, which would otherwise branch for all possible position attributes.

#### Syntax

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

#### Significance

<POSITIONING COMMAND> [ ]	The following positioning commands can be programmed together with the key word BP:
POS, POSA, SPOS, SPOSA	
Also possible:	
• All axis and spindle identifiers present in the channel: <axis/spindle>	
• Variable axis/spindle identifier AX	
Axis/spindle that is to be positioned	
Key word for positioning	
Parameter 1	
Axis/spindle position as constant or variable	
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	Significance	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
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 instruction 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	Comments
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 Stop Delay Area
N2190 COUPOF(S2,S1)	; Deactivate synchronous spindle coupling
N2200	; Position leading and following spindle
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 spindle
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), Chapter: Replacement of NC functions by subprograms

### 1.3.3 Indirectly programming part program lines (EXECSTRING)

#### Function

Part program command EXECSTRING passes a string as a parameter that already contains the part program line to run.

#### Syntax

```
EXECSTRING (<string_variable>)
```

#### Parameters

EXECSTRING	Transfer of a string variable with the part program line to run
<string_variable>	Parameters with the part program line actually to be executed

---

#### Note

All part program constructions that can be programmed in a part program can be output. This means that PROC and DEF instructions are excluded as well as the general use in INI and DEF files.

---

#### Example: Indirect part program line

Program code	Comments
N100 DEF STRING[100] BLOCK	; String variable to accept part program line
N110 DEF STRING[10] MFCT1="M7"	
N200 EXECSTRING(MFCT1 << "M4711")	; Execute part program line "M7 M4711"
N300 R10=1	
N310 BLOCK="M3"	
N320 IF(R10)	
N330 BLOCK = BLOCK << MFCT1	
N340 ENDIF	
N350 EXECSTRING(BLOCK)	; Execute part program line "M3 M4711"

## 1.4 Assignments

### Function

Values of a suitable type can be assigned to the variables/arithmetic parameters in the program.

### Programming

Assignments to axis addresses (traversing instructions) always require a separate block to variable assignments. Assignment to axis addresses (traverse instructions) must be in a separate block from the variable assignments.

#### Assignment to string variable

CHARs and STRINGs distinguish between upper and lower case.

If you want to include an ' or " in the string, put it in single quotes '...'.

Example:

```
MSG ("Viene lavorata l'ultima figura")
```

displays the text 'Viene lavorata l'ultima figura' on the screen.

The string can contain non-displayable characters if they are specified as binary or hexadecimal constants.

### Example

Program code	Comments
R1=10.518 R2=4 VARI1=45	; Assigning a numerical value.
X=47.11 Y=R2	
R1=R3 VARI1=R4	; Assigning a variable of the appropriate type.
R4=-R5 R7=-VARI8	; Assignment with opposing sign (only permitted for type INT and REAL).

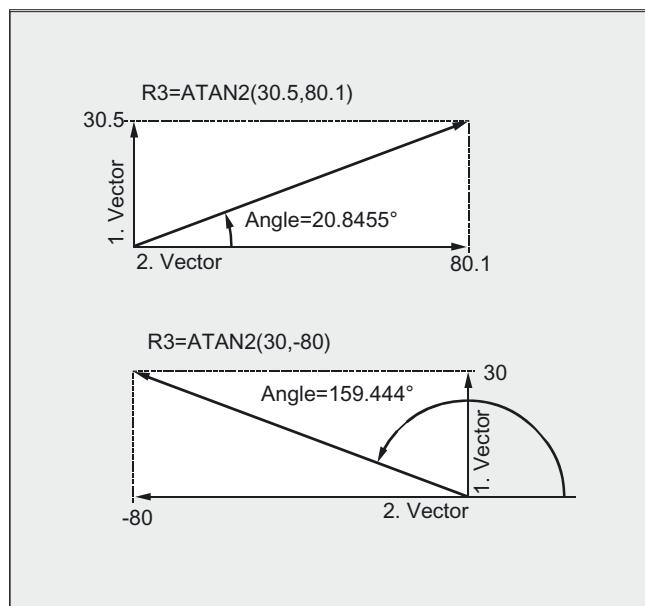
## 1.5 Arithmetic functions

### Function

The arithmetic functions are primarily for R parameters and variables (or constants and functions) of type REAL. The types INT and CHAR are also permitted.

#### Arithmetic function ATAN2( , )

The function calculates the angle of the total vector from two mutually orthogonal vectors. The result is in one of four quadrants ( $-90^\circ < 0 < +180^\circ$ ). The angular reference is always based on the 2nd value in the positive direction.



**The accuracy for comparison commands can be set using TRUNC( )**

See " Accuracy compensation for comparison commands (TRUNC) "

**Variable minimum, maximum and range**

See" Minimum, maximum and range of variables (MINVAL, MAXVAL and BOUND) "

## 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°).

Operator / arithmetic function	Significance
+	Addition
-	Subtraction
*	Multiplication
/	Division
DIV	<b>Notice:</b> (type INT)/(type INT)=(type REAL); example: 3/4 = 0.75 Division, for variable type INT and REAL
MOD	<b>Notice:</b> (type INT)DIV(type INT)=(type INT); example: 3 DIV 4 = 0 Modulo division (only for type INT) supplies remainder of an INT division Example: 3 MOD 4 = 3
:	Chain operator (for FRAME variables)
Sin()	Sine
COS()	Cosine
TAN()	Tangent
ASIN()	Arcsine
ACOS()	Arccosine
ATAN2( , )	Arctangent2
SQRT()	Square root
ABS()	Absolute value
POT()	2. power (square)
TRUNC()	Truncate to integer
ROUND()	Round to integer
LN()	Natural logarithm
EXP()	Exponential function
MINVAL()	Lower value of two variables
MAXVAL()	Larger value of two variables
BOUND()	Variable value within the defined value range
CTRANS()	Translation
CROT()	Rotation
CSCALE()	Change of scale
CMIRROR()	Mirroring

### Example: Initializing complete variable arrays

Program code	Comments
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	; Parentheses are calculated first.
R15=SQRT (POT(R1)+POT(R2))	; Inner parentheses are resolved first: R15 = square root of (R1+R2)
RESFRAME= FRAME1:FRAME2	; The concatenation operator links frames to form a resulting frame or assigns values to frame components.
FRAME3=CTRANS (...) :CROT (...)	

## 1.6 Comparison and logic operations

### Function

**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 logic operators

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

## Programming

<b>Relational operator</b>	<b>Significance</b>
<code>==</code>	equal to
<code>&lt;&gt;</code>	not equal to
<code>&gt;</code>	greater than
<code>&lt;</code>	less than
<code>&gt;=</code>	greater than or equal to
<code>&lt;=</code>	less than or equal to

<b>Logic operator</b>	<b>Significance</b>
<code>AND</code>	AND
<code>OR</code>	OR
<code>NOT</code>	Negation
<code>XOR</code>	Exclusive OR

<b>Bit-by-bit logic operator</b>	<b>Significance</b>
<code>B_AND</code>	Bit-serial AND
<code>B_OR</code>	Bit-serial OR
<code>B_NOT</code>	Bit-serial negation
<code>B_XOR</code>	Bit-serial 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: Relational 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
```

## 1.7 Precision correction on comparison errors (TRUNC)

### Function

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

#### Settable precision for comparison commands

Program data of type REAL are 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 for relative equality.

## Syntax

### Precision correction on comparison errors

TRUNC (R1\*1000)

## Significance

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.

**Example: precision considerations**

<b>Program code</b>	<b>Comments</b>
N40 R1=61.01 R2=61.02 R3=0.01	; Assignment of initial values
N41 IF ABS(R2-R1) > R3 GOTO 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 GOTO ERROR	; Jump is no longer executed
M30	; End of program
ERROR: SETAL(66000)	;

**Example: calculate and evaluate the quotient of both operands**

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

## 1.8 Variable minimum, maximum and range (MINVAL, MAXVAL and BOUND)

### Function

The MINVAL and MAXVAL functions can be used to compare two variables and return either the smaller or the larger value as the result.

The BOUND function can be used to test whether the value of a test variable falls within the defined value range. If this is the case, the variable value is returned. If the test variable value is larger than the maximum value, the maximum value is output. Similarly, if the minimum value is undershot, the minimum value is output.

#### Behavior if values are equal

If the values are equal, the MINVAL and MAXVAL functions output this equal value, while BOUND returns the value of the variable to be tested.

### Syntax

#### Minimum

```
Result smaller value = <MINVAL>(<Variable1>, <Variable2>)
```

#### Maximum

```
Result larger value = <MAXVAL>(<Variable1>, <Variable2>)
```

#### Range

```
Return value = <BOUND>(<Variable min>, <Variable max>, <Variable test>)
```

#### Note

The three MINVAL( ), MAXVAL( ) and BOUND( ) functions can also be programmed as synchronized actions.

## Significance

MINVAL ()	Determines the lower value of two variables
MAXVAL ()	Determines the larger value of two variables
BOUND ()	Tests whether the variable to be tested falls within a defined min./max. value range
Variable1, Variable2	First and second variable whose values are tested against the minimum/maximum
Variable min	Lower defined limit for the test variable value
Variable max	Upper defined limit for the test variable value
Variable test	Variable used to test whether its value falls within the defined range

## Examples

Program code	Comments
DEF REAL rVar1 = 10.5, rVar2 = 33.7, rVar3, rVar4, rVar5, rValMin, rValMax, rRetVar	
rValMin = MINVAL(rVar1, rVar2)	; rValMin set to value 10.5
rValMax = MAXVAL(rVar1, rVar2)	rValMax set to value 33.7
rVar3 = 19.7	rVar3 falls within the limits, rRetVar set to 19.7
rRetVar = BOUND(rVar1, rVar2, rVar3)	
rVar3 = 1.8	
rRetVar = BOUND(rVar1, rVar2, rVar3)	rVar3 falls below the lower limit, rRetVar set to 10.5
rVar3 = 45.2	rVar3 exceeds the upper limit, rRetVar set to 33.7
rRetVar = BOUND(rVar1, rVar2, rVar3)	

## 1.9 Priority of the operations

### Function

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-serial negation
2.	*, /, DIV, MOD	Multiplication, division
3.	+, -	Addition, subtraction
4.	B_AND	Bit AND
5.	B_XOR	Bit-serial exclusive OR
6.	B_OR	Bit-serial 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
```

## 1.10 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 <sup>1)</sup>	yes*	–	–	–
INT	yes	yes	Yes <sup>1)</sup>	Yes <sup>2)</sup>	–	–	–
BOOL	yes	yes	yes	yes	yes	–	–
CHAR	yes	yes	Yes <sup>1)</sup>	yes	yes	–	–
STRING	–	–	Yes <sup>4)</sup>	Yes <sup>3)</sup>	yes	–	–
AXIS	–	–	–	–	–	yes	–
FRAME	–	–	–	–	–	–	yes

#### Explanation

- \* At type conversion from REAL to INT, fractional values that are  $\geq 0.5$  are rounded up, others are rounded down (cf. ROUND function).
- 1) Value  $\neq 0$  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".

---

## 1.11 String operations

### String operations

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

- Type conversion to STRING (AXSTRING)
- Type conversion from STRING (NUMBER, ISNUMBER, AXNAME)
- Concatenation of strings (<<)
- Converting into lower/upper case letters (TOLOWER, TOUPPER)
- Define string length (STRLEN)
- Search for character/string in a string (INDEX, RINDEX, MINDEX, MATCH)
- Select a partial string (SUBSTR)
- Select an individual character (STRINGVAR, STRINGFELD)

### 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	Comments
DEF STRING[20] STRG="axis . stationary"	
STRG[6]="X"	
MSG (STRG)	; Supplies the message "axis X stationary".
STRG[6]=0	
MSG (STRG)	; Supplies the message "axis".

### 1.11.1 Type conversion to STRING (AXSTRING)

#### Function

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

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

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_ERG>=<<<any_type>
<STRING_ERG>=AXSTRING(<axis identifier>)
```

#### Significance

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

---

#### Note

FRAME variables cannot be converted.

---

#### Examples

##### Example 1:

```
MSG ("Position:<<$AA_IM[X])
```

##### Example 2: AXSTRING

Program code	Comments
DEF STRING[32] STRING_ERG STRING_ERG=AXSTRING(X)	; STRING_ERG == "X"

## 1.11.2 Type conversion from STRING (NUMBER, ISNUMBER, AXNAME)

### Function

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_ERG>=NUMBER("<string>")
<BOOL_ERG>=ISNUMBER("<string>")
<AXIS_ERG>=AXNAME("<string>")
```

### Significance

NUMBER	The NUMBER command returns the number represented by the <string> as REAL value.	
<string>	Type STRING variable to be converted	
<REAL_ERG>	Variable for the result of the type conversion with NUMBER Type: REAL	
ISNUMBER	The ISNUMBER command can be used to check whether the <string> can be converted into a valid number.	
<BOOL_ERG>	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.  <b>Note:</b> If the <string> cannot be assigned a configured axis identifier, an alarm is initiated.	
<AXIS_ERG>	Variable for the result of the type conversion with AXNAME Type: AXIS	

## Example

Program code	Comments
DEF BOOL BOOL_ERG	
DEF REAL REAL_ERG	
DEF AXIS AXIS_ERG	
BOOL_ERG=ISNUMBER("1234.9876Ex-7")	; BOOL_ERG == TRUE
BOOL_ERG=ISNUMBER("1234XYZ")	; BOOL_ERG == FALSE
REAL_ERG=NUMBER("1234.9876Ex-7")	; REAL_ERG == 1234.9876Ex-7
AXIS_ERG=AXNAME("X")	; AXIS_ERG == X

### 1.11.3 Concatenation of strings (<<)

#### Function

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>

#### Significance

<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]<<BAUSTEIN_NAME)
```

### CAUTION

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	Comments
<pre>DEF INT IDX=2 DEF REAL VALUE=9.654 DEF STRING[20] STRG="INDEX:2" IF STRG=="Index:"&lt;&lt;IDX GOTO NO_MSG MSG ("Index:"&lt;&lt;IDX&lt;&lt;"/value:"&lt;&lt;VALUE) NO_MSG:</pre>	<p>; Didplay: "Index:2/value:9.654"</p>

### Example 2: Explicit type conversion with <<

Program code	Comments
<pre>DEF REAL VALUE=3.5 &lt;&lt;VALUE</pre>	<p>; The specified type REAL variable is converted into a STRING type.</p>

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

#### Function

The "conversion to lower/upper case letters" function allows all of the letters of a string to be converted into a standard representation.

#### Syntax

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

#### Significance

TOUPPER	Using the TOUPPER command, all of the letters in a character string are converted into <b>upper case</b> letters.
TOLOWER	Using the TOLOWER command, all of the letters in a character string are converted into <b>lower case</b> letters.
<string>	Character string that is to be converted
	Type: STRING
<STRING_ERG>	Variable for the result of the conversion
	Type: STRING

#### Example

Because user inputs can be initiated on the operator interface, they can be given standard capitalization (upper or lower case):

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

### 1.11.5 Determine length of string (STRLEN)

#### Function

The STRLEN command can be used to determine the length of a character string.

#### Syntax

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

#### Significance

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_ERG>	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(BAUSTEIN_NAME)>10) GOTOF ERROR
```

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

#### Function

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_ERG = INDEX	(STRING,CHAR)	Result type: INT
INT_ERG = RINDEX	(STRING,CHAR)	Result type: INT
INT_ERG = MINDEX	(STRING,STRING)	Result type: INT
INT_ERG = MATCH	(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.

#### Significance

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 string into path and block names

Program code	Comments
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") + 1	; The value returned in LISTIDX is 3; because "N" is the first character in the parameter INPUT 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
VARIABLE = SUBSTR (INPUT, PFADIDX, PROGIDX-PFADIDX-1)	The SUBSTR function introduced in the next section can be used to break-up variable INPUT in the components "path" and "module": ; Then returns "_N_MPF_DIR"
VARIABLE = SUBSTR (INPUT, PROGIDX)	; Then returns "_N_EXECUTE_MPF"

### 1.11.7 Selection of a substring (SUBSTR)

#### Function

This functionality extracts a substring from a string. For this purpose, the index of the first character and the desired string length (if applicable) are specified. If no length information is specified, then the string data refers to the remaining string.

#### Syntax

STRING_ERG = SUBSTR (STRING,INT)	<b>Result type:</b> INT
STRING_ERG = SUBSTR(STRING,INT, INT)	<b>Result type:</b> INT

#### Semantics

In the first case, the substring from the position specified by the second parameter is returned up to the end of the string.

In the second case, the result string is limited to the maximum length, specified by the third parameter.

If the initial position is after the end of the string, the empty string (" ") will be returned.

If the initial position or the length is negative, an alarm is output.

#### Example

Program code	Comments
<pre>DEF STRING[29] ERG ERG = SUBSTR ("ACK:10 to 99", 10, 2)</pre>	<p>; Therefore the following applies: ERG == "10"</p>

## 1.11.8 Selection of a single character (STRINGVAR, STRINGFELD)

### Function

This functionality selects a single character from a string. This applies both to read access and write access operations.

### Syntax

CHAR_ERG = STRINGVAR [IDX]	<b>Result type:</b> CHAR
CHAR_ERG = STRINGARRAY [IDX_FELD, IDX_CHAR]	<b>Result type:</b> CHAR

### Semantics

The character at the specified position is read/written within the string. If the position parameter is negative or greater than the string, then an alarm is output.

#### Example messages:

Insertion of an axis identifier into a prepared string.

Program code	Comments
<pre>DEF STRING [50] MESSAGE = "Axis n has reached position" MESSAGE [6] = "X" MSG (MESSAGE)</pre>	<p>; Returns the message "axis X has reached position"</p>

### Parameters

Single character access is possible only to user-defined variables (LUD, GUD, and PUD data).

This type of access is also possible only for "call-by-value" type parameters in subprogram calls.

**Example: single character access to a system, machine data, ...:**

Program code	Comments
<pre>DEF STRING [50] STRG DEF CHAR ACK ... STRG = \$P_MMCA ACK = STRG [0]</pre>	<p style="text-align: right;">; Evaluating the acknowledgement components</p>

**Example: single character access in call-by-reference parameter:**

Program code	Comments
<pre>DEF STRING [50] STRG DEF CHAR CHR1 EXTERN UP_CALL (VAR CHAR1) ... CHR1 = STRG [5] UP_CALL (CHR1) STRG [5] = CHR1</pre>	<p style="text-align: right;">; Call-by-reference parameters!</p> <p style="text-align: right;">; Call-by-reference</p>

## 1.12 Program jumps and branches

### 1.12.1 Return jump to the start of the program (GOTOS)

#### Function

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

Machine data can be used to set that for every return jump to the beginning of the program:

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

#### Syntax

GOTOS

#### Significance

GOTOS	Jump instruction where the destination is the beginning of the program. The execution is controlled via the NC/PLC interface signal: DB21, ... DBX384.0 (...) Value: Significance:
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.

#### Example

Program code	Comments
N10 ...	; Beginning of the program
...	
N90 GOTOS	; Jump to beginning of the program
...	

## 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.

## References

Function Manual Basic Functions; BAG, Channel, Program Operation, Reset Response (K1),  
Chapter: "Program operation" > "Program jumps" > "Return jump to the beginning of the program"

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

### Function

Jump markers (labels) can be set in a program, that 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 instruction that immediately follows the destination marker. This means that branches can be realized within the program.

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

If a jump condition (**IF** ...) is formulated before the jump instruction, 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>
```

## Significance

GOTOB	Jump instruction with jump destination towards the beginning of the program.
GOTOF	Jump instruction with jump destination towards the end of the program.
GOTO	Jump instruction 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 marker>      Jump destination is the jump marker (label) set in the program with a user-defined name: <jump marker>:  <block number>      Jump destination is main block or sub-block number (e.g.: 200, N300)  Variable, type      Variable jump destination. The variable stands STRING      for a jump marker 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 markers (labels)

Jump markers (labels) 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:
    - At least 2
    - A maximum of 32
  - Permitted characters include:
    - Letters
    - Numbers
    - Underscore symbols
  - The first two characters must be letters or underscores.
  - The name of the jump marker is followed by a colon ("").
-

## Examples

### Example 1: Jumps to jump markers

Program code	Comments
N10 ...	
N20 GOTOF Label_1	; Jump towards the end of the program to the jump marker "Label_1".
N30 ...	
N40 Label_0: R1=R2+R3	; Jump marker "Label_0" set.
N50 ...	
N60 Label_1:	; Jump marker "Label_1" set.
N70 ...	
N80 GOTOB Label_0	; Jump in the direction of the beginning of the program to the jump marker "Label_0".
N90 ...	

### Example 2: Indirect jump to the block number

Program code	Comments
N5 R10=100	
N10 GOTOF "N"<<R10	; Jump to the block whose block number is located in R10.
...	
N90 ...	
N100 ...	; Jump destination
N110 ...	
...	

### Example 3: Jump to variable jump destination

Program code	Comments
DEF STRING[20] DESTINATION	
DESTINATION = "Marker2"	
GOTOF DESTINATION	; Jump in the direction of the end of the program to the variable jump destination DESTINATION.
Marker 1: T="Drill1"	
...	
Marker 2: T="Drill2"	; Jump destination
...	

### Example 3: Jump with jump condition

Program code	Comments
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 marker LA1 set.
N42 R1=R1+R3 R4=R4-1	
N43 IF R4>0 GOTOB LA1	; If the jump condition is fulfilled, then a jump in the direction of the beginning of the program to jump marker LA1.
N44 M30	; End of program

### Supplementary conditions

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

### References

/PGA/ Job Planning Programming Manual; Chapter "Flexible NC Programming" >  
"Comparison and Logical Operations"

### 1.12.3 Program branch (CASE, DEFAULT)

#### Function

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 destination_1>
<constant_2> GOTOF <jump destination_2> ... DEFAULT GOTOF <jump
destination_n>
```

## Significance

CASE	Jump statement
<expression>	Variable or arithmetic function
OF	Keyword to formulate conditional program branches.
<constant_1>	First specify constant value for the variable or arithmetic function
Ty INT pe :	
<constant_2>	Second specified constant value for the variable or arithmetic function
Ty INT pe :	
DEFAULT	For the cases where the variable or arithmetic function does not assume any of the specified constant values, the DEFAULT instruction can be used to determine the branch destination.  <b>Note:</b> If the DEFAULT instruction is not programmed, then in these cases, the block following the CASE instruction is the jump destination.
GOTOF	Jump instruction with jump destination 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 destination_1>	A branch is made to this jump destination if the value of the variable or arithmetic function corresponds to the first specific constant.  The jump destination can be specified as follows:
<jump marker>	Jump destination is the jump marker (label) set in the program with a user-defined name: <jump marker>:
<block number>	Jump destination is main block or sub-block number (e.g.: 200, N300)
Variable, type STRING	Variable jump destination. The variable stands for a jump marker or a block number.
<jump destination_2>	A branch is made to this jump destination if the value of the variable or arithmetic function corresponds to the second specified constant.
<jump destination_n>	A branch is made to this jump destination 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 GOTO Label_1 9 GOTO Label_2 DEFAULT GOTO Label_3 N40 Label_1: G0 X1 Y1 N50 Label_2: G0 X2 Y2 N60 Label_3: G0 X3 Y3 ...

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

1. If the value of the arithmetic function  $\text{VAR1}+\text{VAR2}-\text{VAR3} = 7$ , then jump to the block with the jump marker definition "Label\_1" ( $\rightarrow$  N40).
2. If the value of the arithmetic function  $\text{VAR1}+\text{VAR2}-\text{VAR3} = 9$ , then jump to the block with the jump marker definition "Label\_2" ( $\rightarrow$  N50).
3. If the value of the arithmetic function  $\text{VAR1}+\text{VAR2}-\text{VAR3}$  is neither 7 nor 9, then jump to the block with the jump marker definition "Label\_3" ( $\rightarrow$  N60).

## 1.13 Repeating program sections (REPEAT, REPEATB)

### Function

Program section repetition allows you to repeat existing program sections within a program in any order. The block or program sections to be repeated are identified by labels.

For labels (markers), refer to Chapter "Program jumps and branches" and "Check structures"

#### Syntax: Repeat block

```
LABEL: xxx  
YYY  
REPEATB LABEL P=n  
ZZZ
```

The program line identified by a label is repeated P=n times. If P is not specified, the program section is repeated exactly once. After the last repetition, the program is continued at the line zzz following the REPEATB line.

The block identified by the label can appear before or after the REPEATB statement. The search initially commences toward the start of the program. If the label is not found in this direction, the search continues toward the end of the program.

#### Syntax: Repeat area starting at label

```
LABEL: xxxx  
YYY  
REPEAT LABEL P=n  
ZZZ
```

The program section between the label with any name and the REPEAT statement is repeated P=n times. If the block with the label contains further statements, these are executed again on each repetition. If P is not specified, the program section is repeated exactly once. After the last repetition, the program is continued at the line zzz following the REPEAT line.

---

#### Note

The label must appear before the REPEAT statement. The search is performed toward the start of the program only.

---

**Syntax: Repeat an area between two labels**

```
START_LABEL: xxx
ooo
END_LABEL: yyy
ppp
REPEAT START_LABEL END_LABEL P=n
zzz
```

The area between the two labels is repeated P=n times. User-defined names can be assigned to the labels. The first line of the repetition contains the start label, the last line contains the end label. If the line containing the start or end label contains further statements, these are executed again on each run. If P is not specified, the program section is repeated once. After the last repetition, the program is continued at the line zzz following the REPEAT line.

**Note**

The program section to be repeated can appear before or after the REPEAT statement. The search initially commences toward the start of the program. If the start label is not found in this direction, the search resumes from the REPEAT statement toward the end of the program.

It is not possible to nest the REPEAT statement with the two labels within parentheses. If the start label is found before the REPEAT statement and the end label is not reached before the REPEAT statement, the repetition is performed on the section between the start label and the REPEAT statement.

**Syntax: Repeat an area between a label and the end label**

```
LABEL: xxx
ooo
ENDLABEL: yyy
REPEAT LABEL P=n
zzz
```

ENDLABEL is a predefined label with a fixed name. ENDLABEL marks the end of a program section and can be used multiple times in the program. The block marked by ENDLABEL can contain further statements.

The area between a label and the following ENDLABEL is repeated P=n times. Any name can be used to define the start label. If the block with the start label or ENDLABEL contains further statements, these are executed on each repetition.

**Note**

If no ENDLABEL is found between the start label and the block with the REPEAT call, the loop ends before the REPEAT line. The construct therefore has the same effect as described above in "repeat area from label".

If P is not specified, the program section is repeated once.

After the last repetition, the program is continued at the line zzz following the REPEAT line.

**Significance**

LABEL:	Jump destination; the name of the jump destination is followed by a colon
REPEAT	Repeat (repeat several lines)
REPEATB	Repeat block (repeat one line only)

**Example: Repeat positions**

Program code	Comments
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: 5 squares with increasing width are to be produced**

Program code	Comments
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 area from N10 to N70 four times
N90 Z10	
N100 M30	

**Example: Example repeat program section from BEGIN to END**

<b>Program code</b>	<b>Comments</b>
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 END:Z=10	
N80 Z10	
N90 CYCLE(10,20,30)	
N100 REPEAT BEGIN END P=3	; Execute area from N10 to N70 three times
N110 Z10	
N120 M30	

**Example: ENDLABEL**

<b>Program code</b>	<b>Comments</b>
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 area from N110 to N120 three times
N140 REPEAT BEGIN2 P=2	; Execute area from N50 to N80 twice
N150 M100	
N160 REPEAT BEGIN1 P=2	; Execute area from N20 to N80 twice
N170 Z10	
N180 X0 Y0	
N190 M30	

**Example: Milling: Machine drill position with different technologies**

<b>Program code</b>	<b>Comments</b>
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	
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	

## Supplementary conditions

- 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 aborted. 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

Program section repetition is activated by programming.

The *REPEAT* instruction should be placed behind the traveling blocks.

---

## 1.14 Check structures

### Function

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 check structure elements (key words) IF...ELSE, LOOP, FOR, WHILE and REPEAT.

 **CAUTION**

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.

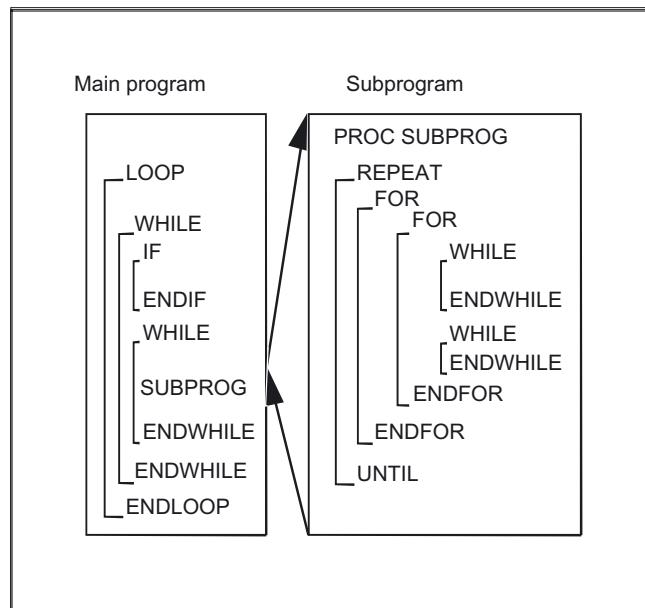
It is not permissible to superimpose macros on keywords for check structures or on branch destinations. No such check is made when the macro is defined.

### Effectiveness

Check structures apply locally within programs.

### 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.

### **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.

### 1.14.1 Program loop with alternative (IF, ELSE, ENDIF)

#### Function

A construction with **IF** and **ELSE** is used if the program loop should contain an alternative program block: If the **IF** condition is fulfilled, then the program block following the **IF** is executed. If the **IF** condition is **not** fulfilled, then the alternative program block following **ELSE** is executed.

---

#### Note

If an alternative is not required, then an IF loop can be programmed also without **ELSE** instruction and the program block following **ELSE**.

---

#### Syntax

```

| IF <condition>
| ...
| ELSE
| ...
| ENDIF
| 
```

#### Significance

<b>IF</b>	Introduces the IF loop.
<b>ELSE</b>	Introduces the alternative program block.
<b>ENDIF</b>	Marks the end of the IF loop and results in a return jump to the beginning of the loop.
<condition>	Condition that determines which program block is executed.

**Example: Tool change subprogram**

<b>Program code</b>	<b>Comments</b>
PROC L6	; Tool change routine
N500 DEF INT TNR_AKTUELL	; Variable for active T number
N510 DEF INT TNR_VORWAHL	; Variable for pre-selected T number
	; Determine current tool
N520 STOPRE	
N530 IF \$P_IESTEST	; 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	

## 1.14.2 Continuous program loop (LOOP, ENDLOOP)

### Function

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
```

### Significance

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") M0 STOPRE ENDLOOP ...

### 1.14.3 Count loop (FOR ... TO ..., ENDFOR)

#### Function

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

```

#### Significance

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
	<b>Note:</b>
	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
<end value>	Condition: The start value must be lower than the end value. End value of the count

## Examples

### Example 1: INTEGER variable or R parameter as count variable

#### INTEGER variable as count variable:

Program code	Comments
<pre>DEF INT iVARIABLE1 R10=R12-R20*R1 R11=6 FOR iVARIABLE1 = R10 TO R11      ; Count variable = INTEGER variable   R20=R21*R22+R33 ENDFOR M30</pre>	

#### R parameter as count variable:

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

### Example 2: Production of a fixed quantity of parts

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

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

#### Function

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

```

#### Significance

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	Comments
<pre> ... WHILE \$AA_IW[DRILL_AXIS] &gt; -10       G1 G91 F250 AX[DRILL_AXIS] = -1 ENDWHILE ... </pre>	<p>; Call the WHILE loop under the following condition: The actual WCS setpoint for the drilling axis must be greater than -10.</p>

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

#### Function

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>
```

#### Significance

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	Comments
...	
REPEAT	; Call the REPEAT loop.
...	
UNTIL ...	; Check whether the condition is fulfilled.
...	

### 1.14.6 Program example with nested check structures

Program code	Comments
LOOP IF NOT \$P_SEARCH G01 G90 X0 Z10 F1000 ; No block search WHILE \$AA_IM[X] <= 100 G1 G91 X10 F500 ; Hole drilling template Z-F100 Z5 ENDWHILE Z10 ELSE MSG("No drilling during block search") ENDIF \$A_OUT[1] = 1 ; Next drilling plate G4 F2 ENDLOOP M30	

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

### Function

#### Channels

A channel can process its own program independently of other channels. It can control the axes and spindles temporarily assigned to it via the program.

Two or more channels can be set up for the control during startup.

#### Program coordination

If several channels are involved in the machining of a workpiece it may be necessary to synchronize the programs.

There are special statements (commands) for this program coordination. Each statement is programmed separately in a block.

#### Note

Program coordination is also possible in its own channels.

### Program coordination statements

- Specification with absolute path

The absolute path is programmed according to the following rules:

INIT (n,"/\_HUGO\_DIR/\_N\_name\_MPF" )  
or

INIT (n,"/\_N\_MPF\_DIR/\_N\_name\_MPF" )

- Current directory/\_N\_name\_MPF  
"current directory" stands for the selected workpiece directory or the standard directory/\_N\_MPF\_DIR.
- Selects a particular program for execution in a particular channel:  
n: Number of the channel, the value depends on the control configuration
- Complete program name

**Example:**

INIT(2,"/\_N\_WKS\_DIR/\_DRESS\_MPF")

G01F0.1

START

INIT

(2,"/\_N\_WKS\_DIR/\_N\_UNDER\_1\_SPF")

- Relative path specification

**Up to SW 3:**

At least one executable block must be programmed between an **init** command (without synchronization) and an **NC start**. With subprogram calls "\_SPF" must be added to the path.

**Example:**

INIT(2,"DRESS")

INIT(3,"UNDER\_1\_SPF")

The same rules apply to relative path definition as for program calls.

With subprogram calls "\_SPF" must be added to the program name.

**Parameters**

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.

INIT(n, path name, acknowledgement mode)	Instruction for execution in a channel. Selection of a particular program with an absolute or relative path name.
START (n, n)	Starts the selected programs in the other channels. n,n: Enumeration of the channel numbers: value depends on control configuration
WAITM (marker no., n, n, ...)	Sets the marker "marker no." in the same channel. Terminate previous block with exact stop. Waits for the markers with the same "marker no." in the specified channels "n" (current channel does not have to be specified). Marker is deleted after synchronization. 10 markers can be set per channel simultaneously.

WAITMC (marker No., n, n,	Sets the marker "marker no." in the same channel. An exact stop is initiated only if the other channels have not yet reached the marker. Waits for the marker with the same "marker No." in the specified channels "n" (current channel does not have to be specified). As soon as marker "marker no." in the specified channels is reached, continue without terminating exact stop.
WAITE (n, n, ...)	Waits for the end of program of the specified channels (current channel not specified) Example: programming a delay time after the Start command.  N30 START(2) N31 G4 F0.01 N40 WAITE(2)
SETM (marker No., marker No.,	Sets the markers "marker no." in the same channel without affecting current processing. SETM() remains valid after RESET and NC START.
CLEARM (marker No., marker No.,	Deletes the markers "Marker No." in the same channel without affecting current processing. All markers can be deleted with CLEARM(). CLEARM (0) deletes the marker "0". CLEARM() remains valid after RESET and NC START.
n	Corresponding channel number or channel name

---

### Note

All the above commands must be programmed in separate blocks.

The number of markers depends on the CPU used.

---

### Channel numbers

Up to 10 channels can be specified as channel numbers (integer value) for the channels requiring coordination.

### Channel names

Channel names must be converted into numbers using variables (see "Variables and arithmetic parameters"). Alternatively, the channel names defined using \$MC\_CHAN\_NAME (identifier or keyword) can also be programmed rather than channel numbers. The defined names must comply with the NC naming conventions (i.e., the first two characters must be either letters or an underscore).

#### CAUTION

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

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

### SETM() and CLEARM()

SETM() and CLEARM() can also be programmed independently of a synchronized action.  
See Chapter "Set/delete wait markers: SETM CLEARM"

### Example

Channel called "MACHINE" is to contain channel number 1,  
channel called "LOADER" is to contain channel number 2:

DEF INT MACHINE=1, LOADER=2

The variables are given the same names as the channels.

The statement START is therefore:

START (MACHINE)

## Example: program coordination

### Channel 1:

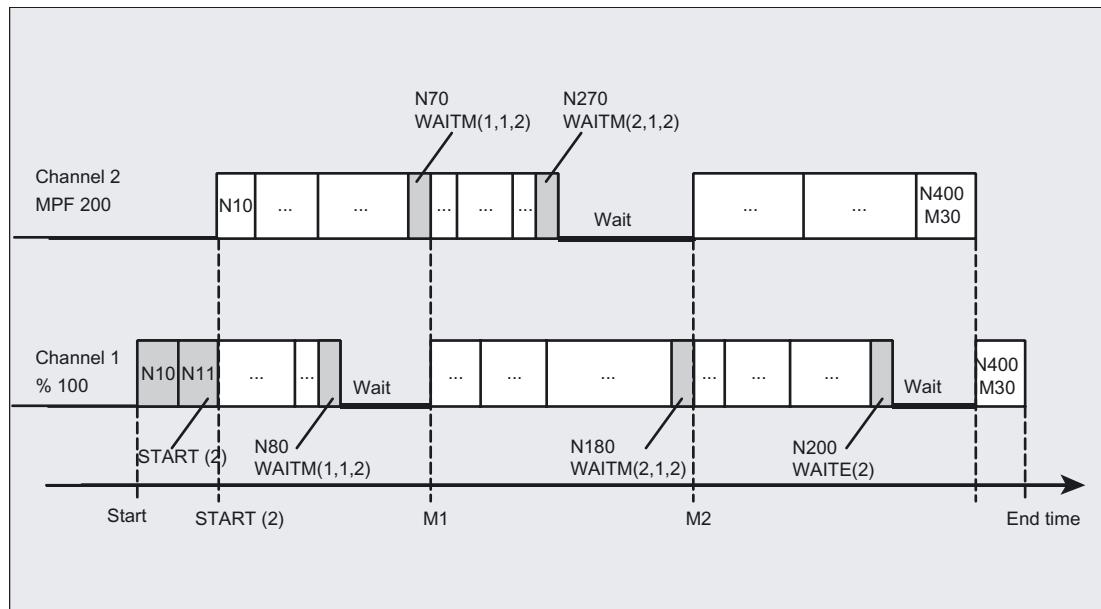
\_N\_MPFI00\_MPFI

Program code	Comments
N10 INIT(2,"MPF200")	
N11 START(2)	; Processing in channel 2
...	
N80 WAITM(1,1,2)	; Wait for WAIT marker 1 in channel 1 and in channel 2 additional processing in channel 1
...	
N180 WAITM(2,1,2)	; Wait for WAIT marker 1 in channel 2 and in channel 2 additional processing in channel 1
...	
N200 WAITE(2)	; Wait for the end of program for channel 2
N201 M30	; End of program channel 1, total end
...	

### Channel 2:

\_N\_MPFI00\_MPFI

Program code	Comments
;\$PATH=/_N_MPFI_DIR	; Processing in channel 2
N70 WAITM(1,1,2)	; Wait for WAIT marker 2 in channel 1 and in channel 2 additional processing in channel 1
...	
N270 WAITM(2,1,2)	; Wait for WAIT marker 2 in channel 1 and in channel 2 additional processing in channel 2
...	
N400 M30	; End of program in channel 2



### Example: Program from workpiece

Program code
N10 INIT(2, "/_N_WKS_DIR/_N_SHAFT1_WPD/_N_CUT1_MPF")

### Example: INIT command with relative path specification

Program /\_N\_MPF\_DIR/\_N\_MAIN\_MPF is selected in channel 1

Program code	Comments
N10 INIT(2, "MYPROG") ; Select program /_N_MPF_DIR/_N_MYPROG_MPF in channel 2	

**Example: Channel name and channel number with integer variable**

```
$MC_CHAN_NAME[0] = "CHAN_X" ;name of 1st channel
$MC_CHAN_NAME[1] = "CHAN_Y" ;name of 2nd channel
```

Program code	Comments
START(1, 2)	; Execute start in the 1st and 2nd channels

Similar to this, programming with the channel identifiers:

Program code	Comments
START(CHAN_X, CHAN_Y)	; Execute start in the 1st and 2nd channels ; The channel_X and channel_Y identifiers represent ; channel numbers 1 and 2 internally, due to the \$MC_CHAN_NAME machine data. They also execute a start in the 1st and 2nd channel accordingly.

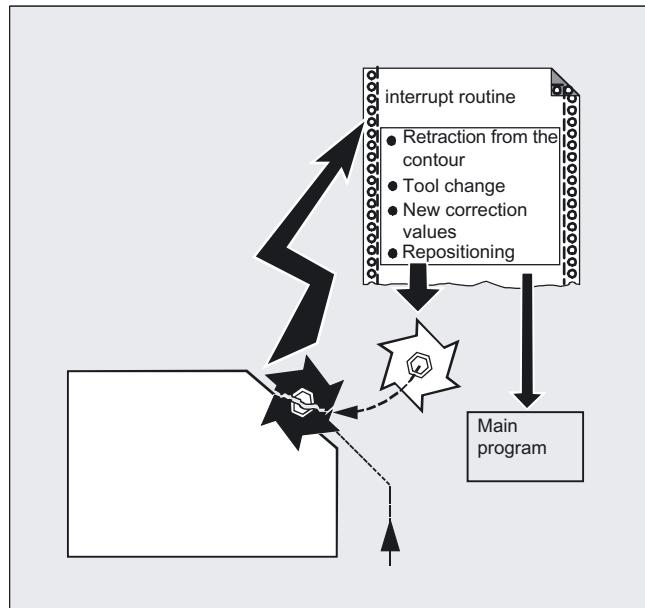
Programming with integer variable:

Program code	Comments
DEF INT chanNo1, chanNo2 chanNo1=CHAN_X chanNo2=CHAN_Y START(chanNo1, chanNo2)	; Define channel number

## 1.16 Interrupt routine (ASUB)

### 1.16.1 Function of an interrupt routine

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 has finished being executed 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 502) ").

#### CAUTION

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.

## 1.16.2 Creating an interrupt routine

### Create interrupt routine as subprogram

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

Example:

Program code	Comments
PROC LIFT_Z	; Program name "ABHEB_Z"
N10 ...	; The NC blocks then follow:
...	
N50 M17	; At the end, the end of 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 saved before calling the interrupt routine and are re-activated after the end of the interrupt routine (see "Subprograms with SAVE mechanism (SAVE) (Page 138)".

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 instructions can be programmed within the interrupt routine (see "Assign and start interrupt routine (SETINT)" (Page 102)) therefore activating additional interrupt routines. They are triggered via the input.

## References

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

### 1.16.3 Assign and start interrupt routine (SETINT, PRIO)

#### Function

The control has signals (inputs 1...8) that initiate that the program being executed is interrupted and a corresponding interrupt routine can be started.

The assignment as to which input starts which program is realized in the part program using the SETINT command.

If several SETINT instructions are in the part program and therefore several signals can be simultaneously received, the assigned interrupt routines must be allocated priorities that define the sequence in which the interrupt routines are executed: PRIO=<value>

If new signals are received while interrupt routines are being executed, the current interrupt routines are interrupted by routines with higher priority.

#### Syntax

SETINT (<n>) PRIO=<value> <NAME>

#### Significance

SETINT (<n>)	Command: Assign input <n> to an interrupt routine. The assigned interrupt routine starts when input <n> switches.
	<b>Note:</b> If an input that is already assigned is allocated to a new routine, then the old assignment is automatically cancelled.
<n>	Parameters: Input number Type: INT Range of values: 1 ... 8
PRIO=	Command: Defining the priority <value> Priority value Type: INT Range of values: 1 ... 128 Priority 1 corresponds to the highest priority.
<NAME>	Name of the subprogram (interrupt routine) that should be executed.

## Examples

### Example 1: Assign interrupt routines and define the priority

Program code	Comments
...	
N20 SETINT(3) PRIO=1 ABHEB_Z	; If input 3 switches, then interrupt routine "ABHEB_Z" should start.
N30 SETINT(2) PRIO=2 ABHEB_X	; If input 2 switches, then interrupt routine "ABHEB_X" should start.
...	

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	Comments
...	
N20 SETINT(3) PRIO=2 ABHEB_Z	; If input 3 switches, then interrupt routine "ABHEB_Z" should start.
...	
N120 SETINT(3) PRIO=1 LIFT_X	; Input 3 is assigned to a new interrupt routine: Instead of "ABHEB_Z", "ABHEB_X" should start if input 3 switches.

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

#### Function

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

#### Syntax

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

#### Significance

DISABLE (<n>)	Command: <b>Deactivating</b> the interrupt routine assignment of input <n>
ENABLE (<n>)	Command: <b>Reactivating</b> the interrupt routine assignment of input <n>
<n>	Parameters: Input number
	Type: INT
	Range of values: 1 ... 8

#### Example

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

### 1.16.5 Delete assignment of interrupt routine (CLRINT)

## Function

An input → interrupt routine defined using SETINT can be deleted with CLRINT.

## Syntax

CLRINT (<n>)

## Significance

**CLRINT (<n>)** Command: Deleting the interrupt assignment of input <n>  
<n> Parameters: Input number  
Type: INT  
Range of values: 1 ... 8

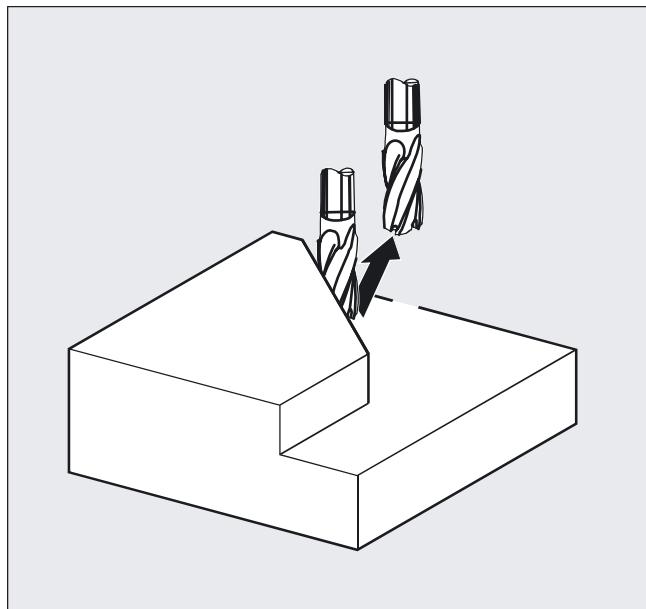
## Example

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

### 1.16.6 Fast retraction from the contour (SETINT LIFTFAST, ALF)

#### Function

For a SETINT instruction 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 instruction 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
```

## Significance

SETINT (<n>)	Command: Assign input <n> to an interrupt routine. The assigned interrupt routine starts when input <n> switches.
<n>	Parameters: Input number Type: INT Range of values: 1 ... 8
PRIOR=	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 should be executed.
LIFTFAST	Command: Fast retraction from 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 108)".

## Example

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

### Main program:

Main program	Comments
N10 SETINT(1) PRIO=1 W_WECHS LIFTFAST	; When input 1 is switched, the tool is immediately retracted from the contour with rapid lift (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	
N50 G41 G1 X16 Y16 F200	
N60 Y35	
N70 X53 Y65	
N90 X71.5 Y16	
N100 X16	
N110 G40 G0 Z100 M30	

**Subprogram:**

Subprogram	Comments
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 REPOS L RMB M3	; Reposition at the contour and return jump into the main program (this is programmed in a block)

**Supplementary conditions****Behavior for active frame with mirroring**

When determining the lift 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 using the MD setting:

MD21202 \$MC\_LIFTFAST\_WITH\_MIRROR = TRUE

**1.16.7 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/machining plane that is selected using G codes G17, G18 or G19. The direction of the retraction movement is not dependent on the path tangent. This allows a fast lift to be programmed parallel to the axis.

- LFPOS

Retraction of the axis declared using POLFMASK to the absolute axis position programmed with POLF.

ALF has no influence on the lift direction for several axes and for several axes in a linear system.

## Programmable traversing direction (ALF=...)

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

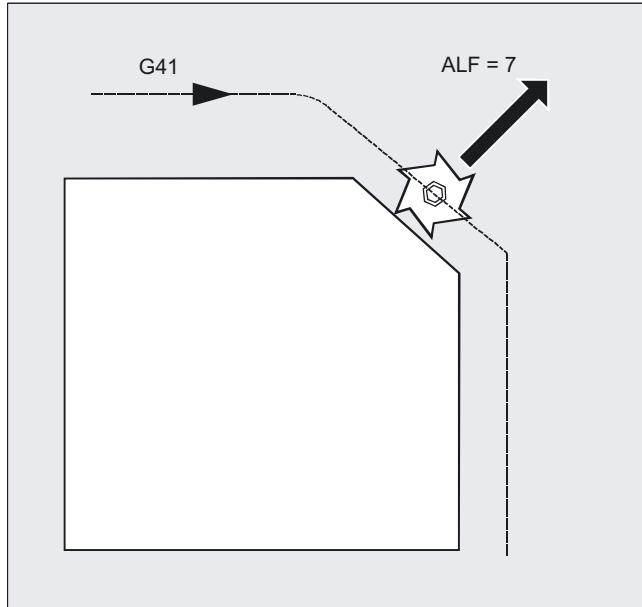
The possible traversing directions are stored in special code numbers on the control and can be called up using these numbers.

Example:

### Program code

```
N10 SETINT(2) PRIO=1 LIFT_Z LIFTFAST
ALF=7
```

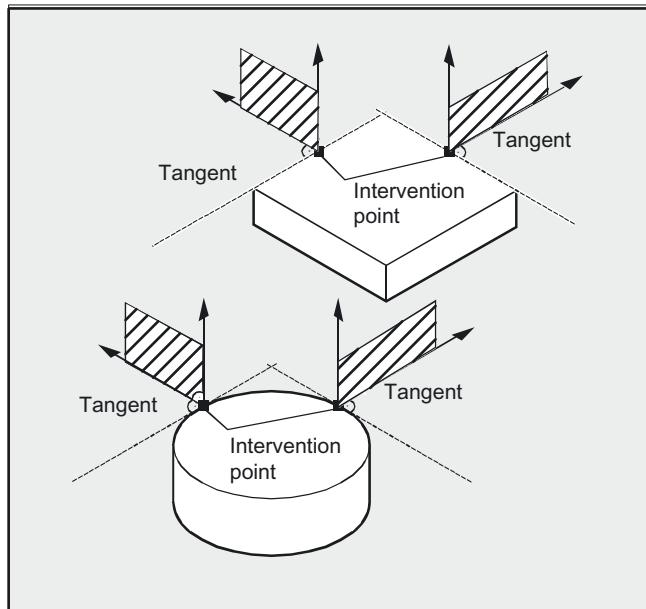
With G41 activated (machining direction to the left of the contour) the tool vertically moves away from the contour.



## Reference plane for defining the traversing direction for LFTXT

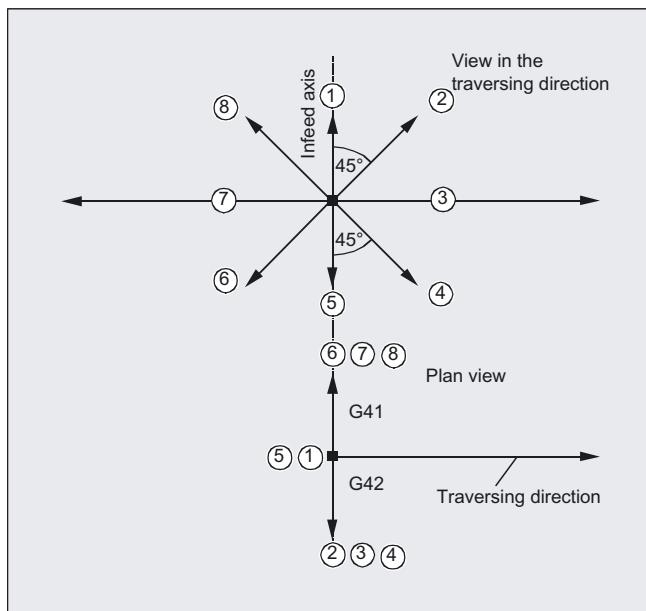
At the point of application of the tool to the programmed contour, the tool is clamped at a plane which is used as a reference for specifying the liftoff movement with the corresponding code number.

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**

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

For LFWP, the direction in the working/machining plane has 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

### References

/PG/ Programming Manual Fundamentals; Chapter: "Motion commands" > "Stop for thread cutting" > "Lift for retraction"

/FB1/ Function Manual, Special Functions; Coupled axes and ESR (M3)

## 1.16.8 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 RMI.

### 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.

---

## 1.17 Axis replacement, spindle replacement (RELEASE, GET, GETD)

### Function

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. That 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).

## Significance

RELEASE (axis name, axis name, ...)	Release the axis (axes)
GET (axis name, axis name, ...)	Accept the axis (axes)
GETD (axis name, axis name, ...)	Directly accept the axis (axes)
Axis name	Axis assignment in 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

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.

### Example: Axis exchange between two channels

Of the 6 axes, the following are used for machining in channel 1: 1., 2., 3. 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:**

<b>Program code</b>	<b>Comments</b>
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 synchronizing in both channels.
...	; Rest of program after axis replacement.
N... M30	

**Program "TRANSFER2" in channel 2:**

<b>Programming</b>	<b>Comments</b>
N... RELEASE (AX2)	
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: Axis replacement without synchronization**

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

<b>Programming</b>	<b>Comments</b>
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, as X position as in N04.
N08 X30	; First block that can be executed after N05.
...	

**Example: Activating an axis replacement without a preprocessing stop**

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

Programming	Comments
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 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.

**Requirements****Preconditions 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 preprocess 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 preprocess 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 /FB2/ Function Manual, Extended Functions, Mode Groups, Channels, Axis Replacement (K5).

## 1.18 Transfer axis to another channel (AXTOCHAN)

### Function

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[,...]])
```

### Significance

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:

/FB2/ Function Manual, Extended Functions; Positioning Axes (P2)

---

## Example of 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	Comments
N110 AXTOCHAN(Y, 2)	; Move Y axis to 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 second channel (axes are neutral).
N131 M0	
N140 AXTOCHAN(Y, 2)	; Move Y axis to second channel (NC program).
N141 M0	

## Description

### 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".

## 1.19 Activate machine data (NEWCONF)

### Function

All machine data of the effectiveness level "NEW\_CONFIG" are set active by means of the NEWCONF language command. 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, that is, the path movement is interrupted.

### Syntax

NEWCONF

### Significance

NEWCONF	All machine data of the "NEW_CONFIG" effectiveness level are set active.
---------	---

#### Cross-channel execution of NEWCONF from the part program

If axial machine data from the part program are changed 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 made, the NEWCONF statement must be executed in every channel in which the axes or functions affected by the changes in the machine data are being calculated.

No axial machine data are effective for NEWCONF.

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

### Example

Milling: Machine drill position with different technologies

Program code	Comments
N10 \$MA_CONTOUR_TOL[AX]=1.0	; Change machine data
N20 NEWCONF	; Activate machine data

## 1.20 Write file (WRITE)

### Function

Using the WRITE command, data (e.g., measurement results for measuring cycles) can be appended to the end of the specified file.

The files created can

- be read, edited, and deleted by all users,
- be written into the part program being executed.

The blocks are inserted at the end of the file, after M30.

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 (error=13)

### Syntax

```
WRITE (VAR INT error, CHAR[160] filename, CHAR[200] STRING)
```

### Significance

#### Machine manufacturer

The WRITE command can be used to store blocks from the part program in a file. The file size for log files (KB) is specified in the machine data.

The maximum length of the protocol files is set in kbyte using MD11420 \$MN\_LEN\_PROTOCOL\_FILE. This length is applicable for all files created using the WRITE command.

Once the file reaches the specified length, an error message is output and the STRING is not saved. If there is sufficient free memory, a new file can be created.

WRITE	Add data at the end of the specified file
error	Error variable for return
0:	No error
1:	Path not allowed
2:	Path not found
3:	File not found
4:	Incorrect file type
10:	File is full
11:	The file is used
12:	No resources free
13:	No access rights
20:	Other errors
filename	Name of file in which the string is to be written. If the filename contains spaces or control characters (characters with decimal ASCII code <= 32), the WRITE command will be terminated with error code 1, "path not permitted".  The file name can be specified with path and file identifier. Path names must be absolute, that is, start with "/". If the file name does not contain a domain identifier (_N_), it is added accordingly. If there is no identifier (_MPF or _SPF), the file name is automatically completed with _MPF. If there is no path specified, the file is saved in the current directory (= directory of selected program). The file name length can be up to 32 bytes, the path length up to 128 bytes.
STRING	<b>Example:</b> PROTFILE _N_PROTFILE _N_PROTFILE_MP /_N_MPF_DIR/_N_PROTFILE_MP/  Text to be written. Internally LF is then added; this means that the text is lengthened by one character.

---

**Note**

If no such file exists in the NC, it is newly created and can be written to by means of the WRITE command.

If a file with the same name exists on the hard disk, it is overwritten after the file is closed (in the NC).

Remedy: Change the name in the NC under the Services operating area using the "Properties" soft key.

---

**Example**

Program code	Comments
N10 DEF INT ERROR	
N20 WRITE(ERROR, "TEST1", "LOG FROM 7.2.97")	; Write the text from LOG FROM 7.2.97 into the TEST1 file.
N30 IF ERROR	
N40 MSG ("Error with WRITE command:" <<ERROR)	
N50 M0	
N60 ENDIF	
...	
WRITE(ERROR, "/_N_WKS_DIR/_N_PROT_WPD/_N_PROT_MPF", "LOG FROM 7.2.97") ; Absolute path data.	

## 1.21 Delete file (DELETE)

### Function

All files can be deleted by means of the DELETE command, 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

```
DELETE (VAR INT error, CHAR[160] filename)
```

### Significance

DELETE	Delete the specified file.
error	Error variable for return
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
filename	Name of the file to be deleted  The file name can be specified with path and file identifier. Path names must be absolute, that is, start with "/". If the file name does not contain a domain identifier (_N_), it is added accordingly. The file identifier ("_" plus 3 characters), e.g., _SPF) is optional. If there is no identifier, the file name is automatically added _MPF. If there is no path specified, the file is saved in the current directory (= directory of selected program). The file name length can be up to 32 bytes, the path length up to 128 bytes.

#### Example:

```
PROTFILE
_N_PROTFILE
_N_PROTFILE_MPF
/_N_MPFILE/_N_PROTFILE_MPFILE/
```

## Example

Program code	Comments
N10 DEF INT ERROR	
N15 STOPRE	; Preprocessing stop
N20 DELETE(ERROR, "/_N_SPF_DIR/_N_TEST1_SPF")	; Deletes file TEST1 in the subprogram branch.
N30 IF ERROR	
N40 MSG("error for DELETE command:" <<ERROR)	
N50 M0	
N60 ENDIF	

## 1.22 Read lines in the file (READ)

### Function

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

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 (error=13).

### Syntax

```
READ(VAR INT error, STRING[160] file, INT line, INT number, VAR  
STRING[255] result[])
```

**Significance**

READ	Read one or more lines in the specified file and store in an array element of an array. The information is available as STRING.
error	Error variable for return (call-by-reference parameter, type INT) 0: No error 1: Path not allowed 2: Path not found 3: File not found 4: Incorrect file type 13: Insufficient access rights 21: Line not present ("line" or "number" parameter larger than the number of lines in the file) 22: Array length of result variable "result" is too small 23: Line range too large ("number" parameter selected so large that the read would go beyond the end of the file)
file	Name/path of the file to be read (call-by-value parameter of type STRING with a max. length of 160 bytes). The file must be stored in the user memory of the NCK (passive file system). The file name can be preceded by the domain identifier _N_. If the domain identifier is missing, it is added correspondingly. The file identifier ("_" plus 3 characters, e.g., _SPF) is optional. If there is no identifier, _MPF is automatically added to the file name. If there is no path specified in "file", the file is searched for in the current directory (=directory of selected program). If a path is specified in "file", it must start with a slash "/" (absolute path indication).
line	Position indication of the line range to be read (call-by-value parameter of type INT). 0: The number of lines specified with the "number" parameter before the file end are read. 1 to n: Number of the first line to be read
number	Number of lines to be read (call-by-value parameter of type INT)
result	Array of type STRING, where the read text is stored (call-by-reference parameter with a length of 255)

If the number of lines specified in the parameter "number" is smaller than the array length of "result", the other array elements are not altered.

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 target variables "result". Read lines are cut off, if the line is longer than the string length of the target variable "result". An error message is not output.

---

#### Note

**Binary files cannot be read in.**

The error message error=4: Wrong type of file is output. The following types of file are not readable: \_BIN, \_EXE, \_OBJ, \_LIB, \_BOT, \_TRC, \_ACC, \_CYC, \_NCK.

---

## Example

Program code	Comments
N10 DEF INT ERROR	; Error variable
N20 STRING[255] RESULT[5]	; Result variable
...	
N30 READ(ERROR,"TESTFILE",1,5,RESULT)	; Filename without domain and file identifier.
...	
N30 READ(ERROR,"TESTFILE_MPF",1,5,RESULT)	; Filename without domain and with file identifier.
...	
N30 READ(ERROR,"_N_TESTFILE_MPF",1,5,RESULT)	; Filename with domain and file identifier.
...	
N30 READ(ERROR,"/_N_CST_DIR/_N_TESTFILE_MPF",1,5,RESULT)	;Filename with domain, file identifier and path specification.
...	
N40 IF ERROR <>0	; Error evaluation
N50 MSG("ERROR"<<ERROR<<" WITH READ COMMAND")	
N60 M0	
N70 ENDIF	
...	

## 1.23 File present in the NCK user memory (ISFILE)

### Function

With the ISFILE command you check whether a file exists in the user memory of the NCK (passive file system). As a result either TRUE (file exists) or FALSE (file does not exist) is returned.

### Syntax

```
result=ISFILE(STRING[160]file)
```

### Significance

ISFILE	Checks whether the file exists in the NCK user memory.
file	Name/path of the file to be read (call-by-value parameter of type STRING with a max. length of 160 bytes).  The file must be stored in the user memory of the NCK (passive file system). The file name can be preceded by the domain identifier _N_. If the domain identifier is missing, it is added correspondingly.  The file identifier ("_" plus 3 characters), e.g., _SPF) is optional. If there is no identifier, the file name is automatically added _MPF.  If there is no path specified in "file", the file is searched for in the current directory (=directory of selected program). If a path is specified in "file", it must start with a slash "/" (absolute path indication).
result	Variable for storage of the result of type BOOL (TRUE or FALSE)

## Example

Program code
<pre>N10 DEF BOOL RESULT N20 RESULT=ISFILE("TESTFILE") N30 IF (RESULT==FALSE) N40 MSG("FILE DOES NOT EXIST") N50 M0 N60 ENDIF ... or: N30 IF (NOT ISFILE("TESTFILE")) N40 MSG("FILE DOES NOT EXIST") N50 M0 N60 ENDIF ...</pre>

## 1.24 File information (FILEDATE, FILETIME, FILESIZE, FILESTAT, FILEINFO)

### Function

The FILEDATE, FILETIME, FILESIZE, FILESTAT and FILEINFO commands can be used to read particular pieces of file information, such as date, time, current file size, file status or the sum of this information from the user memory of the NCK (passive file system).

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 (error=13).

Application:

Provision of new file information if a file has changed for the user and this is for example to be recalculated.

### Syntax

FILExxxx (VAR INT error, STRING[160] file, VAR {STRING[yy] INT}result)

## Significance

FILEDATE	Returns date when file was last accessed and written
FILETIME	Returns time when file was last accessed and written
FILESIZE	Returns the current file size
FILESTAT	Returns file status, such as read, write and execute rights
FILEINFO	Returns the sum of the information from a directory entry
error	Error variable for return
	0: No error
	1: Path not allowed
	2: Path not found
	3: File not found
	13: Insufficient access rights
	22: Array length of result variable "result" is too small
file	Name or path of the file to be read (call-by-value parameter of type STRING with a maximum length of 160 bytes).  The file must be stored in the user memory of the NCK (passive file system). The file name can be preceded by the domain identifier _N_. If the domain identifier is missing, it is added correspondingly.  The file identifier ("_" plus 3 characters), e.g., _SPF) is optional. If there is no identifier, _MPF is automatically added to the file name.  If there is no path specified in "file", the file is searched for in the current directory (=directory of selected program). If a path is specified in "file", it must start with a slash "/" (absolute path indication).
result	Variable with the result in which the file information is saved (Call-by-reference parameter) of a STRING type for: FILEDATE, the length must be 8, format is "dd.mm.yy" FILETIME, the length must be 8, format is "hh:mm:ss" FILESTAT, the length must be 5, format is "rwxsd" FILEINFO, the length must be 32, format is "rwxsd nnnnnnnn dd.mm.yy hh:mm:ss"  (Call-by-reference parameter) of a INT type for: FILESIZE, file size is output in bytes "rwxsd" (read, write, execute, show, delete)

## Examples

Program code	Comments
N10 DEF INT ERROR	; Error variable
N20 STRING[32] RESULT	; Result variable
...	
N30 FILEINFO(ERROR,"TESTFILE",RESULT)	; Filename without domain and file identifier.
...	
N30 FILEINFO(ERROR,"TESTFILE_MPF",RESULT)	; Filename without domain and with file identifier
...	
N30 FILEINFO(ERROR,"_N_TESTFILE_MPF",RESULT)	; Filename with domain and file identifier.
...	
N30 FILEINFO(ERROR,"/_N_MPF_DIR/_N_TESTFILE_MPF",RESULT)	; Filename with domain, file identifier and path specification.
...	
N40 IF ERROR <>0	; Error evaluation
N50 MSG("ERROR "<<ERROR<<" WITH FILEINFO COMMAND")	
N60 M0	
N70 ENDIF	
...	

The example returns in the RESULT event variable: "77777 12345678 26.05.00 13:51:30"

## 1.25 Form the checksum over an array (CHECKSUM)

### Function

With CHECKSUM you form a checksum over an array.

### Application

Check to see whether the initial contour has changed during stock removal.

### Syntax

```
error=CHECKSUM(VAR STRING[16] chksum, STRING[32]array, INT first,
INT last)
```

### Significance

CHECKSUM	Form the checksum over an array
error	Error variable for return representation
0:	No error
1:	Symbol not found
2:	No array
3:	Index 1 too large
4:	Index 2 too large
5:	Invalid data type
10:	Check sum overflow
chksum	Checksum over the array as a STRING (call-by-reference parameter of type STRING, with a defined length of 16).  The checksum is indicated as a character string of 16 hexadecimal numbers. However, no format characters are indicated.  Example: "A6FC3404E534047C"
array	Number of the array over which the checksum is to be formed. (call-by-value parameter of type STRING with a max. length of 32).  Permissible arrays: 1- or 2-dimensional arrays of the types BOOL, CHAR, INT, REAL, STRING  Arrays of machine data are not permissible.
first	Column number of start column (optional)
last	Column number of end column (optional)

---

**Note**

The parameters first and last are optional. If no column indices are indicated, the checksum is formed over the whole array.

The result of the checksum is always definite. If an array element is changed, the result string will also be changed.

---

**Example**

---

**Program code**

```
N10 DEF INT ERROR
N20 DEF STRING[16] MY_CHECKSUM
N30 DEF INT MY_VAR[4,4]
N40 MY_VAR=...
N50 ERROR=CHECKSUM (CHECKSUM;"MY_VAR", 0, 2)
...
returns in MY_CHECKSUM the value "A6FC3404E534047C"
```

---

## 1.26 Roundup (ROUNDUP)

### Function

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>)

### Significance

ROUNDUP      Command to roundup an input value  
<value>      Input value, type REAL

---

### Note

Input value, type INTEGER (an integer number) are 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]) ...

## 1.27 Subprogram technique

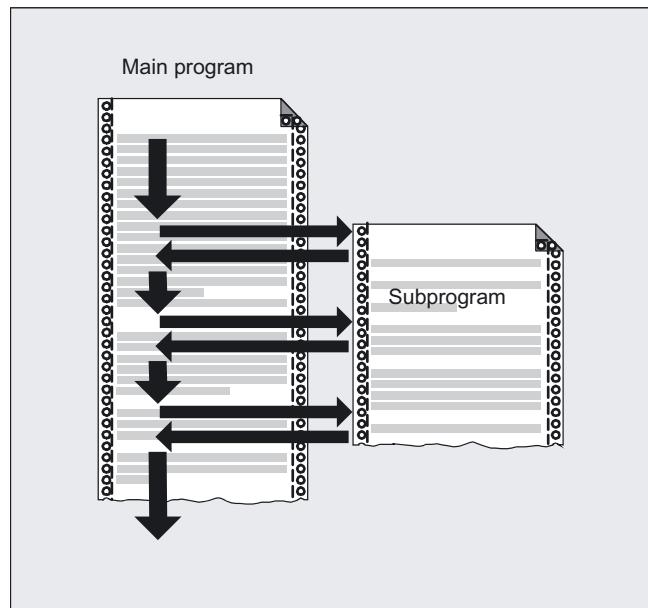
### 1.27.1 Subprograms

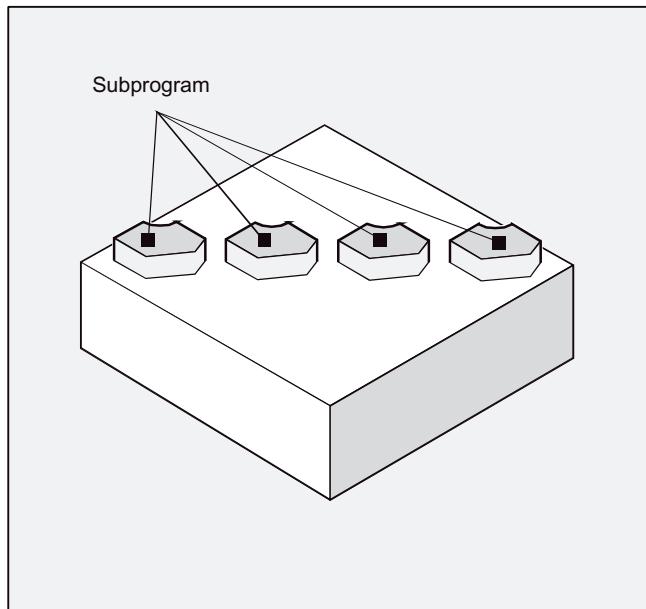
#### Function

Just the same as in all higher-level programming languages, program parts can also be used a multiple number of times in the NC language – these are then shifted out to subprograms.

Subprograms offer the following advantages:

- Increase the transparency and readability of programs
- Increase the quality by re-using tested program parts
- Offer the possibility of creating specific machining libraries
- Save memory space





### Structure of a subprogram

In principle, a subprogram has the same structure as a main program. It consists of NC blocks with traversing and switching commands.

A program header with parameter definitions can also be programmed in the subprogram.

The end of the subprogram is programmed using M17. This means a return to the program plane that called the subprogram.

---

#### Note

It is possible to suppress the M17 end of program in the machine data (e.g., to achieve a better running time).

---

### End of subprogram with RET

The instruction RET can also be used in subprograms as a substitute for the backward jump with M17. RET must be programmed in a separate block.

The RET instruction should then be used if a G64 continuous-path mode (G641, G642, G643) is **not** to be interrupted by the return. This is only possible if the subprogram has **no** SAVE attribute.

If M17 is programmed in a separate block, G64 is interrupted and an exact stop generated.  
 Remedy: Do not write M17 in a subprogram block on its own, instead use it, for example, with a traverse path:

G1 X=YY M17.

The following must be set in the machine data: "No M17 from PLC".

### Subprogram name

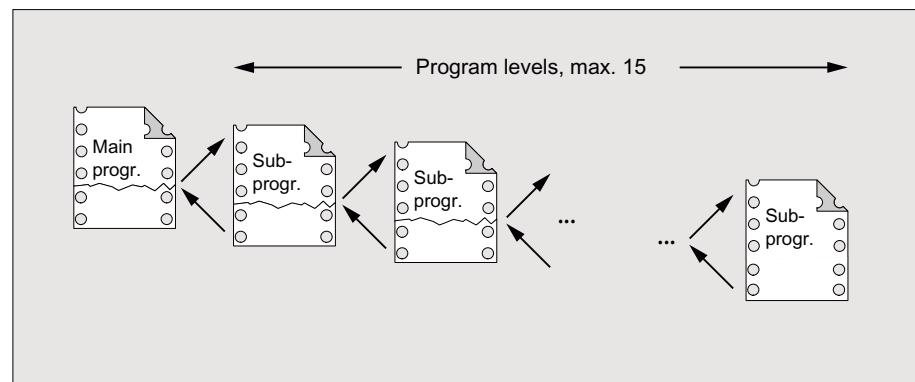
A program name must be assigned when creating a subprogram. When assigning a name, the same rules apply as for the main program.

#### References:

Programming Manual Fundamentals.

### Nesting of subprograms

Subprograms can not only be called from a main program, but also from a subprogram. The overall call hierarchy may not exceed 16 program levels - including the main program level. That means that a main program may contain as many as 15 nested subprogram calls.



---

**Note**

**Interrupt routines (ASUB)**

It is also possible to call subprograms in interrupt routines. In order that the maximum possible number of program levels is not exceeded – and therefore resulting in an alarm - the system reserves 2 additional program levels for interrupt routines. Users cannot use these.

If more than 2 program levels are required for interrupt routines, then this must be taken into account when nesting the subprogram, i.e. the program levels required must be kept free.

---

---

**Note**

**Cycles**

For SIEMENS machining and measuring cycles you require three levels. If a cycle is to be called up from a subprogram, then the maximum level is 14.

---

## **1.27.2 Subprograms with SAVE attribute (SAVE)**

### **Function**

The **SAVE** attribute means that before the subprogram call, active modal G functions are saved and are re-activated after the end of the subprogram.

### **Syntax**

PROC <subprogram name> SAVE

### **Significance**

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	Comments
PROC CONTOUR (REAL VALUE1) SAVE	; Subprogram definition with the SAVE parameter
N10 G91 ...	; Modal G function G91: Incremental dimension
N100 M17	; Subprogram end

Main program:

Program code	Comments
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), Chapter "Subprogram return with SAVE".

### 1.27.3 Subroutines with parameter transfer (PROC, VAR)

#### Function

##### **Program start, PROC**

A subprogram that is to take over parameters from the calling program when the program runs is designated with the keyword PROC.

##### **Subprogram end M17, RET**

The command M17 designates the end of subprogram and is also an instruction to return to the calling main program. As an alternative to M17: The keyword RET stands for end of subprogram without interruption of continuous path mode and without function output to the PLC.

#### Programming

The parameters of a subprogram must be listed with type and name at the beginning of the subprogram definition.

##### **Parameter transfer call-by-value**

```
PROC PROGRAM_NAME (VARIABLE_TYPE1 VARIABLE1, VARIABLE_TYPE2  
VARIABLE2, ...)
```

Example:

```
PROC CONTOUR (REAL LENGTH, REAL WIDTH)
```

##### **Parameter transfer call-by-reference, identification with keyword VAR**

```
PROC PROGRAM_NAME (VAR VARIABLE_TYPE1 VARIABLE1, VAR VARIABLE_TYPE2  
..., )
```

Example:

```
PROC CONTOUR (VAR REAL LENGTH, VAR REAL WIDTH)
```

##### **Array transfer with call-by-reference, identification with keyword VAR**

```
PROC PROGRAM_NAME (VAR VARIABLE_TYPE1 ARRAY_NAME1 [array size],  
VAR VARIABLE_TYPE2 ARRAY_NAME2 [array size],  
VAR VARIABLE_TYPE3 ARRAY_NAME3 [array size1, array size2],  
VAR VARIABLE_TYPE4 ARRAY_NAME4 [ ],  
VAR VARIABLE_TYPE5 ARRAY_NAME5 [,array size])
```

Example:

```
PROC PALLET (VAR INT ARRAY[,10])
```

## Parameters

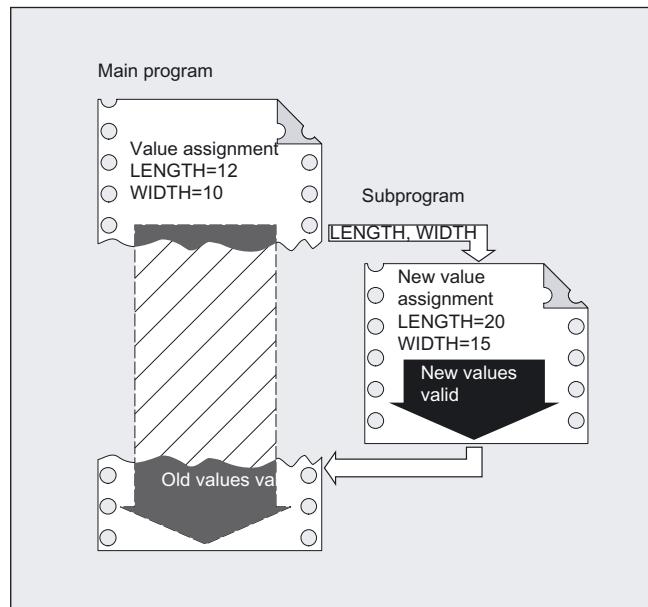
PROC	First instruction in a program
PROGRAM NAME	Subprogram name that should accept the relevant values of the parameters
VARIABLE_TYPE	Variable types with specification of the variable values.
VARIABLE	Several values can be specified.
VAR	Keyword for the type of the parameter transfer
FIELDNAME	Elements of an array with the listed values for the field array
Array size1	For a one-dimensional array
Array size2	For a two-dimensional array

### Note

The definition statement with PROC must be programmed in a separate NC block. A maximum of 127 parameters can be declared for parameter transfer.

## Example: parameter transfer between main program and subprogram

```
N10 DEF REAL LENGTH,WIDTH
N20 LENGTH=12 WIDTH=10
N30 BORDER (LENGTH,WIDTH)
```



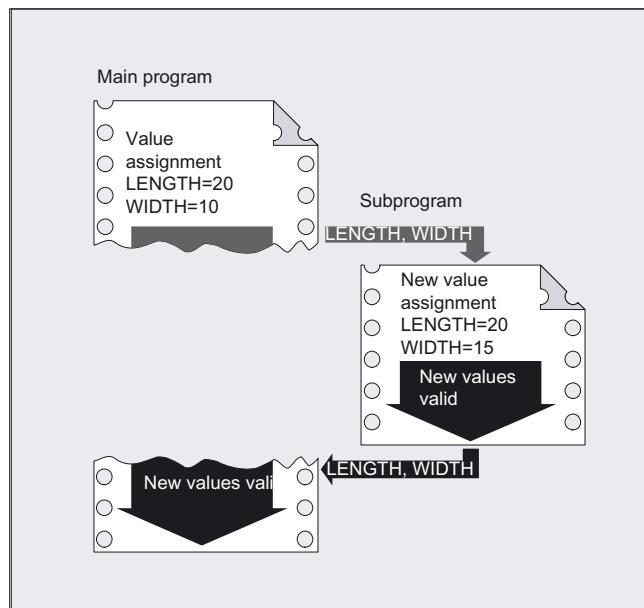
The values assigned in N20 in the main program are transferred in N30 when the subprogram is called. Parameters are transferred in the sequence stated.

The parameter names do not have to be identical in the main programs and subprogram.

#### Second method of parameter transfer:

- **Values are only transferred (call-by-value)**

If the parameters transferred are changed as the subprogram runs this does not have any effect on the main program. The parameters remain unchanged in it (see Fig.).



- **Parameter transfer with data exchange (call-by-reference)**

Any change to the parameters in the subprogram also causes the parameter to change in the main program (see Fig.).

### Example: Variable array lengths

Main program "DRILLING PLATE":

Program code	Comments
N10 DEF REAL TABLE[100,2]	; Defining a position table
N20 EXTERNAL DRILLING PATTERN (VAR REAL[,2],INT)	;
N30 TABLE[0,0]=-17.5	; Defining positions
...	
N200 TABLE[99,1]=45	
N210 DRILLING PATTERN(TABLE,100)	; Subprogram call
N220 M30	

### Example: Creating a drilling pattern using a transferred position table with variable length

Subprogram "DRILLING PATTERN":

Program code	Comments
N10 PROC DRILLING PATTERN(VAR REAL ARRAY[,2], INT NUMBER)	; Definition
N20 DEF INT COUNTER	; Processing sequence
N30 STEP: G1 X=ARRAY[COUNTER,0] Y=ARRAY[COUNTER,1] F100	
N40 Z=IC(-5)	
N50 Z=IC(5)	
N60 COUNTER = COUNTER + 1	
N70 IF COUNTER < NUMBER GOTOB STEP	
N80 RET	

### Interruption of continuous-path mode

In order that continuous-path mode is not interrupted by a subprogram call, it is not permissible to use the `SAVE` attribute when defining a subprogram. The `RET` command must be used as command for the subprogram return jump.

Programming	Comments
PROC CONTOUR	; Start of the subprogram
N10...	
...	
N100 RET	; Return jump

### Parameter transfer between the main program and subprogram

If you work with parameters in the main program, then you can also use them in the subprogram. When calling the subprogram, transfer the values of the **actual parameters** of the main program to the **formal parameters** of the subprogram.

### Arrays of undefined length as formal parameters

With arrays of an undefined array length, subprograms can process arrays of variable length as formal parameter. When defining a two-dimensional array as formal parameter, the length of the 1st dimension is not specified. However, the comma must be written.

Example:

```
PROC <Subprogram name> (VAR REAL ARRAY[, 5])
```

A detailed explanation of defining arrays is provided under:

#### References

/PGA/ Programming instructions Job Planning; Flexible NC programming,  
Chapter: Array definitions (DEF, SET, REP)

## 1.27.4 Call subroutines (L or EXTERN)

### Function

#### Subprogram call without parameter transfer

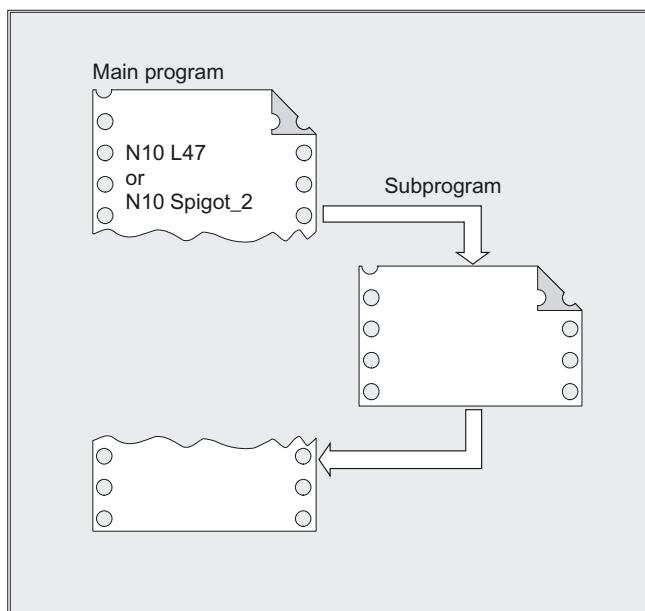
In the main program, call the subprogram either with address L and subprogram number or by specifying the program name.

Example:

N10 L47

or

N10 SPIGOT\_2



#### Subprogram call with parameter transfer

For a subprogram call with parameter transfer, you can directly transfer variables or values (not for 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.

## Syntax

### Subprogram call without parameter transfer:

L<number>  
<program\_name>

### Subprogram call with parameter transfer:

```
EXTERNAL <program name>(<type_Par1>,<type_Par2>,<type_Par3>)
...
<program name>(<value_Par1>,<value_Par2>,<value_Par3>)
```



#### The following applies to every subprogram call:

The subprogram call must always be programmed in a separate NC block.

## Significance

L	Address of the subprogram call
<number>	Name of the subprogram
Type:	INT
Value:	Maximum 7 decimal places
	<b>Notice:</b> Leading zeros are significant in names (⇒ L123, L0123 and L00123 are three different subprograms).
<program name>	Name of subprogram
EXTERNAL	Keyword to declare a subprogram with parameter transfer.
	<b>Note:</b> You only have to specify EXTERNAL if the subprogram is in the workpiece or in the global subprogram directory. You do not have to declare cycles as EXTERN .
<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

## Incomplete parameter transfer

In a subprogram call only mandatory values and parameters can be omitted. In this case, the parameter in question is assigned the value **zero** in the subprogram.

The comma must always be written to indicate the sequence. If the parameters are at the end of the sequence you can omit the comma as well.

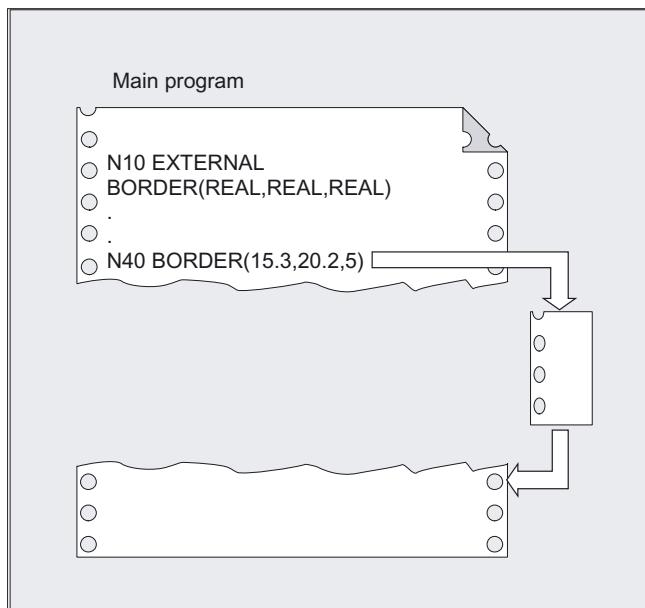
**! CAUTION**

The current parameter of type AXIS must not be omitted. VAR parameters must be transferred completely.

## Examples

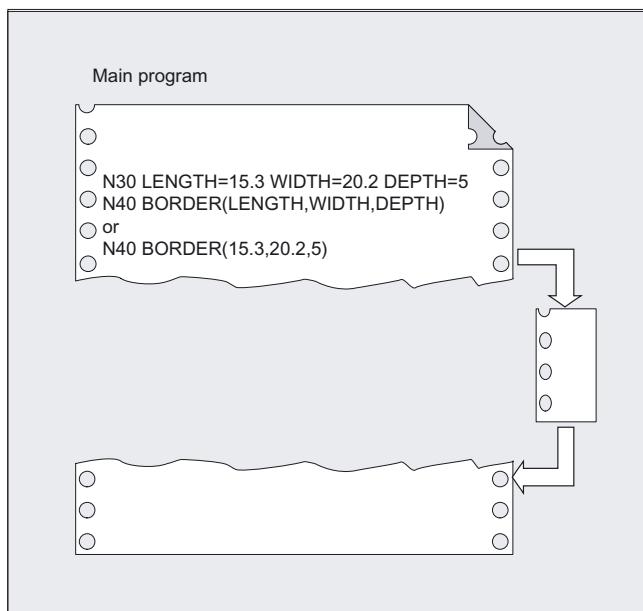
### Example 1: Subprogram call with parameter transfer (with EXTERNAL)

Program code	Comments
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 with parameter transfer (without EXTERNAL)**

Program code	Comments
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)



### Example 3: Subprogram

Program code	Comments
PROC SUB1(INT VAR1,DOUBLE VAR2)	
IF \$P_SUBPAR[1]==TRUE	; Parameter VAR1 was programmed in the subprogram call.
ELSE	; Parameter VAR1 was not programmed in the subprogram call and the system sets a standard value of 0 as default.
ENDIF	
IF \$P_SUBPAR[2]==TRUE	; Parameter VAR2 was programmed in the subprogram call
ELSE	; Parameter VAR2 was not programmed in the subprogram call and the system sets the standard value 0.0 as default.
ENDIF	
IF \$P_SUBPAR[3]==TRUE -> Alarm 17020	; Parameter 3 is not defined.
M17	

### Description

 **CAUTION**

**Subprogram definition corresponds to subprogram call**

Both the variable types and the sequence of transfer must match the definitions declared under PROC in the subprogram name. The parameter names can be different in the main program and subprograms.

Definition in the subprogram:

```
PROC BORDER(REAL LENGTH, REAL WIDTH, REAL DEPTH)
```

Call in the main program:

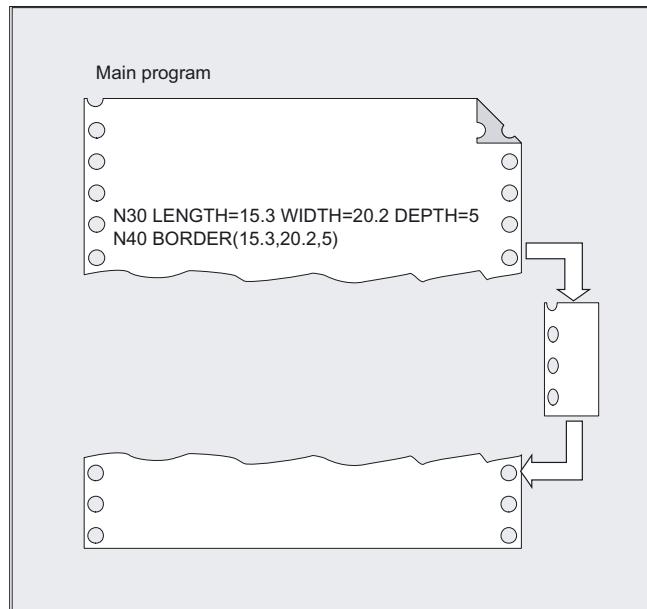
```
N30 BORDER (LENGTH, WIDTH, DEPTH)
```

### Incomplete parameter transfer

Back to the last example:

N40 BORDER(15.3, , 5)

The mean value 20.2 was omitted here.



With incomplete parameter transfer, it is possible to tell by the system variable \$P\_SUBPAR[i] whether the transfer parameter was programmed for subprograms or not. The system variable contains as argument (i) the number of the transfer parameter.

The system variable \$P\_SUBPAR returns

- TRUE, if the transfer parameter was programmed
- FALSE, if no value was set as transfer parameter.

If an impermissible parameter number was specified, part program processing is aborted with alarm output.

## Call main program as subprogram

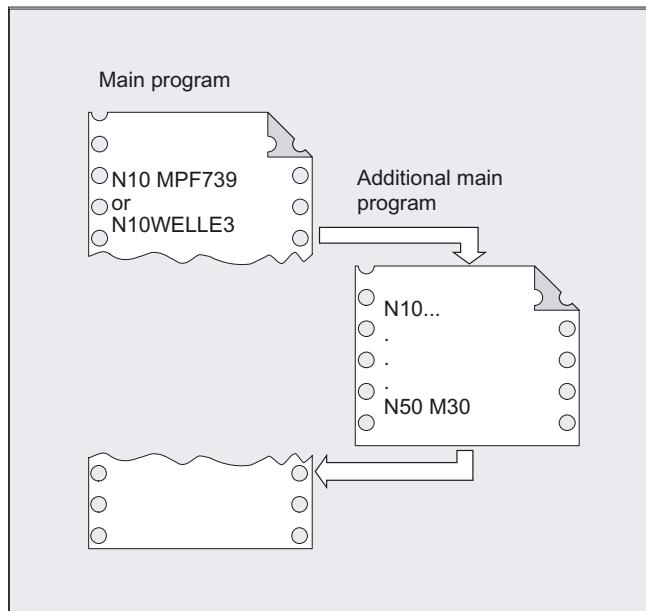
A main program can also be called as a subprogram. The end of program M2 or M30 set in the main program is evaluated as M17 in this case (end of program with return to the calling program).

You program the call specifying the program name.

Example

N10 MPF739 or

N20 Shaft3



A subprogram can also be started as a main program.

---

### Note

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 issues the call is called again. This is generally an undesirable effect and must be avoided by assigning unique names to subprograms and main programs.

---

### **Call the subprograms with theINI file**

Subprograms that do not require parameter assignment can be called from an initialization file:

Example

```
N10 MYINISUB1 ;Subprogram call without parameters
```

## **1.27.5 Parameterized subroutine return (RET)**

### **Function**

Usually, an RET or M17 end of subprogram returns to the program from which the subprogram was called and processing continues with the program line following the subprogram call.

However, on the other hand, there are also applications where the program should be resumed at another position, e.g.:

- 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.

In cases such as these, the RET command is programmed together with "return jump parameters".

### **Syntax**

```
RET("<destination block>")
RET("<destination block>",<block after destination block>")
RET("<destination block>",<block after destination block>,<number of
return jump levels>")
RET("<destination block>,<block after destination block>,<number of
return jump levels>,<number of return jump levels>)
RET("<destination block>",<block after destination 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>)
```

## Significance

RET <destination block>	Subprogram end (use instead of M17) Return jump parameter 1 Declares as jump destination the block where program execution should be resumed. If the return jump parameter 3 is not programmed, then the jump destination is in the program from which the current subprogram was called. Possible data include: "<block number>"      Number of the destination block "<jump marker>"      Jump marker that must be set in the destination block. "<character string>"      Character string that must known in the program (e.g. program or variable name). The following rules apply when programming the character string: <ul style="list-style-type: none"><li>• <b>Blank at the end</b> (contrary to the jump marker, that is designated using ":" at the end).</li><li>• <b>Before the character string</b> only one block number and/or a jump marker may be set, <b>no program commands</b>.</li></ul>
<block after destination block>	Return jump parameter 2 Refers to the return jump parameter 1. Type: INT Value: 0      Return jump is made to the block that was specified using return jump parameter 1. > 0      Return jump is made to the block that follows the block specified with return jump parameter.

<number of  
return jump levels>

Return jump parameter 3

Specifies the number of levels that must be jumped back in order to reach the program level in which program execution should be continued.

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.

...

Value

range: 1 ... 15

Return jump parameter 4

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.

<return jump to the  
beginning of the program>

---

### Note

For a subprogram return jump with a character string to specify the destination block search, initially, a search is always made for a jump marker in the calling program.

If a jump destination 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).

---

## Example

### Example 1: Resuming in the main program after ASUB execution

Programming	Comments
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
N10900	; Resumption in main program
N10910 MCALL	; Deactivate modal subprogram
N10920 G0 G60 G40 M5	; correct additional modal settings

### Example 2: Character string (string>) to specify the destination block search

#### Main program:

<u>Programming</u>	<u>Comments</u>
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:**

Programming	Comments
PROC subProg1 N2000 R10=R20+100 N2010 ... N2200 RET("subProg2")	; Return jump into the main program at block N1400

**Subprogram subProg2:**

Programming	Comments
PROC subProg2 N2000 R10=R20+100 N2010 ... N2200 RET("iVar1")	; Return jump into the main program at block N1500

**Subprogram subProg3:**

Programming	Comments
PROC subProg3 N2000 R10=R20+100 N2010 ... N2200 RET("subProg1")	; Return jump into the main program at block N1550

**Supplementary conditions**

- When making a return jump through several program levels, the **SAVE** instructions 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 de-selection command **MCALL** is programmed for the modal subprogram, then the modal subprogram remains active.

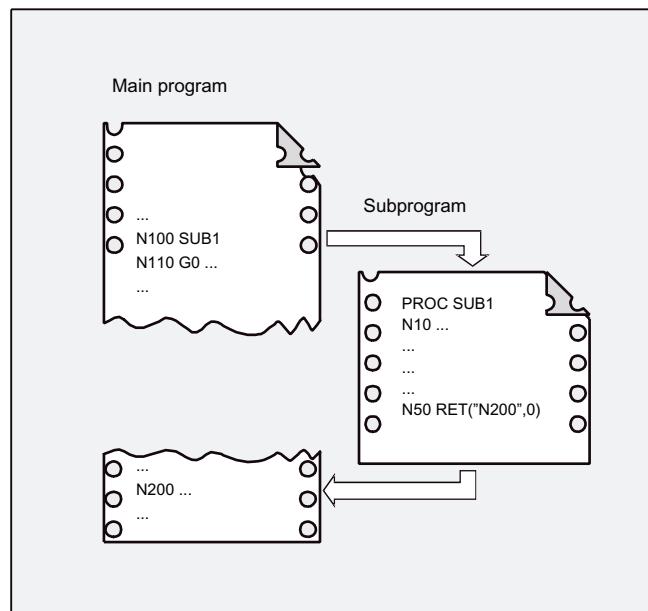
 **CAUTION**

The programmer must carefully ensure that for a return jump over several program levels, the program continues with the correct modal settings. This can be achieved, e.g. by programming an appropriate main block.

## Description

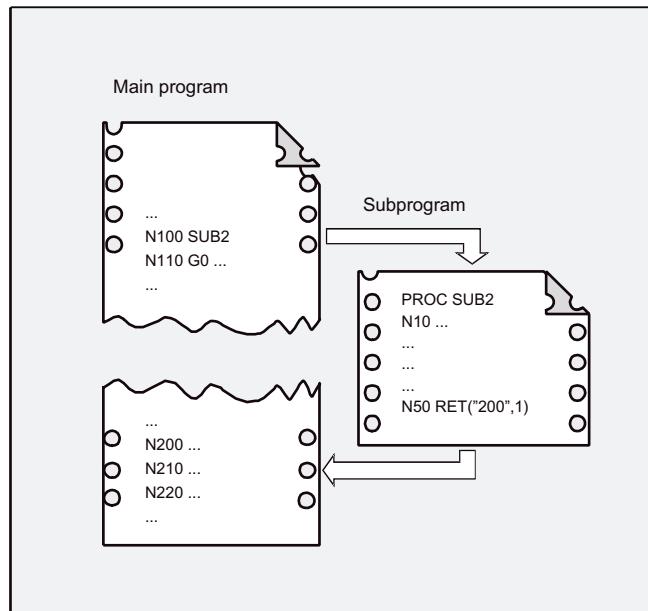
The different effects of return jump parameters 1 to 3 are explained in the following graphics.

**1st return jump parameter 1 = "N200", return jump parameter 2 = 0**



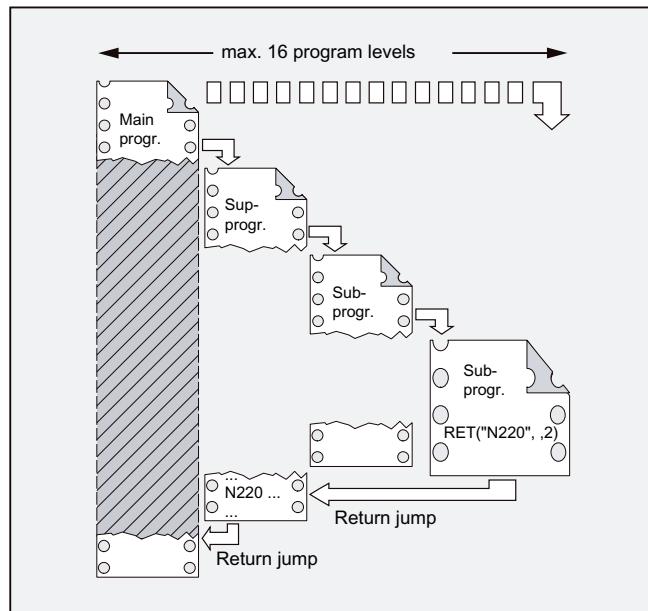
After the `RET` command, program execution is continued with block `N200` in the main program.

2nd return jump parameter 1 = "N200", return jump parameter 2 = 1



After the RET command, program execution is continued with the block (N210) that follows block N200 in the main program.

3rd return jump parameter 1 = "N220", return jump parameter 3 = 2



After the RET command, two program levels are jumped through and program execution is continued with block N220.

## 1.27.6 Subroutine with program repetition (P)

### Function

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.



#### 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

P<number\_program\_repetitions>

### Significance

P  
<number\_program\_repetitions>

Address to program program repetitions

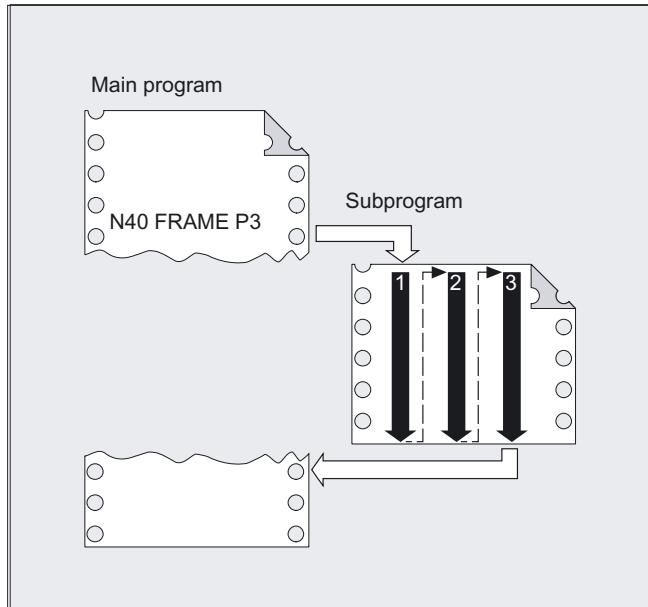
The number of program repetitions is specified using this parameter

Type: INT

Value range: 1 ... 9999  
(unsigned)

## Example

Program code	Comments
...	
N40 FRAME P3	; The BORDER subprogram is to be executed three times one after the other.
...	



### 1.27.7 Modal subprogram call (MCALL)

#### Function

For a modal subprogram call with MCALL, the subprogram is automatically called and executed after each block with path motion. This allows subprogram calls to be automated, which are to be executed at different workpiece positions (for example to create drilling patterns).

MCALL is used to deactivate the function without a subprogram call or by programming a new modal subprogram call for a new subprogram.



#### CAUTION

**Only one** MCALL call can be simultaneously effective in a program run. Parameters are only transferred once for an MCALL call.

In the following situations the modal subprogram is also called without motion programming:

- Programming addresses S and F if G0 or G1 is active.
- If G0/G1 were programmed alone in the block or with additional G codes.

#### Syntax

MCALL L<number>

#### Significance

MCALL	Keyword for the modal subprogram call
L	Address of the subprogram call
<number>	Name of the subprogram

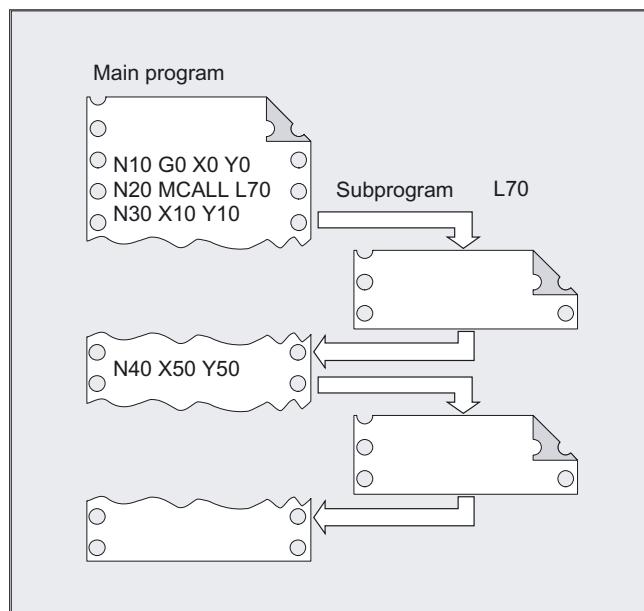
## Examples

### Example 1:

#### Program code

```
N10 G0 X0 Y0  
N20 MCALL L70  
N30 X10 Y10  
N40 X50 Y50
```

In blocks N30 and N40 The programmed position is approached and then subprogram L70 is executed.



### 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.

## 1.27.8 Indirect subroutine call (CALL)

### Function

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 type STRING. The subprogram call is realized with CALL and the variable name.



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>
```

### Parameter

CALL	Keyword for indirect subprogram call
<program name>	Name of the subprogram (variable or constant, type STRING)

### Examples

#### Example 1: Direct call with STRING constant

```
CALL "/_N_WKS_DIR/_N_SUBPROG_WPD/_N_PART1_SPF"
```

#### Example 2: Indirect call via variable

```
Program code
DEF STRING[100] PROGNAME
PROGNAME="/_N_WKS_DIR/_N_SUBPROG_WPD/_N_PART1_SPF"
CALL PROGNAME
```

Subprogram PART1 is assigned the PROGNAME variable. The subprogram is indirectly called using CALL and the path name.

### 1.27.9 Executing a program part in an indirectly called subprogram (CALL BLOCK ... TO ...)

#### Function

Using CALL and the keyword combination BLOCK ... TO, subprogram is indirectly called and the program part designated with start label and end label is executed.

#### Syntax

```
CALL <program name> BLOCK <start label> TO <end label>
CALL BLOCK <start label> TO <end label>
```

#### Significance

CALL	Keyword for the indirect subprogram call
<program name>	Variable or constant for the name of the subprogram that contains the program part to be executed (specification optional).
<b>Note:</b>	If a <program name> has not been programmed, the program part designated with <start label> and <end label> is searched for in the actual program and executed.
BLOCK ... TO ...	Keyword combination for indirect program part repetition
<start label>	Variable that refers to the start of the program part to be executed.
Type: STRING	
<end label>	Variable that refers to the end of the program part to be executed.
Type: STRING	

## Example

Program code	Comments
<pre>DEF STRING[20] STARTLABEL, ENDLABEL STARTLABEL="LABEL_1" ENDLABEL="LABEL_2" ... CALL "CONTUR_1" BLOCK STARTLABEL TO ENDLABEL ... ...</pre>	

Program code	Comments
<pre>PROC CONTUR_1 ... LABEL_1 N1000 G1 ... LABEL_2 ...</pre>	<p>; Start of program part repetition</p> <p>; End of program part repetition</p>

### 1.27.10 Indirect call of a program programmed in ISO language (ISOCALL)

#### Function

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>
```

#### Significance

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	Comments
0122_SPF	; Contour description in the 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)
...	
N2000 R11 = \$AA_IW[X]	
N2010 ISOCALL PROGNAME	
N2020 R10 = R10+1	; Execute program 0122.spf in the ISO mode
...	
N2400 M30	

### 1.27.11 Calling subroutine with path specification and parameters (PCALL)

#### Function

With PCALL you can call subprograms with the absolute path and parameter transfer.

#### Syntax

PCALL <path/program name>(<parameter 1>,...,<parameter n>)

#### Significance

PCALL	Keyword for subprogram call with absolute path name
<path/program_name>	Absolute path name beginning with "/", including subprogram names
	If no absolute path name is specified, PCALL behaves like a standard subprogram call with a program identifier.
	The program identifier is written without the leading _N_ and without an extension.
	If the program name is to be programmed with leading character and extension, then must be explicitly declared with leading character and extension using the command EXTERNAL.
<parameter 1>, ... <parameter n>	Current parameters in accordance with the PROC statement of the subprogram.

#### Example

```
Program code
PCALL/_N_WKS_DIR/_N_WELLE_WPD/WELLE(parameter1,parameter2,...)
```

### 1.27.12 Extend search path for subprogram calls (CALLPATH)

#### Function

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.

The search path extension is made before the entry for user cycles (\_N\_CUS\_DIR).

The search path extension is de-selected again as a result of the following events:

- CALLPATH with blanks
- CALLPATH without parameter
- Part program end
- Reset

#### Syntax

```
CALLPATH("<path_name>")
```

#### Significance

CALLPATH	Keyword for the programmable search path extension. Is programmed in a separate part program line.
<path_name>	Constant or variable, type STRING. Contains the absolute path name of a directory by which the search path should be extended. The path name starts with "/". The path must be completely specified using prefixes and suffixes. The maximum path length is 128 bytes.  If the <path name> contains a blank or if CALLPATH is called without parameter, then search path instruction is reset again.

## Example

### Programming

```
CALLPATH ("/_N_WKS_DIR/_N_MYWPD_WPD")
```

This means that the following search path is set (position 5. is new):

1. Current directory / subprogram name
2. Current directory / subprogram identifier\_SPF
3. Current directory / subprogram identifier\_MPF
4. /\_N\_SPF\_DIR/subprogram identifier\_SPF
5. **/\_N\_WKS\_DIR/\_N\_MYWPD/subprogram identifier\_SPF**
6. /N\_CUS\_DIR/\_N\_MYWPD/subprogram identifier\_SPF
7. /\_N\_CMA\_DIR/subprogram identifier\_SPF
8. /\_N\_CST\_DIR/subprogram identifier\_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 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.

### 1.27.13 Execute external subroutine (EXTCALL)

#### Function

EXTCALL can be used to reload a program from the HMI in "Execution from external source" mode. All programs that can be accessed via the directory structure of HMI can be reloaded and run.

#### Syntax

```
EXTCALL("<path/program name>")
```

#### Significance

EXTCALL	Keyword for subprogram call
<path/program_name>	Constant/variable of type STRING
	An absolute (or relative) path or a program name can be specified.
	The program name is written with/without the leading _N_ and without an extension. An extension can be attached to the program name using the <_> character.
	Example:
	"/_N_WKS_DIR/_N_WELLE_WPD/_N_WELLE_SPF"
	or
	"SHAFT"

---

#### Note

External subprograms must not contain jump statements such as GOTOF, GOTOB, CASE, FOR, LOOP, WHILE, or REPEAT.

IF-ELSE-ENDIF constructions are possible.

Subprogram calls and nested EXTCALL calls may be used.

---

#### RESET, POWER ON

RESET and POWER ON cause external subprogram calls to be interrupted and the associated load memory to be erased.

A subprogram selected for "Execution from external source" remains selected for "Execution from external source" after a RESET/part-program end. A POWER ON deletes the selection.

## Examples

### 1. Execution from local hard disk

Systems: SINUMERIK solution line/powerline with HMI Advanced

The "\_N\_MAIN\_MPF" main program is stored in NC memory and is selected for execution:

```
Program code
N010 PROC MAIN
N020 ...
N030 EXTCALL ("ROUGHING")
N040 ...
N050 M30
```

The "\_N\_ROUGHING\_SPF" subprogram to be reloaded is stored on the local hard disk in the directory "\_N\_WKS\_DIR/\_N\_WST1".

The subprogram path is preset in SD42700:

SD42700 \$SC\_EXT\_PROG\_PATH = "\_N\_WKS\_DIR/\_N\_WST1"

```
Program code
N010 PROC ROUGHING
N020 G1 F1000
N030 X= ... Y= ... Z= ...
N040 ...
...
...
N999999 M17
```

### 2. Execution from network drive

Systems: SINUMERIK solution line/powerline with HMI si/HMI Advanced/HMI Embedded

The "Contour2.spf" program to be reloaded is stored on the network drive in the directory "\\\R4711\\Workpieces".

```
Program code
...
N... EXTCALL ("\\\R4711\Workpieces\Contour2.spf")
...
```

## External program path

The path for the external subprogram directory can be preset using setting data:

SD42700 \$SC\_EXT\_PROG\_PATH

Together with the subprogram path or identifier specified with the EXTCALL call, this forms the entire path for the program to be called.

## **Effects**

### **EXTCALL call with absolute path name**

If the subprogram exists at the specified path, it will be executed following the EXTcall call. If it does not exist, program execution is cancelled.

### **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:

- If a path name is preset in SD42700, 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 results from linking the following characters:
  - The path name preset in SD42700
  - The "/" character as a separator
  - The subprogram path or identifier programmed in EXTcall
- If the called subprogram is not found at the preset path, the data specified in the EXTcall call is then searched for in the user-memory directories.
- If the called subprogram is not found on the program memory currently being searched (e.g., CompactFlash card), the next program memory (e.g., network drive) is searched in accordance with points 1 and 2.
- The search ends when the subprogram is found for the first time. If the search does not produce any hits, the program is canceled.

---

### **Note**

#### **SINUMERIK powerline with HMI Embedded**

An absolute path must always be specified in SINUMERIK powerline with HMI Embedded.

---

## External program memory

Depending on the system (SINUMERIK solution line/powerline), the available user interface (HMI sl/HMI Advanced/HMI Embedded) and the acquired options, external program memories may be stored on the following data carriers:

- CompactFlash card
- Network drive
- USB drive
- Local hard disk

---

### Note

#### Execution from external source via USB interface with SINUMERIK solution line

If external programs are to be transferred from an external USB drive (USB FlashDrive) via a USB interface, only the interface via X203 (named "TCU\_1") can be used.

### NOTICE

- USB FlashDrives are not suitable as persistent memory media.
- USB FlashDrives cannot be recommended for "execution from external".

Reasons:

- USB FlashDrives can drop-out or contact problems can be encountered in operation. The consequence – machining interruptions.
- USB FlashDrives can break-off when accidentally hit and damage the operator panel front.

---

### Note

#### Execution from external source via RS-232 interface with SINUMERIK powerline

In HMI Embedded, the "Execution from external source" softkey can be used to transfer external programs across the RS-232 interface onto the NC.

---

## Adjustable reload memory (FIFO buffer)

A reload memory is required in the NCK in order to run a program in "Execution from external source" mode (main program or subprogram). The size of the reload memory is preset to 30 Kbytes and, like all other memory-related machine data, can only be changed to match requirements by the machine manufacturer.

One reload memory must be set for each program (main program or subprogram) to run concurrently in "Execution from external source" mode.

#### Machine manufacturer

Please contact the machine manufacturer if the size and number of reload memories is to be extended.

For further information about "Execution from external source", see:

#### References:

/FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1).

### Block display, single block and behavior on NC stop

When executing from the hard disk and with `EXTCALL`, only the "Program run" HMI Advanced 3-block display can be used. This setting applies for the single block or NC stop status.

## 1.27.14 Suppress individual block (SBLOF, SBLON)

### Function

#### 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:

- To the next `SBLON`  
or
- To 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

## Significance

PROC	First instruction 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.

## Restrictions

- **Single block suppression and block display**

The current block display can be suppressed in cycles/subprograms using DISPLOF. If DISPLOF is programmed together with SBLOF, 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 re-activated 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	Comments
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:**

<b>Program code</b>	<b>Comments</b>
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:**

<b>Program code</b>	<b>Comments</b>
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:**

<b>Program code</b>	<b>Comments</b>
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.
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 instruction 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 instruction 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 instruction 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 instruction 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.

### 1.27.15 Suppress current block display (DISPLOF)

#### Function

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. Instead of the current block, the call of the cycle or the subprogram is displayed.

#### Syntax

```
PROC ... DISPLOF  
PROC ... DISPLOF ACTBLOCNO
```

#### Significance

DISPLOF

Command to suppress the current block display.

Location: At the end of the PROC instruction

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 this.

ACTBLOCNO

DISPLOF together with the ACTBLOCNO program 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.

#### Supplementary conditions

- If a subprogram with suppressed block display is interrupted by an asynchronous subprogram (ASUB), the blocks of the current subprogram are displayed.

## Examples

### Example 1: Suppress current block display in the cycle

Program code	Comments
PROC CYCLE (AXIS TOMOV, REAL POSITION) SAVE DISPOF	; Suppress current block display Instead, the cycle call should be displayed, e.g.: CYCLE(X,100.0)
DEF REAL DIFF	; Cycle contents
G01 ...	
...	
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	Comments
PROC SUBPROG1 DISPLOF ACTBLOCNO	
N8000 R10 = R33 + R44	
...	
N9040 R10 = 66 X100	; Output Alarm 12080
...	
N10000 M17	

Subprogram SUBPROG2 (without ACTBLOCNO):

Program code	Comments
PROC SUBPROG2 DISPLOF	
N5000 R10 = R33 + R44	
...	
N6040 R10 = 66 X100	; Output Alarm 12080
...	
N7000 M17	

Main program:

Program code	Comments
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	

### 1.27.16 Identifying subprograms with preparation (PREPRO)

#### Function

All files can be identified with the PREPRO keyword at the end of the PROC statement line during power up.

#### Machine manufacturer

This type of program preparation depends on the relevant set machine data. Please see the machine manufacturer's specifications for further details.

/FB3/ Function Manual, Special Functions; Preprocessing (V2)

#### Syntax

##### In the PROC statement line

PROC ... PREPRO

#### Significance

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 addressed in the same order both for subprograms prepared with parameters during startup and during subprogram call

1. \_N\_CUS\_DIR user cycles
2. \_N\_CMA\_DIR manufacturer cycles
3. \_N\_CST\_DIR standard cycles

. In cases of NC programs of the same name with different characteristics, the first PROC instruction found is activated and the other PROC instruction overlooked without an alarm message.

### 1.27.17 Cycles: Setting parameters for user cycles

**Function**

You can use the cov.com and uc.com files to parameterize your own cycles.

The cov.com file is included with the standard cycles at delivery and is to be expanded accordingly. The uc.com file is to be created by the user.

Both files are to be loaded in the passive file system in the "User cycles" directory (or must be given the appropriate path specification):

`; $PATH=/_N_CUS_DIR`

in the program.

**Files and paths**

cov.com\_COM

Overview of cycles

uc.com

Cycle call description

### **Adaptation of cov.com – Overview of cycles**

The cov.com file supplied with the standard cycles has the following structure:

%_N_COV_COM	File name
; \$PATH=/_N_CST_DIR	Path
;Vxxx 11.12.95 Sca cycle overview	Comment line
C1(CYCLE81) drilling, centering	Call for 1st cycle
C2(CYCLE82) drilling, counterboring	Call for 2nd cycle
...	
C24(CYCLE98) chaining of threads	Call for last cycle
M17	End of file

## **Syntax**

For each newly added cycle a line must be added with the following syntax:

C<number> (<cycle\_name>) comment\_text

Number: an integer as long as it has not already been used in the file;

Cycle name: The program name of the cycle to be included

Comment text: Optionally a comment text for the cycle

Example:

C25 (MY\_CYCLE\_1) usercycle\_1

C26 (SPECIAL CYCLE)

### Example: uc.com file - user cycle description

The explanation is based on the continuation of the example:

For the following two cycles a cycle parameterization is to be newly created:

Programming	Comments
<b>PROC MY_CYCLE_1 (REAL PAR1, INT PAR2, CHAR PAR3, STRING[10] PAR4)</b>	
The cycle has the following transfer parameters:	
PAR1:	; Real value in range -1000.001 <= PAR2 <= 123.456, default with 100
PAR2:	; Positive integer value between 0 <= PAR3 <= 999999, default with 0
PAR3:	; 1 ASCII character
PAR4:	; String of length 10 for a subprogram name
...	
M17	;

Programming	Comments
<b>PROC SPECIAL CYCLE (REAL VALUE1, INT VALUE2)</b>	
The cycle has the following transfer parameters:	
VALUE1:	; Real value without value range limitation and default
VALUE2:	; Integer value without value range limitation and default
...	
M17	;

Associated file uc.com:

Programming
%_N_UC_COM
; \$PATH=/_N_CUS_DIR
//C25(MY_CYCLE_1) usercycle_1
(R/-1000.001 123.456 / 100 /Parameter_2 of cycle)
(I/0 999999 / 1 / Integer value)
(C//"/A" / Character parameter)
(S///Subprogram name)
//C26(SPECIALCYCLE)
(R///Entire length)
(I/*123456/3/Machining type)
M17

**Example: both cycles**

Display screen for cycle MY\_CYCLE\_1

Parameter 2 of the cycle	100
Integer value	1
Character parameter	
Subroutines	

Display screen for cycle SPECIAL CYCLE

Total length	100
Machining type	1

## Syntax description for the uc.com file - user cycle description

### Header line for each cycle:

as in the cov.com file preceded by "://"

```
//C <number> (<cycle_name>) comment_text
```

### Example:

```
//C25(MY_CYCLE_1) usercycle_
```

### Line for description for each parameter:

```
><data_type_id> / <minimum_value> <maximum_value>
  <preset_value> /
```

### Data type identifier:

R	for real
I	for integer
C	for character (1 character)
S	for string

### Minimum value, maximum value (can be omitted)

Limitations of the entered values which are checked at input; values outside this range cannot be entered. It is possible to specify an enumeration of values which can be operated via the toggle key; they are listed preceded by "\*\*", other values are then not permissible.

### Example:

```
(I/*123456/1/Machining type)
```

There are no limits for string and character types.

### Default value (can be omitted)

Value which is the default value in the corresponding screen when the cycle is called; it can be changed via operator input.

### Comment

Text of up to 50 characters which is displayed in front of the parameter input field in the call screen for the cycle.

## **1.28    Macro technique (DEFINE ... AS)**



### **CAUTION**

Use of macros can significantly alter the control's programming language! Therefore, exercise caution when using macros.

#### **Function**

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**

Instruction 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.

#### **Activating**

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 <instruction 1> <instruction 2> ...

**Call in the NC program:**

<macro name>

## Significance

DEFINE ... AS	Keyword combination to define a macro
<macro name>	Macro name
	Only identifiers are permissible as macro names.
<instruction>	The macro is called from the NC program using the macro name.
	Programming instruction 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 2 digits.
- M and G functions can be programmed using 3 digits.



Keywords and reserved names may not be re-defined using macros.

## Supplementary conditions

- Nesting of macros is not possible.

## Examples

### Example 1: Macro definition at the beginning of the program

Program code	Comments
DEFINE LINE AS G1 G94 F300	; Macro definition
...	
...	
N70 LINE X10 Y20	; Macro call
...	

**Example 2: Macro definitions in a macro file**

Program code	Comments
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-in the external macro file into the control. Only then can macros be used in the NC program.

Program code	Comments
%_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 counterclockwise, 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	

# 2

## File and Program Management

### 2.1 Program memory

#### Function

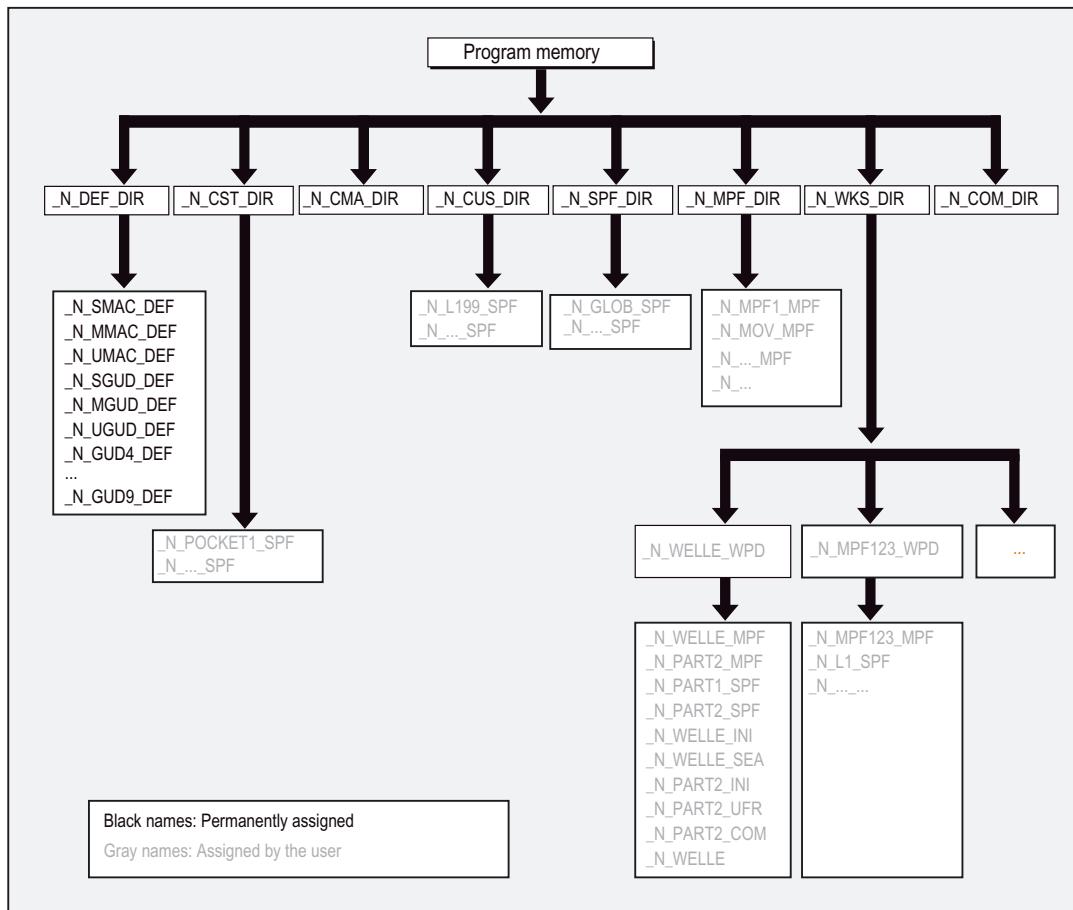
Files and programs (e.g., main programs and subprograms, macro definitions) are persistently saved in the program memory (→ passive file system).

#### References:

/FB2/ Function Manual for Extended Functions; Memory Configuration (S7)

A number of file types are also stored here temporarily and these can be transferred to the working memory as required (e.g., for initialization purposes on machining of a specific workpiece).

## 2.1 Program memory



## Standard directories

Its standard complement of directories is as follows:

Folder	Contents
<code>_N_DEF_DIR</code>	Data modules and macro modules
<code>_N_CST_DIR</code>	Standard cycles
<code>_N_CMA_DIR</code>	Manufacturer cycles
<code>_N_CUS_DIR</code>	User cycles
<code>_N_WKS_DIR</code>	Workpieces
<code>_N_SPF_DIR</code>	Global subprograms
<code>_N_MPF_DIR</code>	Main programs
<code>_N_COM_DIR</code>	Comments

## File types

The following file types can be stored in the main memory:

File type	Significance
name_MPMPF	Main program
name_SPFSPF	Subprogram
name_TEA	Machine data
name_SEA	Setting data
name_TOA	Tool offsets
name_UFR	Zero offsets/frames
name_INI	Initialization files
name_GUDGUD	Global user data
name_RPA	R parameters
name_COM	Comments
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)

To make data and program handling more flexible certain data and programs can be grouped together or stored in individual workpiece directories.

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_WELLE_WPD`, created for SHAFT workpiece contains the following files:

File	Significance
<code>_N_SHAFT_MPF</code>	Main program
<code>_N_PART2_MPF</code>	Main program
<code>_N_PART1_SPF</code>	Subprogram
<code>_N_PART2_SPF</code>	Subprogram
<code>_N_SHAFT_INI</code>	General initialization program for the data of the workpiece
<code>_N_SHAFT_SEA</code>	Setting data initialization program
<code>_N_PART2_INI</code>	General initialization program for the data for the Part 2 program
<code>_N_PART2_UFR</code>	Initialization program for the frame data for the Part 2 program
<code>_N_SHAFT_COM</code>	Comment file

**Creating workpiece directories on an external PC**

The steps described below are performed on an external data station. Please refer to your Operator's Guide for file and program management (from PC to control system) directly on the control.

**Creating a workpiece directory with a path name (\$PATH=...)**

The destination path `$PATH=...` is specified within the second line of the file. The file is then stored at the specified path.

**Example:**

Program code
%_N_SHAFT_MPF ;\$PATH=/_N_WKS_DIR/_N_WELLE_WPD N40 G0 X... Z... ... M2

File `_N_WELLE_MPF` is stored in directory `/_N_WKS_DIR/_N_WELLE_WPD`.

### Creating a workpiece directory without a path name

If the path name is missing, files with the \_SPF extension are stored in directory /\_N\_SPF\_DIR, files with the \_INI extension are stored in the RAM and all other files are stored in directory/\_N\_MPFI\_DIR.

Example:

Program code
%_N_SHAFT_SPF
...
M17

File \_N\_WELLE\_SPF is stored in directory /\_N\_SPF\_DIR.

### 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:

/BAD/ Operating Manual HMI Advanced; "Job list" and "Selecting a program for execution"

### Search paths for subprogram call

If the search path is not specified explicitly in the part program when a subprogram (or initialization file) is called, the calling program searches in a fixed search path.

#### Subprogram call with absolute path

Example

Program code
...
CALL"/_N_CST_DIR/_N_CYCLE1_SPF"
...

### Subprogram call without absolute path

Programs are usually called without specifying a path.

Example

<b>Program code</b>
...   CYCLE1   ...

The directories are searched for the called program in the following sequence:

No.	Folder	Significance
1	Current directory / <i>name</i>	Workpiece main directory or standard directory _N_MPFI_DIR
2	Current directory / <i>name_SPF</i>	
3	Current directory / <i>name_MPFI</i>	
4	/_N_SPF_DIR / <i>name_SPF</i>	Global subprograms
5	/_N_CUS_DIR / <i>name_SPF</i>	User cycles
6	/_N_CMA_DIR / <i>name_SPF</i>	Manufacturer cycles
7	/_N_CST_DIR / <i>name_SPF</i>	Standard cycles

### Programming search paths for subprogram call (CALLPATH)

The CALLPATH part program command is used to extend the search path of a subprogram call.

Example:

<b>Program code</b>
CALLPATH ("/_N_WKS_DIR/_N_MYWPD_WPD")   ...

The search path is stored in front of position 5 (user cycle) in accordance with the specified programming.

For further information about the programmable search path for subprogram calls with CALLPATH, see chapter "Extending the search path for subprogram calls with CALLPATH".

## 2.2 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
- Work offsets
- ...

### Initialization programs

These are programs with which the working memory data are initialized. The following file types can be used for this:

File type	Significance
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

You will find information on all of the file types in the Operator Manual for the operator interface.

### Data areas

The data can be organized in different areas in which they are to apply. For instance, a control can have several channels (not for SINUMERIK 840D NCU 571) or usually, also several axes.

The following exist:

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 are 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.

Program code	Comments
%_N_INITIAL_INI  CHANDATA(1)  \$MC_AXCONF_MACHAX_USED[0]=1 \$MC_AXCONF_MACHAX_USED[1]=2 \$MC_AXCONF_MACHAX_USED[2]=3  CHANDATA(2)  \$MC_AXCONF_MACHAX_USED[0]=4 \$MC_AXCONF_MACHAX_USED[1]=5  CHANDATA(1)  \$MA_STOP_LIMIT_COARSE[AX1]=0.2 \$MA_STOP_LIMIT_COARSE[AX2]=0.2  \$MA_STOP_LIMIT_FINE[AX1]=0.01 \$MA_STOP_LIMIT_FINE[AX1]=0.01	; Machine axis assignment, channel 1:  ; Machine axis assignment, channel 2:  ; Axial machine data: ; Exact stop window coarse: ; Axis 1 ; Axis 2 ; ;Exact stop window fine: ; Axis 1 ; Axis 2

**! CAUTION****CHANDATA statement**

In the part program, the CHANDATA instruction may only be set for that channel in which the NC program is executed; i.e. the instruction 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 instructions.

**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**

If the file is read-in with the name "INITIAL\_INI", then all data that are not supplied in the file are initialized using standard data. Only machine data are 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.

## **2.3 Define user data (DEF)**

### **Function**

#### **NOTICE**

User data (GUD) is defined at the time of start-up. The necessary machine data should be initialized accordingly. The user memory must be configured. All relevant machine data have as a component of their name GUD.

User data (GUD) can be defined in the operator interface in the operator area "Services". This eliminates the time-consuming reimport from data backup (%\_N\_INITIAL\_INI).

The following applies:

- Definition files that are on the hard disk are not active.
- Definition files that are on the NC are always active.

The individual GUD variables are programmed using the DEF command.

### **Syntax**

DEF <range> <VL stop> <type> <name>[ . . . , . . . ]=<value>

### **Significance**

DEF	Command to define GUD variables
<range>	The <range> designates the variable as GUD variable and defines the range of validity: NCK      Throughout the NCK CHAN     Throughout the channel
<VL stop>	Optional attribute preprocessing stop: SYNR      Preprocess stop while reading SYNW      Preprocess stop while writing SYNRW     Preprocessing stop when reading/writing
<type>	Data type: BOOL, REAL, INT, AXIS, FRAME, STRING or CHAR

<code>&lt;name&gt;</code>	Variable name
<code>[..., ...]</code>	Optional run limits for array variables
<code>&lt;value&gt;</code>	Optional default value
	Several values for arrays, separated using a comma or REP (w1), SET (w1, w2, ...), (w1, w2, ...).

**Note:**

Initialization values are not possible for type FRAME.

## Examples

### Example 1: Definition file, global data (Siemens)

Programming	Comments
<code>%_N_SGUD_DEF</code>	
<code>\$PATH=/_N_DEF_DIR</code>	
<code>DEF NCK REAL RTP</code>	; Retraction plane
<code>DEF CHAN INT SDIS</code>	; Safety clearance
<code>M30</code>	

### Example 2: Definition file, global data (machine manufacturer)

Programming	Comments
<code>%_N_MGUD_DEF</code>	
<code>\$PATH=/_N_DEF_DIR</code>	; Global data definitions of the machine manufacturer:
<code>DEF NCK SYNRW INT QUANTITY</code>	; Implicit preprocessing stop when reading/writing, spec. data available in the control, access from all channels.
<code>DEF CHAN INT TOOLTABLE[100]</code>	; Tool table for channel-spec. View of the tool number at the magazine locations, separate table set-up for each channel.
<code>M30</code>	

## Further Information

### Reserved block names

The following blocks can be stored in the directory /\_N\_DEF\_DIR:

_N_SMAC_DEF	contains macro definitions (Siemens system applications)
_N_MMAC_DEF	contains macro definitions (machine manufacturer)
_N_UMAC_DEF	contains macro definitions (user)
_N_SGUD_DEF	contains definitions for global data (Siemens system applications)
_N_MGUD_DEF	contains definitions for global data (machine manufacturer)
_N_UGUD_DEF	contains definitions for global data (user)
_N_GUD4_DEF	freely definable
_N_GUD5_DEF	contains definitions for measuring cycles (Siemens system applications)
_N_GUD6_DEF	Contains definitions for measuring cycles (Siemens system applications)
_N_GUD7_DEF	contains definitions for standard cycles (Siemens system applications)
_N_GUD8_DEF	Freely definable
_N_GUD9_DEF	Freely definable

---

### Note

If there are no measuring cycles/standard cycles, then the blocks reserved for this purpose can also be freely defined.

---

**Procedure: Defining user data (GUD)**

1. **Save block \_N\_INITIAL\_INI**
2. **Creating a definition file for user data**

Definition files can be created on the external PC or in the operator area "Services". There are also pre-defined file names (refer to "reserved block names"):

\_N\_SGUD\_DEF  
\_N\_MGUD\_DEF  
\_N\_UGUD\_DEF  
\_N\_GUD4\_DEF ... \_N\_GUD9\_DEF

Files with these names can contain definitions for GUD variables.

3. **Load definition file into the program memory of the control**

The control always sets-up as standard a directory \_N\_DEF\_DIR. This name is entered as the path in the header of the GUD definition file and evaluated when read in via the corresponding interface.

4. **Activate definition files and re-activate their content**

When the GUD definition file is loaded into the NC ("Load" soft key), it becomes active. If the content of a particular GUD definition file is re-activated, the old GUD data block in the active file system is deleted and the new parameters reset. If this process is undertaken via the dialog "Services" > "Manage data" > "Define and activate user data (GUD)", then the variable contents are saved by INI file and re-established at the end of the process.

5. **Data backup**

When the file \_N\_COMPLETE\_GUD is archived from the working memory, only the data contained in the file are saved. The created definition files for the global user variables must be separately archived.

The variable assignments to global user data are also saved in \_N\_INITIAL\_INI; the names must be identical with the names in the definition files.

## **2.4 Protection levels for user data, MD, SD and NC commands**

### **2.4.1 Defining protection levels for user data (APR, APW)**

#### **Function**

Access criteria can be defined for GUD modules to protect them against manipulation. GUD variables can be interrogated in cycles so that they can be protected against changes through the operator interface or from the program. The access protection applies to **all** variables defined in this module. When an attempt is made to access protected data, the control outputs an appropriate alarm.

#### **Syntax**

Protection levels for the complete block are specified after the header lines:

<b>Program code</b>	<b>Comments</b>
%_N_MGUD_DEF	; Header 1
\$PATH=/_N_DEF_DIR	; Header 2
APR<protection level> APW<protection level>	; Defining access protection

The access protection level is programmed with the desired protection level in the GUD module before any variable is defined. Vocabulary words must be programmed in a separate block.

#### **Significance**

APR	Access Protection to Read GUD variables is defined using the APR command.
APW	Access Protection for Writing to GUD variables is defined using the APW command.

<protection level>	Protection level
Type:	INT
Range of values:	From 0 or 10 (highest level) up to 7 or 17 (lowest level)
0 - 7	The read and write protection acts at the operator interface and in the NC program and in MDA operation.  These values are permissible in GUD blocks and in protection level data for individual variables in the REDEF instruction.
10 - 17	The read and write protection acts at the operator interface.  These values are only permissible for block-specific GUD protection level.
Meaning of the protection levels:	
0 or 10	SIEMENS
1 or 11	OEM_HIGH
2 or 12	OEM_LOW
3 or 13	end user
4 or 14	Keyswitch 3
...	...
7 or 17	Keyswitch 0

---

#### Note

In order to protect a complete file, the commands must be located before the first definitions of the file, otherwise in the REDEF instruction of the data involved (refer to "protection levels for NC language commands").

---

## **Example**

Definition file with:

- Write protection: Machine manufacturer
- Read protection: Keyswitch 2 at the operator interface

<b>Programming</b>	<b>Comments</b>
%_N_GUD6_DEF	
\$PATH=/_N_DEF_DIR	
APR15 APW12	; Protection levels for all of the following variables
DEF CHAN REAL_CORRVAL	
DEF NCK INT MYCOUNT	
...	
M30	

## **Further Information**

### **Activating a GUD definition file for the first time**

When a GUD definition file is first activated any defined access authorization contained therein is evaluated and automatically re-transferred to the read/write access of the GUD definition file.

---

### **Note**

Access authorization entries in the GUD definition file can restrict but not extend the required access authorization for the GUD definition file.

---

### **Example:**

The definition file \_N\_GUD7\_DEF contains: APW2

1. The file \_N\_GUD7\_DEF has value 3 as write protection. The value 3 is then overwritten with value 2.
2. The file \_N\_GUD7\_DEF has value 0 as write protection. There is no change to it.

With the APW statement a retrospective change is made to the file's write access.

With the APR statement a retrospective change is made to the file's read access.

---

### **Note**

If you erroneously enter in the GUD definition file a higher access level than your authorization allows, the archive file must be reimported.

---

## **2.4.2 Automatic activation of GUDs and MACs**

### **Function**

Definition files for GUD and macro definitions for HMI Advanced are edited in the "Services" operator area.

If a definition file is edited in the NC, when exiting the Editor you are prompted whether the definitions are to be set active.

### **Unloading the GUD and macro definitions**

If a definition file is unloaded, the associated data block is deleted after a query is displayed.

### **Loading the GUD and macro definitions**

If a definition file is loaded, a prompt is displayed asking whether to activate the file or retain the data. If you do not activate, the file is not loaded.

If the cursor is positioned on a loaded definition file, the soft key labeling changes from "Load" to "Activate" to activate the definitions. If you select "Activate", another prompt is displayed asking whether you want to retain the data.

Data is only saved for variable definition files, not for macros.

---

### **Note**

#### **HMI Advanced**

If there is not enough memory capacity to activate the definition file, once the memory size has been changed, the file must be transferred from the NC to the PCU and back to the NC again for activation.

---

**Example: Query when exiting the editor**

"Do you want to activate the definitions from file GUD7.DEF?"

"OK": A query is displayed asking if the currently active data should be saved:

"Do you want to keep the previous data of the definitions?"

OK": The GUD modules of the definition file to be edited will be saved, the new definitions will be activated and the saved data will be reloaded.

"Abort": The new definitions will be activated; the old data will be lost.

"Cancel": The changes in the definition file will be rejected; the associated data block is not changed.

### **2.4.3 Changing the protection levels for machine and setting data (REDEF, APR, APW)**

#### **Function**

The user can **change** protection levels for machine and setting data. Only protection levels of lower priority can be assigned to the machine data, setting data can also be assigned protection levels of higher priority.

#### **Syntax**

REDEF <MD/SD> APR<protection level> APW<protection level>

#### **Significance**

REDEF	Command to re-define ( <b>REDEFinition</b> ) the protection level of the machine and setting data
<MD/SD>	Machine data or setting data to which a protection level is to be assigned.
APR	<b>Access Protection to Read</b> the specified machine or setting data
APW	<b>Access Protection to Write</b> to the specified machine or setting data
<protection level>	Protection level
	Type: INT
	Range of values: From 0 (highest level) to 7 (lowest level)

## Resetting machine/setting data

To undo a change to the protection levels, the original protection levels must be written back again.

## Examples

### Example 1: Changing rights for individual MD

Program code
%_N_SGUD_DEF ; \$PATH=/_N_DEF_DIR REDEF \$MA_CTRLOUT_SEGMENT_NR APR2 APW2 REDEF \$MA_ENC_SEGMENT_NR APR2 APW2 REDEF \$SN_JOG_CONT_MODE_LEVELTRIGGRD APR2 APW2 M30

### Example 2: Resetting rights for individual MD to the original value

Program code
%_N_SGUD_DEF ; \$PATH=/_N_DEF_DIR REDEF \$MA_CTRLOUT_SEGMENT_NR APR7 APW2 REDEF \$MA_ENC_SEGMENT_NR APR0 APW0 REDEF \$SN_JOG_CONT_MODE_LEVELTRIGGRD APR7 APW7 M30

## 2.4.4 Changing protection levels for NC language elements and system variables (REDEF, APX, APW)

### Function

The protection level concept to access machine/setting data and GUDs can be extended to NC language elements and system variables.

---

#### Note

An NC language element with access protection is only executed in the part program execution if the corresponding execution right exists.

## Syntax

Access protection for NC language element:

REDEF <NC language element> APX<protection level>

Write access from part program or synchronized actions to system variables:

REDEF <system variable> APW<protection level>

## Significance

REDEF	Command to re-define ( <b>REDEFinition</b> ) the protection level
	Effective: Global for all channels and BAGs (mode groups)
<NC language element>	NC language element that should be assigned a protection level for execution  Possible NC language elements include: <ul style="list-style-type: none"><li>• Predefined subprogram calls</li><li>• Predefined functions</li><li>• DO instruction for synchronized actions</li><li>• G functions/motion conditions</li><li>• Program identifier for cycles The cycle must be saved in one of the cycle directories and must contain a PROC instruction.</li></ul>
APX	Access Protection for execution the specified language element
<system variable>	System variable for which a protection level is to be assigned for write access  <b>Note:</b> Read access to a system variable is always possible.
APW	Access Protection to Write to the specified system variables
<protection level>	Protection level  Type: INT Range of values: From 0 (highest level) to 7 (lowest level) For the significance of the individual protection levels, see " Defining protection levels for user data (APR, APW) ".  <b>Note:</b> Keyswitch setting 0 (protection level 7) corresponds to the pressing of all available NC commands.

## Example: Subprogram call in definition files

Program code
N10 REDEF GEOAX APX3 N20 IF(ISFILE("/_N_CST_DIR/_N_SACCESS_SUB1_SPF") N30 PCALL /_N_CST_DIR/_N_SACCESS_SUB1_SPF N40 ENDIF N40 M17

## Further Information

### Definition files

Like for the GUD definitions, separate definition files exist that are evaluated on control start-up:

End user:	/_N_DEF_DIR/_N_UACCESS_DEF
Manufacturer:	/_N_DEF_DIR/_N_MACCESS_DEF
Siemens:	/_N_DEF_DIR/_N_SACCESS_DEF

### Subprogram call in definition files

It is possible to call subprograms containing REDEF statements from the above definition files. The REDEF instructions must always be at the beginning of the data part just like the DEF statements. The subprograms must have the extension SPF or MPF and inherit the write-protection of the definition files set using machine data MD11170 to MD11172.

### Access protection for machine data/setting data

---

#### Note

As soon as definition files are active for the protection stage new definition of the NC language elements, the previous protection stage new definitions from machine data/setting data - set-up in GUD definition files - must be moved to the new protection level definition files. This means that setting protection levels for machine and setting data is only permitted if the above specified protection level definition files and is rejected in the GUD definition files (Alarm!).

Setting the initialization attributes and synchronization attributes is still only possible in the GUD definition files.

---

#### **Protection levels for system variables**

Protection levels for system variables only apply to value assignments via the part program command. The protection level concept of the particular operator interface is effective in the operator interface.

## **2.5      Changing attributes of NC language elements (REDEF)**

### **Function**

When extended using the REDEF instruction, functions to define data objects and define protection levels described in the Chapter "Protection levels" become general functions to set attributes and values for NC language elements.

### **Syntax**

```
REDEF <NC language element> <attribute> <value>
REDEF <name>
```

### **Significance**

REDEF	Command to re-define ( <b>REDEF</b> inition)
<NC language element>	This includes:
	<ul style="list-style-type: none"><li>• GUD</li><li>• R parameters</li><li>• Machine data/setting data</li><li>• Synchronous variables (\$AC_PARAM, \$AC_MARKER, \$AC_TIMER)</li><li>• System variables that can be written from part programs (refer to the Manual "List of system variables")</li><li>• User frames (G500, etc.)</li><li>• Magazine/tool configurations</li></ul>

&lt;attribute&gt; &lt;value&gt;

**Initializations**

<value>:	Permissible for:
INIPO	GUD, R parameters, synchronous vars
INIRE	GUD, R parameters, synchronous variables
INICF	GUD, R parameters, synchronous variables
PRLOC	Setting data

**Note:**

Refer below for the significance of the attributes

**Synchronization**

<value>:	Permissible for:	Specifying a standard value value
SYNR	GUD	Preprocess stop while reading
SYNW	GUD	Preprocess stop while writing
SYNRW	GUD	Preprocessing stop while reading and writing

**Access right**

<value>:	Permissible for:	Specifying a standard value value
APW	Machine and setting data	Write access rights
APR	Machine and setting data	Access right during read

**Note:**

For machine and setting data, it is possible to subsequently overwrite the previously set access rights. The permissible values extend from "0" (Siemens password) to "7" (keyswitch setting 0).

&lt;name&gt;

The settings for APX, APR, APW are set to standard values and INIPO, INIRE, INICF, PRLOC are reset again.

**Optional parameters:**

<value>

Optional parameters for the attributes INIPO, INIRE, INICF, PRLOC: Subsequent starting value(s)

Forms:

Individual e.g. 5  
value

Value list e.g. (0,1,2,3,4,5,6,7,8,9) for variable with 10  
elements

with:

- REP (<w1>)

<w1>: Value list to be repeated

For a variable with several elements

Example: REP (12)

- SET (<w1>,<w2>,<w3>,...)

- (<w1>,<w2>,<w3>,...)

Value list

<protection level>

Required parameter, protection level for attributes for APR or APW

For **GUD**, the definition can contain a starting value (DEF NCK INT \_MYGUD=5). If this start value is not specified (e.g. in DEF NCK INT \_MYINT), the start value can be defined subsequently in the REDEF instruction.

The initialization value for an array applies to all array elements.

Individual elements can be set using an initialization list or REP ( ).

Examples:

REDEF\_MYGUD INIRE 5

REDEF\_MYGUD INIRE 0,1,2,3,4,5,6,7,8,9

REDEF\_MYGUD INIRE REP(12,14,16,18,20)

**Cannot** be used for R parameters and system variables.

Only constants can be assigned.

Expressions are not permitted values.

**Significance of the attribute:**

INIPO	<b>INIt for Power On</b> For a buffered NC restart, the data are overwritten with standard value(s).
INIRE	<b>INIt for operator panel front Reset or TP end</b> At the end of a main program, with e.g. M2, M30, etc. or an abort with reset, data are overwritten with the standard value. INIRE is also effective for INIPO.
INICF	<b>INIt for NewConf request or TP command NEWCONF</b> For a NewConf request or TP command, NEWCONF data are overwritten with the standard value. INICF is also effective for INIRE and INIPO.
PRLOC	Only local program change If data is changed in a part program, subprogram, cycle or ASUB, then after the end of the main program (end with M2, M30, etc. or when aborting using an operator panel front reset) it reassumes its original value. This attribute is only permissible for programmable setting data (see programmable setting data).

The user must realize the **synchronization** of the event triggering the initialization. This means, for example, if a part program end is executed in two **different channels**, then the variables are initialized for each of these operations. That affects global and axial data!

**Programmable setting data and the writable system variables from the part program**

The following setting data can be initialized with the REDEF instruction:

Number	Name of identifier	GCODE
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_VEL0_G25	G25
43220	\$SA_SPIND_MAX_VEL0_G26	G26
43230	\$SA_SPIND_MAX_VEL0_LIMS	LIMS
43300	\$SA_ASSIGN_FEED_PER_REV_SOURCE	FPRAO
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_VEL0	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

You will find the list of system variables in the Manual "List of system variables". All system variables that are marked W (write) or WS (write with preprocessing stop) in the column "part program" can be initialized with the RESET instruction.

## Example

Program code	Comments
	; Reset behavior with GUD:
/_N_DEF_DIR/_N_SGUD_DEF	
DEF NCK INT _MYGUD1	; Definitions
DEF NCK INT _MYGUD2=2	
DEF NCK INT _MYGUD3=3	
	; Initialization on operator panel front reset/end of part program:
DEF _MYGUD2 INIRE	; Initialization
M17	

This sets "\_MYGUD2" back to "2" on operator panel front reset / end of part program whereas "\_MYGUD1" and "\_MYGUD3" retain their value.

## Example: Modal speed limiting in the part program (setting data)

Program code	Comments
/_N_DEF_DIR/_N_SGUD_DEF	
REDEF \$SA_SPIND_MAX_VELO_LIMS PRLOC	; Setting data for speed limit
M17	

Program code
/_N_MPFI_DIR/_N_MY_MPFI
N10 SETMS (3)
N20 G96 S100 LIMS=2500
...
M30

The speed limit defined in the setting data (SD43230 \$SA\_SPIND\_MAX\_VELO\_LIMS) should be 1200 rpm. In a part program that has been set-up and has been completely tested, a higher speed is permissible, so that LIMS=2500 is programmed here. After the end of the program, the value configured in the setting data takes effect here again.

Set the settings back the the standard values and delete the initializations again

New definition	Attribute	Standard values, reset initializations
REDEF	NC language element	APX=7
REDEF	Machine/setting data	Reset APW=7 APR=7 PRLOC
REDEF	Synchronization variable	Reset APW=7 INIRE, INIPO, INICF
REDEF	GUD, LUD	Reset INIRE, INIPO, INICF

**Example:**

Program code	Comments
REDEF MASLON APX2	
REDEF SYG RS INIRE APW3	
REDEF R[ ] INIRE	
REDEF MASLON	; APX set to 7.
REDEF SYG RS	; APW set to 7 and INIRE deleted.
REDEF R[ ]	; INIRE deleted.

## Restrictions

- The **change** to the attributes of NC objects can only be made **after definition** of the object. In particular, it is necessary to pay attention to the DEF.../ REDEF sequence for GUD. (Setting data/system variables are implicitly created before the definition files are processed). The symbol must always be defined first (implicitly by the system or by the DEF statement) and only then can the REDEF be changed.
- If two or more concurrent attribute changes are programmed, the last change is always active.
- Attributes of arrays** cannot be set for individual elements but only always **for the entire array**:

```

DEF CHAN INT _MYGUD[10,10]
REDEF _MYGUD INIRE                                // ok
REDEF _MYGUD[1,1] INIRE                            // not possible, alarm is output
                                                    // (array value)

```

- Initialization of **GUD arrays** themselves is not affected.

```
| DEF NCK INT _MYGUD[10] =(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
| DEF NCK INT _MYGUD[100,100] = REP (12)
| DEF NCK INT _MYGUD[100,100] ;
```

- REDEF statements with **R parameters** must be enclosed in parentheses.

```
| REDEF R[ ] INIRE
```

- INI attributes**

However, it should be noted that when the INI attributes for these variables are set, an appropriately large **memory for INIT values** must be available, which can be set using MD18150 \$MN\_MM\_GUD\_VAL\_MEM. Bit 1 in MD11270 \$MN\_DEFAULT\_VALUES\_MEM\_MASK must be set to 1 (memory for initialization values active). Too small a memory cause alarm 12261 "Initialization not allowed"

- R parameters and system variables**

For R and system variables, it is not possible to specify a default value that differs from a value that has already been compiled-in. However, resetting to the compiled value is possible with INIPO, INIRE, or INICF.

- For **data type FRAME of GUD** it is not possible to specify a default deviating from the compiled value either (like for definition of the data item).

- GUD (DEF NCK INT\_MYGUD)**

Only the INIPO attribute is permissible for global GUD (DEF NCK INT\_MYGUD).

Only the data in the corresponding channel is initialized for channel-specific GUD (DEF CHAN INT\_MYGUD) with the corresponding result (RESET, BAG-RESET or NewConfig).

**Example:** 2 channels are defined with the channel-specific GUD that is to be initialized during RESET:

```
DEF CHAN INT _MYGUD
REDEF _MYGUD INIRE
```

During a RESET in the first channel, the GUD for this channel is reset and the value in the second channel is not affected.

### Setting a default value

If the behavior of a system variable or GUD is changed using REDEF <name> INIRE, INIPO; INICF; PRLOC, machine data

MD11270 \$MN\_DEFAULT\_VALUES\_MEM\_MASK must be set to 1 (memory for initialization values active). Otherwise, alarm 12261 "Initialization not allowed" is output.

## 2.6 Structuring instruction in step editor (SEFORM)

### Function

The structuring instruction **SEFORM** is evaluated in the step editor (editor-based program support) to generate the step view for HMI Advanced. The step view is used to improve the readability of the NC subprogram.

### Syntax

**SEFORM(<section name>,<level>,<icon>)**

### Significance

<b>SEFORM()</b>	Function call of the structuring instruction with parameters <b>&lt;section name&gt;</b> , <b>&lt;level&gt;</b> and <b>&lt;icon&gt;</b>
<b>&lt;section name&gt;</b>	Identifier of the working step
<b>&lt;level&gt;</b>	Type: STRING Index for the main or sub-level
	Type: INT
	Value: 0 Main level
	1, ..., <n> Lower level 1, ..., lower level <n>
<b>&lt;icon&gt;</b>	Name of the icon displayed for this section. Type: STRING

---

### Note

SEFORM instructions are generated in the Step editor.

The string transferred with the **<section name>** parameter is stored main-run-synchronously in the OPI variable in a similar way to the **MSG** instruction. The information remains until overwritten by the next SEFORM instruction. Reset and end of part program clear the content.

The NCK checks during the part program execution the parameters **<level>** and **<icon>** but does not process them any further.

---

### References

For additional information on editor-based program support, see:  
Operating Manual HMI Advanced.

# 3

## Protection zones

### 3.1 Definition of the protection zones (CPROTDEF, NPROTDEF)

#### Function

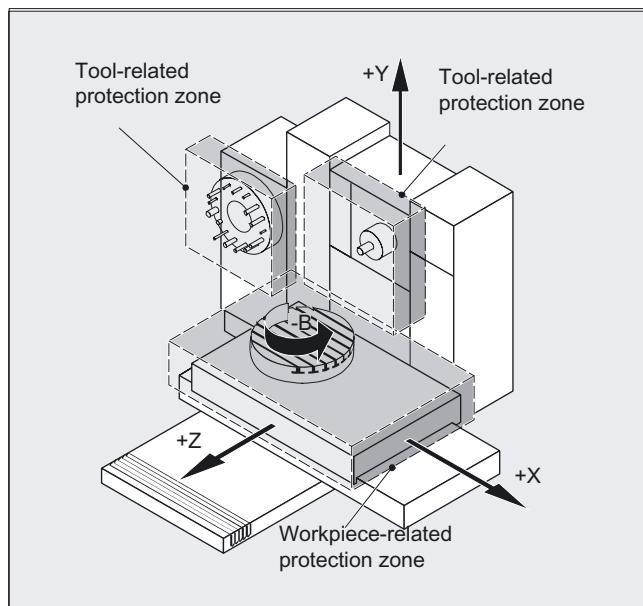
You can use protection zones to protect various elements on the machine, their components and the workpiece against incorrect movements.

**Tool-oriented** 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).



## *Protection zones*

### *3.1 Definition of the protection zones (CPROTDEF, NPROTDEF)*

#### Syntax

```
DEF INT NOT_USED  
CPROTDEF(n,t,applim,appplus,appminus)  
NPROTDEF(n,t,applim,appplus,appminus)  
EXECUTE(NOT_USED)
```

#### Significance

DEF INT NOT_USED	Define local variable, data type integer (see Motion-synchronous action section)
CPROTDEF	Define channel-specific protection zones (for NCU 572/573 only)
NPROTDEF	Defining machine-specific protection zones
EXECUTE	End definition
n	Number of defined protection zone
t	TRUE = <b>Tool-related</b> protection zone FALSE = <b>workpiece</b> protection zone
applim	Type of limitation in the third dimension 0 = No limit 1 = Limit in positive direction 2 = Limit in negative direction 3 = Limit in positive and negative direction
appplus	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 in protection zones with EXECUTE

#### Significance

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-oriented protection zones are defined with reference to the tool carrier reference point F.

## Contour definition of protection zones

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. The travel motions programmed between CPROTDEF or NPROTDEF and EXECUTE are not executed, but merely define the protection zone.

## Plane

The required plane is selected before CPROTDEF and NPROTDEF with G17, G18, G19 and must not be altered before EXECUTE. The applicate must not be programmed between CPROTDEF or NPROTDEF and EXECUTE.

## Contour elements

The following is permissible:

- G0, G1 for straight contour elements
- G2 for clockwise circle segments (only for tool-oriented protection zones)
- G3 for circular segments in the counterclockwise direction.

---

### Note

If a full circle describes the protection zone, it must be divided into two half circles. The order 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.

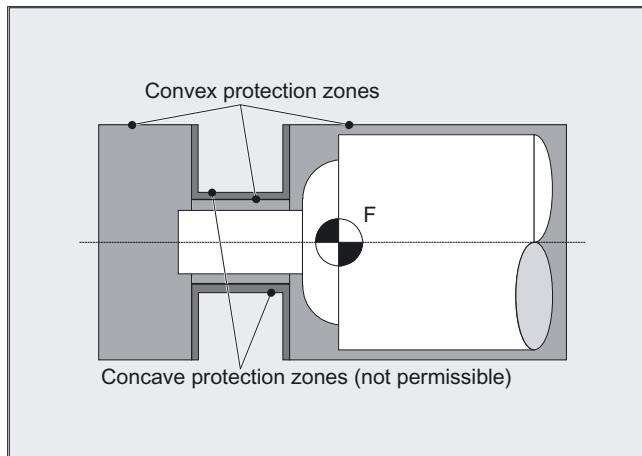
---

**External protection zones** (only possible for workpiece-related protection zones) must be defined in the clockwise direction.

For **rotation-symmetric** protection zones (e.g. spindle chuck), you must describe the **complete contour** and not only up to the center of rotation!.

Tool-oriented protection zones must always be **convex**. If a concave protected zone is desired, this should be subdivided into several convex protection zones.

### *3.2 Activating/deactivating protection zones (CPROT, NPROT)*



During definition of the protection zones

- no cutter or tool nose radius compensation,
- no transformation,
- no frame must be active.

Nor must reference point approach (G74), fixed point approach (G75), block search stop or program end be programmed.

## **3.2 Activating/deactivating protection zones (CPROT, NPROT)**

### **Function**

Activating and preactivating previously defined protection zones for collision monitoring and deactivating protection zones.

The maximum number of protection zones, which can be active simultaneously on the same channel, is defined in machine data.

If no toolrelated 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)
```

## Significance

CPROT	Call channel-specific protection zone (for NCU 572/573 only)
NPROT	Call machine-specific protection zone
n	Number of 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

## Example of milling

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:

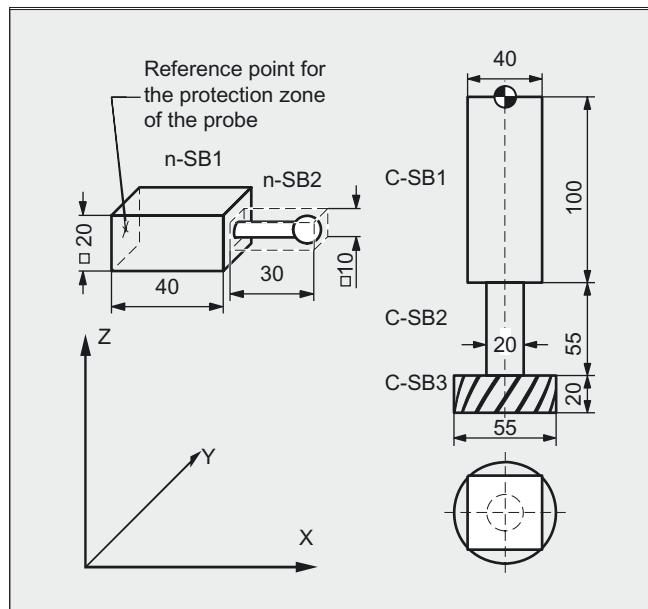
- A machine-specific and a workpiece-related protection zone for both the measuring probe holder (n-SB1) and the measuring probe itself (n-SB2).
- A channel-specific and a tool-oriented protection zone for the milling cutter holder (c-SB1), the cutter shank (c-SB2) and the milling cutter itself (c-SB3).

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.

## Protection zones

### 3.2 Activating/deactivating protection zones (CPROT, NPROT)



Program code	Comments
DEF INT PROTECTB	; Definition of a Help variable
Definition of protection zone G17	
NPROTDEF(1, FALSE, 3, 10, -10) G01 X0 Y-10	; Set orientation
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)	

## 3.2 Activating/deactivating protection zones (CPROT, NPROT)

Program code	Comments
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

**Activation status**

A protection zone is generally activated in the parts program with status = 2.

The status is always channel-specific even for machine-oriented protection zones.

If a PLC user program provides for a protection zone to be effectively set by a PLC user program, the required preactivation is implemented with status = 1.

The protection zones are deactivated and therefore disabled with Status = 0. No offset is necessary.

**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

`$SN_PA_ACTIV_IMMED [n] = TRUE` must be set for this.

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-oriented protection zones must have the same orientation on both channels.

## **3.3 Checking for protection zone violation, working area limitation and software limits**

### **Function**

The CALCPOSI function is for checking whether, starting from a defined starting point, the geometry axes can traverse a defined path without violating the axis limits (software limits), working area limitations, or protection zones.

If the defined path cannot be traversed, the maximum permissible path is returned.

The CALCPOSI function is a predefined subprogram. It must be alone in a block.

### **Syntax**

```
Status=CALCPOSI(_STARTPOS, _MOVEDIST, _DLIMIT, _MAXDIST, _BASE_SYS,  
_TESTLIM)
```

---

3.3 Checking for protection zone violation, working area limitation and software limits**Significance**

Status	<p>0: Function OK, the specified path can be completely traversed.</p> <ul style="list-style-type: none"> <li>-: At least one component is negative in _DLIMIT</li> <li>-: An error has occurred in a transformation calculation</li> </ul> <p>If the specified path cannot be completely traversed, then a positive, decimal-coded value is returned:</p> <p><b>Units digit (type of violated limit):</b></p> <ul style="list-style-type: none"> <li>1: Software limits are limiting the traversing path.</li> <li>2: Working area limits are limiting the traversing path.</li> <li>3: Software limits are limiting the traversing path.</li> </ul> <p>If several limits are violated simultaneously (e.g. software limits and protection zones), the limit that results in the most significant restriction in the specified traversing path is indicated in the units digit.</p> <p><b>Tens digit</b></p> <p>10: The initial value violates the limit</p> <p>20: 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 zone, curved software limits in the WCS for non-linear transformations, e.g. transmit).</p> <p><b>Hundreds digit</b></p> <p>100: The positive limit value is violated (only if the units digit is 1 or 2, i.e. for software limits and working area limits)</p> <p>100: An NCK protection zone is violated (only if the units digit is 3).</p> <p>200: The negative limit value is violated (only if the units digit is 1 or 2, i.e. for software limits and working area limits)</p> <p>200: A channel-specific protection zone is violated (only if the units digit is 3).</p>
--------	---

**Thousands digit**

1000:

Factor, by which the number of the axis that violates the limit is multiplied (only if the units digit is either 1 or 2, i.e. for software limits and working area limits).

The axis count starts at 1 and refers in the case of violated software limits (units digit = 1) to the machine axes and for violated working area limits (units digit = 2) to the geometry axes.

1000:

Factor by which the number of the violated protection zone is multiplied (only if the units digit is 3).

If several protection zones are violated, the limit resulting in the greatest limitation of the traversing path is indicated in the hundreds and thousands digit of the protection zone.

\_STARTPOS

Starting value for abscissa [0], ordinate [1] and applicate [2] in the (WCS)

\_MOVEDIST

Incremental path definition for abscissa [0], ordinate [1] and applicate [2]

\_DLIMIT

[0] - [2]: Minimum clearances assigned to the geometry axes.

[3]: Minimum clearance assigned to a linear machine axis for non-linear transformation, if no geometry axis can be uniquely assigned.

[4]: Minimum clearance assigned to a rotary machine axis for non-linear transformation, if no geometry axis can be uniquely assigned. Only for special transformations, if SW limits are to be monitored.

\_MAXDIST

Array [0] - [2] for the return value. Incremental path in all three geometry axes without violating the defined minimum clearance of an axis limit in the machine axes involved.

If the traversing path is not restricted, the contents of this return parameter are the same as the contents of \_MOVEDIST.

*3.3 Checking for protection zone violation, working area limitation and software limits*\_BASE\_SYS

FALSE or parameters not specified:

When evaluating the position and length data, the G code of group 13 (G70, G71, G700, G710; inch/metric) is evaluated. If G70 is active and the basic system is metric (or G71 is active and inch), the WCS system variables \$AA\_IW[X] and \$AA\_MW[X] are provided in the basic system and must, if necessary, be reconverted using the CALCPOSI function.

TRUE:

When evaluating the position and length data, the basic system of the control is always used independent of the value of the active G of group 13.

\_TESTLIM

Limits to be checked (binary-coded):

- 1: Monitor software limits
- 2: Monitor working area limits
- 3: Monitor activated protection zones
- 4: Monitor pre-activated protection zones

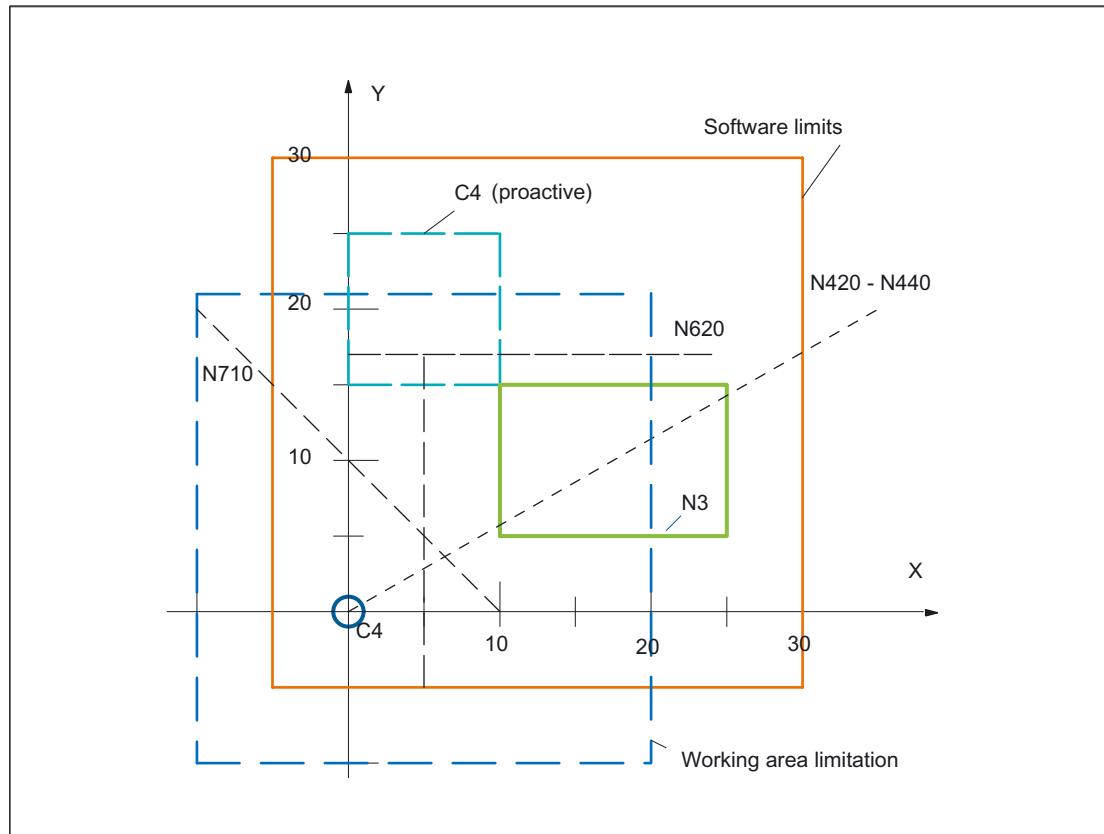
Combinations by adding the values. Default: 15; check all.

**Example**

In the example (refer to the diagram), X software limits and working area limits are shown. In addition, three protection zones are defined, the two channel-specific protection zones C2 and C4 and well as the NCK protection zone N3. C2 is a circular, active, tool-related protection zone with a radius of 2 mm. C4 is a square, pre-activated and workpiece-related protection zone with 10 mm side length and N3 is a rectangular, active protection zone with 10 mm and 15 mm side lengths. In the following NC, initially, the protection zones and the working field limits are defined as sketched, and the CALCPOSI function is called with various parameter assignments. The results of each individual CALCPOSI call are summarized in the table at the end of the example.

## Protection zones

### 3.3 Checking for protection zone violation, working area limitation and software limits



Program code	Comments
N10 def real _STARTPOS[3]	
N20 def real _MOVEDIST[3]	
N30 def real _DLIMIT[5]	
N40 def real _MAXDIST[3]	
N50 def int _SB	
N60 def int _STATUS	
N70 cprotdef(2, true, 0)	; Toolrelated protection zone
N80 g17 g1 x-y0	
N90 g3 i2 x2	
N100 i-x-	
N110 execute(_SB)	
N120 cprotdef(4, false, 0)	; Workpiece-related protection zone
N130 g17 g1 x0 y15	
N140 x10	
N150 y25	
N160 x0	
N170 y15	
N180 execute(_SB)	

## 3.3 Checking for protection zone violation, working area limitation and software limits

Program code	Comments
N190 nprotdef(3, false, 0)	; Machine-related protection zone
N200 g17 g1 x10 y5	
N210 x25	
N220 y15	
N230 x10	
N240 y5	
N250 execute(_SB)	
N260 cprot(2,2,0, 0, 0)	; Activate or pre-activate protection zones
N270 cprot(4,1,0, 0, 0)	
N280 nprot(3,2,0, 0, 0)	
N290 g25 XX=-YY=-	; Define working area limit groups
N300 g26 xx= 20 yy= 21	
N310 _STARTPOS[0] = 0.	
N320 _STARTPOS[1] = 0.	
N330 _STARTPOS[2] = 0.	
N340 _MOVEDIST[0] = 35.	
N350 _MOVEDIST[1] = 20.	
N360 _MOVEDIST[2] = 0.	
N370 _DLIMIT[0] = 0.	
N380 _DLIMIT[1] = 0.	
N390 _DLIMIT[2] = 0.	
N400 _DLIMIT[3] = 0.	
N410 _DLIMIT[4] = 0.	
;Various function calls	; Other starting point
N420 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST)	
N430 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST,,3)	
N440 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST,,1)	
N450 _STARTPOS[0] = 5.	; Other destination
N460 _STARTPOS[1] = 17.	
N470 _STARTPOS[2] = 0.	
N480 _MOVEDIST[0] = 0.	
N490 _MOVEDIST[1] =-.	
N500 _MOVEDIST[2] = 0.	

Program code	Comments
<pre>;Various function calls N510 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST,,14) N520 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST,, 6) N530 _DLIMIT[1] = 2. N540 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST,, 6) N550 _STARTPOS[0] = 27. N560 _STARTPOS[1] = 17.1 N570 _STARTPOS[2] = 0. N580 _MOVEDIST[0] =-. N590 _MOVEDIST[1] = 0. N600 _MOVEDIST[2] = 0. N610 _DLIMIT[3] = 2. N620 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST,, 12) N630 _STARTPOS[0] = 0. N640 _STARTPOS[1] = 0. N650 _STARTPOS[2] = 0. N660 _MOVEDIST[0] = 0. N670 _MOVEDIST[1] = 30. N680 _MOVEDIST[2] = 0. N690 trans x10 N700 arot z45 N710 _STATUS = calcposi(_STARTPOS,_MOVEDIST, _DLIMIT, _MAXDIST) N720 M30</pre>	

## 3.3 Checking for protection zone violation, working area limitation and software limits

**Results of the tests in the example:**

Block No. N...	_STATUS	_MAXDIST [0] (= X)	_MAXDIST [1] (= Y)	Remarks
420	3123	8.040	4.594	Protection zone SB N3 violated.
430	1122	20.000	11.429	No protection zone monitoring, working area limits violated.
440	1121	30.000	17.143	Now only monitoring of the software limits active.
510	4213	0.000	0.000	Starting point violates protection C4
520	0000	0.000	−.000	Pre-activated protection zone C4 is not monitored. Defined path can be traversed completely.
540	2222	0.000	−.000	Because _DLIMIT[1]=2, the traversing path is restricted by the working area limits.
620	4223	−.000	0.000	Clearance to C4 is a total of 4 mm due to C2 and _DLIMIT[3]. Clearance C2 – N3 of 0.1 mm does not lead to limiting the traversing path.
710	1221	0.000	21.213	Frame with translation and rotation active The permissible traversing path in _MOVEDIST applies in the shifted and rotated coordinate system (WCS).

### Special cases and further details

All path data are always entered as radii even if for a facing axis with active G code "DIAMON". If the part of one of the involved axes cannot be traversed completely, the paths of the other axes will also be reduced accordingly in the \_MAXDIST return value so that the resulting end point lies on the specified path.

It is permissible that no software limits, operating range limits or protection zones are defined for one or more of the axes involved. All limits are only monitored if the axes involved are referenced. Any involved rotary axes are monitored only if they are not modulo axes.

---

*3.3 Checking for protection zone violation, working area limitation and software limits*

As in the normal traversing operation, the monitoring of the software limits and the operating range limits depends on the active settings (interface signals for selecting the software limits 1 or software limits 2, GWALIMON/WALIMOF, setting data for the specific activation of the operating range limits and for the specification whether or not the radius of the active tool is to be considered for the monitoring of the operating range limits).

For certain kinematic transformations (e.g. TRANSMIT), the position of the machine axes cannot be determined uniquely from the positions in the workpiece coordinate system (WCS) (non-uniqueness). In the normal traversing operation, the uniqueness normally results from the previous history and the condition that a continuous movement in the WCS must correspond to a continuous movement in the machine axes. When monitoring the software limits using the CALCPOSI function, the current machine position is therefore used to resolve non-unique determinability in such cases. If necessary, a **STOPRE** must be programmed in front of CALCPOSI to input valid machine axis positions to the function.

It is not guaranteed that the separation to the protection zones specified in \_DLIMIT[3] can always be maintained for a movement on the specified traversal path. Therefore if the end point returned in \_MOVEDIST is lengthened by this distance, no protection zone is violated, even though the straight line may pass extremely close to a protection zone.

---

**Note**

You will find details on working area limitations in the  
/PG/ Fundamentals Programming Guide,

on the software limits in  
/FB1/ Function Manual, Basic Functions; Axis Monitoring, Protection Zones (A3).

---

# 4

## Special Motion Commands

### 4.1 Approaching coded positions (CAC, CIC, CDC, CACP, CACN)

#### Function

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>)

#### Significance

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 Value range: 0, 1, ... (max. number of table locations - 1)

## *Special Motion Commands*

---

### **4.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)**

#### **Example: Approach coded positions of a positioning axis**

<b>Programming code</b>	<b>Comments</b>
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**

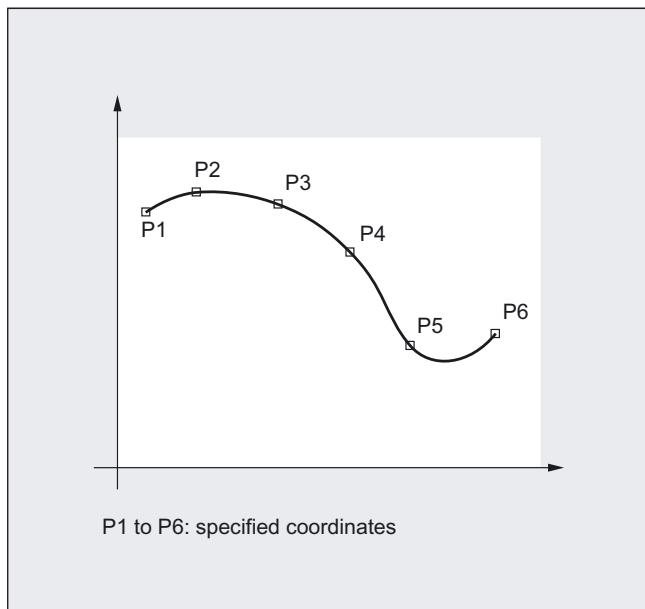
- Function Manual Expanded Functions; Indexing Axes (T1)
- Function Manual, Synchronized Actions

## **4.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)**

#### **Function**

Random curved workpiece contours cannot be precisely defined in an analytic form. This is the reason that these type of contours are approximated using a limit 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
```

## *Special Motion Commands*

---

### *4.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)*

## **Significance**

### **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...      Positions in Cartesian coordinates  
A... B... C...     

### **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: $0 \leq n \leq 3$
	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):**

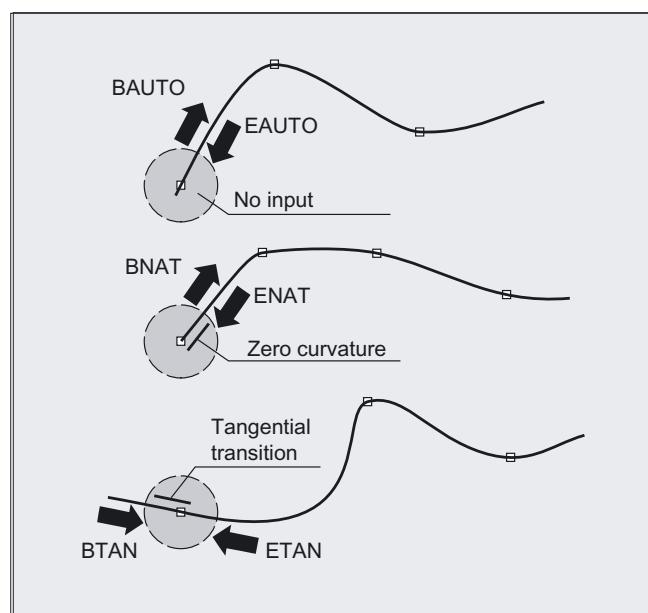
PL	The distances between nodes are suitably calculated internally. The control can also machine pre-defined node clearances that are specified in the so-called parameter-interval-length using the PL command.
<value>	Parameter interval length Range of values: As for path dimension

**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.

## Special Motion Commands

### 4.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)

## 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
```

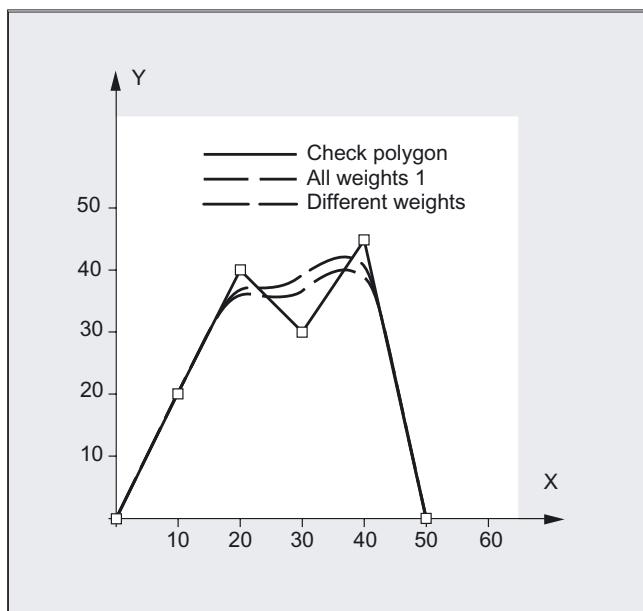
#### Program code 2 (different weights)

```
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)

#### Comments

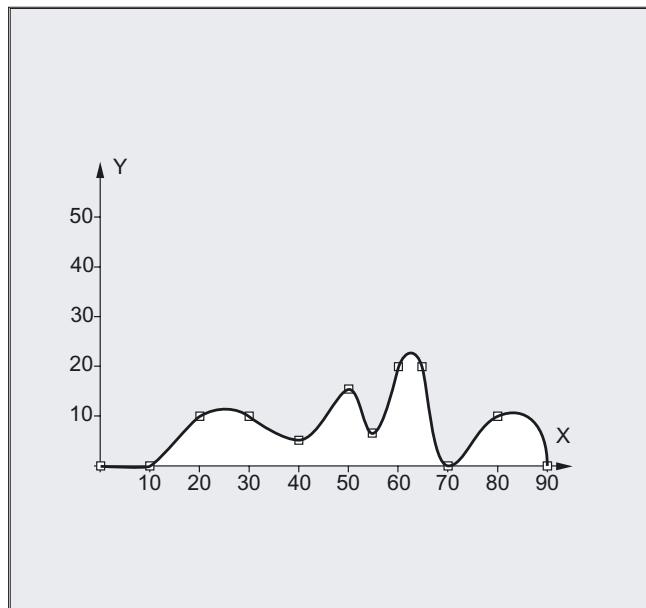
N10 G1 X0 Y0 F300 G64 N20 ; N/A N30 X10 Y20 N40 X20 Y40 N50 X30 Y30 N60 X40 Y45 N70 X50 Y0
--



**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
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
```



## *Special Motion Commands*

---

### *4.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)*

#### **Example 3: Spline interpolation (A spline) and coordinate transformation (ROT)**

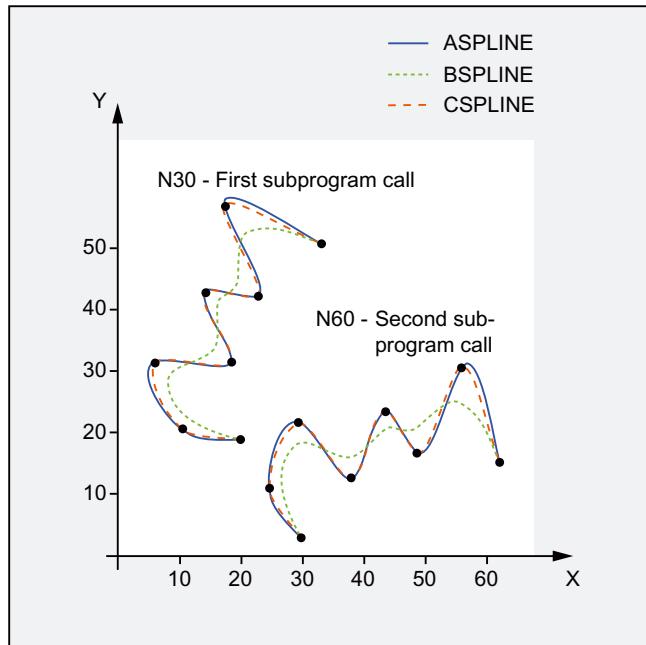
Main program:

<b>Program code</b>	<b>Comments</b>
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):

<b>Program code</b>
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):



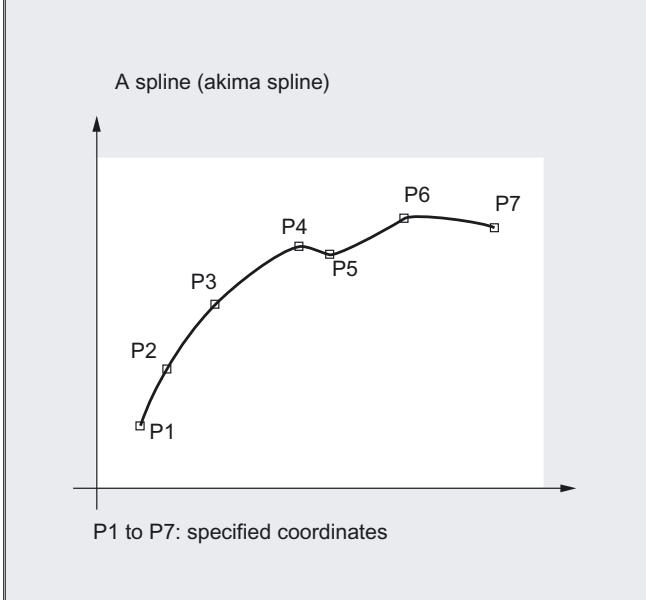
## Further Information

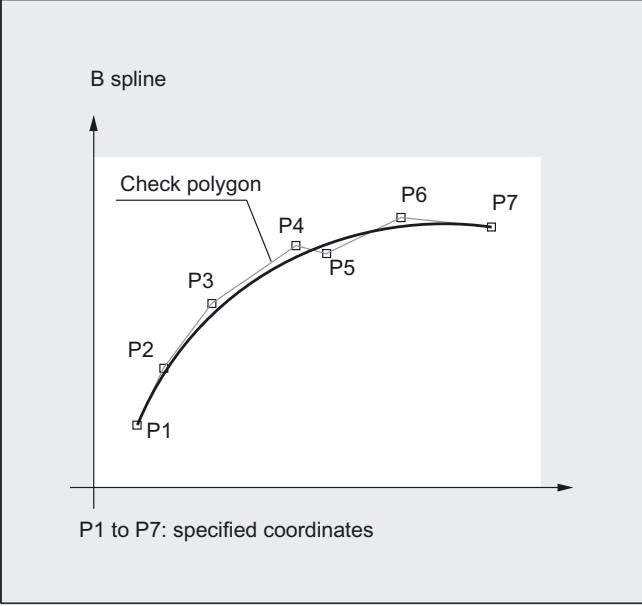
### Advantages of spline interpolation

By using 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.

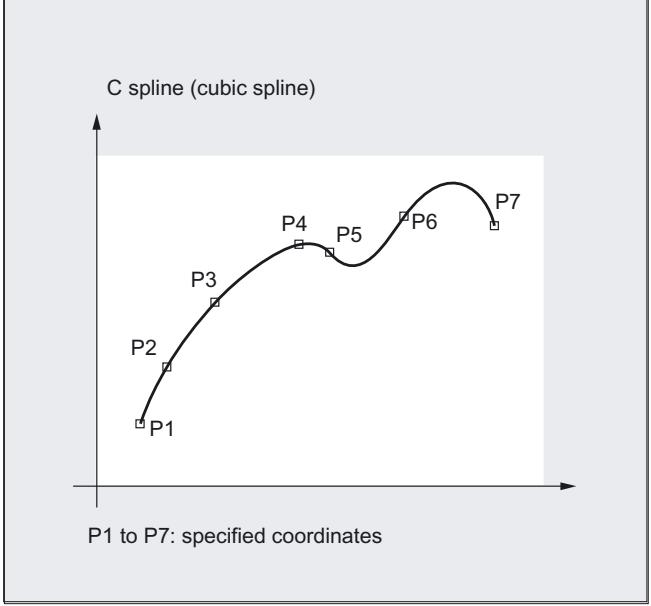
**Properties and use of the various spline types**

Spline type	Properties and use
<b>A spline</b>	<p>A spline (akima spline)</p>  <p>P1 to P7: specified coordinates</p> <p><b>Properties:</b></p> <ul style="list-style-type: none"> <li>The passes precisely through the specified intermediate points along the curve.</li> <li>The curve characteristic is tangential, but does not have continuous curvature.</li> <li>Produces hardly any undesirable oscillations.</li> <li>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.</li> </ul> <p><b>Application:</b></p> <p>The A spline is especially suitable for interpolating curves with large changes in the gradient (e.g. staircase-type curves and characteristics).</p>

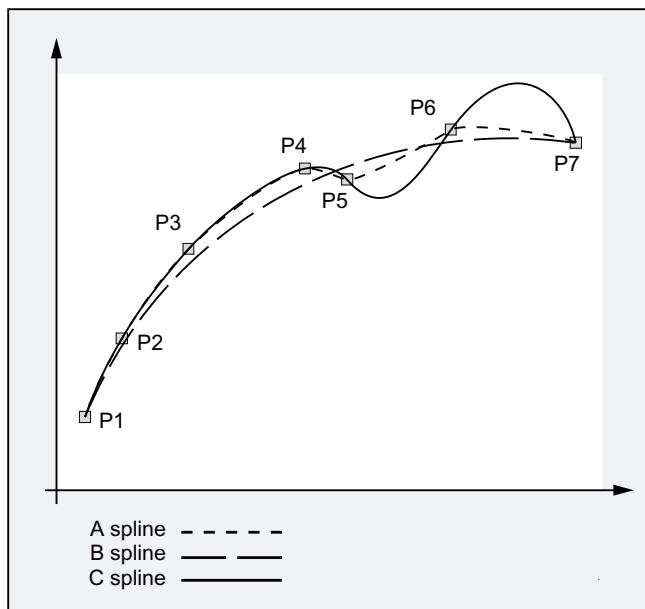
Spline type	Properties and use
B spline	<p><b>B spline</b></p>  <p>P1 to P7: specified coordinates</p> <p><b>Features:</b></p> <ul style="list-style-type: none"> <li>• Does not run through the specified intermediate points along the curve, but only close to them. The intermediate points do not attract the curve. The curve characteristic can be additionally influenced by weighting the intermediate points using a factor.</li> <li>• The curve characteristic is tangential with continuous curvature.</li> <li>• Does not generate any undesirable oscillations.</li> <li>• 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.</li> </ul> <p><b>Application:</b></p> <p>Their B spline is primarily intended as interface to CAD systems.</p>

## Special Motion Commands

### 4.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN, EAUTO, ENAT, ETAN, PW, SD, PL)

Spline type	Properties and use
C spline	<p>C spline (cubic spline)</p>  <p>P1 to P7: specified coordinates</p> <p><b>Features:</b></p> <ul style="list-style-type: none"> <li>The passes precisely through the specified intermediate points along the curve.</li> <li>The curve characteristic is tangential with continuous curvature.</li> <li>Frequently generates undesirable oscillations, especially at positions where the gradient changes significantly.</li> <li>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.</li> </ul> <p><b>Application:</b></p> <p>The C spline can be well used if the intermediate points lie on a curve that can be analytically calculated defined (circle, parabola, hyperbola).</p>

## 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 controller needs a certain number of spline end-point blocks for every 10 blocks:

Spline type	Minimum number of spline blocks
A spline:	At least <b>4</b> blocks out of every 10 must be spline blocks. These do not include comment blocks or parameter calculations.
B spline:	At least <b>6</b> 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 used to calculate the spline segment is entered in MD20160. The default setting is 8. As standard, at least <b>9</b> 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

## **4.3 Spline grouping (SPLINEPATH)**

### **Function**

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:

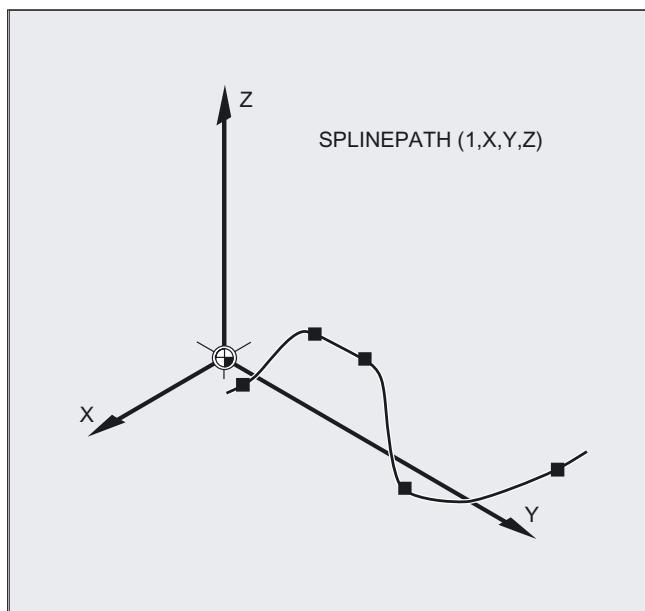
SPLINEPATH (n, X, Y, Z, ...)

## Significance

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	Comments
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 interpolation



## 4.4 NC block compression (**COMPON**, **COMPCURV**, **COMPCAD**, **COMPOF**)

### Function

CAD/CAM systems normally produce linear blocks, which meet the configured accuracy specifications. In the case of complex contours, a large volume of data and short path sections can result. The short path sections restrict the processing rate.

By using a compressor function, the contour, specified by using linear blocks, is approached using polynomial blocks. This has the following advantages:

- Reduction of the number of required part program blocks for describing the workpiece contour
- Continuous block transitions
- Higher maximum path velocities

The following compressor functions are available:

- **COMPON**

The block transitions are only constant in the **velocity**, while acceleration of the participating axes can be in jumps at block transitions.

- **COMPCURV**

Block transitions have **continuous acceleration**. This ensures both smooth velocity and acceleration of all axes at block transitions.

- **COMPCAD**

The compression that uses a lot of computation time and memory space is optimized regarding surface quality and speed. COMPCAD should only be used if measures to improve the surface cannot be taken by the CAD/CAM program in advance.

**COMPOF** terminates the compressor function.

### Syntax

COMPON  
COMPCURV  
COMPCAD  
COMPOF

## Significance

COMPON	Command to activate the compressor function COMPON. Effective: modal
COMPCURV	Command to activate the compressor function COMPCURV. Effective: modal
COMPCAD	Command to activate the compressor function COMPCAD. Effective: modal
COMPOF	Command to deactivate the currently active compressor function.

---

### Note

The rounding function G642 and jerk limitation SOFT can be used to achieve further improvements in surface quality. These commands must be written at the beginning of the program.

---

## Supplementary conditions

- The NC block compression can only be executed for linear blocks (G1).
- Only blocks that comply with a simple syntax are compressed:  
  
N... G1X... Y... Z... F... ;comment  
All other blocks are executed unchanged (no compression).
- Motion blocks with extended addresses such as C=100 or A=AC(100) are also condensed.
- The position values do not have to be programmed directly, but can also be indirectly specified using parameter assignments, e.g. X=R1\*(R2+R3).
- If the option "orientation transformation" is available, then NC blocks in which the tool orientation (and where relevant, also the tool rotation) is programmed using direction vectors can also be compressed (see "Compressing the orientation (Page 369)").
- It is interrupted by any other type of NC instruction, e.g., an auxiliary function output.

## *Special Motion Commands*

### *4.4 NC block compression (COMPON, COMPCURV, COMPCAD, COMPOF)*

## Examples

### Example 1: COMPON

Program code	Comments
N10 COMPON	; Compressor function COMPON on.
N11 G1 X0.37 Y2.9 F600	; G1 before end point and feed.
N12 X16.87 Y-.698	
N13 X16.865 Y-.72	
N14 X16.91 Y-.799	
...	
N1037 COMPOF	; Compressor function off.
...	

### Example 2: COMPCAD

Program code	Comments
G00 X30 Y6 Z40	
G1 F10000 G642	; Blending function G642 on.
SOFT	; Jerk limiting SOFT on.
COMPCAD	; Compressor function COMPCAD on.
STOP FIFO	
N24050 Z32.499	
N24051 X41.365 Z32.500	
N24052 X43.115 Z32.497	
N24053 X43.365 Z32.477	
N24054 X43.556 Z32.449	
N24055 X43.818 Z32.387	
N24056 X44.076 Z32.300	
...	
COMPOF	; Compressor function off.
G00 Z50	
M30	

## References

Function Manual, Basic Functions; Continuous-Path Mode, Exact Stop, Look Ahead (B1), Chapter: "NC block compression"

## 4.5 Polynomial interpolation (POLY, POLYPATH, PO, PL)

### Function

Polynomial interpolation (POLY) is not spline interpolation in the true sense. 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 applied optimally in cases where the coefficients are supplied directly by a CAD system or postprocessor.

### Syntax

3rd degree polynomial:

```
POLY PO[X]=(xe,a2,a3) PO[Y]=(ye,b2,b3) PO[Z]=(ze,c2,c3) PL=n
```

or extension to 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")
```

### Significance

POLY	Activation of polynomial interpolation with a block containing POLY.
POLYPATH	Polynomial interpolation can be selected for the 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 <sub>2</sub> , a <sub>3</sub> , a <sub>4</sub> , and a <sub>5</sub> 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.

### Activate/deactivate POLY

Polynomial interpolation belongs to G0, G1, G2, G3, A spline, B spline and C spline in the first G group. If it is active, then it is not necessary to program the polynomial syntax: Axes that are only programmed with their name and end point are linearly traversed to their end point. If all axes are programmed in this way, the control behaves as for G1.

Polynomial interpolation is deactivated using another command of the G group (e.g. 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.

New polynomial syntax with PO: The previous syntax still remains valid.

### Subprogram call POLYPATH

The polynomial interpolation can be selectively specified for the following axis groups using POLYPATH:

- POLYPATH ("AXES")  
All path axes and supplementary axes.
- POLYPATH ("VECT")  
Orientation transformation (for orientation transformation).

The programmed polynomials are also interpolated as polynomials as standard for both axis groups.

**Examples:**

POLYPATH ("VECT")

Only the orientation axes are selected for polynomial interpolation - all other axes traverse linearly.

POLYPATH ( )

Deactivates polynomial interpolation for all axes

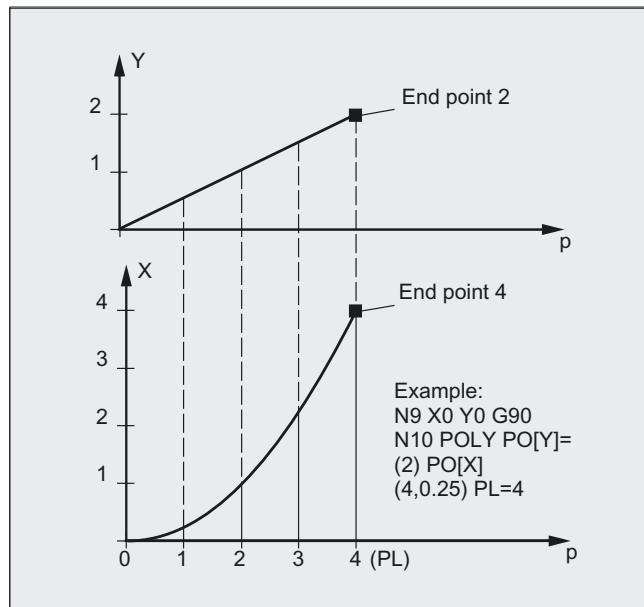
## Example

Program code	Comments
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: Applicable polynomial syntax with PO

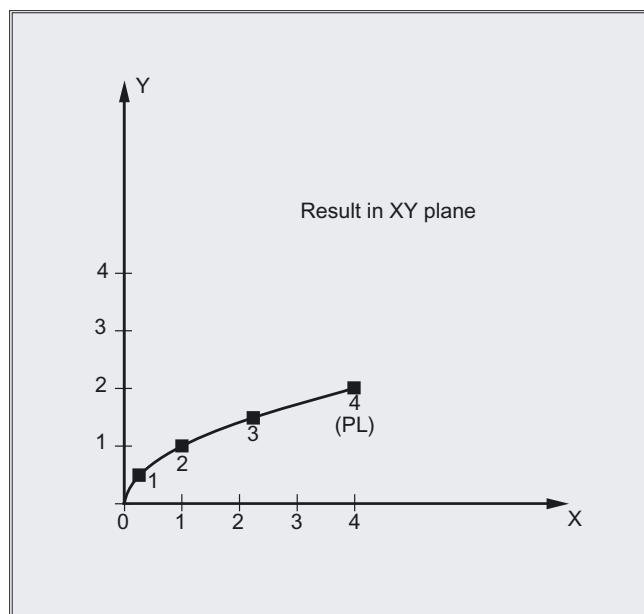
Previous polynomial syntax remains valid	New polynomial syntax (from SW 6)
PO[axis identifier]=(.. , ..)	Axis identifier=PO(.. , ..)
PO[PHI]=(.. , ..)	PHI=PO(.. , ..)
PO[PSI]=(.. , ..)	PSI=PO(.. , ..)
PO[THT]=(.. , ..)	THT=PO(.. , ..)
PO[]=(.. , ..)	PO(.. , ..)
PO[variable]=IC(.. , ..)	variable=PO IC(.. , ..)

**Example: Curve in the X/Y plane.**



**Program code**

```
N9 X0 Y0 G90 F100  
N10 POLY PO[Y]=(2) PO[X]=(4,0.25) PL=4
```



## Description

The control system is capable of traveling curves (paths), in which every selected path axis is operating as a polynomial function up to the 5th order.

The equation used to express the polynomial function is generally as follows:

$$f(p) = a_0 + a_1 p + a_2 p^2 + a_3 p^3$$

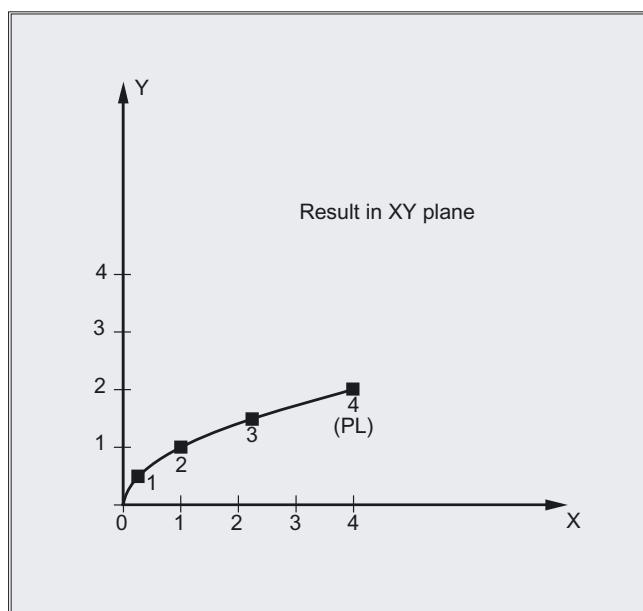
or

$$f(p) = a_0 + a_1 p + a_2 p^2 + a_3 p^3 + a_4 p^4 + a_5 p^5$$

Key:

$a_n$ : Constant coefficients

$p$ : Parameter



By assigning concrete values to these coefficients, it is possible to generate a wide variety of curve shapes such as line, parabola and power functions.

For setting the coefficients  $a_2 = a_3 = 0$  or  $a_2 = a_3 = a_4 = a_5 = 0$  yields, for example, a straight line with:

$$f(p) = a_0 + a_1 p$$

The following 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 POLY G code being active. In this case, however, the programmed polynomials are not interpolated; instead the respective programmed end point of each axis is approached linearly (G1). The polynomial interpolation is then activated by programming POLY.

Also, if G code POLY is active, with the predefined subprogram POLYPATH(...), you can select which axes are to be interpolated with polynomial.

### Special features of the 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 forms such as conics (circle, ellipse, parabola, hyperbola) exactly.

#### Example

Program code	Comments
POLY G90 X10 Y0 F100	; Geometry axes traverse linearly to position X10 Y0.
PO[X]=(0,-) 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 specified end point is not dependent on G90 / G91.

The result obtained from the above example is as follows:

$$X(p) = 10(1) / (1+p^2) \text{ and}$$

$$Y(p) = 20p / (1+p^2)$$

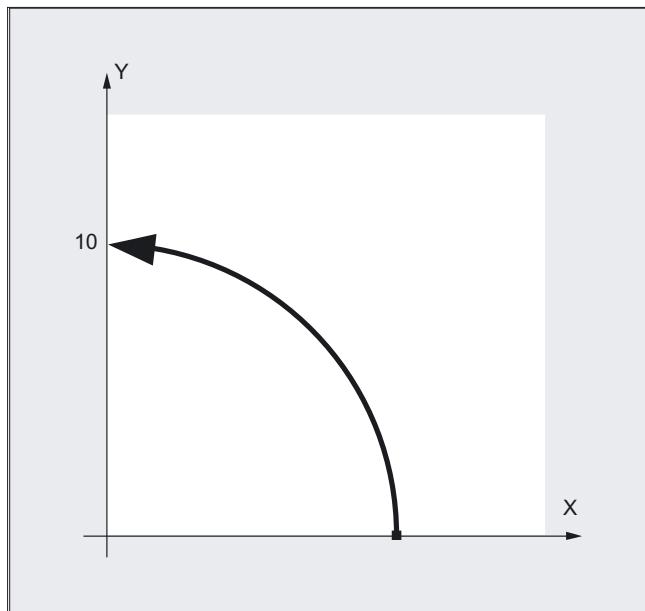
with  $0 \leq p \leq 1$

As a result of the programmed start points, end points, coefficient  $a_2$  and PL=1, the intermediate values are as follows:

$$\text{Numerator (X)} = 10+0*p-p^2$$

$$\text{Numerator (Y)} = 0+20*p+0*p^2$$

$$\text{Denominator} = 1+2*p+1*p^2$$



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.

---

## 4.6 Settable path reference (SPATH, UPATH)

### Function

During polynomial interpolation, the user may require two different relationships between the velocity determining FGROUP axes and the other path axes: The latter should either be controlled, synchronized to the path of the FGROUP axes or synchronized to the curve parameters.

Therefore, for the axes not contained in FGROUP, there are two ways to follow the path:

- In synchronism to path S (SPATH)  
or
- In synchronism to curve parameter U of the FGROUP axes (UPATH)

Both types of path interpolation are used in different applications and can be changed over via G codes SPATH and UPATH.

### Syntax

SPATH  
UPATH

### Significance

SPATH	Path reference for FGROUP axes is arc length
UPATH	Path reference for FGROUP axes is curve parameter
FGROUP	Definition of axes with path feed

### SPATH, UPATH

One of the two G codes (SPATH, UPATH) can be used to select and program the required behavior.

The commands are modal. If SPATH is active, the axes are traversed synchronized with the path; if UPATH is active, traversal is synchronized with the curve parameter.

UPATH and SPATH also define the interrelationship of the F word polynomial (FPOLY, FCUB, FLIN) with path motion.

### FGROUP activation

The path reference for the axes that are not contained in FGROUP, is set via the two language commands SPATH and UPATH contained in the 45th G code group.

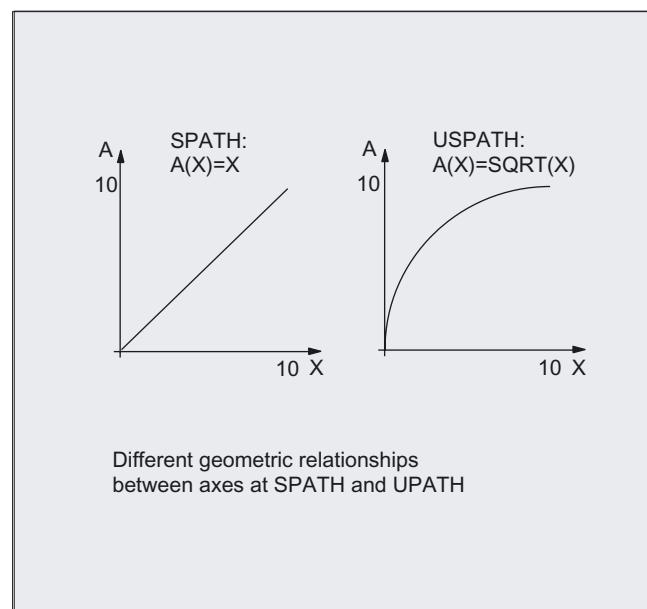
## Example 1

The following example shows a square with 20 mm side lengths and corners rounded-off with G643. The maximum deviations from the precise contour are defined for each axis using machine data MD33100 \$MA\_COMPRESS\_POS\_TOL[...].

Program code	Comments
N10 G1 X... Y... Z... F500	
N20 G643	; Block-internal corner rounding with G643
N30 X0 Y0	
N40 X20 Y0	; mm edge length for the axes
N50 X20 Y20	
N60 X0 Y20	
N70 X0 Y0	
N100 M30	

## Example 2

The following example shows the difference between both types of motion control. Both times the default setting FGROUP(X,Y,Z) is active.



**Programming**

```
N10 G1 X0 A0 F1000 SPATH  
N20 POLY PO[X]=(10,10) A10
```

or:

**Programming**

```
N10 G1 X0 F1000 UPATH  
N20 POLY PO[X]=(10,10) A10
```

In block N20, path S of the FGROUP axes 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.

### Restrictions

The path reference set is of no importance with

- linear and circular interpolation,
- in thread blocks and
- if all path axes are contained in FGROUP.

### Description

During polynomial interpolation - and thus the polynomial interpolation is always understood

- in the narrow sense (POLY),
  - all spline interpolation types (ASPLINE, BSPLINE, CSPLINE) and
  - linear interpolation with compressor (COMPON, COMPCURV)
- are the positions of all path axes i specified by the polynomials  $\pi_i(U)$ . Curve parameter U moves from 0 to 1 within an NC block, therefore it is standardized.

The axes to which the programmed path feed is to relate can be selected from the path axes by means of language command FGROUP. However, during polynomial interpolation, an interpolation with constant velocity on path S of these axes usually means a non constant change of curve parameter U.

## Control behavior for reset and machine option data

After reset, MD 20150: GCODE\_RESET\_VALUES [44] makes certain G codes effective (45th G code group).

The initial state for the type of the smoothing is specified with MD 20150: GCODE\_RESET\_VALUES [9] (10th G code group).

The G code group value active after Reset is determined via machine data MD 20150: GCODE\_RESET\_VALUES [44]. In order to maintain compatibility with existing installations, SPATH is set as default value.

The axial machine data MD 33100: COMPRESS\_POS\_TOL have an extended meaning: They contain the tolerances for the compressor function and for rounding with G642.

## 4.7 Measurements with touch trigger probe (MEAS, MEAW)

### Function

The positions coinciding with the switching edge of the probe are acquired for all axes programmed in the NC block and written for each specific axis to the appropriate memory cell. Maximum two probes exist.

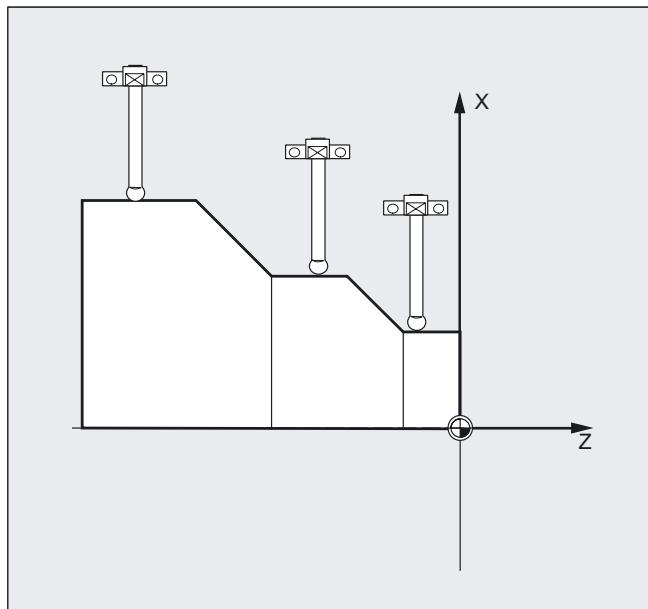
#### Read measurement result

The measurement result is available for the axes acquired with probes in the following variables:

- Under \$AA\_MM[axis] in the machine coordinate system
- Under \$AA\_MW[axis] in the workpiece coordinate system

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. The system will otherwise read false values.



## Syntax

### Programming measurement blocks, MEAS, MEAW

When command **MEAS** is programmed in conjunction with an interpolation mode, actual positions on the workpiece are approached and measured values recorded simultaneously. The distance-to-go between the actual and setpoint positions is deleted.

The **MEAW** function is employed in the case of special measuring tasks where a programmed position must always be approached. **MEAS** and **MEAW** are non-modal commands.

MEAS=±1	G... X... Y... Z...	(+1/+2 measurement with deletion of distance-to-go and rising edge)
MEAS=±2	G... X... Y... Z...	(-1/-2 measurement with deletion of distance-to-go and falling edge)
MEAW=±1	G... X... Y... Z...	(+1/+2 measurement without deletion of distance-to-go and rising edge)
MEAW=±2	G... X... Y... Z...	(-1/-2 measurement without deletion of distance-to-go and falling edge)

## Significance

MEAS=±1	Measurement with probe 1 at measuring input 1
MEAS=±2*	Measurement with probe 2 at measuring input 2
MEAW=±1	Measuring with probe 1 at measuring input 1
MEAW=±2*	Measuring with probe 2 at measuring input 2
G...	Interpolation type, e.g., G0, G1, G2 or G3
X... Y... Z...	End point in Cartesian coordinates

\*Max. of two inputs depending on configuration level

## Example for programming measurement blocks

MEAS and MEAW are programmed in a block with motion commands. The feeds and interpolation types (G0, G1, ...) must be selected to suit the measuring task in hand; this also applies to the number of axes.

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 job status

If an evaluation is required in the program as to whether the probe has been triggered or not, status variable \$AC\_MEA[n] (n= number of the measuring probe) can be interrogated:

0 measurement task not fulfilled

1 measurement task successfully completed (the probe has been triggered)

---

### Note

If the program is deflected in the program, the variable is set to 1. When starting 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). For MEAS, the motion is braked in a defined fashion after the probe has been triggered.

---

**Note**

If a GEO axis is programmed in a measurement block, the measured values are stored for all current GEO 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 saved.

---

## **4.8      Extended measuring function (MEASA, MEAWA, MEAC, TE) (option)**

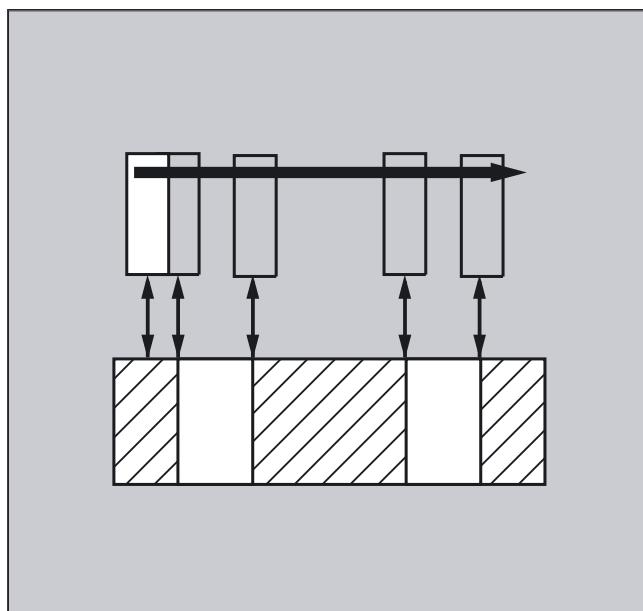
### **Function**

Several probes and several measuring systems can be used for the axial measuring.

For **MEASA**, **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. The maximum number of measured values per measurement is also four with **MEAC**:

- Under **\$AA\_MM1 to 4 [axes]** in the machine coordinate system
- Under **\$AA\_MM1 to 4 [axes]** in the workpiece coordinate system



## Syntax

MEASA and MEAWA act blockwise and can be programmed in a block. If MEASA/MEAWA is programmed with MEAS/MEAW in the same block, an error message is output.

MEASA[axis]=(mode, TE1, ..., TE4)

MEAWA[axis]=(modes, TE1, ..., TE4)

MEAC[axis]=(mode, measuring memory, TE1, ..., TE4)

## Significance

MEASA	Measurement with deletion of distance-to-go
MEAWA	Measurement without deletion of distance-to-go
MEAC	Continuous measurement without deleting distance-to-go
Axis	Name of channel axis used for measurement
Mode	Double digit specification for the operating mode; comprising <b>measuring mode</b> and <b>measuring system</b> .
	<b>Measuring mode</b> (units decade):
0	Mode 0: Cancel measuring task.
1	Mode 1: Up to 4 different trigger events can be activated <b>simultaneously</b> .
2	Mode 2: Up to 4 trigger events can be activated <b>consecutively</b> .
3	Mode 3: Up to 4 trigger events can be activated <b>consecutively</b> , but with no monitoring of trigger event 1 on START (alarms 21700/21703 are suppressed).
	<b>Note:</b> For MEAC, mode 3 is not possible.
	<b>Measuring system</b> (tens decade):
0 (or no data)	active measuring system
1	Measuring system 1
2	Measuring system 2
3	Both measuring systems
TE	<b>Trigger event:</b>
1	rising edge, probe 1
-1	falling edge, probe 1
2	rising edge, probe 2
-2	falling edge, probe 2
Measurement memory	Number of FIFO (circulating storage)

**Example of measuring with delete distance-to-go in mode 1**

(evaluation in chronological sequence)

**a) with 1 measuring system**

Program code	Comments
...	
N100 MEASA[X]=(1,1,-1) G01 X100 F100	; Measuring in mode 1 with active measuring 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 successful.
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:	

**Example, measuring with delete distance-to-go in mode 1**

**b) with 2 measuring systems**

Program code	Comments
...	
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 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 of measuring with delete distance-to-go in mode 2**

(evaluation in programmed sequence)

<b>Program code</b>	<b>Comments</b>
...	
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 following sequence rising edge from ;probe 1, falling edge probe 1, rising edge from probe 2, falling edge probe 2 on travel path to X=100.
N110 STOPRE	; Preprocessing stop
N120 IF \$AC_MEA[1]==FALSE gotof	; Check that the measurement with probe 1 is successful.
PROBE2	;
N130 R10=\$AA_MM1[X]	; Save measured value acquired on first programmed trigger event (rising edge, probe 1).
N140 R11=\$AA_MM2[X]	; Save measured value acquired on second programmed trigger event (rising edge, 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 on third programmed trigger event (rising edge, probe 2).
N180 R13=\$AA_MM4[X]	; Save measured value acquired on fourth programmed trigger event (rising edge, probe 2).
N190 END:	

**Example of continuous measuring in mode 1**

(evaluate in the sequence in time)

**a) Measurement of up to 100 measured values**

Program code	Comments
...	
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	

**Example, continuous measuring in mode 1**

(evaluate in the sequence in time)

**b) Measuring with deletion of distance-to-go after 10 measured values**

Program code	Comments
...	
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
...	

## Description

The measurements can be programmed in the parts program or from a synchronized action (see "Motion-synchronous action" section). 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 guaranteed only at feed rates with which no more than one trigger event of the same type and no more than four trigger events occur in each position controller cycle.

In the case of continuous measurement with MEAC, the ratio between the interpolation cycle and position control cycle must not exceed 8 : 1.

---

## Trigger events

A trigger event comprises the number of the probe and the trigger criterion (rising or falling edge) of the measuring signal.

Up to four trigger events of the addressed probe can be processed for each measurement, i.e., up to two probes with two measuring signal edges each. The processing sequence and the maximum number of trigger events depend on the selected mode.

---

### Note

The same trigger event is only permitted to be programmed once in a measuring job (only applies to mode 1)!

---

## Operating mode

The first digit in the mode setting selects the desired measuring system. If only one measuring system is installed, but a second programmed, the installed system is automatically selected.

With the second digit, i.e., the **measurement mode**, the measuring process is adapted to the capabilities of the connected control system:

- **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, however no monitoring of trigger event 1 at START.

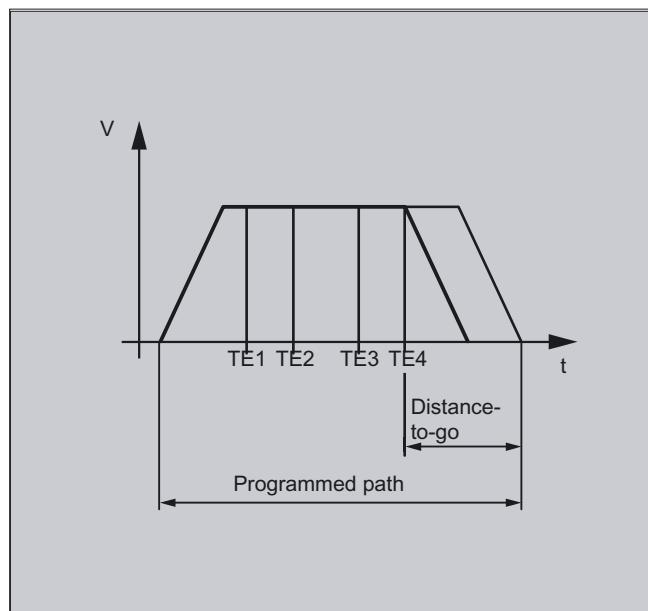
**Note**

No more than two trigger events can be programmed if two measuring systems are in use.

### Measurement with and without delete distance-to-go, MEASA, MEAWA

When command MEASA is programmed, the distance-to-go is not deleted until all required measured values have been recorded.

The MEAWA function is employed in the case of special measuring tasks where a programmed position must always be approached.



- MEASA cannot be programmed in synchronized actions. As an alternative, MEAWA plus the deletion of distance-to-go can be programmed as a synchronized action.
- If the measuring job with MEAWA is started from the synchronized actions, the measured values will only be available in machine coordinates.

## Measurement results for MEASA, MEAWA

The results of measurements are available under the following system variables:

- In 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 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 ("List of Instructions" section) at the appropriate position in the program. False values will otherwise be read in.

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.

### Example:

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 for measuring system 1 for trigger event 1
\$AA_MM2[axis]	or	\$AA_MW2[axis]	Measured value for measuring system 2 for trigger event 1
\$AA_MM3[axis]	or	\$AA_MW3[axis]	Measured value for measuring system 1 for trigger event 2
\$AA_MM4[axis]	or	\$AA_MW4[axis]	Measured value for measuring system 2 for trigger event 2

## Probe status can be read via \$A\_PROBE[n]

n=probe  
1==Probe deflected  
0==Probe not deflected

## Measuring job status for MEASA, MEAWA

If the probe switching state needs to be evaluated in the program, then the measuring job status can be interrogated via \$AC\_MEAS[n], with n = number of probe. Once all the trigger events of probe "n" that are programmed in a block have occurred, this variable switches to the "1" stage. Its value is otherwise 0.

---

### Note

If measuring is started from synchronized actions, \$AC\_MEAS is not updated. In this case, new PLC status signals DB(31-48) DBB62 bit 3 or the equivalent variable \$AA\_MEAACT["Axis"] must be interrogated.

Meaning:

\$AA\_MEAACT==1: Measurement active  
\$AA\_MEAACT==0: Measurement not active

---

### References:

/FB2/ Function Manual, Extended Functions; Measurements (M5).

## Continuous measurement MEAC

The measured values for MEAC are available in the machine coordinate system and stored in the programmed FIFO[n] memory (circulating memory). 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 circulating memory in which measured values are written to \$AC\_FIFO variables according to the circulation principle, see section "Motion Synchronous Actions" ..

### Note

FIFO contents can be read only once from the circulating storage. If these measured data are to be used multiply, they must be buffered in 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.

## Recognized programming errors

The following programming errors are detected and indicated appropriately:

- If MEASA/MEAWA is 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 GEO axis

Example:

N01 MEASA[X]=(1,1) MESA[Y]=(1,1) G01 X50 Y50 Z50 F100 ;GEO axis  
X/Y/Z

- Inconsistent measuring job with GEO axes

Example:

N01 MEASA[X]=(1,1) MEASA[Y]=(1,1) MEASA[Z]=(1,1,2)  
G01 X50 Y50 Z50 F100

## **4.9      Special functions for OEM users (OEMIPO1, OEMIPO2, G810 to G829)**

### **Function**

#### **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. The address identifiers are settable. OEM addresses can be programmed in any block.

### **Parameters**

#### **Reserved G groups**

Group 1 with OEMIPO1, OEMIPO2

The OEM user can define two additional names of G functions OEMIPO1, OEMIPO2 . Their functionality is incorporated by means of compile cycles and is reserved for the OEM user.

- Group 31 with **G810 to G819**
- Group 32 with **G820 to G829**

Two G groups with ten OEM G functions each are reserved for OEM users. **These allow the functions incorporated by an OEM user to be accessed for external applications.**

#### **Functions and subroutines**

OEM users can also set up predefined functions and subroutines with parameter transfer.

## 4.10 Feed reduction with corner deceleration (FENDNORM, G62, G621)

### Function

With automatic corner deceleration the feed rate is reduced according to a bell 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

### Significance

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:

$F * (\text{override for feed rate reduction}) * \text{feed rate override}$

---

#### 4.11 Programmed end-of-motion criterion (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)

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.

## 4.11 Programmed end-of-motion criterion (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)

### Function

Similar to the block change criterion for path interpolation (G601, G602 and G603) it is also possible to program the movement end criterion for singleaxis 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 singleaxis movements are completed. The same applies for PLC via FC15/16/18.

### Syntax

```
FINEA[<axis>]  
COARSEA[<axis>]  
IPOENDA[<axis>]  
IPOBRKA(<axis>, [, [<instant in time>]]) ; multiple data possible  
ADISPOSA(<axis>, [<mode>] [, [<>window size>]]) ; multiple data possible
```

### Significance

FINEA	Motion end when "Exact stop FINE" reached
COARSEA	Motion end when "Exact stop COARSE" reached
IPOENDA	Motion end when "Interpolator stop" reached
IPOBRKA	A block change is possible in the braking ramp.
ADISPOSA	Size of the tolerance window for the end of motion criteria
<axis>	Channel axis name (X, Y, ....)
<instant in time>	Instant in time of the block change, referred to the braking ramp as a %

## 4.11 Programmed end-of-motion criterion (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)

<code>&lt;mode&gt;</code>	<b>Mode</b>
	Type: INT
	Range of values:
	0 Tolerance window not active
	1 Tolerance window with respect to set position
	2 Tolerance window with respect to actual position
<code>&lt;window size&gt;</code>	<b>Size of the tolerance window</b>
	This value is entered synchronized with the main run in the setting data SD43610 \$SA_ADISPOSA_VALUE.
	Type: REAL

**Example: End-of-motion when reaching the interpolator stop**

Program code	Comments
...	
N110 G01 POS[X]=100 FA[X]=1000 ACC[X]=90 IPOENDA[X]	Traversing to position X100 when input 1 is active, with a path velocity of 1000 rpm, an acceleration value of 90% and end-of-motion on reaching the interpolator stop.
...	
N120 EVERY \$A_IN[1] DO POS[X]=50 FA[X]=2000 ACC[X]=140 IPOENDA[X]	; Traversing to position X50 when input 1 is active, with a path velocity of 2000 rpm, an acceleration value of 140% and end-of-motion on reaching the interpolator stop.
...	

## *Special Motion Commands*

### **4.11 Programmed end-of-motion criterion (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)**

#### **Example: Block change criteria, "braking ramp" in the part program**

<b>Program code</b>	<b>Comments</b>
	; Default setting is effective
N40 POS[X]=100	; The block is changed if the X axis has reached position 100 and exact stop fine is reached.
N20 IPOBRKA(X,100)	; Block change criteria, activate braking ramp.
N30 POS[X]=200	; The block is changed as soon as the X axis starts to brake.
N40 POS[X]=250	; The X axis does not brake at position 200, but moves further to position 250, as soon as the X axis starts to brake, the block is changed.
N50 POS[X]=0	; The X axis brakes and returns to position 0 - the block changed at position 0 and exact stop fine.
N60 X10 F100	
N70 M30	
...	

#### **Example for the braking ramp in synchronous actions block change condition**

<b>Program code</b>	<b>Comments</b>
	; In the technology cycle:
FINEA	; End of motion criteria, exact stop fine.
POS[X]=100	; Technology cycle block change if the X axis has reached position 100 and exact stop fine is reached.
IPOBRKA(X,100)	; Block change criteria, activate braking ramp.
POS[X]=100	; POS[X]=100; technology cycle block change as soon as the X axis starts to brake.
POS[X]=250	; The X axis does not brake at position 200, but moves further to position 250, as soon as the X axis starts to brake, the block is changed in the technology cycle.
POS[X]=250	; The X axis brakes and returns to position 0 - the block change is made at position 0 and exact stop fine.
M17	

## Description

### \$AA\_MOTEND system variable

The set end-of-motion criterion can be scanned by system variable \$AA\_MOTEND[axis]

\$AA_MOTEND[Axis] = 1	End-of-motion with "Exact stop fine"
\$AA_MOTEND[Axis] = 2	End-of-motion with "Exact stop coarse"
\$AA_MOTEND[Axis] = 3	End-of-motion with "IPO-Stop"
\$AA_MOTEND[Axis] = 4	Block change criterion braking ramp of axis motion
\$AA_MOTEND[Axis] = 5	Block change in braking ramp with tolerance window relative to "set position"
\$AA_MOTEND[Axis] = 6	Block change in braking ramp with tolerance window relative to "actual position"

---

### Note

The last programmed value is retained after RESET.

### References:

/FB1/ Function Manual Basic Functions; Feedrates (V1).

---

### Block change criterion in braking ramp

The percentage value is entered in SD 43600: IPOBRAKE\_BLOCK\_EXCHANGE. If no value is specified, the current value of this setting data is effective. The range is adjustable from 0% to 100%.

### Additional tolerance window for IPOBRKA

An additional block change criterion tolerance window can be selected as well as the existing block change criterion in the braking ramp. Release will only occur when the axis

- as before has reached the specified % value of its braking ramp **and**
- its current actual or set position is no further than a tolerance from the end of the axis in the block.

For more information on the block change criterion of the positioning axes, please refer to:

### References:

/FB2/ Function Manual, Extended Functions; Positioning Axes (P2).

/PG/ Fundamentals Programming Guide; "Feed Control and Spindle Motion".

## 4.12 Programmable servo parameter set (SCPARA)

### Function

The parameter set (comprising MDs) in the part program and in synchronized actions can be programmed using SCPARA (up until now, only via the PLC).

#### DB3n DBB9 bit3

To ensure no conflicts occur between PLC and NCK, an additional bit is defined on the PLC → NCK interface:

DB3n DBB9 bit3 "Parameter set selection by SCPARA disabled".

If the parameter set input for SCPARA is inhibited, then an error message is not output if this is even programmed.

### Syntax

SCPARA [<axis>]=<value>

### Significance

SCPARA	Define parameter block
<axis>	Channel axis name (X, Y, ...)
<value>	Desired parameter block (1≤ value ≤6)

---

### Note

The actual parameter set can be interrogated using the system variables \$AA\_SCPAR [<axis>].

For G33, G331 or G332, the most suitable parameter set is selected by the control.

The PLC user program must be expanded if the **servo parameter set** is to be **changed** in a part program, synchronized action or the PLC.

---

### References:

/FB1/ Function Manual Basic Functions; Feedrates (V1),  
"Feedrate Impact" section.

### Example

Program code	Comments
...	
N110 SCPARA[X]= 3	; The 3rd parameter set is selected for axis X.
...	

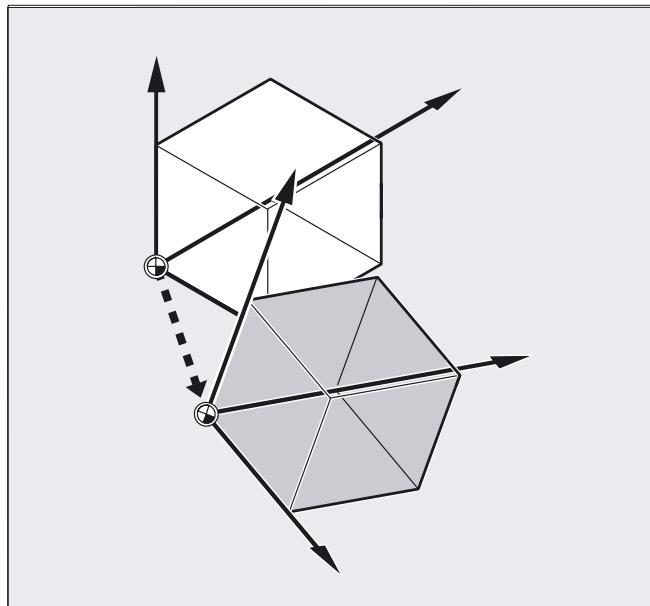
# 5

## Coordinate transformation (FRAMES)

### 5.1 Coordinate transformation via frame variables

#### Function

In addition to the programming options already described in the Programming Guide "Fundamentals", you can also define coordinate systems with predefined frame variables.



The following coordinate systems are defined:

**MCS:** Machine coordinate system

**BCS:** Basic coordinate system

**BZS:** Basic origin system

**SZS:** Settable zero system

**WCS:** Workpiece coordinate system

## *5.1 Coordinate transformation via frame variables*

### **What is a predefined frame variable?**

Predefined frame variables are keywords whose use and effect are already defined in the control language and that can be processed in the NC program.

Possible frame variable:

- Basic frame (basic offset)
- Settable frames
- Programmable frame

### **Value assignments and reading the actual values**

#### **Frame variable/frame relationship**

A coordinate transformation can be activated by assigning the value of a frame to a frame variable.

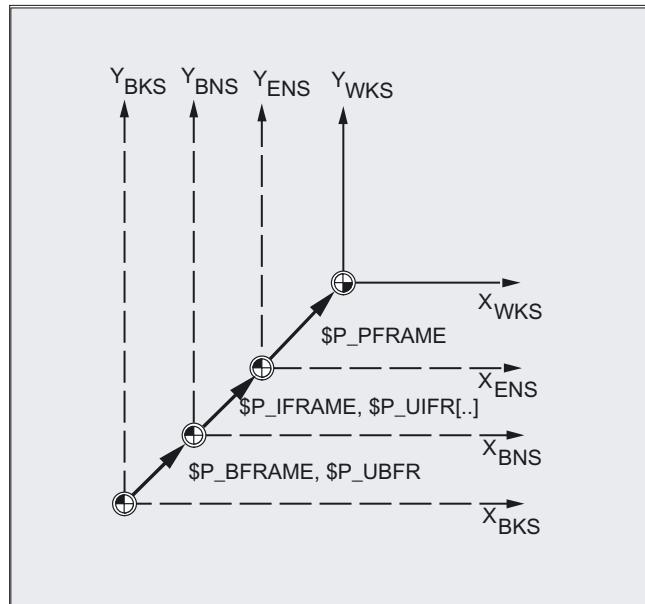
**Example:** \$P\_PFRAME=CTRANS (X, 10)

Frame variable:

\$P\_PFRAME means: current programmable frame.

Frame:

CTRANS (X, 10) means: programmable zero offset of X axis by 10 mm.



### Reading the actual values

The current actual values of the coordinate system can be read out via predefined variables in the parts program:

- \$AA\_IM[axis]: Read actual value in MCS
- \$AA\_IB[axis]: Read actual value in BCS
- \$AA\_IBN[axis]: Read actual value in BOS
- \$AA\_IEN[axis]: Read actual value in Szs
- \$AA\_IW[axis]: Read actual value in WCS

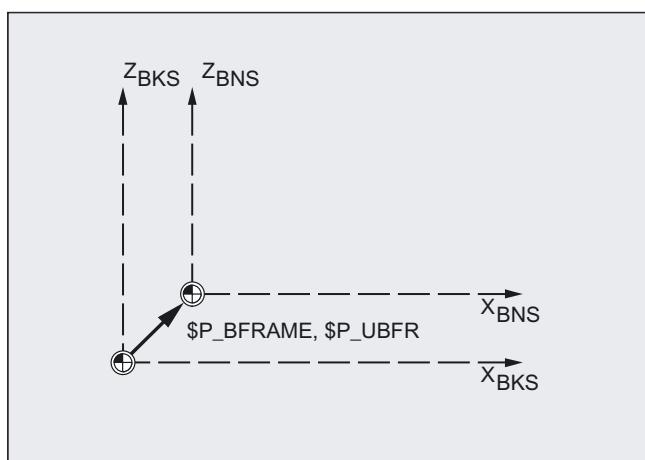
## 5.1.1 Predefined frame variable (\$P\_BFRAME, \$P\_IFRAME, \$P\_PFRAME, \$P\_ACTFRAME)

### \$P\_BFRAME

Current basic frame variable that establishes the reference between the basic coordinate system (BCS) and the basic origin system (BOS).

For the basic frame described via \$P\_UBFR to be immediately active in the program, either

- you have to program a G500, G54...G599, or
- you have to describe \$P\_BFRAME with \$P\_UBFR

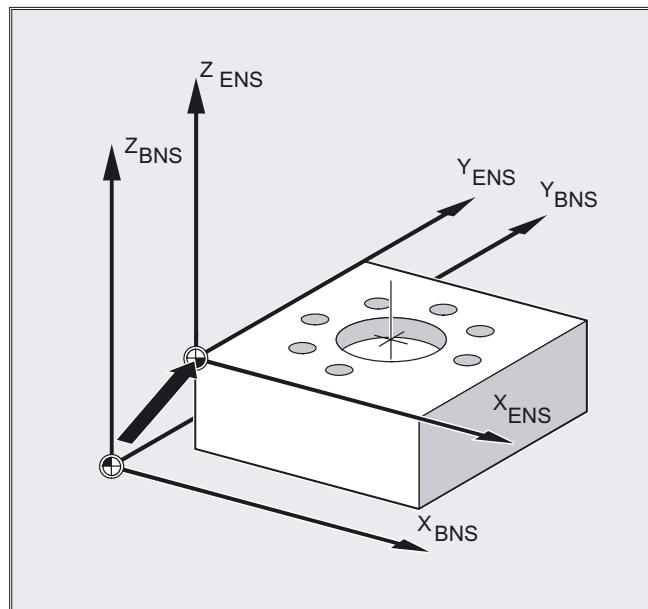


*5.1 Coordinate transformation via frame variables*

**\$P\_IFRAME**

Current, settable frame variable that establishes 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.

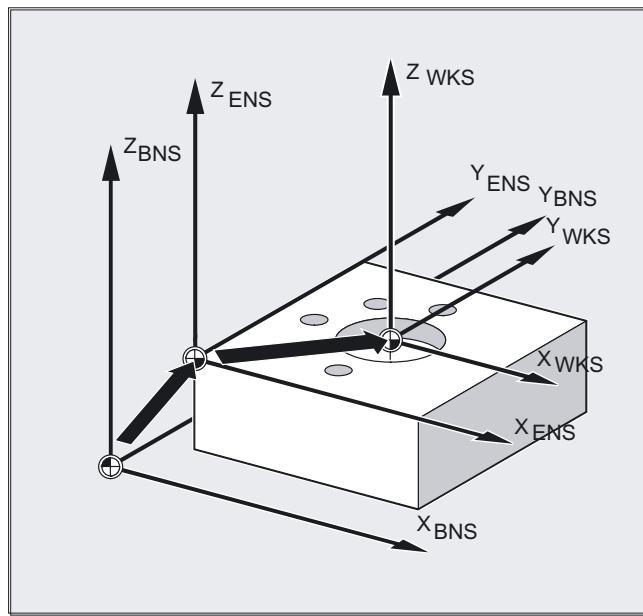


**\$P\_PFRAME**

Current, programmable frame variable that establishes 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



## *Coordinate transformation (FRAMES)*

### *5.1 Coordinate transformation via frame variables*

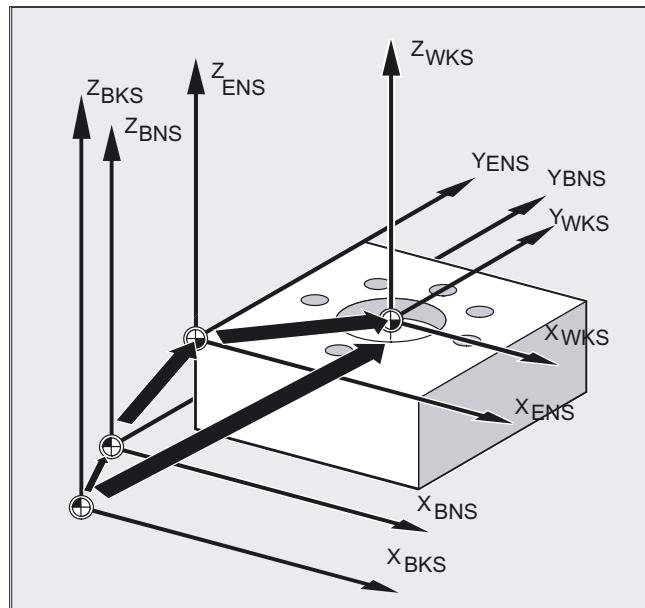
#### **\$P\_ACTFRAME**

Current, resulting complete frame that results from chaining

- the current basic frame variable **\$P\_BFRAME**,
- the currently settable frame variable **\$P\_IFRAME** with system frames and
- the currently programmable frame variable **\$P\_IFRAME** with system frames.

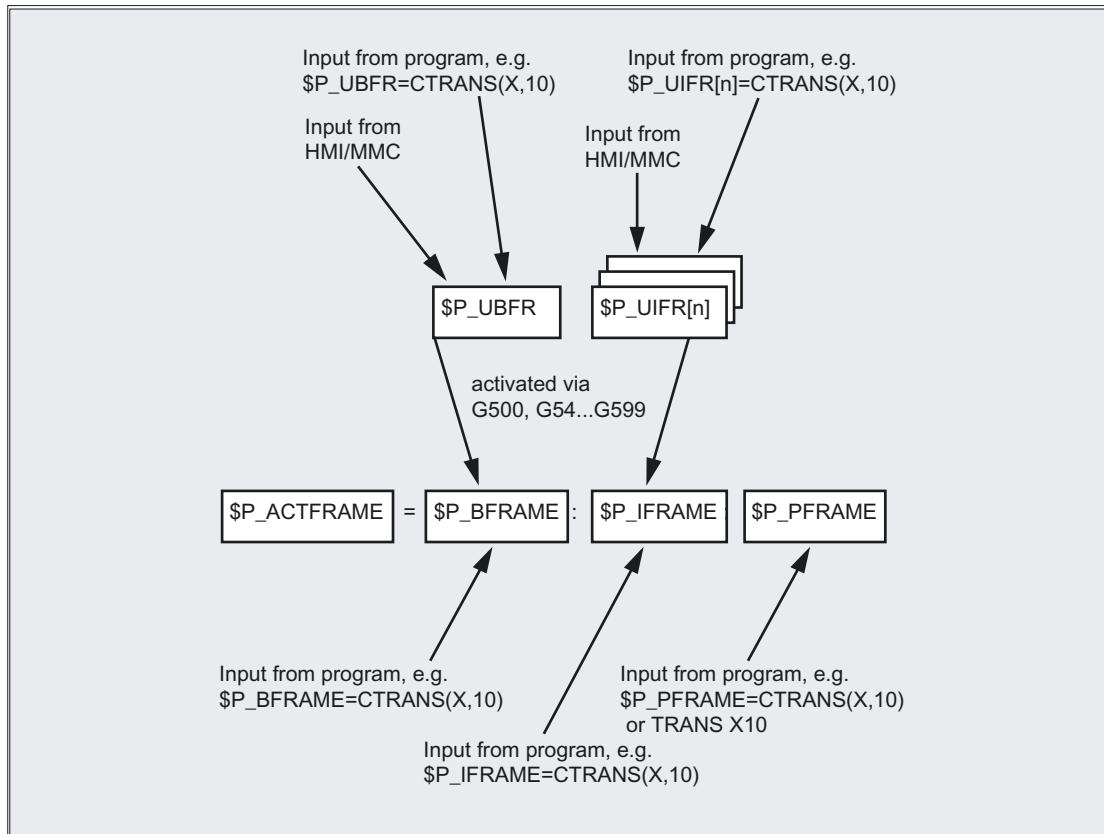
System frames, see Section "Frames that Act in the Channel"

**\$P\_ACTFRAME** describes the currently valid workpiece zero.



If **\$P\_IFRAME**, **\$P\_BFRAME** or **\$P\_PFRAME** are changed, **\$P\_ACTFRAME** is recalculated.

**\$P\_ACTFRAME** corresponds to **\$P\_BFRAME:\$P\_IFRAME:\$P\_PFRAME**



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

### Predefined settable frames \$P\_UBFR

The basic frame is programmed with \$P\_UBFR, but it is not simultaneously active in the parts program. The basic frame programmed with \$P\_UBFR is included in the calculation if

- Reset was activated and bits 0 and 14 are set in MD RESET\_MODE\_MASK and
- the statements G500,G54...G599 were executed.

---

## *5.1 Coordinate transformation via frame variables*

### **Predefined settable frames \$P\_UIFR[n]**

The predefined frame variable \$P\_UIFR[n] can be used to read or write the settable zero offsets G54 to G599 from the parts program.

These variables produce a one-dimensional array of type FRAME called \$P\_UIFR[n].

### **Assignment to G commands**

As standard, five settable frames \$P\_UIFR[0]...\$P\_UIFR[4] or five equivalent G commands – G500 and G54 to G57, can be saved using their address values.

\$P\_IFRAME=\$P\_UIFR[0] corresponds to G500

\$P\_IFRAME=\$P\_UIFR[1] corresponds to G54

\$P\_IFRAME=\$P\_UIFR[2] corresponds to G55

\$P\_IFRAME=\$P\_UIFR[3] corresponds to G56

\$P\_IFRAME=\$P\_UIFR[4] corresponds to G57

You can change the number of frames with machine data:

\$P\_IFRAME=\$P\_UIFR[5] corresponds to G505

... ... ...

\$P\_IFRAME=\$P\_UIFR[99] corresponds to G599

---

#### **Note**

This allows you to generate up to 100 coordinate systems, which can be called up globally in different programs, for example, as zero point for various fixtures.



#### **CAUTION**

Frame variables must be programmed in a separate NC block in the NC program.

**Exception:** programming of a settable frame with G54, G55, ...

## 5.2 Frame variables / assigning values to frames

### 5.2.1 Assigning direct values (axis value, angle, scale)

#### Function

You can directly assign values to frames or frame variables in the NC program.

#### Syntax

```
$P_PFRAME=CTRANS (X, axis value, Y, axis value, Z, axis value, ...)
$P_PFRAME=CROT (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)
```

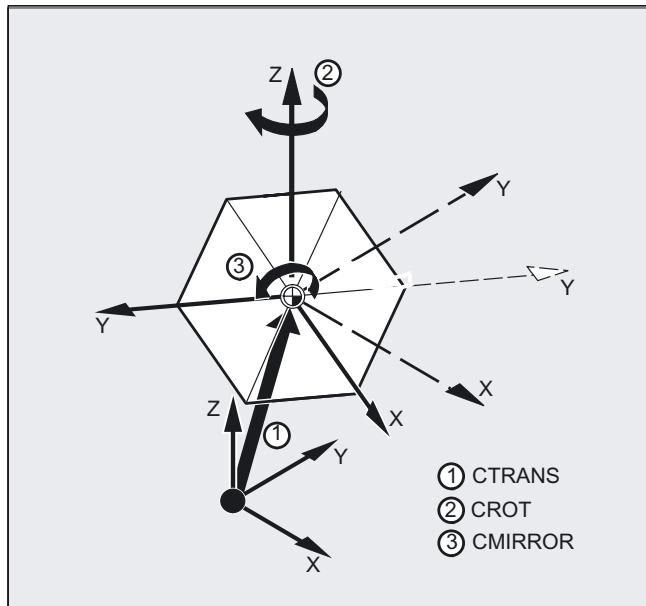
Programming \$P\_BFRAME is carried out analog to \$P\_PFRAME.

#### Significance

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
Axis value	Assigning the axis value of the offset
Angle	Assigning the angle of rotation around the specified axes
Scale	Changing the scale

### Example

Translation, rotation and mirroring are activated by value assignment to the current programmable frame.



```
N10 $P_PFRAME=CTRANS (X,10,Y,20,Z,5) :CROT (Z,45) :CMIRROR (Y)
```

### Frame-red components are pre-assigned other values

With CROT, pre-assign all three UIFR components with values

Program code	Comments
\$P_UIFR[5] = CROT(X, 0, Y, 0, Z, 0)	
N100 \$P_UIFR[5, y, rt]=0	
N100 \$P_UIFR[5, x, rt]=0	
N100 \$P_UIFR[5, z, rt]=0	

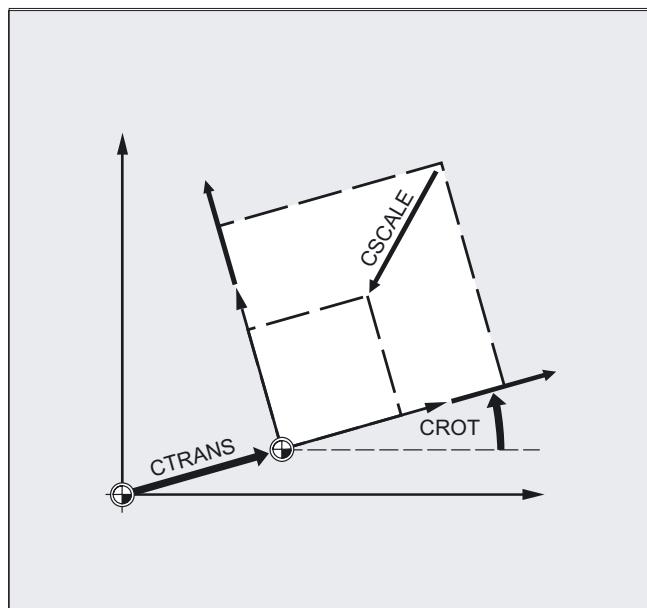
## Description

You can program several arithmetic rules in succession.

Example:

`$P_PFRAME=CTRANS (...):CROT (...):CSCALE...`

Please note that the commands must be connected by the colon chain operator: (...):(...). This causes the commands firstly to be linked and secondly to be executed additively in the programmed sequence.



---

### Note

The values programmed with the above commands are assigned to the frames and stored.

The values are not activated until they are assigned to the frame of an active frame variable `$P_BFRAME` or `$P_PFRAME`.

---

### 5.2.2     Reading and changing frame components (TR, FI, RT, SC, MI)

#### Function

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_UIFRNUM to the variable R10.
R12=\$P_UIFR[25,Z,TR]	Assign the offset value TR in Z from the data record 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.

#### Significance

\$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**

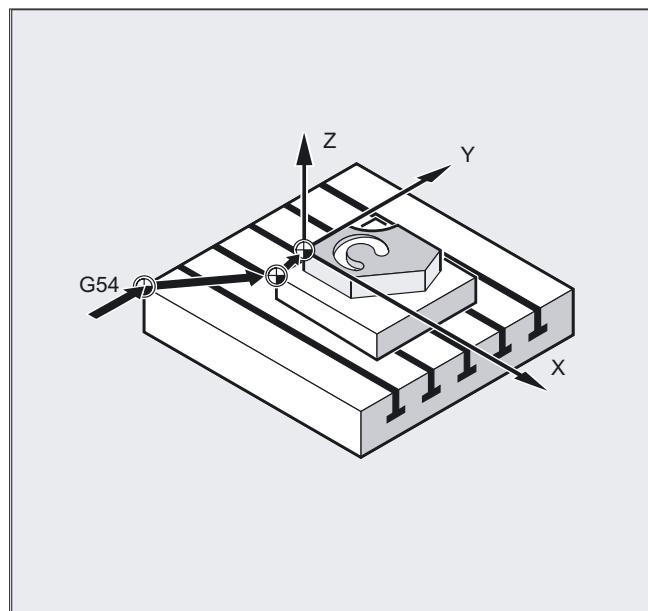
The axis name and the frame component of the value you want to access or modify are written in square brackets, e.g., [X, RT] or [Z, MI].

### 5.2.3 Linking complete frames

#### Function

A complete frame can be assigned to another frame or frames can be chained to each other in the NC program.

Frame chaining is 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.

#### Syntax

##### Assigning frames

```
| DEF FRAME SETTING1  
| SETTING1=CTRANS(X,10)  
| $P_PFRAME=SETTING1  
|  
| DEF FRAME SETTING4  
| SETTING4=$P_PFRAME  
| $P_PFRAME=SETTING4
```

Assign the values of the user frame SETTING1 to the current programmable frame.

The current programmable frame is stored temporarily and can be recalled.

**Frame chains**

The frames are chained in the programmed sequence. The frame components (translations, rotations, etc.) are executed additively in succession.

`$P_IFRAME=$P_UIFR[15]:$P_UIFR[16]`

`$P_UIFR[15]` contains, for example, data for zero offsets. The data of `$P_UIFR[16]`, e.g., data for rotations, are subsequently processed additively.

`$P_UIFR[3]=$P_UIFR[4]:$P_UIFR[5]`

The settable frame 3 is created by chaining the settable frames 4 and 5.

**Note**

The frames must be linked with each other using the concatenation colon : .

## 5.2.4 Defining new frames (DEF FRAME)

**Function**

In addition to the predefined settable frames described above, you also have the option of creating new frames. This is achieved by creating variables of type FRAME to which you can assign a name of your choice.

You can use the functions CTRANS, CROT, CSCALE and CMIRROR to assign values to your frames in the NC program.

**Syntax**

```
DEF FRAME PALETTE1
PALETTE1=CTRANS (...) :CROT (...) ...
```

**Significance**

DEF FRAME	Creating new frames.
PALETTE1	Name of the new frame
=CTRANS (...) : CROT (...) ...	Assign values to the possible functions

## 5.3 Coarse and fine offsets (CFINE, CTRANS)

### Function

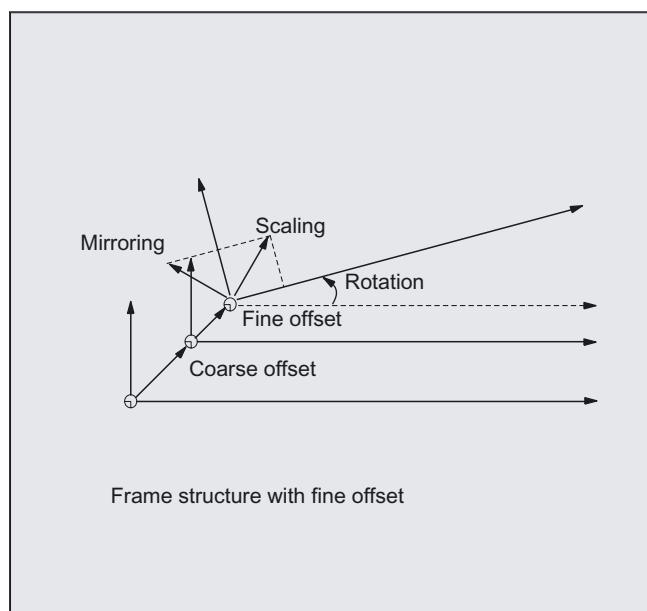
#### Fine offset

A fine offset of the basic frames and of all other settable frames can be programmed with command CFINE (X, ..., Y, ...).

A fine offset can only be made if MD18600 \$MN\_MM\_FRAME\_FINE\_TRANS=1.

#### Rough offset

The coarse offset is defined with CTRANS (....).



Coarse and fine offset add up to the total offset.

### Syntax

```
$P_UBFR=CTRANS(x, 10) : CFINE(x, 0.1) ;Chaining of offset,  
: CROT(x, 45) ;fine offset and rotation  
$P_UIFR[1]=CFINE(x, 0.5 y, 1.0, z, 0.I) ;The complete frame will be  
;overwritten with CFINE  
;including the coarse offset
```

Access to the individual components of the fine offset is achieved through component specification FI (Translation Fine).

```
DEF REAL FINEX ;Definition of the FINEX variable
FINEX=$P_UIFR[$P_UIFNUM, x, FI] ;Fetching the fine offset
;using the FINEX variable
FINEX=$P_UIFR[3, x, FI]$P ;Fetching the fine offset
;of the X axis in the 3rd frame
;using the FINEX variable
```

## Significance

CFINE(x, value, y, value, z, value)	Fine offset for multiple axes. Additive offset (translation).
CTRANS(x, value, y, value, z, value)	Coarse offset for multiple axes. Absolute offset (translation).
x y z	Zero shift of the axes (max. 8)
Value	Translation part

## Machine manufacturer

The fine offset can be configured in the following versions using MD18600 \$MN\_MM\_FRAME\_FINE\_TRANS:

0:

The fine offset cannot be entered or programmed. G58 and G59 are not possible.

1:

Fine offset for settable frames, basic frames, programmable frames, G58 and G59 can be entered/programmed.

## Description

A fine offset changed with the HMI operation does not apply until after activation of the corresponding frame, i.e. activation via G500, G54...G599. Once activated, a fine offset of a frame remains active the whole time the frame is active.

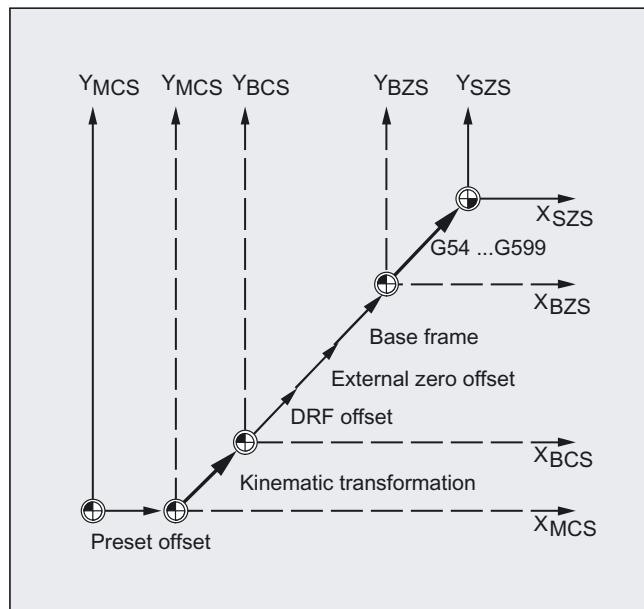
The programmable frame has no fine offset. If the programmable frame is assigned a frame with fine offset, then the total offset is established by adding the coarse and the fine offset. When reading the programmable frame the fine offset is always zero.

## 5.4 External zero offset

### Function

This is another way of moving the zero point between the basic and workpiece coordinate system.

Only linear translations can be programmed with the external zero offset.



### Programming

The \$AA\_ETRANS offset values are programmed by assigning the axis-specific system variables.

#### Assigning offset value

`$AA_ETRANS [axis] = RI`

RI is the arithmetic variable of type REAL that contains the new value.

The external offset is generally set by the PLC and not specified in the parts program.

---

#### Note

The value entered in the parts program only becomes active when the corresponding signal is enabled at the VDI interface (NCU-PLC interface).

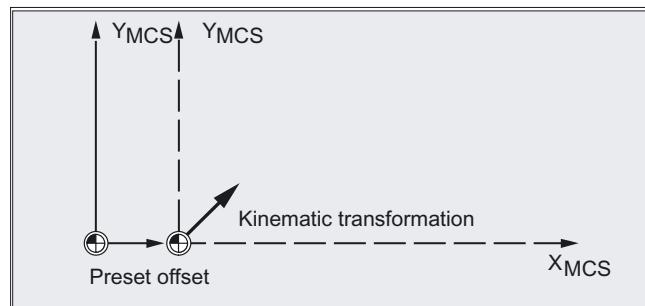
## 5.5 Preset offset (PRESETON)

### Function

In special applications, it can be necessary to assign a new programmed actual value to one or more axes at the current position (stationary).



The reference point becomes invalid with the function PRESETON. You should therefore only use this function for axes which do not require a reference point. If the original system is to be restored, the reference point must be approached with G74 – see the "File and Program Management" section.



### Syntax

`PRESETON(axis, value)`

### Significance

PRESETON	Preset actual value memory
Axis	Machine axis parameter
Value	New actual value to apply to the specified axis

---

### Note

Preset mode with synchronized actions should only be implemented with the keyword "WHEN" or "EVERY".

---

### **Example**

The actual values are assigned to the machine coordinate system – the values refer to the machine axes.

```
N10 G0 A760  
N20 PRESETON(A1, 60)
```

Axis A travels to position 760. At position 760, machine axis A1 is assigned the new actual value 60. From this point, positioning is performed in the new actual value system.

## **5.6 Frame calculation from three measuring points in space (MEAFRAME)**

### **Function**

MEAFRAME is an extension of the 840D language used for supporting measuring cycles.

The function MEAFRAME calculates the frame from three ideal and the corresponding measured points.

When a workpiece is positioned for machining, its position relative to the Cartesian machine coordinate system is generally both shifted and rotated referring 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_POINT, MEAS_POINT, FIT_QUALITY)
```

### **Significance**

MEAFRAME	Frame calculation of three measured points in space
IDEAL_POINT	Array of real data containing the three coordinates of the ideal points
MEAS_POINT	Array of real data containing the three coordinates of the measured points

FIT_QUALITY	REAL variable,	returning the following information:
	-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.
	Positive value:	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 as from SW 6.3.

Example: chaining of frames "concatenation with ADDFRAME".

Further information for the parameters for ADDFRAME (FRAME, STRING) see /FB1/ Function Manual Basic Functions; Axes, Coordinate Systems, Frames (K2), "FRAME Chaining".

## *Coordinate transformation (FRAMES)*

### *5.6 Frame calculation from three measuring points in space (MEAFRAME)*

#### **Example**

<b>Program code</b>	<b>Comments</b>
	; Part program 1
DEF FRAME CORR_FRAME	

#### **Setting measuring points**

<b>Programming</b>	<b>Comments</b>
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] = SET (10.1,0.2,-0.2, -0.2,10.2,0.1, -0.2,0.2,9.8)	; for test
DEF REAL FIT_QUALITY = 0	
DEF REAL ROT_FRAME_LIMIT = 5	; Permits max. 5 degrees rotation of the part position
DEF REAL FIT_QUALITY_LIMIT = 3	; Permits max. 3 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]	

<b>Program code</b>	<b>Comments</b>
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	

## 5.6 Frame calculation from three measuring points in space (MEAFRAME)

Program code	Comments
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	; Activating 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]	
N410 SHOW_MCS_POS1[1]=\$AA_IM[X]	
N430 SHOW_MCS_POS1[2]=\$AA_IM[Z]	
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 of concatenating frames

### Chaining of MEAFRAME for offsets

The MEAFRAME( ) function provides an offset frame. If this offset frame is concatenated with a set frame \$P\_UIFR[1] that was active when the function was called, e.g., G54, one receives a settable frame for further conversions for the procedure or machining.

### Concatenation 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
- with reference to the radius (DIAMOF)

in the basic system of the controller.

## 5.7 NCU global frames

### Function

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 achieved using wait markers (WAITMC) for example.

#### Machine manufacturer

The number of global basic frames is configured using machine data, see /FB1/ 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 \$MN\_MM\_NUM\_GLOBAL\_USER\_FRAMES.

Channel axis identifiers and machine axis identifiers can be used as axis identifiers in frame program commands. Programming of geometry identifiers is rejected with an alarm.

### **5.7.1 Channel-specific frames (\$P\_CHBFR, \$P\_UBFR)**

#### **Function**

Settable frames or basic frames can be read and written by an operator action or from the PLC:

- via the parts program, or
- via the operator panel interface.

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 MD 28081 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 instruction. 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 field device 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 instruction. 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: instruction not permissible".

## 5.7.2 Frames active in the channel

### Function

Frames active in the channel are entered from the parts program via the associated system variables of these frames. System frames also belong here. The current system frame can be read and written via these system variables in the parts program.

### Frames active in the channel

#### Overview

<b>Current system frames</b>	For:
\$P_PARTFRAME	TCARR and PAROT
\$P_SETFRAME	PRESET and scratching
\$P_EXTFRAME	External zero offset
<b>\$P_NCBFRAME[n]</b>	Current NCU-global basic frames
<b>\$P_CHBFRAME[n]</b>	Current channel basic frames
<b>\$P_BFRAME</b>	Current first basic frame in the channel
<b>\$P_ACTBFRAME</b>	Complete basic frame
<b>\$P_CHBFRMASK and \$P_NCBFRMASK</b>	Complete basic frame
<b>\$P_IFRAME</b>	Current settable frame
<b>Current system frames</b>	For:
\$P_TOOLFRAME	TOROT and TOFRAME
\$P_WPFRAME	Workpiece reference points
\$P_TRAFRAME	Transformations
<b>\$P_PFRAME</b>	Current programmable frame
<b>Current system frame</b>	For:
\$P_CYCFRAME	cycles
<b>P_ACTFRAME</b>	Current total frame
<b>FRAME chaining</b>	The current frame consists of the total 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.

## 5.7 NCU global frames

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 in the channel as a result of the write operation. 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 field device of 0, which is valid in the channel, in the parts program. The written basic frame is immediately included in the calculation.

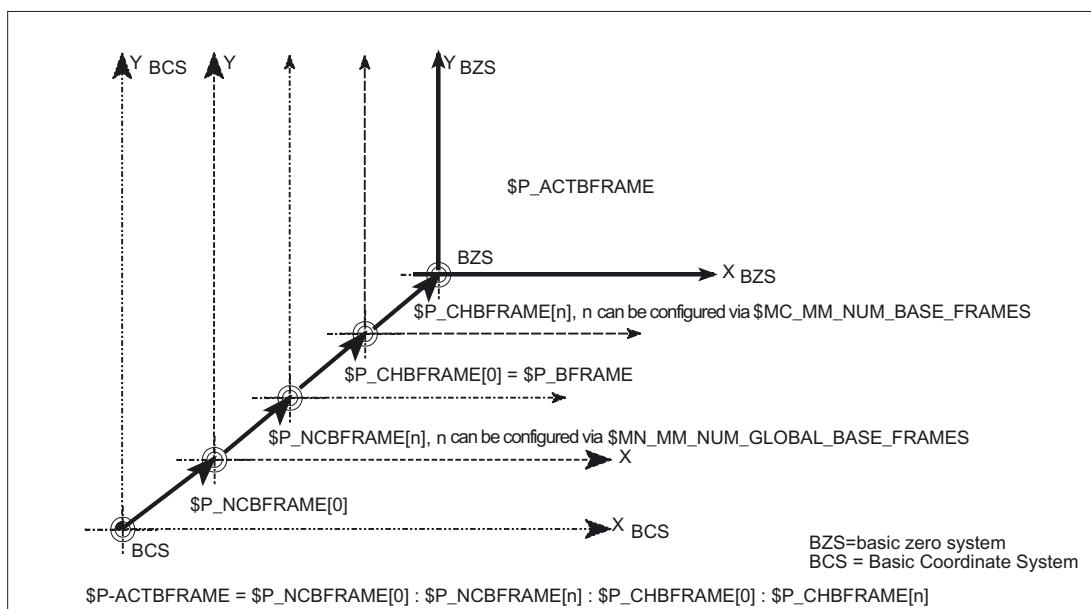
\$P\_UBFR is identical to \$P\_CHBFR[0]. The system variable always has a valid default value. If there is no channel-specific basic frame, an alarm is issued at read/write: "Frame: instruction not permissible".

### \$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 system variables `$P_CHBFRMASK` and `$P_NCBFRMASK` can be used to select, which basic frames to include in the calculation of the "complete" basic frame. The variables can only be programmed in the program and read via the operator panel interface. The value of the variable is interpreted as bit mask and determines which basic frame field element of `$P_ACTFRAME` is included in the calculation.

`$P_CHBFRMASK` can be used to define which channel-specific basic frames are included, and `$P_NCBFRMASK` can be used to define which NCU-global basic frames are included in the calculation.

When the variables are programmed, the total basic frame and the total frame are calculated again. After a reset and in the default setting, the value of

`$P_CHBFRMASK = $MC_CHBFRAME_RESET_MASK` and

`$P_NCBFRMASK = $MC_CHBFRAME_RESET_MASK.`

e.g.,

`$P_NCBFRMASK = 'H81' ;$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 parts 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 programmed 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 programmed FRAME.

Current, programmable frame variable that establishes the reference between the settable

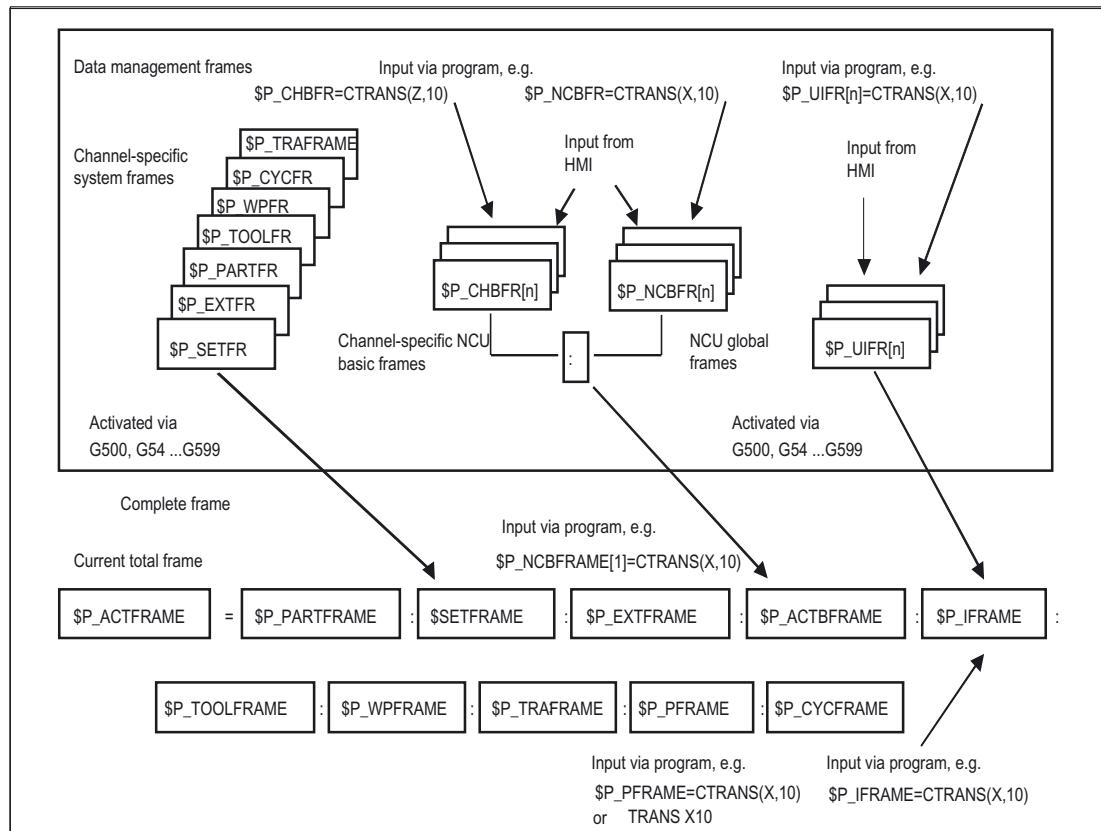
- zero system (Szs) and the
- workpiece coordinate system (Wcs).

### 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.

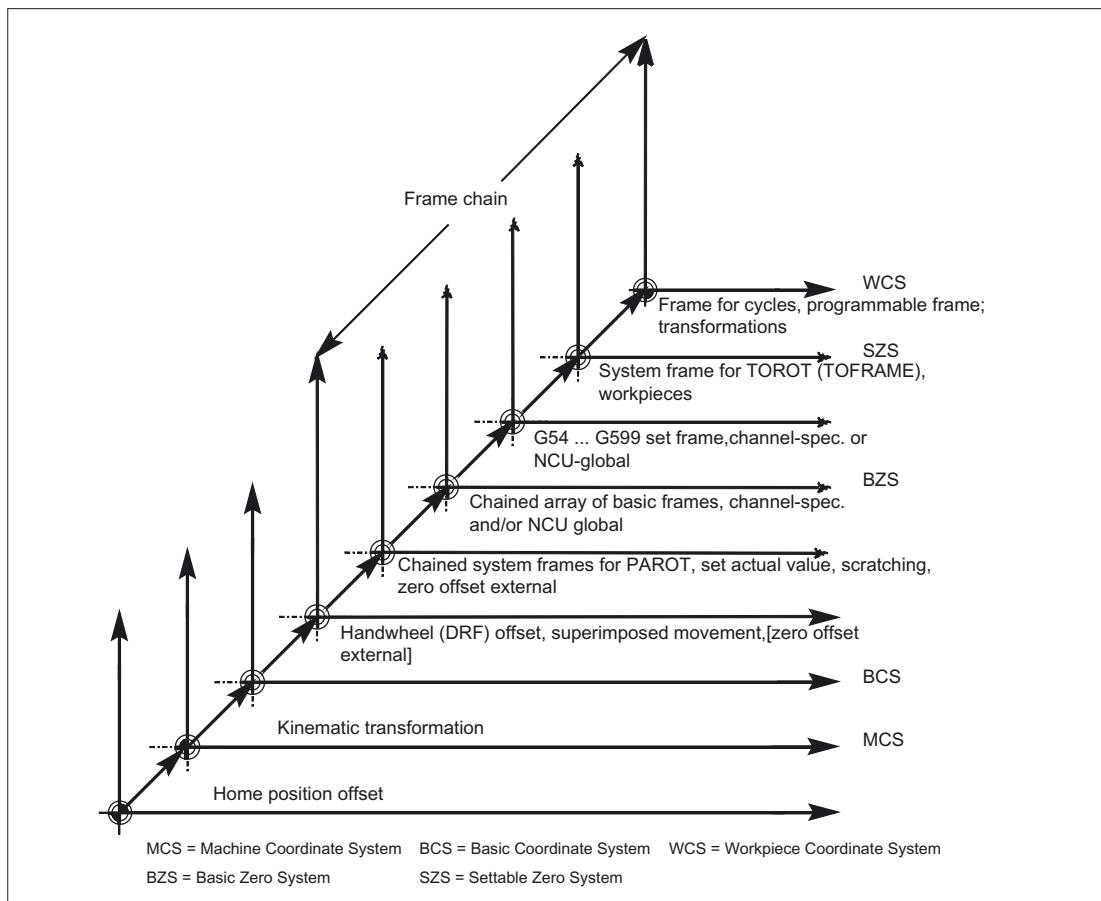
\$P\_ACTFRAME corresponds to

```
$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 consists of the total basic frame, the settable frame, the system frame, and the programmable frame according to the current total frame mentioned above.





# 6

## Transformations

### 6.1 General programming of transformation types

#### General function

You can choose to program transformation types with suitable parameters in order to adapt the control 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 control 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)

---

## *6.1 General programming of transformation types*

### **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/ Description of Functions Extended Functions; Kinematic Transformation (M1)

---

## *6.1 General programming of transformation types*

### **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.

#### **6.1.1 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.

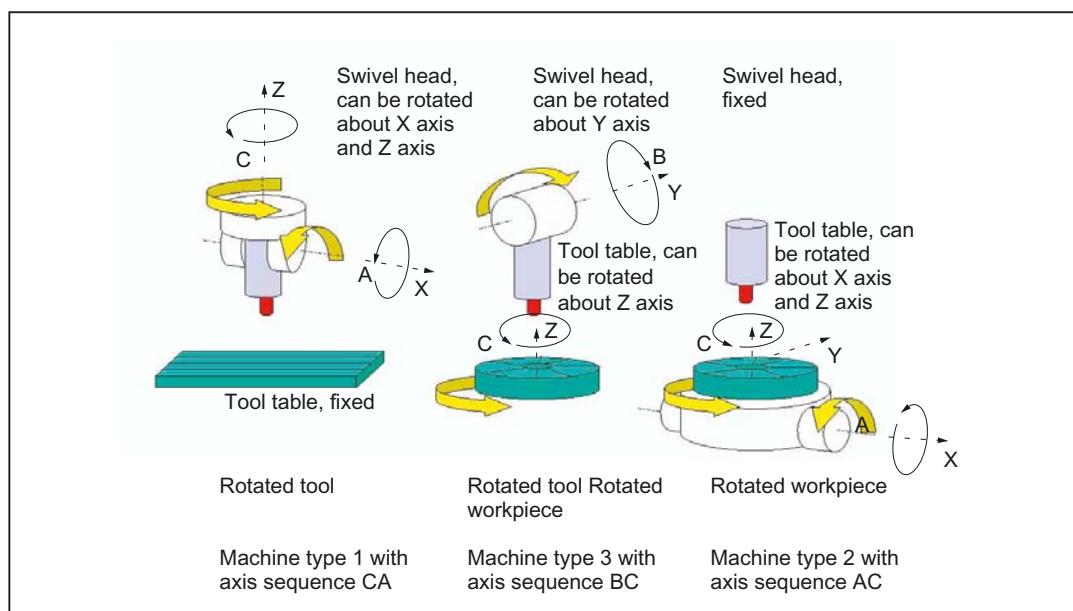
Changes in the position of the rotary axes involved in the orientation transformation will induce compensating movements on the remaining machine axes. The position of the tool tip remains unchanged.

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 control 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 vertical to the rotary axis. There are <b>two</b> translatory axes (linear axes) and <b>one</b> 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 <b>three</b> translatory axes (linear axes) and <b>one</b> axis of rotation (rotary axis).
Five-axis transformation machine types 3	Programming of orientation transformation. Kinematics with <b>three</b> linear axes and <b>two</b> 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	Programming of orientation transformation. Kinematics with <b>three</b> linear axes and <b>three</b> orthogonal rotary axes. The rotary axes are parallel to two of the three linear axes.
Two-axis swivel head with tool which rotates around itself and single-axis rotary table	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 (BCS)
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.

### 6.1.2 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.	<p>The axis sequence of the orientation axes and the orientation direction of the tool can either be configured on a <b>machine-specific</b> basis using machine data depending on the machine kinematics or on a <b>workpiece-specific</b> basis with programmable orientation independently of the machine kinematics.</p> <p>The directions of rotation of the orientation axes in the reference system are programmed with:</p> <ul style="list-style-type: none"><li>- ORIMKS reference system = machine coordinate system</li><li>- ORIWKS reference system = workpiece coordinate system</li></ul> <p>The default setting is ORIWKS.</p> <p>Programming of orientation axes with:</p> <ul style="list-style-type: none"><li>A, B, C of the machine axis position direct</li><li>A2, B2, C2 angle programming virtual axes with</li><li>- ORIEULER via Euler angle (standard)</li><li>- ORIRPY via RPY angle</li><li>- ORIVIRT1 via virtual orientation axes 1st definition</li><li>- ORIVIRT2 via virtual orientation axes 2nd definition</li></ul> <p>with differentiation between the interpolation type: <b>linear interpolation</b></p> <ul style="list-style-type: none"><li>- ORIAxes of orientation axes or machine axes</li></ul> <p><b>large radius circle interpolation</b> (interpolation of the orientation vector)</p> <ul style="list-style-type: none"><li>- ORIVECT from orientation axes</li></ul> <p>Programming orientation axes by specifying</p> <ul style="list-style-type: none"><li>A3, B3, C3 of the vector components (direction/surface normal)</li></ul> <p>Programming the resulting tool orientation</p> <ul style="list-style-type: none"><li>A4, B4, C4 of the vector surface normal at the beginning of the block</li><li>A5, B5, C5 of the vector perpendicular to the surface at the end of the block</li></ul> <p>LEAD leading angle for tool orientation</p> <p>TILT tilt angle for the tool orientation</p>

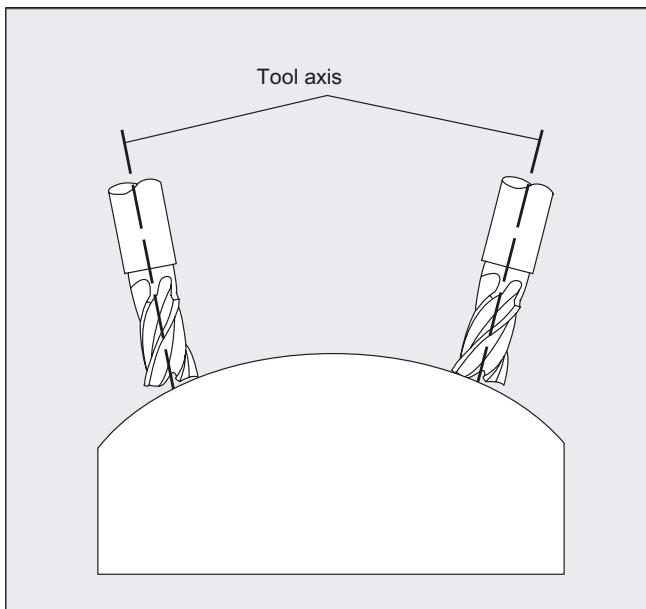
Machine type	Programming with active transformation TRAORI
	<p><b>Interpolation of the orientation vector on a taper peripheral surface</b>  Orientation changes to a taper peripheral surface <b>anywhere in space</b> using  interpolation:</p> <ul style="list-style-type: none"> <li>- ORIPLANE in the plane (large radius circle interpolation)</li> <li>- ORICONCW on a taper peripheral surface in the clockwise direction</li> <li>- ORICONCCW on a taper peripheral surface in the counter-clockwise direction</li> </ul> <p>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</p> <p>Changes in orientation in relation <b>to a path</b> with</p> <ul style="list-style-type: none"> <li>- ORICURVE specification of the movement of two contact points using</li> </ul> <p>PO[XH]=(xe, x2, x3, x4, x5) orientation polynomials up to the fifth degree</p> <p>PO[YH]=(ye, y2, y3, y4, y5) orientation polynomials up to the fifth degree</p> <p>PO[ZH]=(ze, z2, z3, z4, z5) orientation polynomials up to the fifth degree</p> <ul style="list-style-type: none"> <li>- ORIPATHS smoothing of orientation characteristic with</li> </ul> <p>A8, B8, C8 reorientation phase of tool corresponding to: direction and path length of tool during retraction movement</p>
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.	<p>Programming of <b>rotations</b> for tool orientation with</p> <p>LEAD angle, angle relative to surface normal vector</p> <p>PO[PHI] programming of a polynomial up to the fifth degree</p> <p>TILT angle rotation about path tangent (Z direction)</p> <p>PO[PSI] programming of a polynomial up to the fifth degree</p> <p>THETA angle of rotation (rotation about tool direction in Z)</p> <p>THETA= value reached at end of block</p> <p>THETA=AC(...) absolute non-modal switching to dimensions</p> <p>THETA=IC(...) non-modal switching to chain dimensions</p> <p>THETA=Θe interpolate programmed angle G90/G91</p> <p>PO[THT]=(..) programming of a polynomial up to the fifth degree</p> <p>programming of the rotation vector</p> <ul style="list-style-type: none"> <li>- ORIROTA rotation, absolute</li> <li>- ORIROTR relative rotation vector</li> <li>- ORIROTT tangential rotation vector</li> </ul>
Orientation relative to the path for orientation changes relative to the path or rotation of the rotary vector tangentially to the path	<p>Changes in orientation <b>relative to the path</b> with</p> <ul style="list-style-type: none"> <li>- ORIPATH tool orientation relative to the path</li> <li>- ORIPATHS also in the event of a blip in the orientation characteristic</li> </ul> <p>programming of rotation vector</p> <ul style="list-style-type: none"> <li>- ORIROTC tangential rotation vector, rotation to path tangent</li> </ul>

## 6.2 Three, four and five axis transformation (TRAORI)

### 6.2.1 General relationships of universal tool head

#### Function

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.

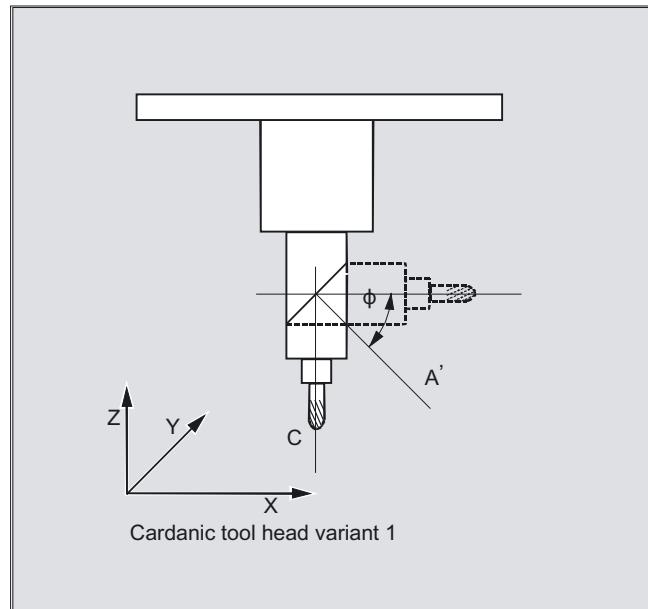


The machine design to achieve this is stored in the axis data.

## 5-Axis Transformation

### Cardanic tool head

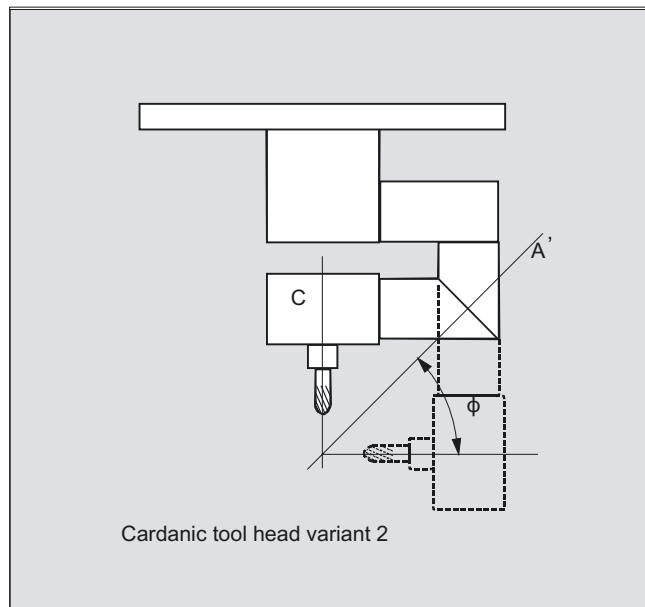
Three linear axes (X, Y, Z) and two orientation axes (C, A) define the setting angle and the operating point of the tool here. One of the two orientation axes is created as an inclined axis, in our example A' - in many cases, placed at 45°.



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.



In this example, A' lies below the angle  $\phi$  to the X axis.

The following possible relations are generally valid:

A' lies below the angle  $\phi$  to the X axis

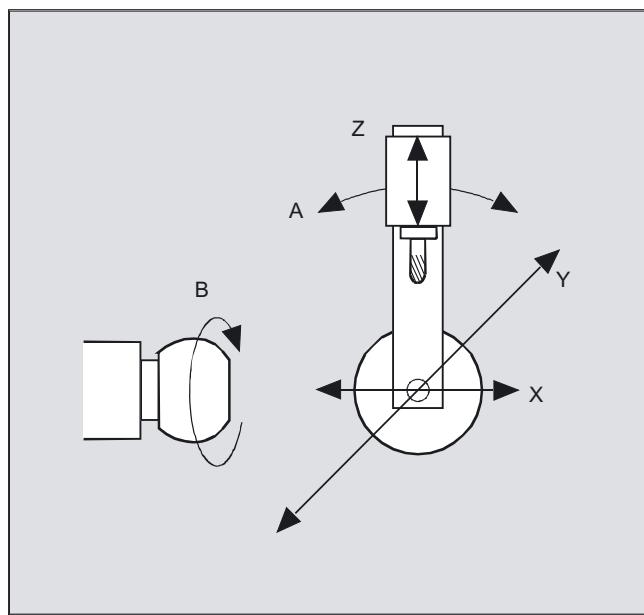
B' lies below the angle  $\phi$  to the Y axis

C' lies below the angle  $\phi$  to the Z axis

Angle  $\phi$  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), Universal Milling Head section, "Parameter Setting".

### **6.2.2 Three, four and five axis transformation (TRAORI)**

#### **Function**

The user can configure two or three transulatory 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.

#### **References:**

/FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformations (F2)

#### **Programming**

```
TRAORI (n)
or
TRAORI (n, X, Y, Z, A, B)
or
TRAFOOF
```

#### **Parameter**

TRAORI	Activates the first specified orientation transformation
TRAORI (n)	Activates the orientation transformation specified by n
n	The number of the transformation (n = 1 or 2), TRAORI(1) corresponds to orientation transformation on
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 position of the rotary axes involved in the transformation causes so many compensating movements of the remaining machine axes that the position of the tool tip is unchanged.

---

Orientation transformation always points from the tool tip to the tool adapter.

### Example of generic transformations

The basic orientation of the tool is indicated as follows:

TRAORI(1, 0, 0, 1) Z direction

TRAORI(1, 0, 1, 0) Y direction

TRAORI(1, 0, 1, 1) Y/Z direction (corresponds to the position -45°)

### 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.

### **6.2.3 Variants of orientation programming and initial setting (OTIRESET)**

#### **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 round 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.

#### **6.2.4 Programming of the tool orientation (A..., B..., C..., LEAD, TILT)**

##### **Function**

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 of 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 of 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 of 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, ORIAxisPOS 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, ORIAxisPOS 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.
-

## *Transformations*

### *6.2 Three, four and five axis transformation (TRAORI)*

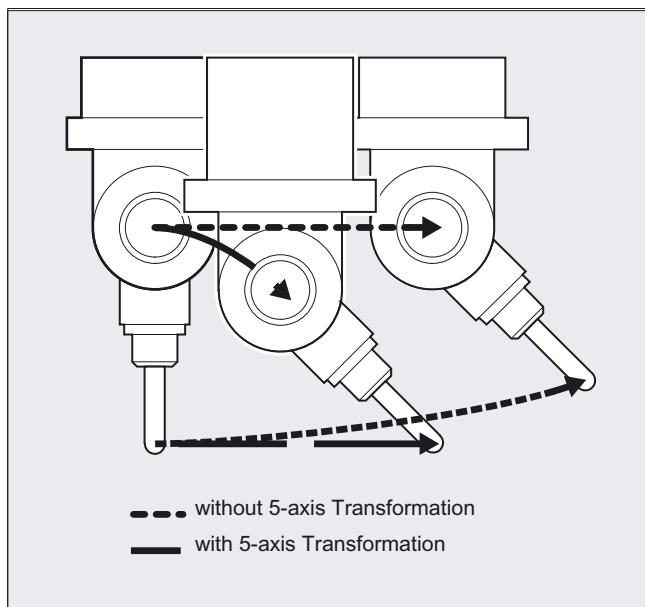
## **Programming**

G1 X Y Z A B C	Programming of rotary axis motion
G1 X Y Z A2= B2= C2=	Programming in Euler angles
G1 X Y Z A3== B3== C3==	Programming of 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 put up 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



### Description

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:

- 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 round axis positions
- ORIPY2 via RPY angle (rotation sequence XYZ)

### Machine manufacturer

The machine manufacturer can use machine data to define various variants. Please refer to the machine manufacturer's instructions.

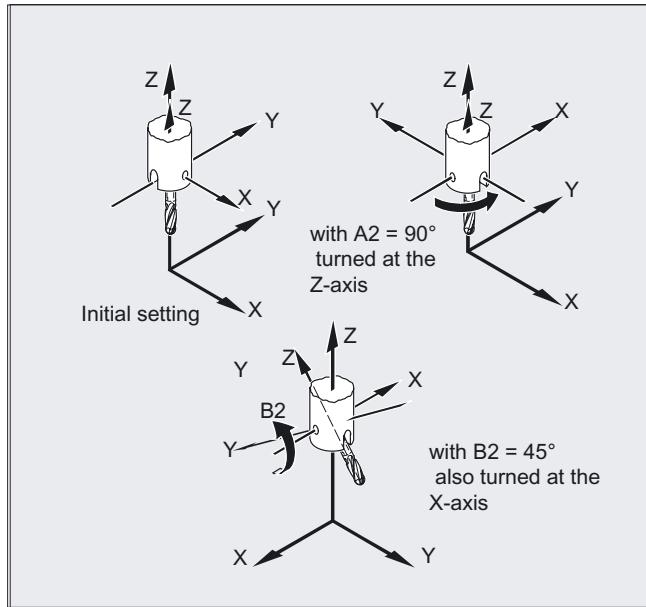
---

## 6.2 Three, four and five axis transformation (TRAORI)

### 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

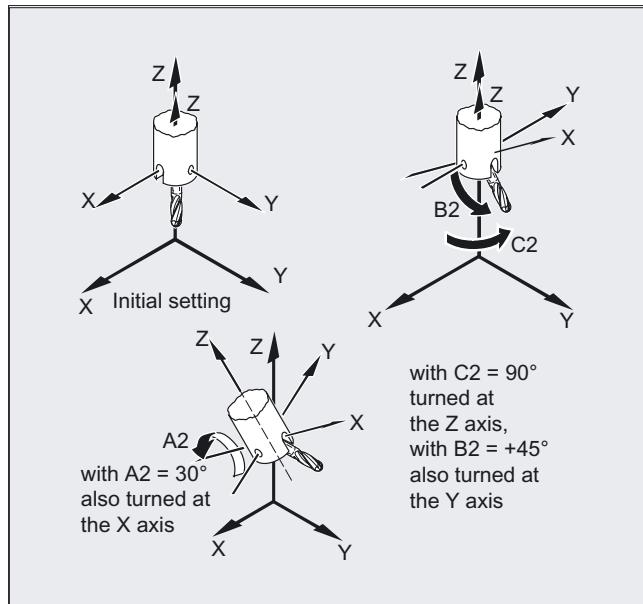
In contrast to Euler angle programming, all three values here have an effect on the orientation vector.

---

**Machine manufacturer**

When defining angles with orientation angles via RPY angle, 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.



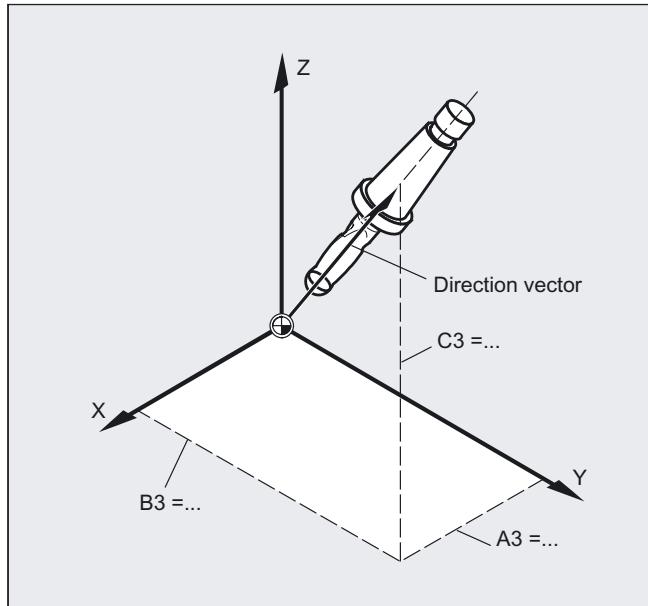
By defining the orientation axes via G code, if the machine data  
 $\$MC\_ORI\_DEF\_WITH\_G\_CODE = 1$ , then:

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 X axis and lastly with C2 around the new Z axis.

**Programming of 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.



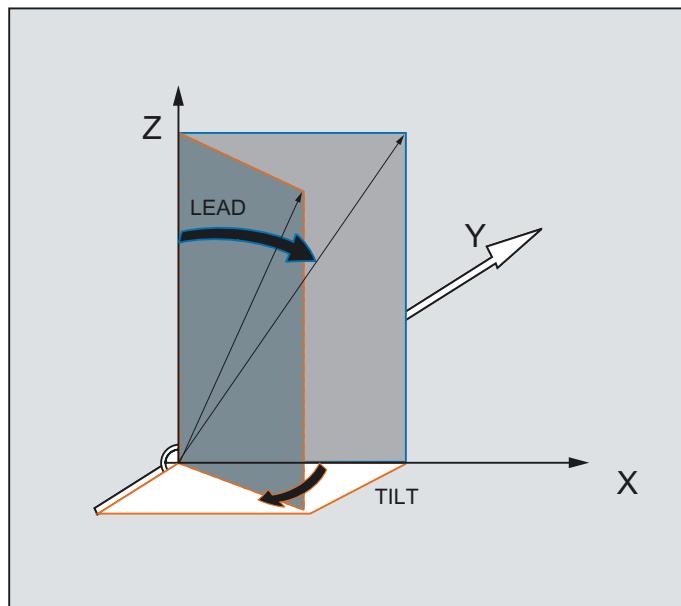
**Programming the tool orientation with LEAD= and TILT=**

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 compensation)**

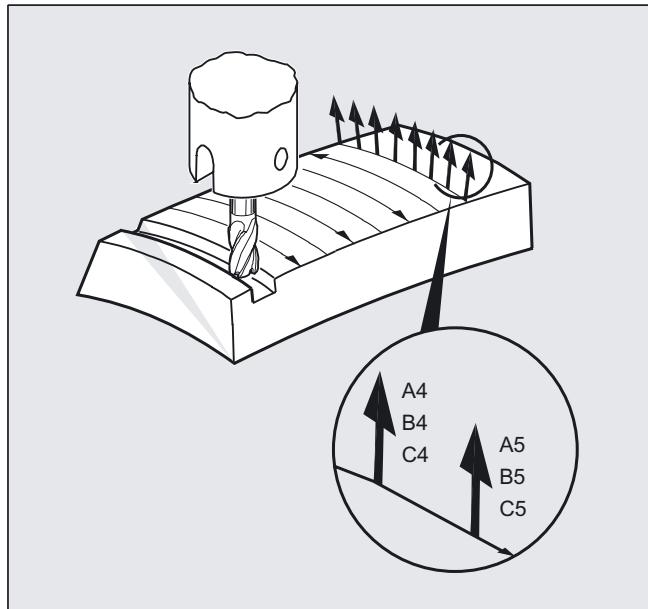
If the block is shortened at an inside corner, the resulting tool orientation is also achieved at the end of the block.

**Definition of tool orientation with LEAD= and TILT=**

### 6.2.5 Face milling (3D-milling A4, B4, C4, A5, B5, C5)

#### Function

Face milling is used to machine curved surfaces of any kind.



For this type of 3D milling, you require line-by-line definition of 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-radius circular interpolation to the programmed end value.

If both start and end vectors are programmed, interpolation runs between the two directions, also via large-radius circular 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.

With active ORIWKS (see "Reference of the orientation axes (ORIWKS, ORIMKS)") , the surface normal vectors relate to the active frame and rotate when the frame rotates.

#### **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.

## **6.2.6 Orientation axis reference (ORIWKS, ORIMKS)**

### **Function**

For orientation programming in the workpiece coordinate system using

- Euler or RPY angle or
- orientation vector

the motion of the rotary motion can be set using ORIMKS/ORIWKS.

#### **Machine manufacturer**

Machine data \$MC\_ORI\_IPO\_WITH\_G\_CODE specifies the active interpolation mode:

ORIMKS/ORIWKS

or

ORIMACHAX/ORIVIRTAX.

## Programming

N.. ORIMKS=

or

N.. ORIWKS=

## Parameters

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, always choose ORIWKS. Which movements the machine actually executes depend on the machine kinematics.

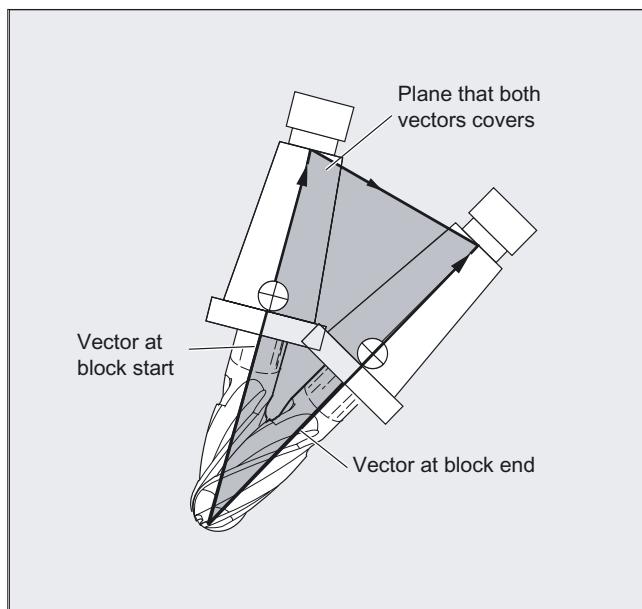
---

With ORIMKS you can program actual machine movements, for example, to avoid collisions with devices, etc.

## Description

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.

### **6.2.7 Programming the orientation axes (ORIAXES, ORIVECT, ORIEULER, ORIRPY)**

#### **Function**

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/TR/TT, ORIROTC, THETA)").

#### **Programming**

Axis identifiers A2, B2 and C2 are used to program the orientation axes.

N... ORIAXES or ORIVECT	Linear or large-radius circular interpolation
N... G1 X Y Z A B C	
or	
N... ORIPLANE	orientation interpolation of the plane
or	
N ... ORIEULER or ORIRPY and/or ORIRPY2	Orientation angle Euler/RPY angle
N... G1 X Y Z A2= B2= C2=	Angle programming of virtual axes
or	
N... ORIVIRT1 or ORIVIRT2	
N... G1 X Y Z A3= B3= C3=	definition 1 or 2 direction vector programming of virtual orientation axes

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, ORICONxx)".

## Parameters

ORIAxes	Linear interpolation of machine or orientation axes
ORIVECT	Large-radius circular interpolation (identical to ORIPLANE)
ORIMKS	Rotation in the machine coordinate system
ORIWKS	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. The rotation sequence is XYZ and: A2 is the rotation angle around X B2 is the rotation angle around Y C2 is the rotation angle around Z
ORIRPY2	Orientation programming via RPY angle. The rotation sequence is ZYX and: A2 is the rotation angle around Z B2 is the rotation angle around Y C2 is the rotation angle around X
A2= B2= C2=	Angle programming of virtual axes
ORIVIRT1	Orientation programming using virtual orientation axes
ORIVIRT2	(definition 1), definition according to MD \$MC_ORIAxis_TURN_TAB_1 (definition 2), definition according to MD \$MC_ORIAxis_TURN_TAB_2
A3= B3= C3=	Direction vector programming of direction axis

## Description

### Machine manufacturer

MD \$MC\_ORI\_DEF\_WITH\_G\_CODE is used to specify how the programmed angles A2, B2, C2 are defined:

The definition is according to MD \$MC\_ORIENTATION\_IS\_EULER (standard) or the definition is according to G group 50 (ORIEULER, ORIRPY, ORIVIRT1, ORIVIRT2).

MD \$MC\_ORI\_IPO\_WITH\_G\_CODE is used to define which interpolation mode type is active: ORIWKS/ORIMKS or ORIAxes/ORIVECT.

*6.2 Three, four and five axis transformation (TRAORI)*

**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:

```
$MC_JOG_VEL0_RAPID_GEO  
$MC_JOG_VEL0_GEO  
$MC_JOG_VEL0_RAPID_ORI  
$MC_JOG_VEL0_ORI
```

---

**Note**

**SINUMERIK 840D 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:**

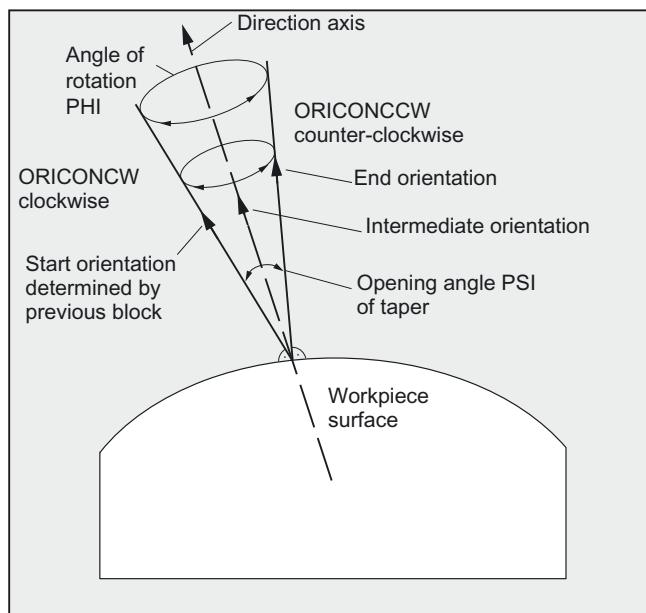
/FB2/ Description of Functions Extended Functions; Kinematic Transformation (M1)

---

### 6.2.8 Orientation programming along the peripheral surface of a taper (ORIPLANE, ORICONxx)

#### Function

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-radius circular 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  
N... A6= B6= C6= A3= B3= C3=  
or  
N... ORICONTO  
N... G1 X Y Z A6= B6= C6=  
or  
N... ORICONIO  
N... G1 X Y Z A7= B7= C7=  
N... PO[PHI]=(a2, a3, a4, a5)  
N... PO[PSI]=(b2, b3, b4, b5)

**Interpolation** on the outside of a taper with  
direction vector in the  
clockwise/counterclockwise direction of  
the taper and end orientation or  
tangential transition and  
specification of end orientation  
or  
specification of end orientation and  
intermediate orientation on the outside of  
the taper with  
polynomials for angle of rotation and  
polynomials for opening angle

## Parameters

ORIPLANE	Interpolation in the plane (large-radius circular 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 counterclockwise 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 smaller 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 of different changes to orientation**

```
...
N10 G1 X0 Y0 F5000
N20 TRAORI(1) ;Orientation transformation ON
N30 ORIVECT    ;Interpolate tool orientation as a vector
...
N40 ORIPLANE   ;Tool orientation in the plane
;Select large-radius circular interpolation
N50 A3=0 B3=0 C3=1
N60 A3=0 B3=1 C3=1 ;Orientation in the Y/Z plane is rotated about
;45 degrees
;at the end of block, the
;orientation (0, 1/√2, 1/√2) is reached.
...
N70 ORICONCW   ;Orientation vector is interpolated in the
;clockwise direction on the
;outside of the taper with the
N80 A6=0 B6=0 C6=1 A3=0 B3=0 C3=1 ;direction (0,0,1) to orientation
; ;(1/√2, 0, 1/√2)
;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.
```

**Description**

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.

### **6.2.9 Specification of orientation for two contact points (ORICURVE, PO[XH]=, PO[YH]=, PO[ZH]=)**

#### **Function**

##### **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:**

/FB1/ Function Manual, Basic Functions; Continuous-Path Mode, Exact Stop, Look Ahead (B1), chapter: 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.

---

### 6.3 Orientation polynomials (PO[angle], PO[coordinate])

## Function

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

### 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)

or

N... PO[THT]=(d2, d3, d4, d5)

Interpolation of the rotation relative to the path

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

## Description

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.

## **6.4 Rotations of the tool orientation (ORIROTA, ORIROTR/TT, ORIROTC, THETA)**

### **Function**

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 ORIROTR or ORIROTT 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]=(d2, d3, d4, d5)	Interpolate angle of rotation with a 5th order polynomial

## 6.4 Rotations of the tool orientation (ORIROTA, ORIROTR/TT, ORIROTC, THETA)

**Significance**

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 $\Theta_e$ 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 of rotations of orientations**

Program code	Comments
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  $\theta(u) = +90u^2$ . 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.

## Description

### **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.

## **6.5 Orientations relative to the path**

### **6.5.1 Orientation types relative to the path**

#### **Function**

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-radius circular 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), Chapter "Orientation"

## **Significance**

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.

---

## **6.5.2 Rotation of the tool orientation relative to the path (ORIPATH, ORIPATHS, angle of rotation)**

### **Function**

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  
N... ORIPATHS

Activate type of orientation relative to the path  
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)  
N... PO[PSI]=(b2, b3, b4, b5)  
N... PO[THT]=(d2, d3, d4, d5)

Polynomial for the leading angle LEAD  
Polynomial for the tilt angle TILT  
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.

## Significance

### Tool orientation relative to the path

ORIPATH	Tool orientation in relation to path
ORIPATHS	Tool orientation relative to the path; blip in orientation characteristic is 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.

---

## 6.5.3 Interpolation of the tool rotation relative to the path (ORIROTC, THETA)

### Function

#### 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.

## Significance

### 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 $\Theta_e$ of rotation vector
THETA=AC (...)	Switch over to absolute dimensions per block
THETA=IC (...)	Switch over to incremental dimensions per block
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-radius circular 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-radius circular interpolation (identical to ORIPLANE)
ORIMKS	Rotation in the machine coordinate system
ORIWKS	Rotation in the workpiece coordinate system
A= B= C=	Description, see the Rotations of the tool orientation section
ORIEULER	Programming the machine axis position
ORIRPY	Orientation programming via Euler angle
A2= B2= C2=	Orientation programming via RPY angle
ORIVIRT1	Angle programming of virtual axes
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-radius circular 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 counterclockwise 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 smaller than or equal to 180 degrees
NUT=-181	Traverse angle greater than or equal to 180 degrees

ORICONIO A7= B7= C7=	Interpolation on the peripheral surface of a taper
ORICURVE XH YH ZH, e.g., with polynomials PO[XH]=(xe, x2, x3, x4, x5)	Intermediate orientation (programming as normalized vector) 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.

## 6.5.4 Smoothing of orientation characteristic (ORIPATHS A8=, B8=, C8=)

### Function

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 more detailed information about the "Orientation relative to the path" function, please refer to

**References:** /FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformation (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

## Significance

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-radius circular interpolation.

---

## 6.6 Compression of the orientation (COMPON, COMPCURV, COMPCAD)

### Function

NC programs, in which orientation transformation (**TRAORI**) is active, and the orientation is programmed using direction vectors, can be compressed if kept within specified limits.

---

#### Note

The orientation movement is only compressed if large-radius circular interpolation is active and depends, therefore, on the G code for orientation interpolation. This can be set via machine data, as can the maximum path length and a permissible tolerance for each axis or for the path feedrate for the compressor function. Please refer to the machine manufacturer's instructions.

---

### 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 **Eulerangles** or **RPY-angles** via:  
`A2=<...> B2=<...> C2=<...>`

#### Rotation of the tool

For **six-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 358) ".

---

#### Note

NC blocks in which additional rotation is programmed, can only be compressed if the angle of rotation changes **linearly**, meaning that a polynomial with `PO[THT]=(...)` for the angle of rotation should not be programmed.

---

---

## 6.6 Compression of the orientation (*COMPON*, *COMPCURV*, *COMPCAD*)

### General structure of an NC block that 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),  
Chapter: "Compression of the orientation"

**Activation/deactivation**

Compressor functions are activated using the modal G codes COMPON, COMPCURV or COMPCAD.

COMPOF terminates the compressor function.

See " NC block compression (COMPON, COMPCURV, COMPCAD) (Page 252) ".

---

**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.

---

### *6.6 Compression of the orientation (COMPON, COMPCURV, COMPCAD)*

## 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	Comments
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 angle 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	

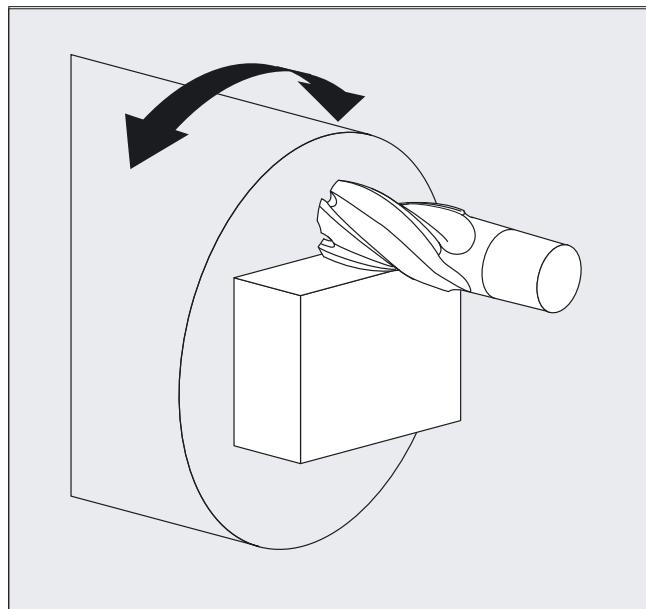
## 6.7 Kinematic transformation

### 6.7.1 Milling on turned parts (TRANSMIT)

#### Function

The TRANSMIT function enables the following:

- Face machining on turned parts in the turning clamp (drill-holes, contours).
- A cartesian coordinate system can be used to program these machining operations.
- The control maps the programmed traversing movements of the Cartesian coordinate system onto the traversing movements of the real machine axes (standard situation):
  - Rotary axis
  - Infeed axis perpendicular to rotary axis
  - Longitudinal axis parallel to rotary axis
  - The linear axes are positioned perpendicular to one another.
- A tool center offset relative to the turning center is permitted.
- The velocity control makes allowance for the limits defined for the rotations.



### **TRANSMIT transformation types**

The TRANSMIT machining operations have two parameterizable forms:

- TRANSMIT in the standard case with (TRAFO\_TYPE\_n = 256)
- TRANSMIT with additional Y linear axis (TRAFO\_TYPE\_n = 257)

The extended transformation type 257 can be used, for example, to compensate clamping compensations of a tool with real Y axis.

### **Syntax**

TRANSMIT or TRANSMIT(n)  
TRAFOOF

#### **Rotary axis**

The rotary axis cannot be programmed because it is occupied by a geometry axis and cannot thus be programmed directly as a channel axis.

### **Significance**

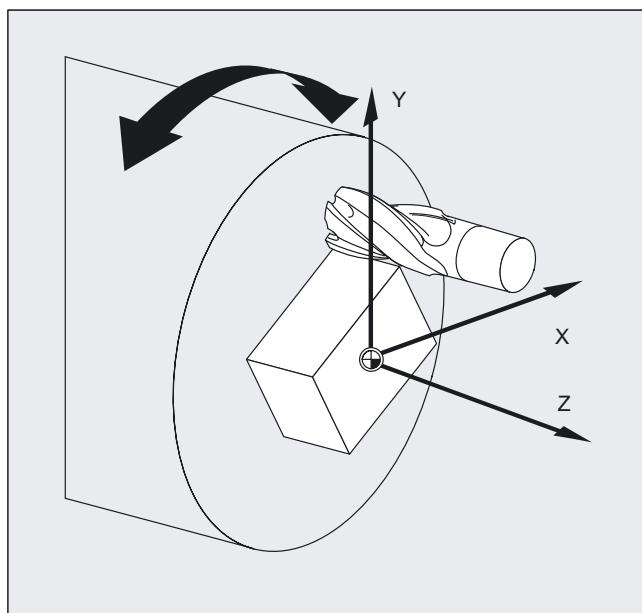
TRANSMIT	Activates the first declared TRANSMIT function. This function is also designated as polar transformation.
TRANSMIT(n) )	Activates the nth declared TRANSMIT function; n can be up to 2 (TRANSMIT(1) is the same as TRANSMIT).
TRAFOOF	Deactivates an active transformation.
OFFN	Offset contour normal: Distance of the face machining from the programmed reference contour.

---

#### **Note**

An active TRANSMIT transformation is likewise deactivated if one of the other transformations is activated in the relevant channel (e.g., TRACYL, TRAANG, TRAORI).

---

**Example**

Program code	Comments
N10 T1 D1 G54 G17 G90 F5000 G94	; Tool selection
N20 G0 X20 Z10 SPOS=45	; Approach the starting position
N30 TRANSMIT	; Activate TRANSMIT function
N40 ROT RPL=-45	; Set frame
N50 ATRANS X-2 Y10	
N60 G1 X10 Y-10 G41 OFFN=1OFFN	; Square roughing; 1 mm tolerance
N70 X-10	
N80 Y10	
N90 X10	
N100 Y-10	
N110 G0 Z20 G40 OFFN=	; Tool change
N120 T2 D1 X15 Y-15	
N130 Z10 G41	
N140 G1 X10 Y-10	; Square finishing
N150 X-10	
N160 Y10	
N170 X10	
N180 Y-10	
N190 Z20 G40	; Deselect frame
N200 TRANS	
N210 TRAFOOF	
N220 G0 X20 Z10 SPOS=45	; Approach the starting position
N230 M30	

## Description

### Pole

There are two ways of passing through the pole:

- Traversal along linear axis
- Traverse to the pole, rotate the rotary axis at the pole and traveling away from the pole

Make the selection using MD 24911 and 24951.

### TRANSMIT with additional Y linear axis (transformation type 257):

This transformation variant of the polar transformation makes use of the redundancy for a machine with another linear axis in order to perform an improved tool compensation. The following conditions then apply to the second linear axis:

- A smaller working area and
- The second linear axis should not be used for the retraction of the parts program.

Certain machine data settings are assumed for the parts program and the assignment of the corresponding axes in the BCS or MCS, see

### References

/FB2/ Function Manual Extended Functions; Kinematic Transformations (M1)

## 6.7.2 Cylinder surface transformation (TRACYL)

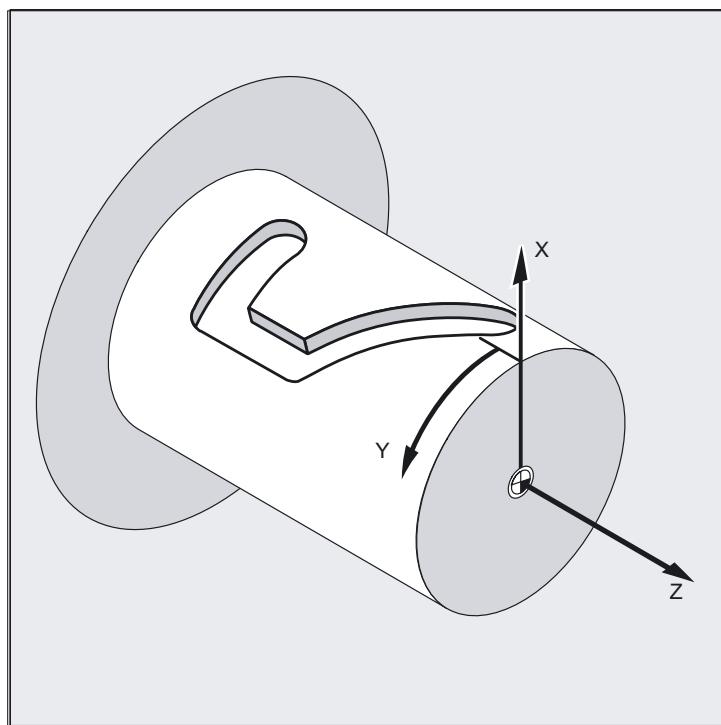
### Function

The TRACYL cylinder surface transformation function can be used to:

#### Machine

- longitudinal grooves on cylindrical bodies,
- Transverse grooves on cylindrical objects,
- grooves with any path on cylindrical bodies.

The path of the grooves is programmed with reference to the unwrapped, level surface of the cylinder.



### TRACYL transformation types

There are three forms of cylinder surface coordinate transformation:

- TRACYL without groove wall offset (TRAFO\_TYPE\_n=512)
- TRACYL with groove wall offset: (TRAFO\_TYPE\_n=513)
- TRACYL with additional linear axis and groove wall offset: (TRAFO\_TYPE\_n=514)  
The groove wall offset is parameterized with TRACYL using the third parameter.

For cylinder peripheral curve transformation with groove side compensation, the axis used for compensation should be positioned at zero ( $y=0$ ), so that the groove centric to the programmed groove center line is finished.

### Axis utilization

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).

## Syntax

TRACYL (d) or TRACYL (d, n) or  
for transformation type 514  
TRACYL (d, n, groove side offset)  
TRAFOOF

### Rotary axis

The rotary axis cannot be programmed as it is occupied by a geometry axis and thus cannot be programmed directly as channel axis.

## Significance

TRACYL (d)	Activates the first TRACYL function specified in the channel machine data. d is the parameter for the working diameter.
TRACYL (d, n)	Activates the n-th TRACYL function specified in the channel machine data. The maximum for n is 2, TRACYL(d,1) corresponds to TRACYL(d).
D	Value for the working diameter. The working diameter is double the distance between the tool tip and the turning center. This diameter must always be specified and be larger than 1.
n	Optional 2nd parameter for the TRACYL data block 1 (preselected) or 2.
Slot side compensation	Optional 3rd parameter whose value for TRACYL is preselected using the mode for machine data. Value range: 0: Transformation type 514 without groove wall offset as previous 1: Transformation type 514 with groove wall offset
TRAFOOF	Transformation OFF (BCS and MCS are once again identical).
OFFN	Offset contour normal: Distance of the groove side from the programmed reference contour.

---

### Note

An active TRACYL transformation is likewise deactivated if one of the other transformations is activated in the relevant channel (e.g., TRANSMIT, TRAANG, TRAORI).

---

### Example: Tool definition

The following example is suitable for testing the parameterization of the TRACYL cylinder transformation:

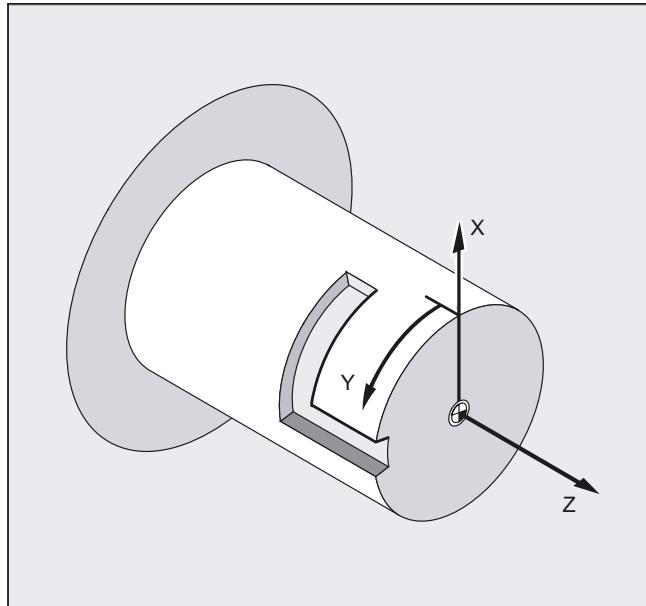
Program code	Comments
Tool parameters	Significance
Number (DP)	Comment
\$TC_DP1[1,1]=120	Tool type
\$TC_DP2[1,1]=0	Tool nose position only for turning tools

Program code	Comments
Geometry	Length compensation
\$TC_DP3[1,1]=8.	Length offset vector
\$TC_DP4[1,1]=9.	Calculation acc. to type and plane
\$TC_DP5[1,1]=7.	

Program code	Comments
Geometry	Radius
\$TC_DP6[1,1]=6.	Radius
\$TC_DP7[1,1]=0	Slot width b for slotting saw, rounding radius for milling tools
\$TC_DP8[1,1]=0	Projection k
\$TC_DP9[1,1]=0	
\$TC_DP10[1,1]=0	
\$TC_DP11[1,1]=0	Angle for taper milling tools

Program code	Comments
Wear	Tool length and radius compensation
\$TC_DP12[1,1]=0	Remaining parameters to \$TC_DP24=0
	Tool base dimension/ adapter

**Example: Making a hook-shaped groove**



**Activate cylinder surface transformation:**

Program code	Comments
N10 T1 D1 G54 G90 F5000 G94	; Tool selection, clamping compensation
N20 SPOS=0	; Approach the starting position
N30 G0 X25 Y0 Z105 CC=200	
N40 TRACYL (40)	; Activate cylinder surface ;transformation
N50 G19	; Plane selection

**Machining a hook-shaped groove:**

Program code	Comments
N60 G1 X20	; Infeed tool to groove base
N70 OFFN=12	; Define 12 mm groove side spacing relative to groove center line
N80 G1 Z100 G42	; Approach right side of groove
N90 G1 Z50	; Groove cut parallel to cylinder axis
N100 G1 Y10	; Groove cut parallel to circumference
N110 OFFN=4 G42	; Approach left side of the groove; define 4 mm groove side spacing relative to the groove center line
N120 G1 Y70	; Groove cut parallel to circumference
N130 G1 Z100	; Groove cut parallel to cylinder axis
N140 G1 Z105 G40	; Retract from groove wall

Program code	Comments
N150 G1 X25	; Retract
N160 TRAFOOF	
N170 G0 X25 Y0 Z105 CC=200	; Approach the starting position
N180 M30	

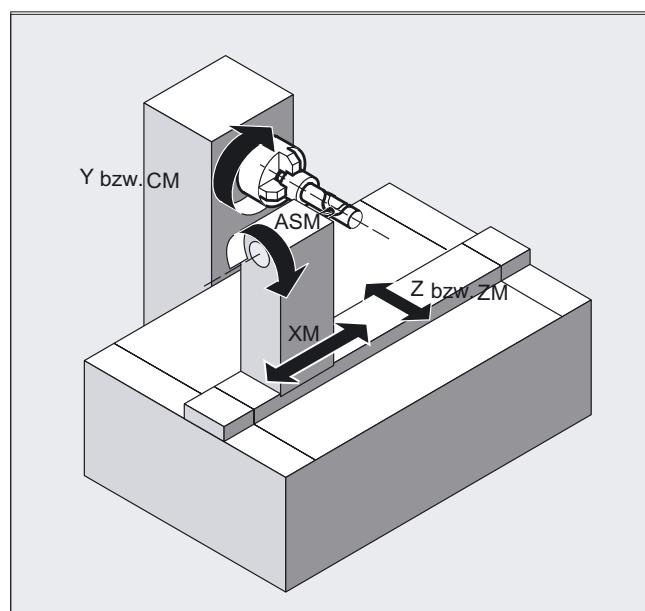
## Description

### Without groove wall offset (transformation type 512):

The control transforms the programmed traversing movements of the cylinder coordinate system to the traversing movements of the real machine axes:

- Rotary axis
- Infeed axis perpendicular to rotary axis
- Longitudinal axis parallel to rotary axis

The linear axes are positioned perpendicular to one another. The infeed axis cuts the rotary axis.

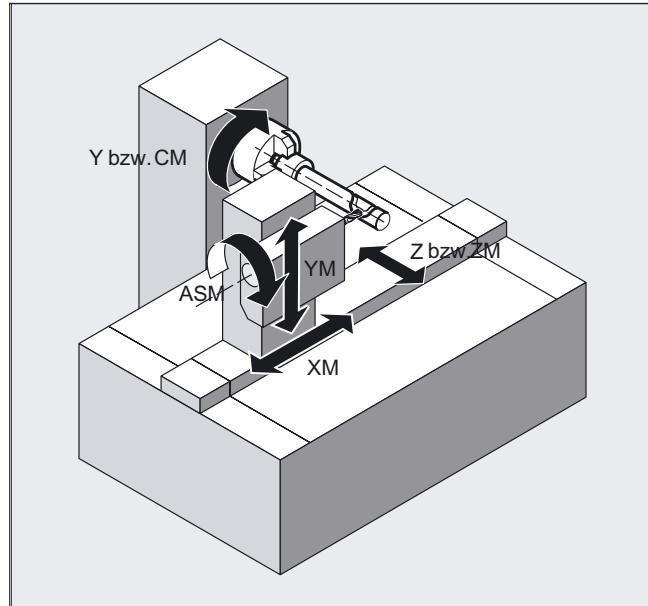


**With groove wall offset (transformation type 513):**

Kinematics as above, but an additional longitudinal axis parallel to the peripheral direction

The linear axes are positioned perpendicular to one another.

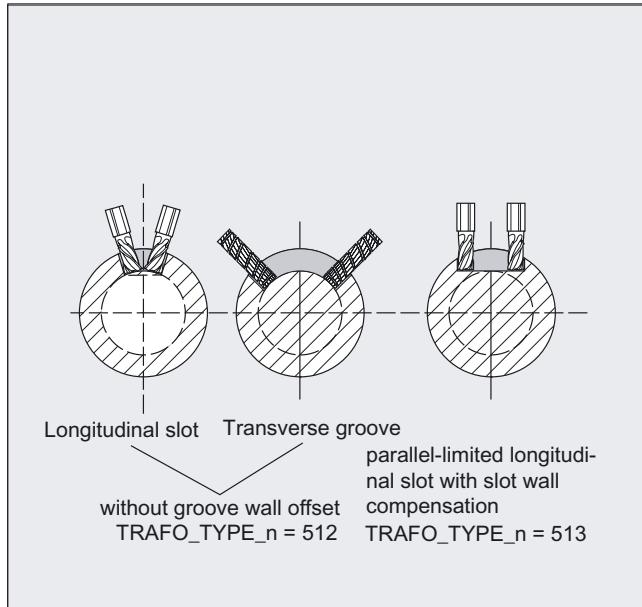
The velocity control makes allowance for the limits defined for the rotations.



### Groove traversing-section

In the case of axis configuration 1, longitudinal grooves along the rotary axis are subject to parallel limits only if the groove width corresponds exactly to the tool radius.

Grooves in parallel to the periphery (transverse grooves) are not parallel at the beginning and end.



### With additional linear axis and groove wall offset (transformation type 514):

On a machine with a second linear axis, this transformation variant makes use of redundancy in order to perform improved tool compensation. The following conditions then apply to the second linear axis:

- a smaller working area and
- the second linear axis should not be used for the travel through the parts program.

Certain machine data settings are assumed for the parts program and the assignment of the corresponding axes in the BCS or MCS, see

### References

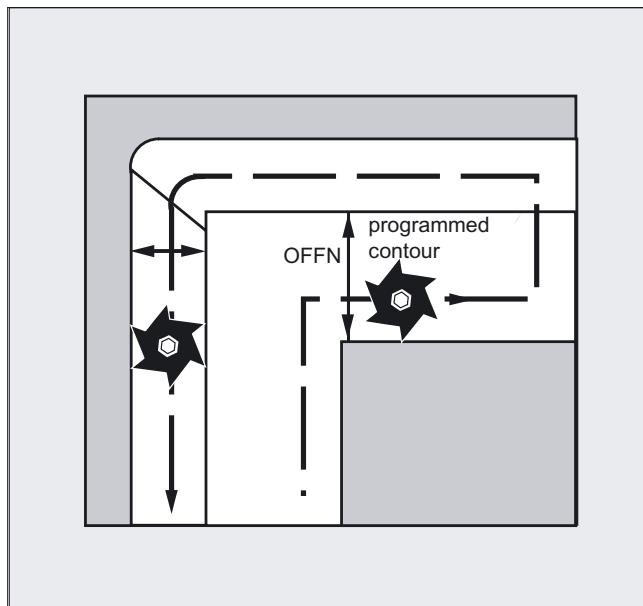
/FB2/ Function Manual Extended Functions; Kinematic Transformations (M1)

### Offset contour normal OFFN (transformation type 513)

To mill grooves with TRACYL, the following is programmed:

- groove center line in the part program,
- **half the groove width** programmed using OFFN.

To avoid damage to the groove side OFFN acts only when the tool radius compensation is active. Furthermore, OFFN should also be  $\geq$  the tool radius to avoid damage occurring to the opposite side of the groove.



A parts program for milling a groove generally comprises the following steps:

1. Selecting a tool
2. Select TRACYL
3. Select suitable coordinate offset (frame)
4. Position
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. Position

12. TRAFOOF

13. Re-select original coordinate shift (frame)

#### Special features

- TRC selection:

TRC is not programmed in relation to the groove side, but relative to the programmed groove center line. To prevent the tool traveling to the left of the groove side, G42 is entered (instead of G41). You avoid this if in OFFN, the groove width is entered with a negative sign.

- OFFN acts differently with TRACYL than it does without TRACYL. As, even without TRACYL, OFFN is included when TRC is active, OFFN should be reset to zero after TRAFOOF.
- It is possible to change OFFN within a parts program. This could be used to shift the groove center line from the center (see diagram).
- Guiding grooves:

TRACYL does not create the same groove for guiding grooves as it would be with a tool with the diameter producing the width of the groove. It is basically not possible to create the same groove side geometry with a smaller cylindrical tool as it is with a larger one. TRACYL minimizes the error. To avoid problems of accuracy, the tool radius should only be slightly smaller than half the groove width.

---

#### Note

#### OFFN and TRC

With TRAFO\_TYPE\_n = 512, the value is effective under OFFN as an allowance for TRC.

With TRAFO\_TYPE\_n = 513, half the groove width is programmed in OFFN. The contour is retracted with OFFN-TRC.

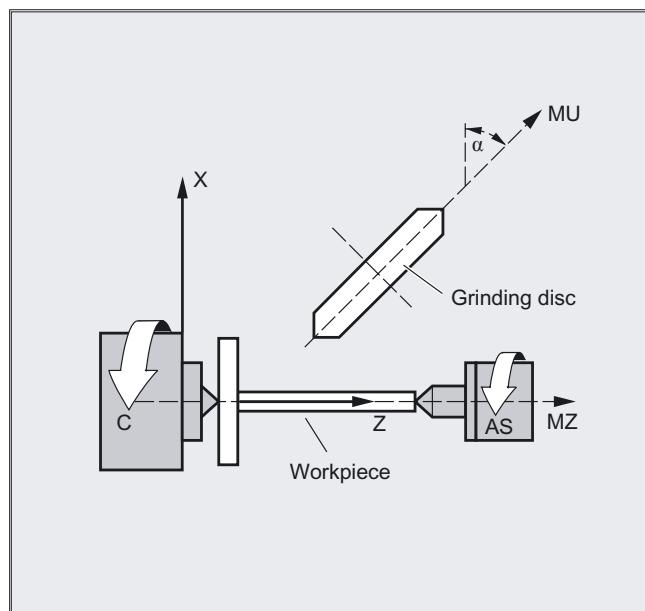
---

### 6.7.3 Inclined axis (TRAANG)

#### Function

The inclined axis function is intended for grinding technology and facilitates the following performance:

- Machining with an oblique infeed axis
- A Cartesian coordinate system can be used for programming purposes.
- The control maps the programmed traversing movements of the Cartesian coordinate system onto the traversing movements of the real machine axes (standard situation): Inclined infeed axis.



#### Syntax

TRAANG ( $\alpha$ ) or TRAANG ( $\alpha$ , n)

TRAFOOF

## Significance

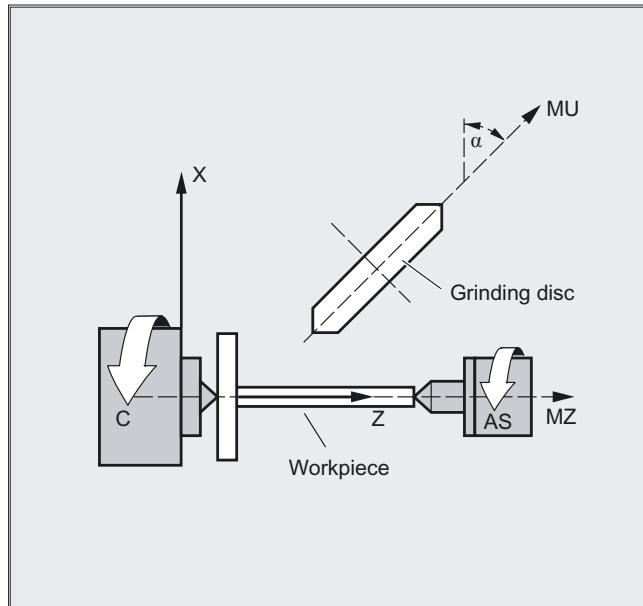
TRAANG( ) or TRAANG( ,n)	Activate transformation with the parameterization of the previous selection.
TRAANG( $\alpha$ )	Activates the first specified inclined axis transformation
TRAANG( $\alpha$ , n)	Activates the nth agreed inclined axis transformation. The maximum value of n is 2. TRAANG( $\alpha$ ,1) corresponds to TRAANG( $\alpha$ ).
$\alpha$ A	Angle of the inclined axis Permissible values for $\alpha$ are: -90 degrees < $\alpha$ < + 90 degrees
TRAFOOF	Transformation off
n	Number of agreed transformations

### Angle $\alpha$ omitted or zero

If  $\alpha$  (angle) is omitted (e.g., TRAANG( ), TRAANG( , n ) ), the transformation is activated with the parameterization of the previous selection. On the first selection, the default settings according to the machine data apply.

An angle  $\alpha = 0$  (e.g., TRAANG( 0 ), TRAANG( 0, n ) ) is a valid parameter setting and is no longer equivalent to the omission of the parameter, as in the case of older versions.

## Example

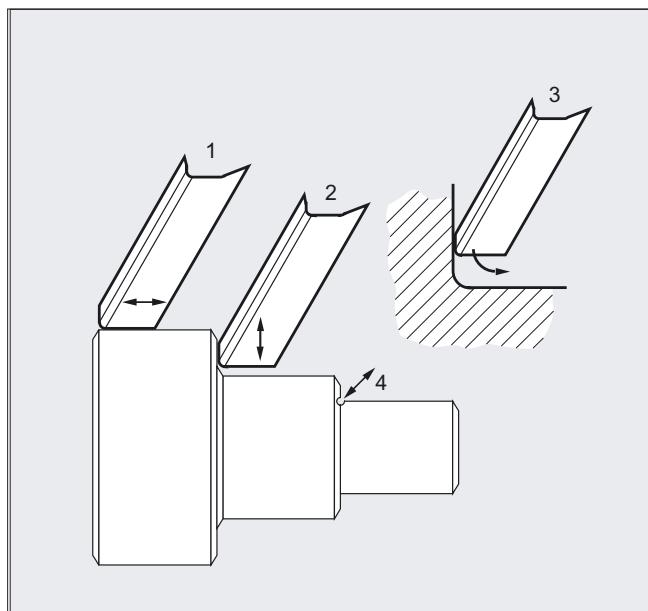


Program code	Comments
N10 G0 G90 Z0 MU=10 G54 F5000 ->	; Tool selection, clamping compensation,
-> G18 G64 T1 D1	Plane selection
N20 TRAANG(45)	; Activate inclined axis transformation
N30 G0 Z10 X5	; Approach the starting position
N40 WAITP(Z)	; Release axis for oscillation
N50 OSP[Z]=10 OSP2[Z]=5 OST1[Z]=-2 ->	; Oscillation until the dimension is reached
-> OST2[Z]=-2 FA[Z]=5000	(Oscillation, see Chapter "Oscillation")
N60 OS[Z]=1	
N70 POS[X]=4.5 FA[X]=50	
N80 OS[Z]=0	
N90 WAITP(Z)	; Release oscillating axes as positioning axes
N100 TRAFOOF	; Deactivate transformation
N110 G0 Z10 MU=10	; Retract
N120 M30	;
-> program in a single block	

**Description**

The following machining operations are possible:

1. Longitudinal grinding
2. Face grinding
3. Grinding of a specific contour
4. Oblique plunge-cut grinding.



### Machine manufacturer

The following settings are defined in machine data:

- The angle between a machine axis and the oblique axis,
- The position of the zero point of the tool relative to the origin of the coordinate system specified by the "inclined axis" function,
- The speed reserve held ready on the parallel axis for the compensating movement,
- The axis acceleration reserve held ready on the parallel axis for the compensating movement.

### Axis configuration

To program in the Cartesian coordinate system, it is necessary to inform the control of the correlation between this coordinate system and the actually existing machine axes (MU,MZ):

- Assignment of names to geometry axes
- Assignment of geometry axes to channel axes
  - general situation (inclined axis not active)
  - inclined axis active
- Assignment of channel axes to machine axis numbers
- Identification of spindles
- Allocation of machine axis names.

Apart from "inclined axis active", the procedure corresponds to the procedure for normal axis configuration.

## 6.7.4 Inclined axis programming (G05, G07)

### Function

In Jog mode, the movement of the grinding wheel can either be cartesian or in the direction of the inclined axis (the display stays cartesian). All that moves is the real U axis, the Z axis display is updated.

In jog-mode, REPOS-offsets must be traversed using Cartesian coordinates.

In jog-mode with active

"PTP-travel", the Cartesian operating range limit is monitored for overtravel and the relevant axis is braked beforehand. If "PTP travel" is not active, the axis can be traversed right up to the operating range limit.

### References

/FB2/ Description of Functions Extended Functions; Kinematic Transformation (M1)

## Syntax

G07

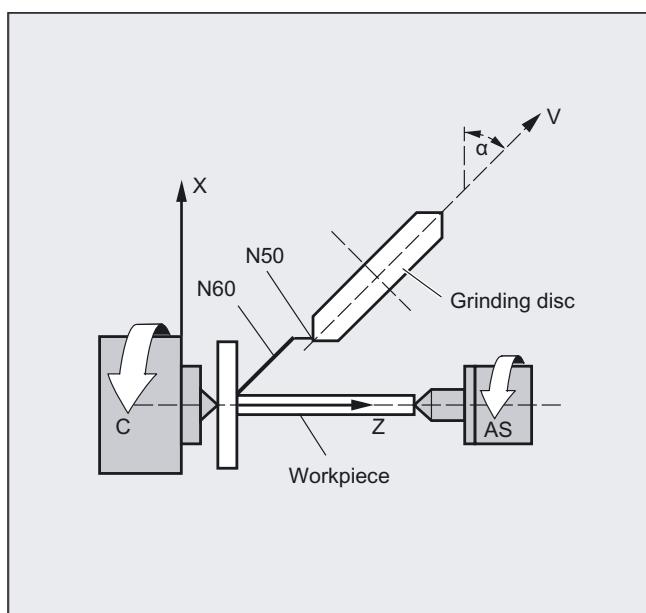
G05

The commands G07/G05 are used to make it easier to program the inclined axes. Positions can be programmed and displayed in the Cartesian coordinate system. Tool compensation and zero offset are included in Cartesian coordinates. After the angle for the inclined axis is programmed in the NC program, the starting position can be approached (G07) and then the oblique plunge-cutting (G05) performed.

## Significance

G07	Approach starting position
G05	Activates oblique plunge-cutting

## Example



<b>Programming</b>	<b>Comments</b>
N.. G18	; Program angle for inclined axis
N50 G07 X70 Z40 F4000	; Approach starting position
N60 G05 X70 F100	; Oblique plunge-cutting
N70 ...	;

## 6.8      Cartesian PTP travel

### Function

This function can be used to program a position in a cartesian coordinate system, however, the movement of the machine occurs in the machine coordinates. The function can be used, for example, when changing the position of the articulated joint, if the movement runs through a singularity.

---

#### Note

The function can only be used meaningfully in conjunction with an active transformation. Furthermore, "PTP travel" is only permissible in conjunction with G0 and G1.

---

### Syntax

```
N... TRAORI  
N... STAT='B10' TU='B100' PTP  
N... CP
```

#### PTP transversal with generic 5/6-axis transformation

If point-to-point transversal is activated in the machine coordinate system (ORIMKS) during an active generic 5/6-axis transformation with PTP, tool orientation can be programmed both with rotary axis positions

```
N... G1 X Y Z A B C
```

as well as with Euler and/or RPY angle vectors irrespective of the kinematics

```
N... ORIEULER or ORIRPY  
N... G1 X Y Z A2 B2 C2
```

or the direction vectors

```
N... G1 X Y Z A3 B3 C3
```

are programmed. Both rotary axis interpolation, vector interpolation with large circle interpolation ORIVECT or interpolation of the orientation vector on a peripheral surface of a taper ORICONxx may be active.

### **Non-uniqueness of orientation with vectors**

When programming the orientation with vectors, there is non-uniqueness in the rotary axis positions available. The rotary axis positions to be approached can be selected by programming STAT = <...>. If

If STAT = 0 is programmed (this is equivalent to the default setting), the positions which are at the shortest distance from the start positions are approached. If

STAT = 1 is programmed, the positions which are at a greater distance from the start positions are approached.

### **Significance**

The PTP and CP commands act in a modal manner. CP is the default setting.

If modal applies when programming the STAT value, TU programming is = <...> non-modal.

Another difference is that programming a STAT value only has an effect during vector interpolation, while programming TU is also evaluated during active rotary axis interpolation.

PTP      point to point (point to point motion)  
The movement is executed as a synchronized axis movement; the slowest axis involved in the movement is the dominating axis for the velocity.

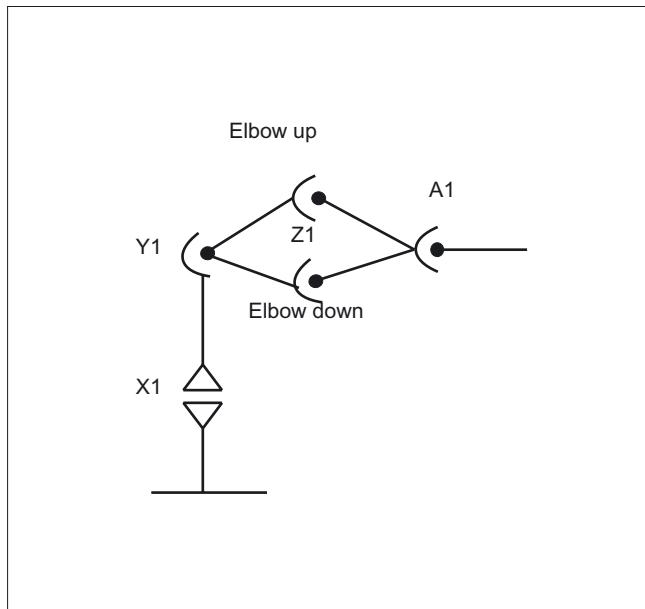
CP      continuous path (path motion)

The movement is executed as Cartesian path motion.

STAT=      Position of the articulated joints; this value is dependent on the transformation.

TU=      TURN information acts clockwise. This makes it possible to clearly approach axis angles between -360 degrees and +360 degrees.

## Example



N10 G0 X0 Y-30 Z60 A-30 F10000

Initial setting

→ 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

**PTP transversal with generic 5-axis transformation**

Assumption: This is based on a right-angled CA kinematics.

Program code	Comments
TRAORI	; Transformation CA kinematics on
PTP	; Activate PTP traversing
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, by programming STAT = 1, the rotary axes then travel the long route from their starting point (C=90, A=90) to the end point (C=270, A=-45), rather than the case would be if STAT = 0 where they would travel the shortest route to the end point (C=90, A=45).

**Description**

The commands PTP and CP effect the changeover between Cartesian traversal and traversing the machine axes.

**PTP transversal with generic 5/6-axis transformation**

During PTP transversal, unlike 5/6-axis transformation, the TCP generally does not remain stationary if only the orientation changes. The transformed end positions of all transformation axes (3 linear axes and up to 3 round axes) are approached in linear fashion without the transformation still actually being active.

The PTP transversal is deactivated by programming the modal G code CP.

The various transformations are included in the document:

/FB3/ Function Manual Special Functions; Handling Transformation Package (TE4).

### Programming the position (STAT=)

A machine position is not uniquely determined just by positional data with Cartesian coordinates and the orientation of the tool. Depending on the kinematics involved, there can be as many as eight different and crucial articulated joint positions. These are specific to the transformation. To be able to uniquely convert a Cartesian position into the axis angle, the position of the articulated joints must be specified with the command `STAT=`. The "STAT" command contains a bit for each of the possible positions as a binary value.

For information about the setting bits to be programmed for "STAT", see:  
/FB2/ Description of Functions Extended Functions; Kinematic Transformation (M1),  
"Cartesian PTP travel" section.

### Programming the axis angle (TU=)

To be able to clearly approach axis angles  $< \pm 360$  degrees, this information must be programmed using the command "`TU=`".

The axes traverse by the shortest path:

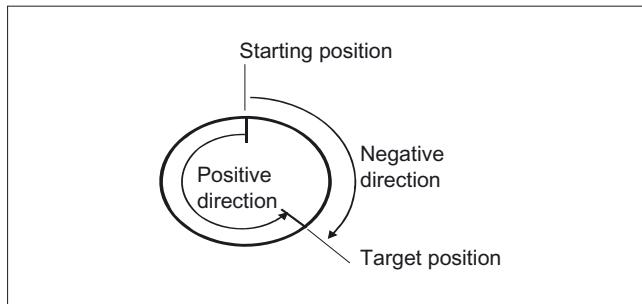
- when no `TU` is programmed for a position,
- with axes that have a traversing range  $> \pm 360$  degrees.

#### Example:

The target position shown in the diagram can be approached in the negative or positive direction. The direction is programmed under address A1.

`A1=225°, TU=Bit 0, → positive direction`

`A1=-135°, TU=Bit 1, → negative direction`



**Example of evaluation of TU for generic 5/6-axis transformation and target positions**

Variable **TU** contains a bit, which indicates the traversing direction for every axis involved in the transformation. The assignment of TU bits matches the channel axis view of the round axes. The TU information is only evaluated for the up to 3 possible round axes which are included in the transformation:

Bit0: Axis 1, TU bit = 0 : 0 degrees <= round axis angle < 360 degrees

Bit1: Axis 2, TU bit = 1 : -360 degrees < round axis angle < 0 degrees

The start position of a round axis is **C** = 0. By programming **C** = 270, the round axis travels to the following target positions:

**C** = 270: TU bit 0, positive direction of rotation

**C** = -90: TU bit 1, negative direction of rotation

**Further behavior**

**Mode change**

The "Cartesian PTP travel" function is only useful in the AUTO and MDA modes of operation. When changing the mode to JOG, the current setting is retained.

When the G code **PTP** is set, the axes will traverse in MCS. When the G code **CP** is set, the axes will traverse in WCS.

**Power On/RESET**

After a power ON or after a RESET, the setting is dependent on the machine data **\$MC\_GCODE\_REST\_VALUES[48]**. The default traversal mode setting is "**CP**".

**REPOS**

If the function "Cartesian PTP travel" was set during the interruption block, **PTP** can also be used for repositioning.

**Overlaid movements**

DRF offset or external zero offset are only possible to a limited extent in Cartesian PTP travel. When changing from PTP to CP movement, there must be no overrides in the BCS.

### **Smoothing between CP and PTP motion**

A programmable transition rounding between the blocks is possible with G641.

The size of the rounding area is the path in mm or inch, from which or to which the block transition is to be rounded. The size must be specified as follows:

- for G0 blocks with ADISPOS
- for all the other motion commands with ADIS.

The path calculation corresponds to considering of the F addresses for non-G0 blocks. The feed is kept to the axes specified in FGROUP ( . . ).

### **Feed calculation**

For CP blocks, the Cartesian axes of the basic coordinate system are used for the calculation.

For PTP blocks, the corresponding axes of the machine coordinate system are used for the calculation.

### 6.8.1 PTP for TRANSMIT

#### Function

PTP for TRANSMIT can be used to approach G0 and G1 blocks time-optimized. Rather than traversing the axes of the Basic Coordinate System linearly (CP), the machine axes are traversed linearly (PTP). The effect is that the machine axis motion near the pole causes the block end point to be reached much faster.

The part program is still written in the Cartesian workpiece coordinate system and all coordinate offsets, rotations and frame programming settings remain valid. The simulation on HMI, is also displayed in the Cartesian Workpiece coordinate system.

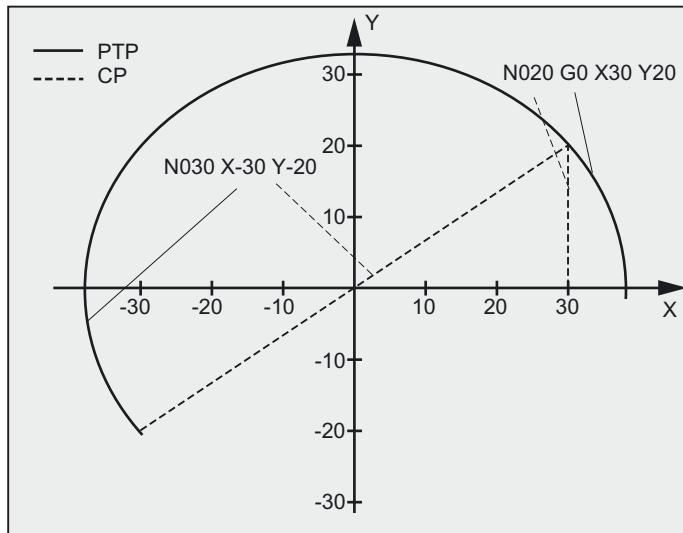
#### Syntax

```
N... TRANSMIT  
N... PTPG0  
N... G0 ...  
...  
N... G1 ...
```

#### Significance

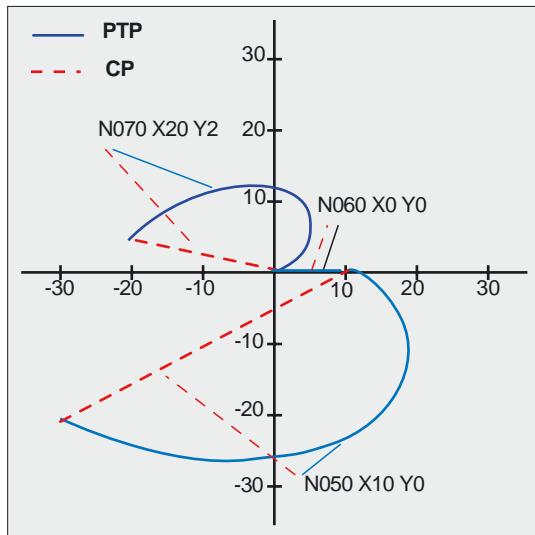
TRANSMIT	Activates the first declared TRANSMIT function (see section "Milling on turned parts: TRANSMIT")
PTPG0	<b>Point to Point G0</b> (point-to-point motion automatic at each G0 block and then set CP again)  Because STAT and TU are modal, the most recently programmed value always acts.
PTP	<b>point to point; (point to point motion)</b>  For TRANSMIT, PTP means that in the Cartesian spirals will be retracted to Archimedean spirals either about the pole or from the pole. The resulting tool motions run significantly different as for CP and are represented in the associated programming examples.
STAT=	Resolving the non-uniqueness with regard to the pole.
TU=	TU is not relevant for PTP with TRANSMIT

### Example of circumnavigation of the pole with PTP and TRANSMIT



Program code	Comments
N001 G0 X30 Z0 F10000 T1 D1 G90	; Initial setting, absolute dimension
N002 SPOS=0	
N003 TRANSMIT	; Transformation TRANSMIT
N010 PTPGO	; For each G0 block, automatically PTP followed by CP
N020 G0 X30 Y20	
N030 X-30 Y-20	
N120 G1 X30 Y20	
N110 X30 Y0	
M30	

**Example of the retraction from the pole with PTP and TRANSMIT**



<b>Programming</b>	<b>Comments</b>
N001 G0 X90 Z0 F10000 T1 D1 G90	; Initial setting
N002 SPOS=0	
N003 TRANSMIT	; Transformation TRANSMIT
N010 PTPG0	; For each G0 block, automatically PTP followed by CP
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	

## Description

### PTP and PTPG0

PTPG0 is considered for all transformations that can process PTP. PTPG0 is not relevant in all other cases.

G0 blocks are processed in CP mode.

The selection of PTP or PTPG0 is performed in the parts program or by the deselection of CP in the machine data \$MC\_GCODE\_RESET\_VALUES [48].

### CAUTION

#### Restrictions

With regard to tool motions and collision, a number of restrictions and certain function exclusions apply, such as:

no tool radius compensation (TRC) may be active with PTP.

With PTPG0, for active tool radius compensation (TRC), is traversed by CP.

PTP does not permit smooth approach and retraction (SAR).

With PTPG0, CP traversal is used for smooth approach and retraction (SAR).

PTP does not permit cutting cycles (CONTPRON, CONTDCON).

With PTPG0 cutting cycles (CONTPRON, CONTDCON) are traversed by CP.

Chamfer (CHF, CHR) and rounding (RND, RNDM) are ignored.

Compressor is not compatible with PTP and will automatically be deselected in PTP blocks.

An axis superimposing in the interpolation may not change during the PTP section.

If G643 is specified, an automatic switch to G642 is made after smoothing with axial accuracy.

For active PTP, the transformation axes cannot be simultaneously positioning axes.

#### References:

/FB2/ Function Manual Extended Functions; Kinematic Transformation (M1), "Cartesian PTP travel" section

---

## 6.9 Constraints when selecting a transformation

### PTP for TRACON:

PTP can also be used with TRACON, provided the first chained transformation supports PTP.

### Meaning of STAT= and TU= for TRANSMIT

If a rotary axis is to turn by 180 degrees or the contour for CP passes through the pole, rotary axes depending on the machine data \$MC\_TRANSMIT\_POLE\_SIDE\_FIX\_1/2 [48] can be turned by -/+ 180 degrees and traversed in clockwise or counter-clockwise direction. It can also be set whether traversal is to go through the pole or whether rotation around the pole is to be performed.

## 6.9 Constraints when selecting a transformation

### Function

Transformations can be selected via a parts 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).

## 6.10 Deselect transformation (TRAFOOF)

### Function

The TRAFOOF command disables all the active transformations and frames.

---

### Note

Frames required after this must be activated by renewed programming.

Please note:

The same restrictions as for selection are applicable to deselecting the transformation (see section "Constraints when selecting a transformation").

---

### Syntax

TRAFOOF

### Significance

Program code	Comments
TRAFOOF	; Disables all the active transformations/frames

## 6.11 Chained transformations (TRACON, TRAFOOF)

### Function

Two transformations can be chained so that the motion components for the axes from the first transformation are used as input data for the chained second transformation. The motion parts from the second transformation act on the machine axes.

The chain may include **two** transformations.

---

### Note

A tool is always assigned to the first transformation in a chain. The subsequent transformation then behaves as if the active tool length were zero. Only the basic tool lengths set in the machine data (`_BASE_TOOL_`) are valid for the first transformation in the chain.

---

### Machine manufacturer

Take note of information provided by the machine manufacturer on any transformations predefined by the machine data.

Transformations and chained transformations are options. The current catalog always provides information about the availability of specific transformations in the chain in specific controls.

### Applications

- Grinding contours that are programmed as a side line of a cylinder (TRACYL) using an inclined grinding wheel, e.g., tool grinding.
- Finish cutting of a contour that is not round and was generated with TRANSMIT using inclined grinding wheel.

### Syntax

TRACON(*trf,par*)

This activates a chained transformation.

TRAFOOF

## Significance

TRACON	This activates the chained transformation. If another transformation was previously activated, it is implicitly disabled by means of TRACON().
TRAFOOF	The most recently activated (chained) transformation will be disabled.
trf	Number of the chained transformation: 0 or 1 for first/single chained transformation. If nothing is programmed here, then this has the same meaning as specifying value 0 or 1, i.e., the first/single transformation is activated. 2 for the second chained transformation. (Values not equal to 0 - 2 generate an error alarm).
par	One or more parameters separated by a comma for the transformations in the chain expecting parameters, for example, the 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 trf. For example: TRACON( , 3.7).

## Requirements

The **second** transformation must be "**Inclined axis**" (**TRAANG**). The first transformation can be:

- Orientation transformations (**TRAORI**), including universal milling head
- **TRANSMIT**
- **TRACYL**
- **TRAANG**

It is a condition of using the activate command for a chained transformation that the individual transformations to be chained and the chained transformation to be activated are defined by the machine data.

The supplementary conditions and special cases indicated in the individual transformation descriptions are also applicable for use in chained transformations.

Information on configuring the machine data of the transformations can be found in:

/FB2/ Function Manual Extended Functions; Kinematic Transformations (M1) and

/FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformations (F2).



# Tool offsets

## 7.1 Offset memory

### Function

#### 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.

<sup>1</sup>Also applies to milling tools for 3D face milling

<sup>2</sup>Tool type for slotting saw

<sup>3</sup>Reserved: Is not used by SINUMERIK 840D

---

Tool parameter number (DP)	Meaning of system variables	Remarks
\$TC_DP1	Tool type	For overview see list
\$TC_DP2	Tool nose position	only for turning tools
<b>Geometry</b>	<b>Length compensation</b>	
\$TC_DP3	Length 1	Allocation to
\$TC_DP4	Length 2	Type and level
\$TC_DP5	Length 3	
<b>Geometry</b>	<b>Radius</b>	
\$TC_DP6 <sup>1</sup>	Radius 1 / length 1	Milling/turning/grinding tool
\$TC_DP6 <sup>2</sup>	diameter d	Slotting saw
\$TC_DP7 <sup>1</sup>	Length 2 / corner radius, tapered milling tool	Milling tools
\$TC_DP7 <sup>2</sup>	Slot width b corner radius	slotting saw
\$TC_DP8 <sup>1</sup>	Rounding radius 1 for milling tools	Milling tools
\$TC_DP8 <sup>2</sup>	projecting length k	slotting saw
\$TC_DP9 <sup>1..3</sup>	Rounding radius 2	Reserved
\$TC_DP10 <sup>1</sup>	Angle 1 face end of tool	Tapered milling tools
\$TC_DP11 <sup>1</sup>	Angle 2 tool longitudinal axis	Tapered milling tools
<b>Wear</b>	<b>Length and radius compensation</b>	
\$TC_DP12	Length 1	
\$TC_DP13	Length 2	
\$TC_DP14	Length 3	
\$TC_DP15 <sup>1</sup>	Radius 1 / length 1	Milling/turning/grinding tool
\$TC_DP15 <sup>2</sup>	diameter d	slotting saw
\$TC_DP16 <sup>1</sup>	Length 2 / corner radius, tapered milling tool,	Milling tools
\$TC_DP16 <sup>3</sup>	slot width b corner radius	slotting saw
\$TC_DP17 <sup>1</sup>	Rounding radius 1 for milling tools	Milling / 3D face milling
\$TC_DP17 <sup>2</sup>	projecting length k	slotting saw
\$TC_DP18 <sup>1..3</sup>	Rounding radius 2	Reserved
\$TC_DP19 <sup>1</sup>	Angle 1 face end of tool	Tapered milling tools
\$TC_DP20 <sup>1</sup>	Angle 2 tool longitudinal axis	Tapered milling tools
<b>Tool base dimension/adapter</b>	<b>Tool length offsets</b>	
\$TC_DP21	Length 1	
\$TC_DP22	Length 2	
\$TC_DP23	Length 3	
<b>Technology</b>		
\$TC_DP24	Clearance angle	only for turning tools
\$TC_DP25		Reserved

**Comments**

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)	Significance	Cutting Dn		Remarks
\$TC_DP1	Tool type			400 ... 599
\$TC_DP2	Tool nose position			
<b>Geometry</b>	<b>Length compensation</b>			
\$TC_DP3	Length 1			
\$TC_DP4	Length 2			
\$TC_DP5	Length 3			
<b>Geometry</b>	<b>Radius</b>			
\$TC_DP6	Radius			
<b>Geometry</b>	<b>Limit angle</b>			
\$TC_DP10	minimum limit angle			
\$TC_DP11	maximum limit angle			
<b>Wear</b>	<b>Length and radius compensation</b>			
\$TC_DP12	Wear length 1			
\$TC_DP13	Wear length 2			
\$TC_DP14	Wear length 3			
\$TC_DP15	Wear radius			
<b>Wear</b>	<b>Limit angle</b>			
\$TC_DP19	Wear min. limit angle			
\$TC_DP20	Wear max. limit angle			
<b>Tool base dimension/ adapter</b>	<b>Length compensations</b>			
\$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.

## **7.2 Additive offsets**

### **7.2.1 Selecting additive offsets (DL)**

#### **Function**

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 are addressed using a DL number (DL: Locationdependent; offsets regarding the location of use) and entered via the operator interface.

#### **Application**

Dimension errors caused by the location of use can be compensated using additive offsets.

#### **Syntax**

DL=<number>

## Significance

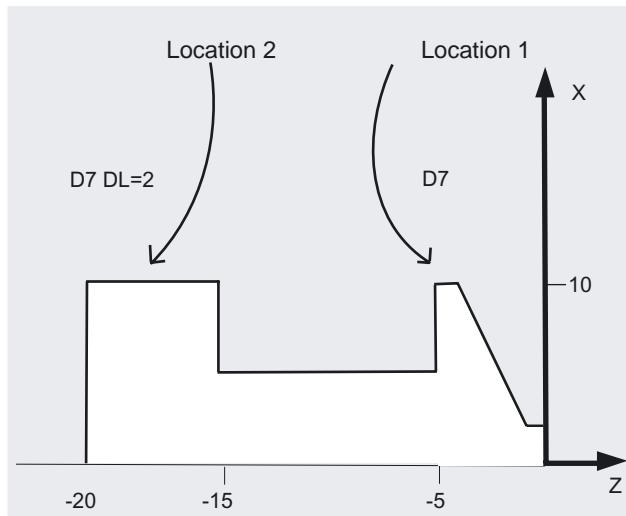
**DL** Command to activate an additive offset  
**<number>** The additive tool offset data to be activated is specified using the **<number>** parameter.

### 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 2 bearing seats:



Program code	Comments
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 position.
...	

### 7.2.2 Specify wear and setup values ( $\$TC\_SCPxy[t,d]$ , $\$TC\_ECPxy[t,d]$ )

#### Function

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

System variable	Significance
$\$TC\_SCPxy[<t>,<d>]$	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:  $\$TC\_DP3$  (length 1, with turning tools)

Wear values:  $\$TC\_SCP13$  to  $\$TC\_SCP63$

Setup values:  $\$TC\_ECP13$  to  $\$TC\_ECP63$

$\$TC\_SCP43 [<t>,<d>] = 1.0$

### 7.2.3 Delete additive offsets (DELDL)

#### Function

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>]
```

#### Significance

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: Significance: 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).

---

## 7.3 Special handling of tool offsets

### Function

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).

More information on summed offsets, refer to:

#### References:

Function Manual, Tool Management

### Setting data

Setting Data	Significance
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 tool nose 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 are 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 are 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 have 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.

---

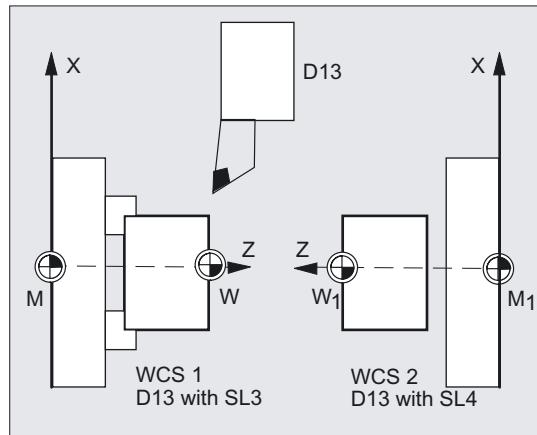
### Further application options

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.

### 7.3.1 Mirroring of tool lengths

#### Function

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 to 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.

### 7.3.2 Wear sign evaluation

#### Function

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 tool nose position (turning and grinding tools, tool types 400), then the sign evaluation of the wear components in the machining plane depends on the tool nose position. This setting data is of no significance for tool types without relevant tool nose position.

In the following table, the dimensions, whose sign is inverted using SD42920 (not equal to zero), are designed using an X:

Tool nose 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 e.g. the 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".

### 7.3.3 Coordinate system of the active machining operation (TOWSTD, TOWMCS, TOWWCS, TOWBCS, TOWTCS, TOWKCS)

#### Function

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

## Significance

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 of tool length at toolholder reference point (orientable toolholder)
TOKCS	Offsets of 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

### 7.3.4 Tool length and plane change

#### Function

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).

---

**7.3 Special handling of tool offsets**

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

<sup>\*)</sup> 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

<sup>\*)</sup> 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.

---

## 7.4 Language commands for tool management (T, NEWT, DELT, GETT, SETPIECE, GETSELT) (option)

### Function

The tool management can be used to change and update the tool data.

You can use predefined functions to perform the following tasks in the NC program:

- Create and fetch tools with names.
- Create a new tool or delete an existing tool.
- Assign a required T number to a tool with known name.
- Update piece number monitoring data.
- Read the T number of the tool preselected for the spindle.

### Syntax

```
T=<WZ>
<return parameter>=NEWT ("<WZ>", DUPLO_NR)
DELT ("WZ", DUPLO_NR)
<return parameter>=GETT ("<WZ>", DUPLO_NR)
SETPIECE(<x>,<y>)
GETSELT (<y>)
```

### Significance

T	Command to select tool
NEWT	Command to set-up a new tool With the NEWT function you can create a new tool with name in the NC program. The function automatically returns the T number created, which can subsequently be used to address the tool. Specification of the duplo number is optional. If none is specified, the duplo number is automatically generated in the tool manager.
DELT	Command to delete a tool The DELT function can be used to delete a tool without referring to the T number. Specification of the duplo number is optional.
GETT	Command to read the T number of the tool preselected for the spindle
SETPIECE	Command to update piece number monitoring data
GETSELT	Command to read the T number of the tool preselected for the spindle

## Tool offsets

### 7.4 Language commands for tool management (*T*, *NEWT*, *DELT*, *GETT*, *SETPIECE*, *GETSELT*) (option)

GETT	Command to assign a T number The GETT function returns the T number required to set the tool data for a tool known only by its name. If several tools with the specified name exist, the T number of the first possible tool is returned. Return value = -1: The tool name or duplo number cannot be assigned to a tool.
<WZ>	Tool identifier (e.g. "Drill", "123")
<DUPLO_NR>	Duplo number (unit quantity)
SETPIECE	Command to set the unit quantity This SETPIECE function is used to update the unit quantity monitoring data. The function counts all of the tool edges which have been changed since the last activation of SETPIECE for the stated spindle number.
GETSELT	Read preselected tool number (T No.) This GETSELT function returns the T number of the tool preselected for the spindle. This function allows access to the tool offset data before M6 and thus establishes main run synchronization slightly earlier.
<x>	Number of machined workpieces
<y>	Spindle number (data is optional) 0 stands for master spindle (default setting).

## Examples

### Example 1: NEWT function

Program code	Comments
DEF INT DUPLO_NO DEF INT T_NO DUPLO_NO=7 T_NO=NEWT("DRILL", DUPLO_NO)	; Create new tool "DRILL" with duplo number 7. The generated T number is saved in T_NO.

7.4 Language commands for tool management (*T, NEWT, DELT, GETT, SETPIECE, GETSEL*) (option)**Example 2: GETT function**

Program code	Comments
T="DRILL" R10=GETT("DRILL", DUPLO_NO)	; Determine T number for DRILL with duplo number = DUPLO_NO.

The "DRILL" must first be declared with NEWT or \$TC\_DP1 [ ].

Program code	Comments
\$TC_DP1[GETT("DRILL", DUPLO_NO),1]=100	; Write to a tool parameter (system variable) with tool name.

**Example 3: Tool change with tool manager**

- T1      Tool preselection, i.e. the tool magazine can be put in a tool position parallel to the machining.
- M6      Loading a preselected tool (depending on the default setting in the machine data, can also be programmed without M6).

Program code	Comments
T1 M6	; Load tool 1.
D1	; Select tool length compensation.
G1 X10 ...	; Machining with T1.
T="DRILL"	; Tool preselection, drill.
D2 Y20 ...	; Cutting edge change T1
X10 ...	; Machining with T1.
M6	; Load drill
SETPIECE(4)	; Number of workpieces that have been machined.
D1 G1 X10 ...	; Machining with drill.

**References**

For the complete list of all variables for tool management, see:

List Manual, system variables

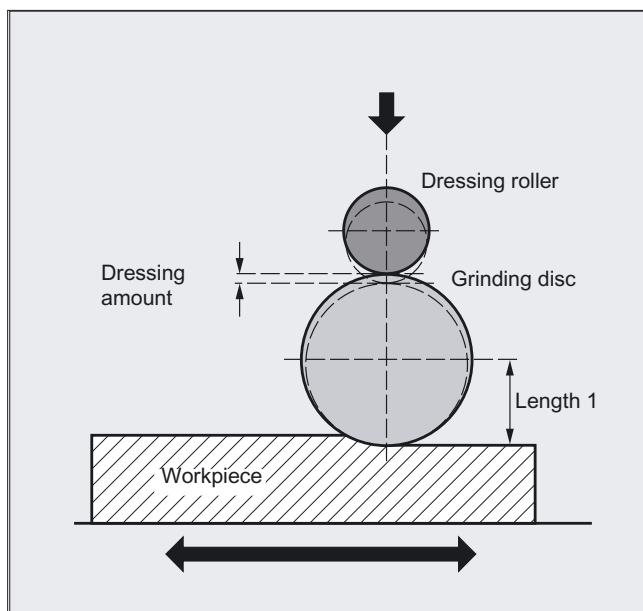
## Tool offsets

### 7.5 Online tool offset (PUTFTOCF, FCTDEF, PUTFTOC, FTOCON, FTOCOF)

## 7.5 Online tool offset (PUTFTOCF, FCTDEF, PUTFTOC, FTOCON, FTOCOF)

### Function

The function makes immediate allowance for tool offsets resulting from machining by means of online tool length offset (e.g., CD dressing: The grinding wheel is dressed parallel to machining). The tool length offset can be changed from the machining channel or a parallel channel (dresser channel).



### Note

Online tool offset can be applied only to grinding tools.

### Syntax

```
FCTDEF(Polynomial No.,LLimit,ULimit,a0,a1,a2,a3)
PUTFTOCF(polynomial No.,reference value,length1_2_3,channel,spindle)
PUTFTOC(value,length1_2_3,channel,spindle)
FTOCON
FTOCOF
```

## Significance

PUTFTOCF	Write online tool offset continuously
FCTDEF	Define parameters for PUTFTOCF function
PUTFTOC	Write online tool offset discretely
FTOCON	Activation of online tool offset
FTOCOF	Deactivation of online tool offset
Polynomial_No.	Values 1 to 3: up to 3 polynomials are possible at one time; polynomial up to 3rd order
Ref_value	Reference value from which the offset is derived
Length1_2_3	Wear parameter into which the tool offset value is added
Channel	Number of channel in which the tool offset is activated; specified only if the channel is different to the present one
Spindle	Number of the spindle on which the online tool offset acts; only needs to be specified for inactive grinding wheels
LLimit	Upper limit value
ULimit	Lower limit value
a0,a1,a2,a3	Coefficients of polynomial function
Value	Value added in the wear parameter

## Example

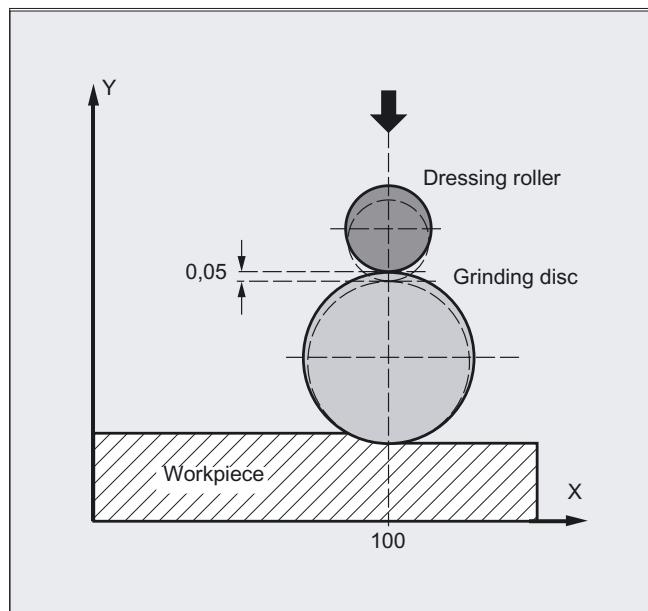
On a surface grinding machine with the following parameters, the grinding wheel is to be dressed by the amount 0.05 after the start of the grinding movement at X100. The dressing amount is to be active with write online offset continuously.

Y: Infeed axis for grinding wheel

V: Infeed axis for dressing roller

Machine: Channel 1 with axes X, Z, Y

Dressing: Channel 2 with axis V



**Machining program in channel 1:**

Program code	Comments
...	
N110 G1 G18 F10 G90	; Basic position.
N120 T1 D1	; Select current tool.
N130 S100 M3 X100	; Spindle on, traverse to starting position.
N140 INIT(2,"DRESS","S")	; Select the dressing program in channel 2.
N150 START(2)	; Start the dressing program in channel 2.
N160 X200	; Traverse to the target point.
N170 FTOCON	; Activate online offset.
N... G1 X100	; Additional machining.
N... M30	

**Dressing program in channel 2:**

Program code	Comments
...	
N40 FCTDEF(1,-1000,1000,-\$AA_IW[V],1)	; Define function: Straight
N50 PUTFTOCF(1,\$AA_IW[V],3,1)	; Continuously write online tool offset: Derived from the motion of the V axis ;the length 3 of the active grinding wheel is compensated in channel 1.
N60 V-0.05 G1 F0.01 G91	; Infeed motion for dressing, PUTFTOCF is only effective in this block.
...	
N... M30	

**Dressing program, modal:**

Program code	Comments
FCTDEF(1,-1000,1000,-\$AA_IW[V],1)	; Function definition.
ID=1 DO FTOC(1,\$AA_IW[V],3,1)	; Select online tool offset: Actual value of the V axis is the input value for polynomial 1. Result is added in channel 1 as compensation value to length 3 of the active grinding disk.
WAITM(1,1,2)	; Synchronization with machining channel.
G1 V-0.05 F0.01 G91	; Infeed motion for dressing.
G1 V-0.05 F0.02	
...	
CANCEL(1)	; Deselect online offset.
...	

**Description****General information about online TO**

Depending on the timing of the dressing process, the following functions are used to write the online tool offsets:

- Continuous write, non-modal: PUTFTOCF
- Continuous write, modal: ID=1 DO FTOC (see section synchronized actions)
- Discrete write: PUTFTOC

In the case of a continuous write (for each interpolation pulse) following activation of the evaluation function each change is calculated additively in the wear memory in order to prevent setpoint jumps. In both cases: The online tool offset can act on each spindle and lengths 1, 2 or 3 of the wear parameters.

The assignment of the lengths to the geometry axes is made with reference to the current plane.

The assignment of the spindle to the tool is made using the tool data for `GWPSON` or `TMON` provided it does not concern the active grinding wheel (see the "Fundamentals" programming manual). An offset is always applied for the wear parameters for the current tool side or for the left-hand tool side on inactive tools.

---

**Note**

Where the offset is identical for several tool sides, the values should be transferred automatically to the second tool side by means of a chaining rule (see Operator's Guide for description).

If online offsets are defined for a machining channel, you cannot change the wear values for the current tool on this channel from the machining program or by means of an operator action.

The online tool offset is also applied with respect to the constant grinding wheel peripheral speed (`GWPS`) in addition to tool monitoring (`TMON`).

---

**PUTFTOCF = Continuous write**

The dressing process is performed at the same time as machining: Dress across complete grinding wheel width with dresser roll or dresser diamond from one side of a grinding wheel to the other.

Machining and dressing can be performed on different channels. If no channel is programmed, the offset takes effect in the active channel.

`PUTFTOCF(Polynomial_No., Ref_value, Length1_2_3, Channel, Spindle)`

Tool offset is changed continuously on the machining channel according to a polynomial function of the first, second or third order, which must have been defined previously with `FCTDEF`. The offset, e.g. changing actual value, is derived from the "Reference value" variable. If a spindle number is not programmed, the offset applies to the active tool.

### Set parameters for FCTDEF function

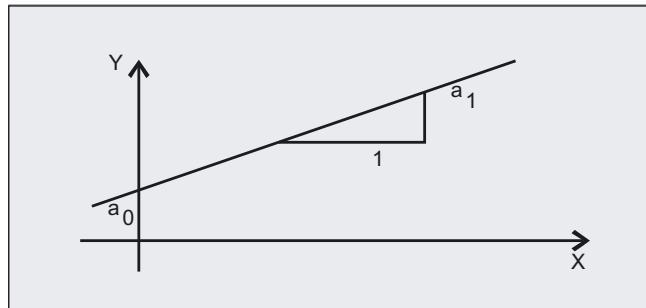
The parameters are defined in a separate block:

```
FCTDEF(Polynomial_no., LLimit, ULimit,a0,a1,a2,a3)
```

The polynomial can be a 1st, 2nd or 3rd order polynomial. The limit identifies the limit values (LLimit = lower limit, ULimit = upper limit).

Example: Straight line ( $y = a_0 + a_1x$ ) with gradient 1

```
FCTDEF(1, -1000, 1000, -$AA_IW[X], 1)
```



### Write online offset discretely: PUTFTOC

This command can be used to write an offset value **once**. The offset is activated immediately on the target channel.

Application of PUTFTOC: The grinding wheel is dressed from a parallel channel, but not at the same time as machining.

```
PUTFTOC(Value, Length1_2_3, Channel, Spindle)
```

The online tool offset for the specified length 1, 2 or 3 is changed by the specified value, i.e. the value is added to the wear parameter.

### Include online tool offset: FTOCON, FTOCOF

The target channel can only receive online tool offsets when FTOCON is active.

- FTOCON must be written in the channel on which the offset is to be activated. With FTOCOF, the offset is no longer applied, however the complete value written with PUTFTOC is corrected in the tool edge-specific offset data.
- FTOCOF is always the reset setting.
- PUTFTOCF always acts non-modally, i.e. in the subsequent traversing block.
- The online tool offset can also be selected modally with FTOC. Please refer to Section "Motion-synchronized actions" for more information.

## 7.6 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

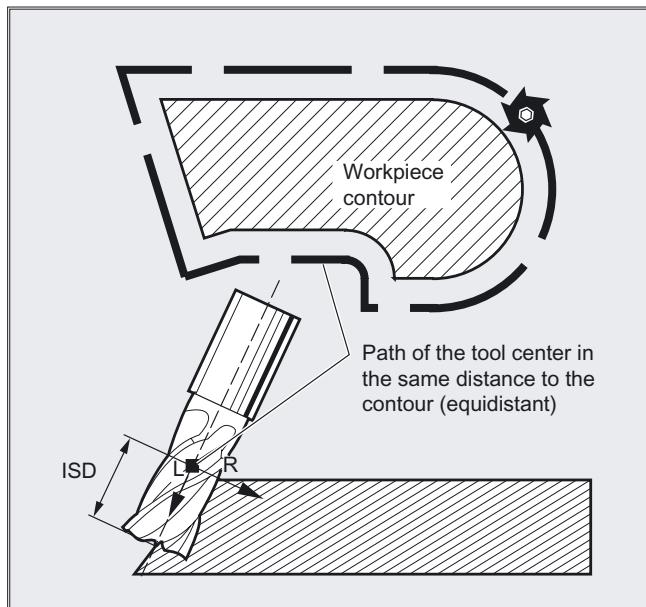
### 7.6.1 Activating 3D tool offsets (CUT3DC, CUT3DF, CUT3DFS, CUT3DFF, ISD)

#### Function

Tool orientation change is taken into account in tool radius compensation for cylindrical tools.

The same programming commands apply to 3D tool radius compensation as to 2D tool radius compensation. The left/right offset is specified in the direction of motion using G41/G42. The approach response is always controlled with NORM. The 3D radius compensation is only effective when 5-axis transformation is selected.

3D tool radius compensation is also called 5D tool radius compensation, because in this case 5 degrees of freedom are available for the orientation of the tool in space.



#### Difference between 2 1/2 D and 3D tool radius compensation

In 3D tool radius compensation tool orientation can be changed. With the 2 1/2D tool radius compensation, it is assumed that only a tool with constant orientation is being used.

## Syntax

```
CUT3DC
CUT3DFS
CUT3DFF
CUT3DF
ISD=<value>
```

## Significance

CUT3DC	Activation of 3D radius offset for circumferential milling
CUT3DFS	3D 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	3D 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 modally effective and written 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.

## Tool offsets

### 7.6 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

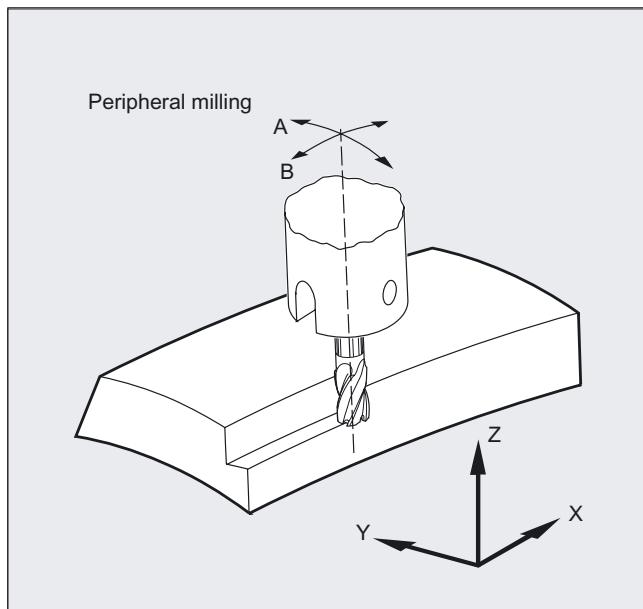
#### Example

Program code	Comments
N10 A0 B0 X0 Y0 Z0 F5000	
N20 T1 D1	; Tool call, call tool offset values.
N30 TRAORI(1)	; Transformation selection
N40 CUT3DC	; 3D tool radius compensation selection
N50 G42 X10 Y10	; Tool radius compensation selection
N60 X60	
N70 ...	

## 7.6.2 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.

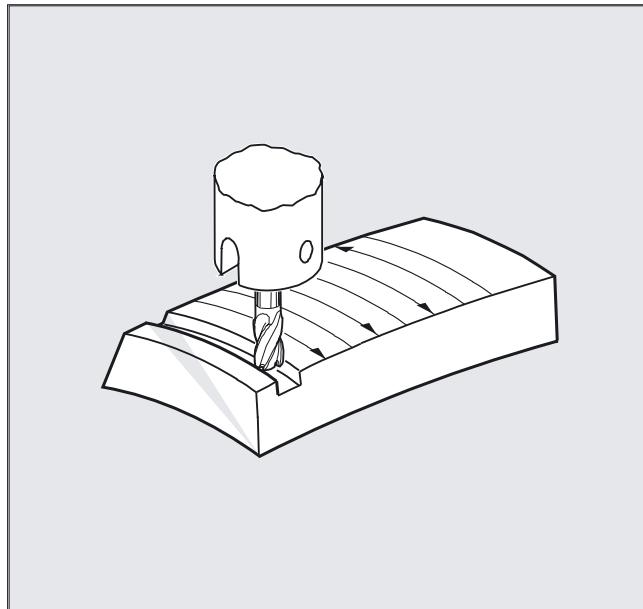


#### Note

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.

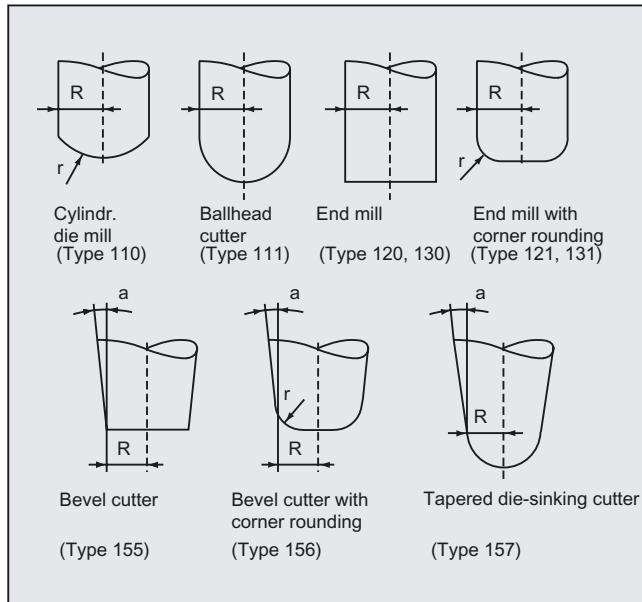
## Tool offsets

### 7.6 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

#### 7.6.3 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 tool shaft shape is not taken into account – tool types 120 and 156 have identical effects.



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 mill	110	>0	X	X
Ball end mill	111	>0	>R	X
End mill, angle head cutter	120, 130	>0	X	X
End mill, angle head cutter with corner rounding	121, 131	>r	>0	X
Bevel cutter	155	>0	X	>0
Bevel cutter with corner rounding	156	>0	>0	>0
Tapered die-sinking cutter	157	>0	X	>0

R = shaft radius (tool radius)  
 r = corner radius  
 a = angle between the tool longitudinal axis and upper end of the torus surface  
 X = 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

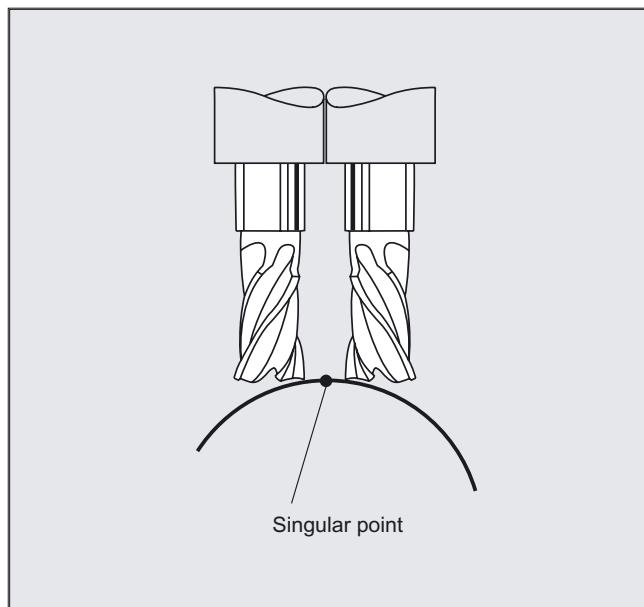
A new tool with modified dimensions (R, r, a) or a different shaft may only be specified with the programming of G41 or G42 (transition G40 to G41 or G42, reprogramming of G41 or G42). This rule does not apply to any other tool data, e.g., tool lengths, so that tools can be loaded without reprogramming G41 or G42.

#### 7.6.4 3D tool offset Compensation on the path, path curvature, insertion depth (*CUT3DC*, *ISD*)

##### Function

###### Compensation on 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. The system outputs an alarm if the limit values stored in the machine data are violated.

###### 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.

Program 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.

### Syntax

3D tool compensation circumference milling

`CUT3DC`

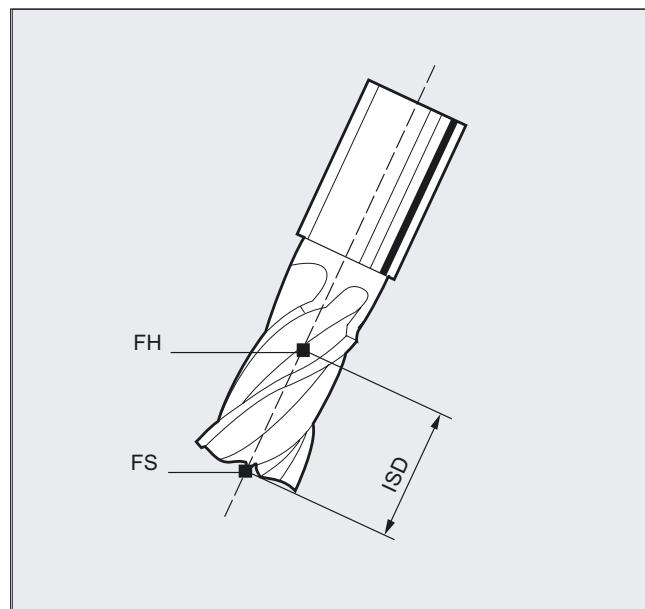
`ISD=<value>`

### Significance

<code>CUT3DC</code>	Activate 3D tool offset for circumferential milling, e.g., for pocket milling with oblique side walls.
<code>ISD</code>	The clearance ( <code>&lt;value&gt;</code> ) between the milling tool tip (FS) and the milling tool construction point (FH) are specified using the command <code>ISD</code> .

### Milling tool reference point

The milling tool reference point (FH) is obtained by projecting the programmed machining point onto the tool axis.



## *Tool offsets*

### 7.6 Activate 3D tool offsets (*CUT3DC...*, *CUT3DF...*)

#### 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".

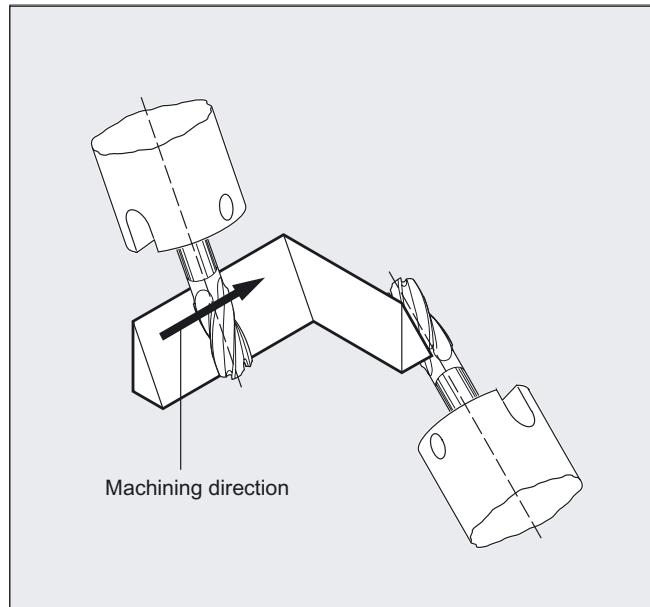
### 7.6.5 3D tool offset Inside/outside corners and intersection procedure (G450/G451)

#### Function

##### **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

**Significance**

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)

## 7.6.6 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.

## 7.6.7 3D tool offset Taking into consideration a limitation surface (CUT3DCC, CUT3DCCD)

### Function

#### 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

### Significance

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.

---

## Tool offsets

### 7.6 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

#### 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

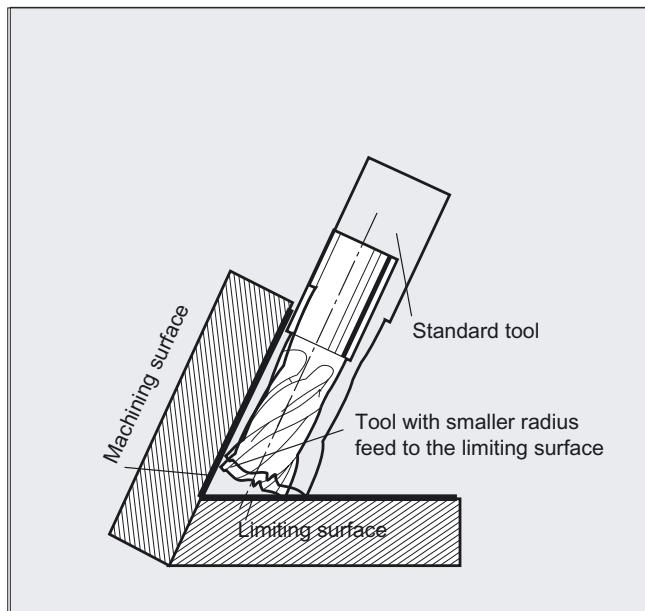
Tool dimensions of a toroidal miller with reduced radius as compared with the standard tool.

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.

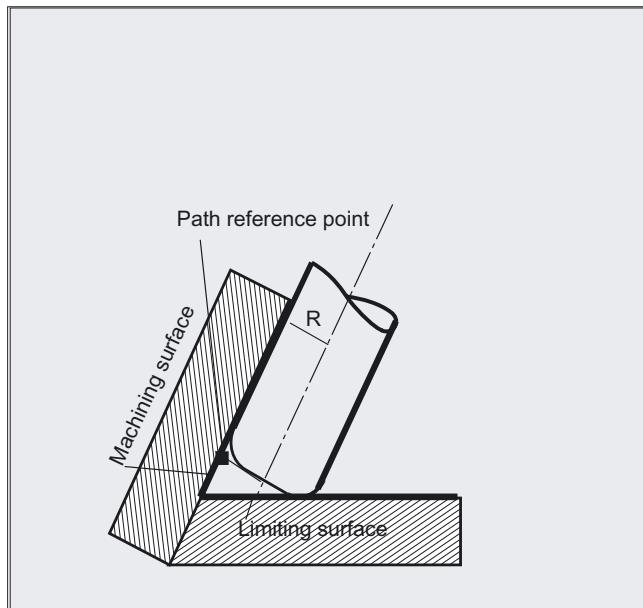
#### Using cylindrical tools

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.

## 7.6 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

### 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.

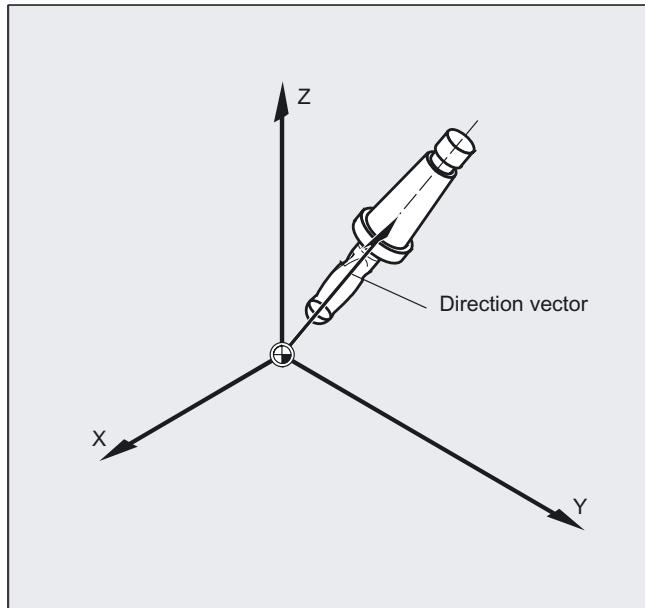
The position of the limitation surface is determined from the difference between these two values:

- Dimensions of the standard tool
- Tool radius (tool parameter `$TC_DP6`)

## 7.7 Tool orientation (ORIC, ORID, OSOF, OSC, OSS, OSSE, OSD, OST)

### Function

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.

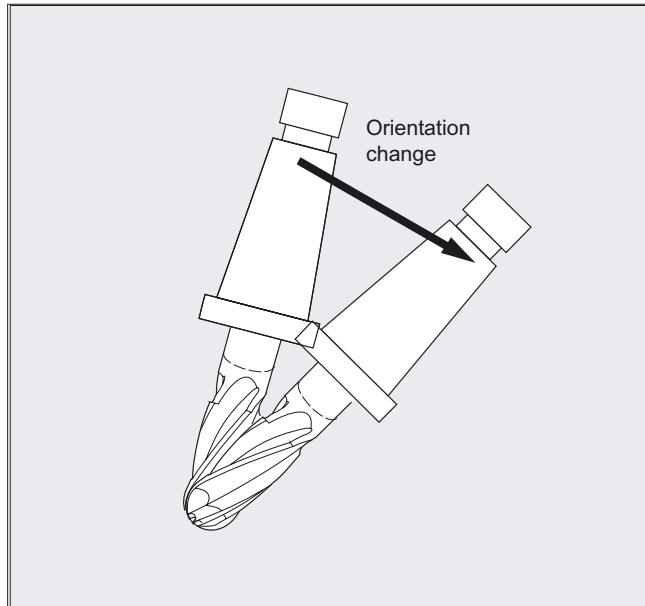
### Programming

#### 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).



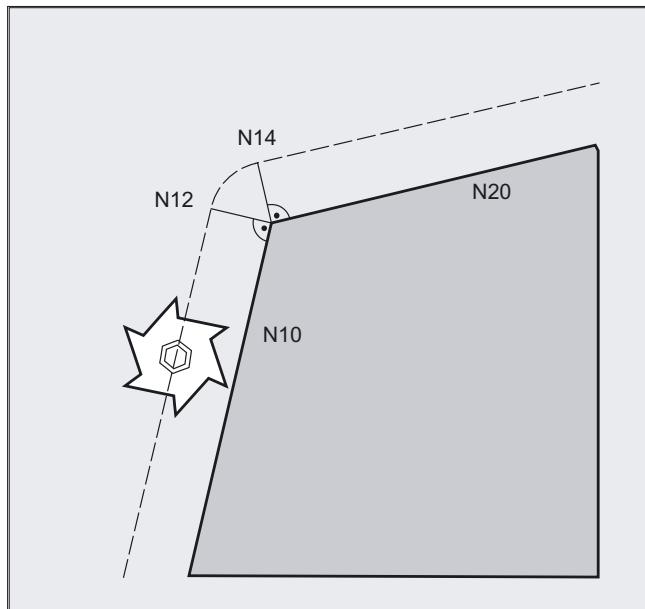
**Programming tool orientation:**

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	Speed of the orientation change for activated orientation smoothing in degrees per mm; applies to OSS and OSSE
OSD	Blending the orientation by specifying the blending length using setting data SD42674 \$SC_ORI_SMOOTH_DIST.
OST	Blending orientation by specifying the angular tolerance in degrees for vector interpolation using 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.

## Examples

### Example 1: ORIC

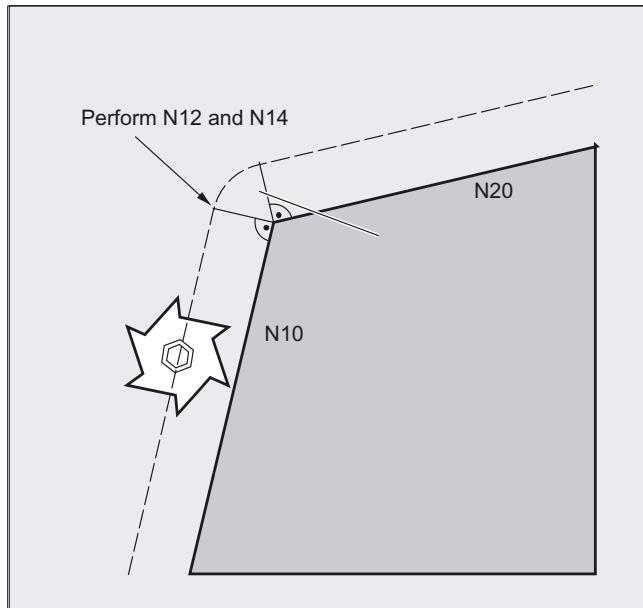
If ORIC is active and there are two or more blocks with changes in orientation (e.g. A2=B2=C2=) programmed between traversing blocks N10 and N20, then the inserted circle block is distributed among these intermediate blocks according to the absolute changes in angle.



Program code	Comments
ORIC	
N8 A2=... B2=... C2=...	
N10 X... Y... Z...	
N12 C2=... B2=...	
N14 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.
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.



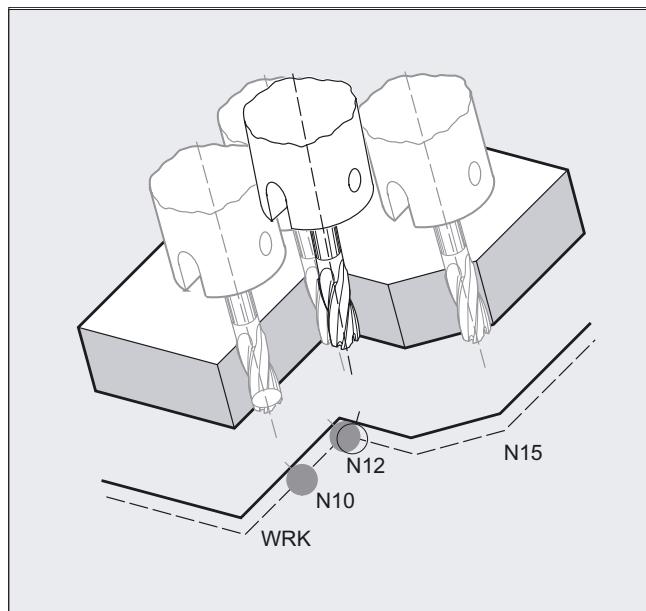
Program code	Comments
ORID	
N8 A2=... B2=... C2=...	
N10 X... Y... Z...	
N12 A2=... B2=... C2=...	; The N12 and N14 blocks are executed at the end of N10. The circle block is then executed with the actual orientation.
N14 M20	; Help functions, etc.
N20 X... Y... Z...	

**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: Changing the orientation at an inner corner**



**Program code**

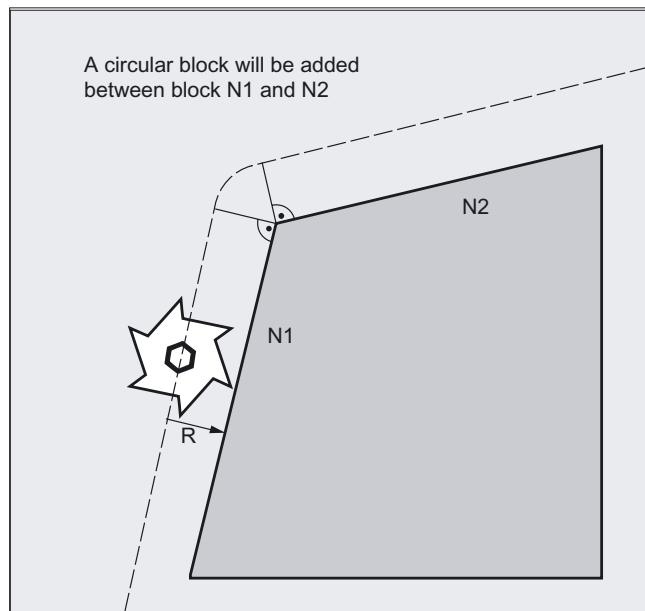
```
ORIC  
N10 X ...Y... Z... G1 F500  
N12 X ...Y... Z... A2=... B2=..., C2=...  
N15 X Y Z A2 B2 C2
```

## 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.

### Rounding orientation with OSD and 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 blending motion and/or angular tolerance to strongly smooth the orientation characteristic without having to accept higher contour deviations.

By activating OSD and OST, very small variances to the orientation characteristics can be smoothed with a specified rounding length and angle tolerance without serious "large" contour variances.

---

#### 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.

---

## 7.8 Free assignment of D numbers, cutting edge numbers

### 7.8.1 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)

## 7.8.2 Free assignment of D numbers: Checking D numbers (CHKDNO)

### Function

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)
```

### Significance

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.

## 7.8.3 Free assignment of D numbers: Rename D numbers (GETDNO, SETDNO)

### Function

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)
```

**Significance**

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	Comments
\$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 are addressed via D number 17; both via the system variables and in the programming with the NC address.

### 7.8.4 Free assignment of D numbers: Determine T number to the specified D number (GETACTTD)

#### Function

You determine the T number associated with an absolute D number using the GETACTTD command. There is not 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)
```

#### Significance

Dno	D number for which the T number shall be searched.	
Tno	T number found	
status	Value:	Significance:
	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.

### 7.8.5 Free assignment of D numbers: Invalidate D numbers (DZERO)

#### Function

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**

#### Significance

**DZERO** Marks all D numbers of the TO unit as invalid.

## 7.9 Tool holder kinematics

#### Prerequisites

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:

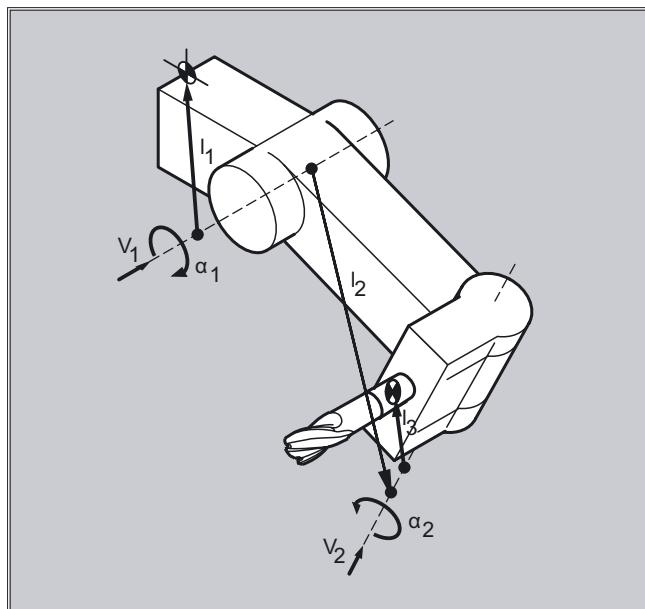
- the tool longitudinal axis must be perpendicular to the first rotary axis  $v_1$ .

#### 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 rotational angles  $\alpha_1$ ,  $\alpha_2$  at 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]
$\alpha_1$ angle of rotation $\alpha_2$ 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			
Name	x component	y component	z component
I <sub>4</sub> offset vector	\$TC_CARR18[m]	\$TC_CARR19[m]	\$TC_CARR20[m]
<b>Axis identifier</b> rotary axis v <sub>1</sub> rotary axis v <sub>2</sub>	Axis identifier of the rotary axes v <sub>1</sub> and v <sub>2</sub> (initialized with zero) \$TC_CARR21[m] \$TC_CARR22[m]		
<b>Kinematic type</b> Tool Part Mixed mode	\$TC_CARR23[m]		
	Kinematics type T ->	Kinematics type P ->	Kinematics type M
	Only the tool can rotate	Only the part can rotate	Part and tool can rotate (default).
<b>Offset</b> of the rotary axis v <sub>1</sub> rotary axis v <sub>2</sub>	Angle in degrees of the rotary axes v <sub>1</sub> and v <sub>2</sub> on assuming the initial setting \$TC_CARR24[m] \$TC_CARR25[m]		
<b>Angle offset</b> of the rotary axis v <sub>1</sub> rotary axis v <sub>2</sub>	Offset of the Hirth tooth system in degrees for rotary axes v <sub>1</sub> and v <sub>2</sub> \$TC_CARR26[m] \$TC_CARR27[m]		
<b>Angle increment</b> v <sub>1</sub> rotary axis v <sub>2</sub> rotary axis	Offset of the Hirth tooth system in degrees for rotary axes v <sub>1</sub> and v <sub>2</sub> \$TC_CARR28[m] \$TC_CARR29[m]		
<b>Min. position</b> rotary axis v <sub>1</sub> rotary axis v <sub>2</sub>	Software limit for the minimum position of the rotary axes v <sub>1</sub> and v <sub>2</sub> \$TC_CARR30[m] \$TC_CARR31[m]		
<b>Max. position</b> rotary axis v <sub>1</sub> rotary axis v <sub>2</sub>	Software limits for the maximum position of the rotary axes v <sub>1</sub> and v <sub>2</sub> \$TC_CARR32[m] \$TC_CARR33[m]		
<b>Toolholder name</b>	A toolholder can be given a name instead of a number. \$TC_CARR34[m]		
<b>User:</b> Axis name 1 Axis name 2 Identifier <b>Position</b>	Intended use in user measuring cycles \$TC_CARR35[m] \$TC_CARR36[m] \$TC_CARR37[m]		
	\$TC_CARR38[m]      \$TC_CARR39[m]      \$TC_CARR40[m]		
<b>Fine offset</b>	Parameters that can be added to the values in the basic parameters.		
I <sub>1</sub> Offset vector	\$TC_CARR41[m]	\$TC_CARR42[m]	\$TC_CARR43[m]
I <sub>2</sub> Offset vector	\$TC_CARR44[m]	\$TC_CARR45[m]	\$TC_CARR46[m]
I <sub>3</sub> Offset vector	\$TC_CARR55[m]	\$TC_CARR56[m]	\$TC_CARR57[m]
I <sub>4</sub> Offset vector	\$TC_CARR58[m]	\$TC_CARR59[m]	\$TC_CARR60[m]
v <sub>1</sub> rotary axis	\$TC_CARR64[m]		
v <sub>2</sub> 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/endpoints of the distance vectors on the axes can be freely selected. The rotation angles  $\alpha_1$ ,  $\alpha_2$  about the two axes are defined in the initial state of the toolholder by  $0^\circ$ . 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 compensations 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:

<b>Offset of rotary axes v1, v2</b>	Changing the position of the rotary axis v1 or v2 for the initial setting of the oriented toolholder.
<b>The angle offset/angle increment of the rotary axes v1, v2</b>	The offset or the angle increment of the Hirth tooth system of the rotary axes v1 and v2. Programmed or calculated angle is rounded up to the next value that results from $\phi = s + n * d$ when n is an integer.
<b>The minimum and maximum position of the rotary axes v1, v2</b>	The minimum and maximum position of the rotary axis limit angle (software limit) of the rotary axes v1 and v2.

**Parameters for the user**

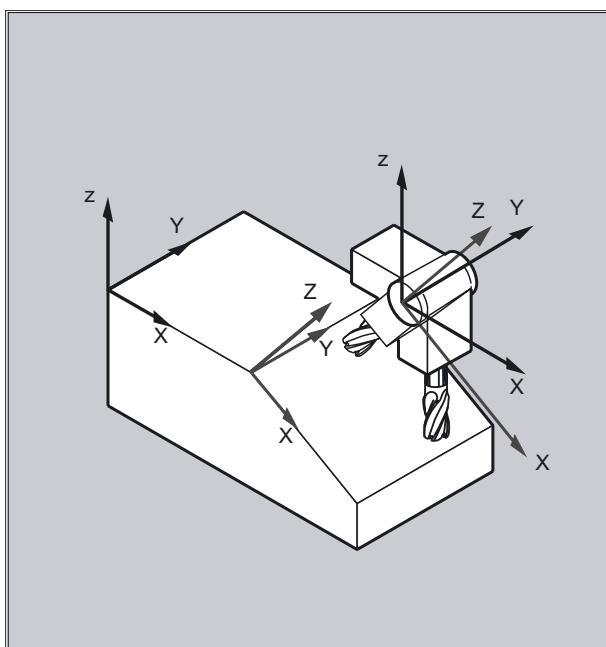
`$TC_CARR34` to `$TC_CARR40` contain parameters that are freely available to users and up to SW 6.4 were as standard, not further evaluated within the NCK or had no significance.  
?Check - S

**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	Comments
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 starting 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 upper or lower case 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.

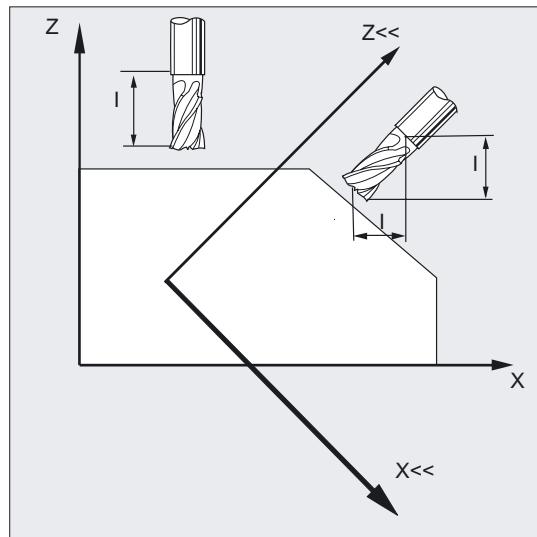
## Tool offsets

### 7.10 Tool length compensation for orientable toolholders (TCARR, TCOABS, TCOFR)

## 7.10 Tool length compensation for orientable toolholders (TCARR, TCOABS, TCOFR)

### Function

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

## Significance

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 compensation from the orientation of the toolholder (TCOABS)

TCOABS calculates the tool length compensation 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 458) ".

In order to make a new calculation of the tool length compensation 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 compensation 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 become 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 compensation (TCOABS) for a frame change**

In order to make a new calculation of the tool length compensation 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 compensation 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 compensation 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

---

## 7.11 Online tool length compensation (TOFFON, TOFFOF)

### Function

Use the system variable \$AA\_TOFF[ ] 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 the index. This defines the number of active directions of compensation by the geometry axes active at the same time.

All compensations can be active at the same time.

The online tool length compensation function can be used for:

- orientation transformation TRAORI
- orientable toolholder TCARR

### Machine manufacturer

Online tool length compensation 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

```
N.. TRAORI  
N.. TOFFON(X,25)  
N.. WHEN TRUE DO $AA_TOFF[tool direction] in synchronized actions
```

For more information about programming online tool length compensation in motion-synchronous actions, see "Actions in synchronized actions".

### Significance

TOFFON	<b>Tool Offset ON</b> (activate online tool length compensation) When activating, an offset value can be specified for the relevant direction of compensation and this is immediately recovered.
TOFFOF	<b>Tool Offset ON</b> (reset online tool length compensation) The relevant compensation values are reset and a preprocessing stop is initiated.
X, Y, Z	Direction of compensation for the offset value indicated for TOFFON

## Example of tool length compensation selection

Program code	Comments
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	; A TLC of 10 is interpolated for the Z tool direction
G4 F5	
...	
N100 XOFFSET = \$AA_TOFF_VAL[X]	; Assign actual compensation in the X direction
N120 TOFFON(X, -XOFFSET)	for the X tool direction, the tool length compensation is again reduced to 0
G4 F5	

## Example of tool length compensation deselection

Program code	Comments
N10 TRAORI(1)	; Transformation on
N20 TOFFON(X)	; Activating the Z tool direction
N30 WHEN TRUE DO \$AA_TOFF[X] = 10	; A TLC of 10 is interpolated for the X tool direction
G4 F5	
...	
N80 TOFFOF(X)	; Position offset of the X tool direction is deleted: ...\$AA_TOFF[X] = 0 No axis is traversed, the position offset is added to the actual position in WCS in accordance with the current orientation

## Description

### 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 set up.

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[ ].

### Adjusting machine data and setting data

The following machine data is available for online tool length offset:

- MD 20610: ADD\_MOVE\_ACCEL\_RESERVE acceleration margin for overlaid motion
- MD 21190: TOFF\_MODE: content of system variable \$AA\_TOFF[ ] is recovered or integrated as an absolute value
- MD 21194: TOFF\_VEL0 velocity of online tool length offset.
- MD 21196: TOFF\_ACCEL acceleration of online tool length offset.
- Setting data for presetting limit values  
SD 42970: TOFF\_LIMIT upper limit of tool length offset value.

**References:** /FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformations (F2).

## 7.12 Cutting data modification for tools that can be rotated (CUTMOD)

### Function

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.

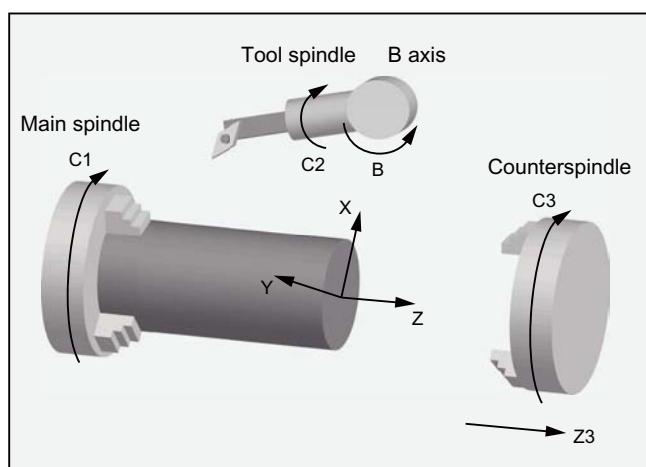


Figure 7-1 Rotatable tool for a lathe

The actual tool rotation is always determined from a currently active toolholder that can be orientated (refer to " Tool length compensation for toolholders that can be orientated (Page 464) ").

The function is activated using the command `_movePathCircular()`.

### Syntax

`CUTMOD=<value>`

### Significance

<code>CUTMOD</code>	Command to switch-in the function "cutting data modification for tools that can be rotated"
<code>&lt;value&gt;</code>	The following values can be assigned to the <code>CUTMOD</code> command:
0	The function is deactivated. The values supplied from system variables <code>\$P_AD...</code> are the same as the corresponding tool parameters.

---

### 7.12 Cutting data modification for tools that can be rotated (CUTMOD)

> 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 that 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 function can be activated using 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 tool nose 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	Comments
N10 \$TC_DP1[1,1]=500	
N20 \$TC_DP2[1,1]=3	; Tool nose 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	

---

## 7.12 Cutting data modification for tools that can be rotated (CUTMOD)

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 tool nose position is not modified due to CUTMOD=0, the old tool nose reference point is decisive just as before. This is the reason that 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 tool nose 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 tool nose 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 tool nose positions only become effective again in the deselect blocks N260 and/or N300.

### Further Information

#### Effectiveness of the modified cutting data

The modified tool nose position and the modified tool nose 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 tool nose 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), a valid value not equal to zero (plus or minus 17, 18 or 19), then its contents define the plane in which the relevant quantities are evaluated.

### System variables

The following system variables are available:

System variables	Significance
\$P_CUTMOD_ANG / \$AC_CUTMOD_ANG	<p>Supplies the (non-rounded) angle in the active machining plane, that was used as basis for the modification of the cutting data (tool nose position, cut direction, clearance angle and holder angle) for the functions activated using CUTMOD and/or \$SC_CUTDIRMOD.</p> <p>\$P_CUTMOD_ANG refers to the actual state in the preprocessing, \$AC_CUTMOD_ANG to the actual main run block.</p>
\$P_CUTMOD / \$AC_CUTMOD	<p>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).</p> <p>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 the number of the active toolholder that can be orientated at the time of programming.</p> <p>\$P_CUTMOD refers to the actual state in the preprocessing, \$AC_CUTMOD to the actual main run block.</p>
\$P_CUT_INV / \$AC_CUT_INV	<p>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:</p> <ol style="list-style-type: none"> <li>1. If a turning or grinding tool is active (tool types 400 to 599 and / or SD42950 \$SC_TOOL_LENGTH_TYPE = 2).</li> <li>2. The cutting influence was activated using the language command CUTMOD.</li> <li>3. A toolholder that can be orientated is active, which was designated using the numerical value of CUTMOD. Check - I reformulated.</li> <li>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°).</li> </ol> <p>The contents of the variable is FALSE if at least one of the specified four conditions is not fulfilled. For tools whose tool nose position is not defined, the value of the variable is always FALSE.</p> <p>\$P_CUT_INV refers to the actual state in the preprocessing and \$AC_CUT_INV to the actual main run block.</p>

---

### 7.12 Cutting data modification for tools that can be rotated (CUTMOD)

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 are made available in the following system variables:

System variable	Significance
\$P_AD[2]	Tool nose position
\$P_AD[10]	Holder angle
\$P_AD[11]	Cut direction
\$P_AD[24]	Clearance angle

---

#### Note

The data are 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:

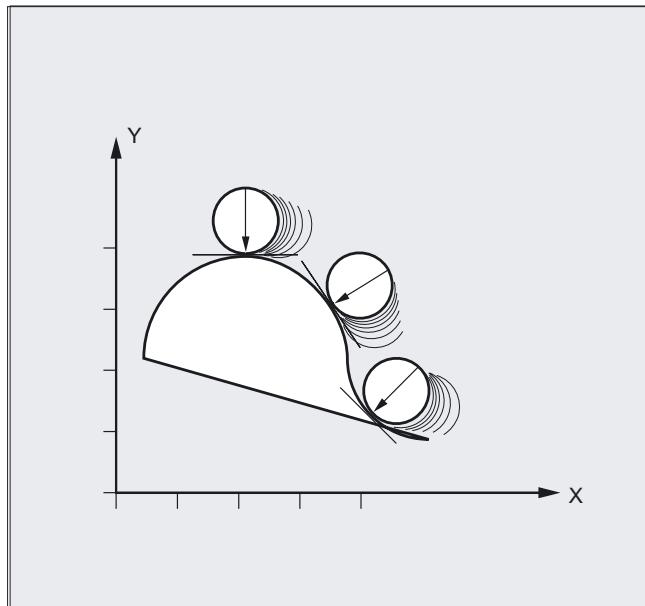
Function Manual Basic Functions; Tool Offset (W1)

## Path traversing behavior

### 8.1 Tangential control (TANG, TANGON, TANGOF, TLIFT, TANGDEL)

#### Function

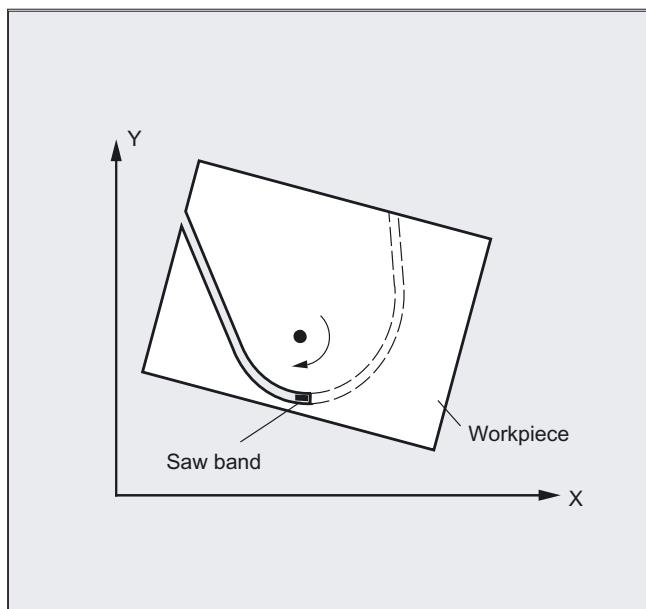
The following axis follows the path of the leading axis along the tangent. This allows alignment of the tool parallel to the contour. Using the angle programmed in the TANGON instruction, the tool can be positioned relative to the tangent.



### Applications

Tangential control can be used in applications such as:

- Tangential positioning of a rotatable tool during nibbling
- Follow-up of workpiece alignment for a bandsaw (s. illustration).
- Positioning of a dressing tool on a grinding wheel
- Positioning of a cutting wheel for glass or paper working
- Tangential feed of a wire for 5-axis welding.



### Syntax

```
TANG (Faxis,Laxis1,Laxis2,Coupling,CS,Opt)
TANGON (Faxis,Angle, Dist, Angletol)
TANGOF (Faxis)
TLIFT (Faxis)
TANGDEL (FAxis)
```

**Simplified programming:**

A coupling factor of 1 does not have to be programmed explicitly.

`TANG(C, X, Y, 1, "B", "P")` can be abbreviated to `TANG(C, X, Y, , , "P")`. As before, `TANG(C, X, Y, 1, "B", "S")` can be written as `TANG(C, X, Y)`.

The `TLIFT(...)` statement must be programmed immediately after the axis assignment with `TANG(...)`. Example:

```
TANG(C,X,Y...)
TLIFT(C)
```

**Deactivate TLIFT**

Repeat axis assignment `TANG(...)` without following it by `TLIFT(...)`.

**TANGDEL Delete definition of a tangential follow-up**

An existing user-defined tangential follow-up must be deleted if a new tangential follow-up with the same following axis is defined in the preparation call `TANG`. Deletion is only possible if the coupling with `TANGOF(Faxis)` is deactivated.

**Significance**

<code>TANG</code>	Preparatory statement for the definition of a tangential follow-up; default setting: 1  <b>TANG(C,X,Y,1,"B") means:</b> Rotary axis C follows the geometry axes X and switch-out Y.TLIFT
<code>TANGON</code>	Activate tangential control specifying following axis and required offset angle of the following axis and, if necessary, rounding path, angle deviation.  <b>TANGON(C,90) means:</b> The C axis is the following axis. On every movement of the path axes, it is rotated into a position at 90° to the path tangent.
<code>TANGOF</code>	Deactivate tangential control specifying following axis.  The following axis is specified in order to deactivate the tangential control: <b>TANGOF(C)</b>

---

**8.1 Tangential control (TANG, TANGON, TANGOF, TLIFT, TANGDEL)**

TLIFT	Insert intermediate block at contour corners
TANGDEL	Delete definition of a tangential follow-up. <b>Example: TANGDEL (FAxis)</b>
Faxis	Following axis: additional tangential following rotary axis.
Laxis1, Laxis2	Leading axes: path axes, which determine the tangent for the following axis.
Couple	Coupling factor: relationship between the angle change of the tangent and the following axis. Parameter optional; default: 1
C	Identifying letter for coordinate system "B" = Basic coordinate system; entry is optional; default setting "W" = Workpiece coordinate system is not available
Opt	Optimization: "S" Standard, Default "P" automatic adaptation of the characteristic over time of the tangential axis and the contour
Angle	Offset angle of following axis
Dist	Smoothing path of following axis, required with Opt "P"
Angletol	Angle tolerance of following axis, (optional), evaluation only with Opt= "P"

**Opt, Dist and Angletol optimization possibility**

Opt="P" specifies that the dynamic behavior of the following axis for the speed limitation of the leading axes and, in particular, is recommended when kinematic transformations are used.

The parameters (Dist and Angletol) limit the error between the following axis and the tangent of the leading axes precisely.

**Example: Plane change**

Program code	Comments
N10 TANG(A, X, Y, 1)	; 1. Defining the tangential tracking.
N20 TANGON(A)	; Activating the coupling.
N30 X10 Y20	; Radius
...	
N80 TANGOF(A)	; Deactivate the 1st coupling.
N90 TANGDEL(A)	; Delete the 1st definition.
...	
TANG(A, X, Z)	; 2. Defining the tangential tracking.
TANGON(A)	; Activating the new coupling
...	
N200 M30	

**Example of the geometry axis switching and TANGDEL**

No alarm is produced.

Program code	Comments
N10 GEOAX(2,Y1)	; Y1 is geometry axis 2.
N20 TANG(A, X, Y)	
N30 TANGON(A, 90)	
N40 G2 F8000 X0 Y0 I0 J50	
N50 TANGOF(A)	; Deactivating tracking with Y1.
N60 TANGDEL(A)	; Delete the 1st definition.
N70 GEOAX(2, Y2)	; Y2 is the new geometry axis 2
N80 TANG(A, X, Y)	; 2. Defining the tangential tracking.
N90 TANGON(A, 90)	; Activating tracking with the 2nd definition.
...	

**Example: Tangential tracking with automatic optimization**

Automatic optimization using Dist and angular tolerance.

<b>Program code</b>	<b>Comments</b>
N80 G0 C0	; Y1 is geometry axis 2.
N100 F=50000	
N110 G1 X1000 Y500	
N120 TRAORI	; Rounding with axial tolerance.
N130 G642	
N171 TRANS X-Y-	; Automatic optimization of the path velocity.
N180 TANG(C,X,Y, 1,,,"P")	; Rounding (blending) distance 5 mm
N190 TANGON(C, 0, 5.0, 2.0)	; Angular tolerance, 2 degrees
N210 G1 X1310 Y500	; Activating tracking with the 2nd definition.
N215 G1 X1420 Y500	
N220 G3 X1500 Y580 I=AC(1420) J=AC(580)	
N230 G1 X1500 Y760	
N240 G3 X1360 Y900 I=AC(1360) J=AC(760)	
N250 G1 X1000 Y900	
N280 TANGOF(C)	
N290 TRAFOOF	
N300 M02	

**Defining following axis and leading axis**

*TANG* is used to define the following and leading axes.

A coupling factor specifies the relationship between an angle change on the tangent and the following axis. Its value is generally 1 (default).

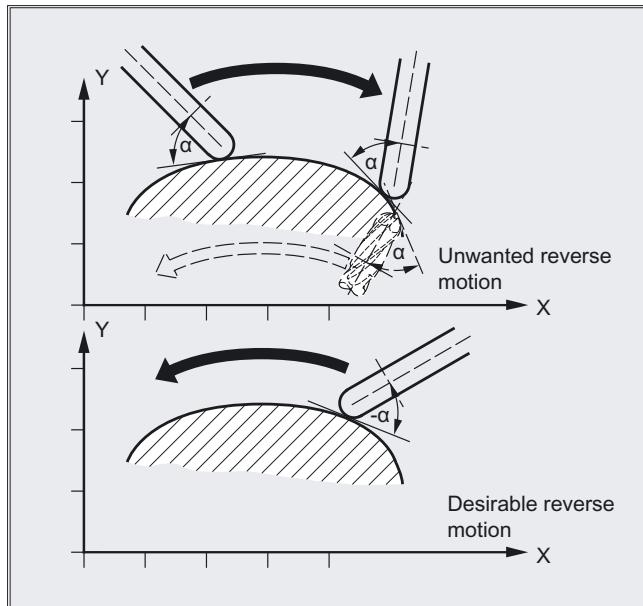
### Limit angle using the working area limitation

For path movements, which oscillate back and forth, the tangent jumps through 180° at the turning point on the path and the orientation of the following axis changes accordingly.

This behavior is generally inappropriate: The return movement should be traversed at the same negative offset angle as the approach movement.

To do this, limit the working area of the following axis (G25, G26). The working area limit must be active at the instant of path reversal (WALIMON).

If the offset angle lies outside the working area limit, an attempt is made to return to the permissible working area with the negative offset angle.



### Insert intermediate block at contour corners, TLIFT

At one corner of the contour the tangent changes and thus the setpoint position of the following axis. The axis normally tries to compensate this step change at its maximum possible velocity. However, this causes a deviation from the desired tangential position over a certain distance on the contour after the corner. If such a deviation is unacceptable for technological reasons, the instruction **TLIFT** can be used to force the control to stop at the corner and to turn the following axis to the new tangent direction in an automatically generated intermediate block.

## Path traversing behavior

### 8.1 Tangential control (TANG, TANGON, TANGOF, TLIFT, TANGDEL)

The path axis is used for turning if the following axis has been used once as the path axis. A maximum axis velocity of the following axis can be achieved with function `TFGREF[ax] = 0.001`.

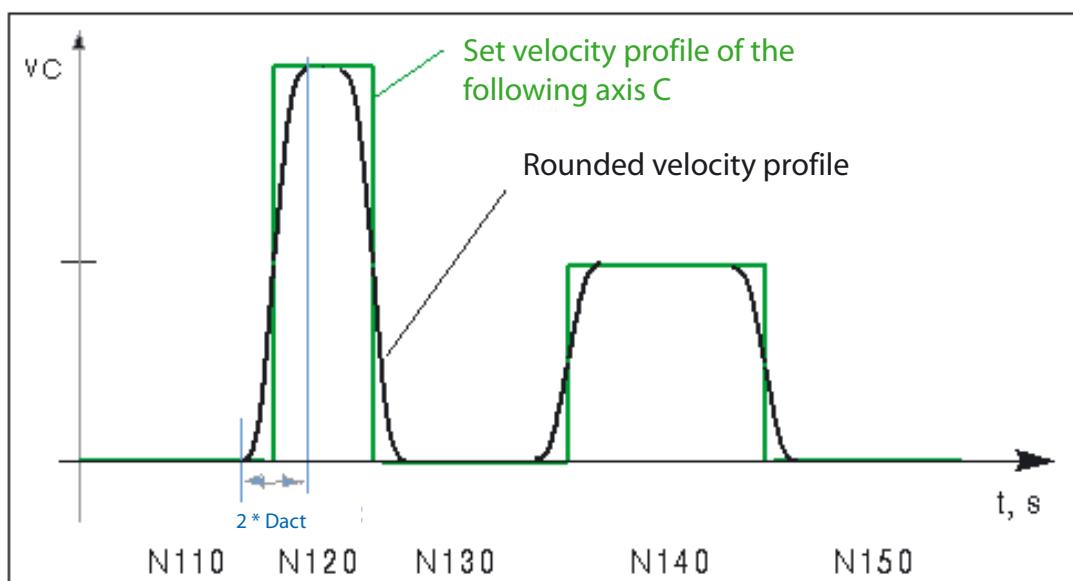
If the follow-up axis was not previously traversed as a path axis it is now traversed as a positioning axis. The velocity is then dependent on the positioning velocity in the machine data.

The axis is rotated at its maximum possible velocity.

## Optimization possibility

Velocity jumps of the following axis caused by jumps in the leading axis contour are rounded and smoothed with (Dist and Angletol).

The following axis is controlled with look-ahead (see diagram) to keep deviations as small as possible.



## Defining the angle change

The angular change limit at which an intermediate block is automatically inserted is defined via machine data `$MA_EPS_TLIFT_TANG_STEP`.

### **Effect on transformations**

The position of the rotary axis to which follow-up control is applied can act as the input value for a transformation.

### **Explicit positioning of the following axis**

If an axis, which is following your lead axes, is positioned explicitly the position is added to the programmed offset angle.

All path definitions are possible: Path and positioning axis movements.

### **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

1,2,3: Tangential follow-up active

## **8.2 Feedrate response (FNORM, FLIN, FCUB, FPO)**

### **Function**

To permit flexible definition of the feed characteristic, the feed programming according to DIN 66205 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 (... , ... , ... )

## Significance

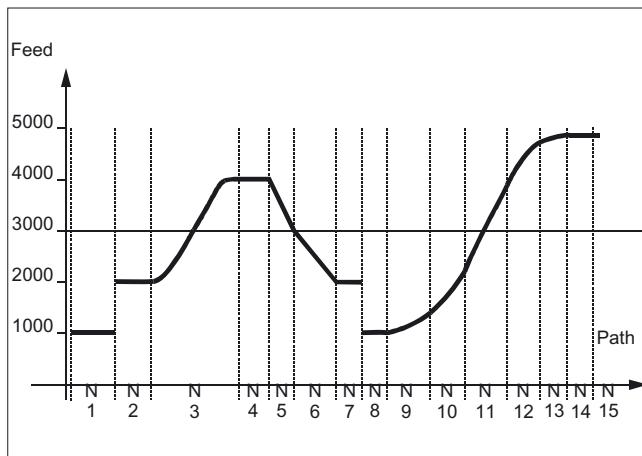
FNORM	Basic setting. The feed value is specified as a function of the traverse path of the block and is then valid as a modal value.
FLIN	<b>Path velocity profile linear:</b> The feed value is approached linearly via the traverse path from the current value at the block beginning to the block end and is then valid as a modal value. The response can be combined with G93 and G94.
FCUB	<b>Path velocity profile cubic:</b> 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...	<b>Polynomial path velocity profile:</b> 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.

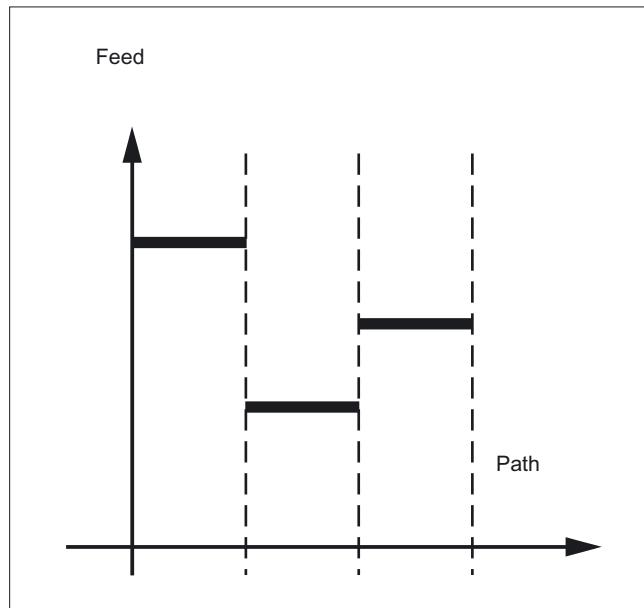


Program code	Comments
N1 F1000 FNORM G1 X8 G91 G64	; Constant feedrate profile, incremental dimension data
N2 F2000 X7	; Setpoint velocity step change
N3 F=FPO(4000, 6000, -4000)	; Feed profile via polynomial with feed 4000 at the end of the block
N4 X6	; Polynomial feedrate 4000 is valid as modal value
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	

## FNORM

The feed address F defines the path feed as a constant value according to DIN 66025.

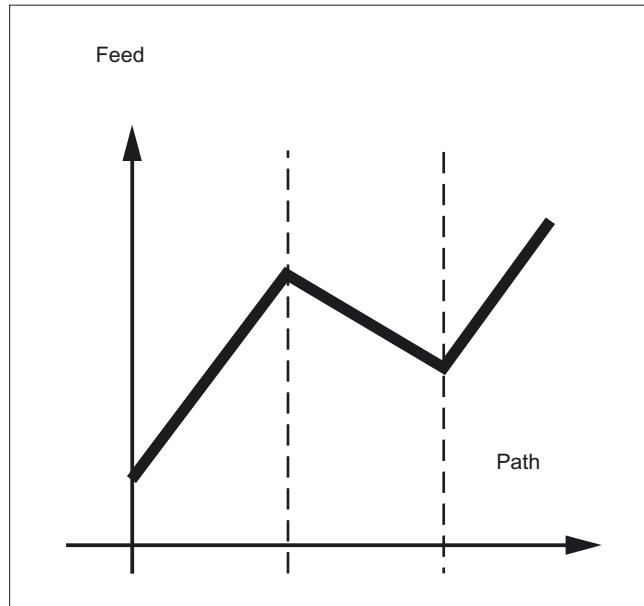
Please refer to Programming Manual "Fundamentals" for more detailed information on this subject.



## FLIN

The feed characteristic is approached linearly from the current feed value to the programmed F value until the end of the block.

Example: N30 F1400 FLIN X50

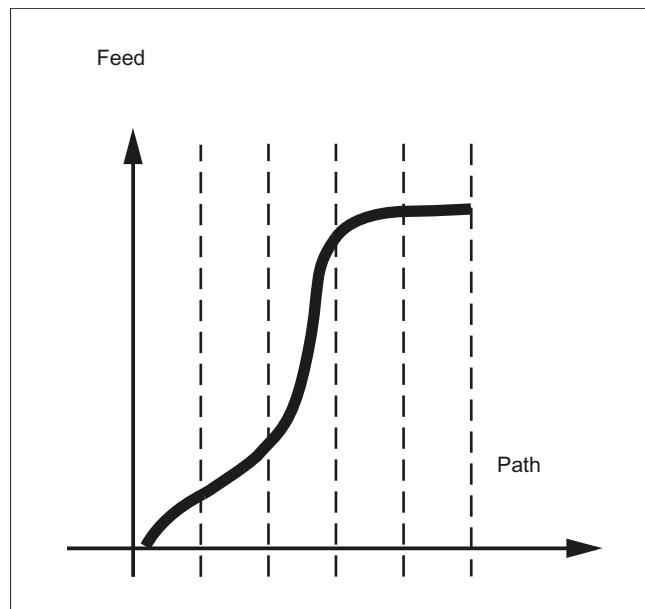


**FCUB**

The feed is approached according to a cubic characteristic from the current feed value to the programmed F value until the end of the block. The control uses splines to connect all the feed values programmed non-modally that have an active FCUB. The feed 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(...,...,...)**

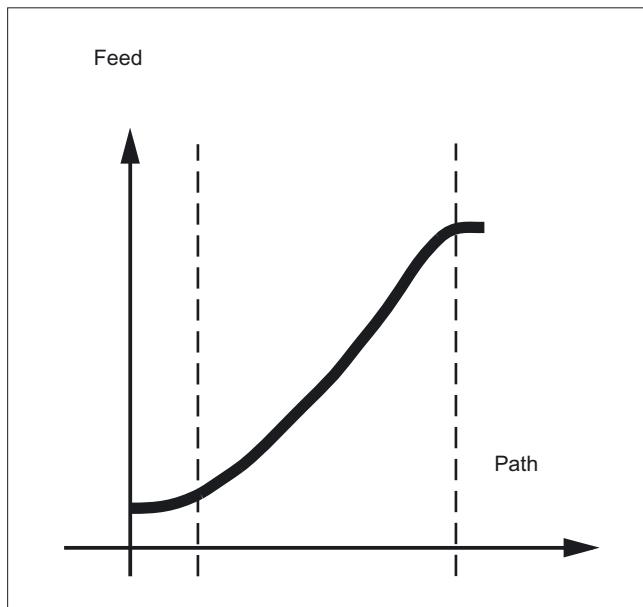
The feed characteristic is programmed directly via a polynomial. The polynomial coefficients are specified according to the same method used for polynomial interpolation.

**Example:**

```
F=FPO(endfeed, quadf, cubf)
endfeed, quadf and cubf are previously defined variables.
```

endfeed:	Feed 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.



## Restrictions

The functions for programming the path traversing characteristics apply regardless of the programmed feed characteristic.

The programmed feed characteristic is always absolute regardless of G90 or G91.

**Feed response FLIN and FCUB are active with**

G93 and G94.

**FLIN and FCUB is not active with**

G95, G96/G961 and G97/G971.

## Active compressor COMPON

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 feed spline deviates from the programmed end points by an amount not exceeding the value set in machine data \$MC\_COMPRESS\_VELO\_TOL.

**F=FPO(...,...,...)**

These blocks are not compressed.

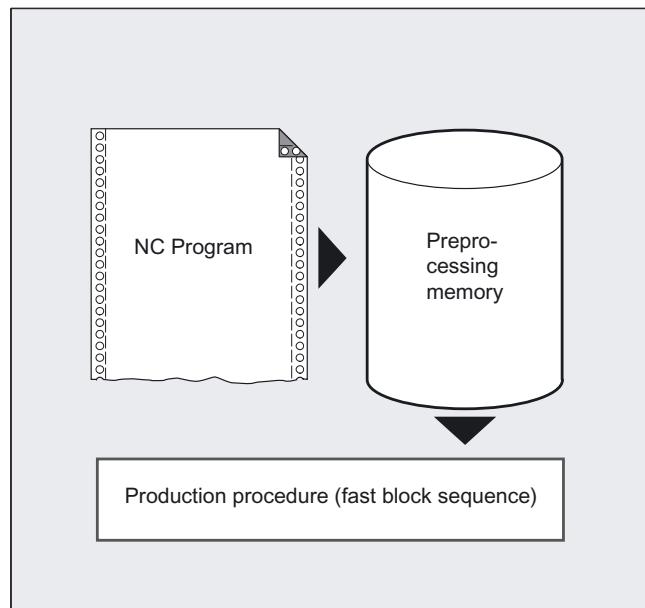
## 8.3 Program run with preprocessing memory (STARTFIFO, STOPFIFO, STOPPRE)

### Function

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.



## Syntax

STARTFIFO  
STOPFIFO  
STOPRE

## Significance

STOPFIFO	Stop high-speed processing section, fill preprocessing memory, until STARTFIFO, "Preprocessing memory full" or "End of program" is detected.
STARTFIFO	Start of high-speed processing section, in parallel to filling the preprocessing memory
STOPRE	Preprocessing stop

---

### Note

STOPFIFO stops the machining until the preprocessing memory has been filled or STARTFIFO or STOPRE is detected.

---

## Example: Designate machining step

The high-speed processing section to be buffered in the preprocessing memory is marked at the beginning and end with STARTFIFO or STOPFIFO respectively.

N10 STOPFIFO  
N20...  
N100  
N110 STARTFIFO

Execution of these blocks does not begin until the preprocessing memory is full or command STARTFIFO is detected.

Exception:

The preprocessing memory is not filled or filling is interrupted if the processing section contains commands that require unbuffered operation (reference point approach, measuring functions, ...).

### Example: Stop preprocessing STOPRE

If **STOPRE** is programmed the following block is not executed until all preprocessed and saved blocks are executed in full. The preceding block is halted in exact stop (as with G9).

Example:

```
N10
N30 MEAW=1 G1 F1000 X100 Y100 Z50
N40 STOPRE
```

The control generates an internal preprocessor stop upon access to machine status data (\$SA...).

Example:

Program code	Comments
R10 = \$AA_IM[X]	; Read actual values of the X axis.



#### CAUTION

When a tool compensation or spline interpolations are active, you should not program the **STOPRE** command as this will lead to interruption in contiguous block sequences.

## 8.4 Conditionally interruptible program sections (DELAYFSTON, DELAYFSTOF)

### Function

Conditionally interruptible part program sections are called stop delay sections. No **stopping** should occur and the **feed** should not be changed within certain program sections.

Essentially, short program sections - e.g. for machining a thread - should be protected from almost all stop events. Stops do not take effect until the program section has been completed.

## Syntax

DELAYFSTON  
DELAYFSTOF

The commands are programmed separately in a part program line.

Both commands are only permitted in part programs but not in synchronous actions.

## Significance

DELAYFSTON	Define a start of a range in which "soft" stops are delayed up to the end of the stop delay section.
DELAYFSTOF	Define the end of the stop delay section

---

### Note

For machine data MD11550 \$MN\_STOP\_MODE\_MASK Bit 0 = 0 (default), a stop delay section is implicitly defined if G331/G332 is active and a path motion or G4 is programmed.

---

## Example: Stop events

In the stop delay section, a change in the **feedrate and feedrate inhibit** are ignored. They only become effective after the stop delay section.

Stop events are divided into:

"Soft" stop events	Response: delayed
"Hard" stop events	Response: immediate

Selection of a number of stop events, which induce at least short stopping:

Event name	Response	interruption parameters
RESET	immediate	IS: DB21,... DBX7.7 and DB11, ... DBX20.7
PROG_END	Alarm 16954	NC prog.: M30
INTERRUPT	delayed	IS: FC-9 and ASUP DB10, ... DBB1
SINGLEBLOCKSTOP	delayed	Single block mode in the stop delay section is activated: NC stops at the end of the first block outside the stop delay section. Single block already selected before the stop delay section: NST: "NC Stop at block limit" DB21, ... DBX7.2

## 8.4 Conditionally interruptible program sections (DELAYFSTON, DELAYFSTOF)

Event name	Response	interruption parameters
STOPPROG	delayed	IS: DB21,... DBX7.3 and DB11, ... DBX20.5
PROG_STOP	Alarm 16954	NC prog.: M0 and M1
WAITM	Alarm 16954	NC prog.: WAITM
WAITE	Alarm 16954	NC prog.: WAITE
STOP_ALARM	immediate	Alarm: Alarm configuration STOPBYALARM
RETREAT_MOVE_THREAD	Alarm 16954	NC prog.: Alarm 16954 with LFON (stop and fastlift in G33 not possible)
WAITMC	Alarm 16954	NC prog.: WAITMC
NEWCONF_PREP_STOP	Alarm 16954	NC prog.: NEWCONF
SYSTEM_SHUTDOWN	immediate	System shutdown with 840Di
ESR	delayed	Extended stop and retract
EXT_ZERO_POINT	delayed	External zero offset
STOPRUN	Alarm 16955	OPI: PI "_N_FINDST" STOPRUN

**Explanation of the responses**

immediate ("hard" stop event)	Stops immediately even in stop delay section.
delayed ("soft" stop event)	Does not stop (even short-term) until after stop delay section.
Alarm 16954	Program is aborted because illegal program commands have been used in stop delay section.
Alarm 16955	Program is continued, an illegal action has taken place in the stop delay section.
Alarm 16957	The program section (stop delay section) enclosed by DELAYFSTON and DELAYFSTOF could not be activated. Every stop will take effect immediately in the section and is not subject to a delay.

For a list of other responses to stop events, see:

**References:**

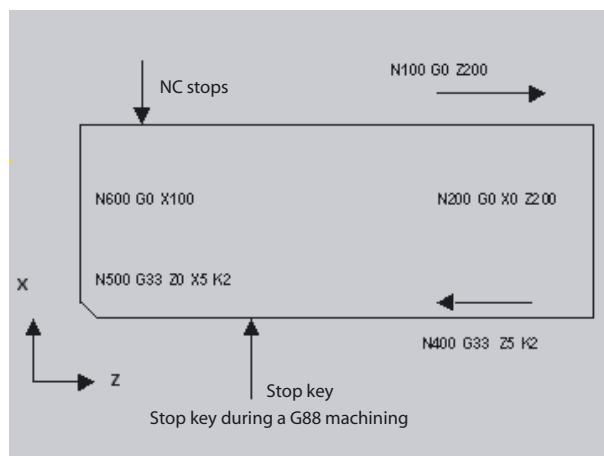
Function Manual Basic Functions; Mode Group, Channel, Program Operation, (K1), "Influencing and Impact on Stop Events"

**Example: Nesting stop delay sections in two program levels**

<b>Program code</b>	<b>Comments</b>
N10010 DELAYFSTON()	; Blocks with N10xxx program level 1.
N10020 R1 = R1 + 1	
N10030 G4 F1	; Stop delay section 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 section end at the same level.
...	
N10060 R2 = R2 + 2	
N10070 G4 F1	; Stop delay section ends. From now, stops act immediately.

**Example: Program excerpt**

The following program block is repeated in a loop:



As shown in the diagram, the user presses "Stop" in the stop delay section and the NC starts deceleration outside the stop delay section, i.e., in block N100. That causes the NC to stop at the beginning of N100.

Program code	Comments
<pre> ... N99 MY_LOOP: N100 G0 Z200 N200 G0 X0 Z200 N300 DELAYFSTON() N400 G33 Z5 K2 M3 S1000 N500 G33 Z0 X5 K3 N600 G0 X100 N700 DELAYFSTOF() N800 GOTOB MY_LOOP </pre>	

Details on SERUPRO type block searches and feeds in conjunction with G331/G332 Feed for tapping without compensating chuck, see:

**References:**

Function Manual Basic Functions; Mode Group, Channel, Program Operation (K1)  
Function Manual Basic Functions; Feedrates (V1)

### Advantages of the stop delay section

A program section is processed without a drop in velocity.

If the user aborts the program after a stop with RESET, the aborted program block is after the protected section. This program block is a suitable search target for a subsequent block search.

The following main run axes are not stopped as long as a stop delay section is in progress:

- Command axes and
- Positioning axes that travel with POSA

Parts program command G4 is permitted in a stop delay section whereas other parts program commands that cause a temporary stop (e.g., WAITM) are not permitted.

Like a path movement, G4 activates the stop delay section and/or keeps it active.

#### Example: Feedrate intervention

If the override is reduced to 6% before a stop delay section, the override becomes active in the stop delay section.

---

#### *8.4 Conditionally interruptible program sections (DELAYFSTON, DELAYFSTOF)*

If the override is reduced from 100% to 6% in the stop delay section, the stop delay section is completed with 100% and beyond that the program continues with 6%.

The feed disable has no effect in the stop delay section; the program does not stop until after the stop delay section.

#### **Overlapping/nesting:**

If two stop delay sections overlap, one from the NC commands and the other from machine data MD 11550: STOP\_MODE\_MASK, the largest possible stop delay section will be generated.

The following features regulate the interaction between NC commands **DELAYFSTON** and **DELAYFSTOF** with nesting and end of subroutine:

1. **DELAYFSTOF** is activated implicitly at the end of the subroutine in which **DELAYFSTON** is called.
2. **DELAYFSTON** stop delay section has no effect.
3. If subroutine 1 calls subroutine 2 in a stop delay section, the whole of subroutine 2 is a stop delay section. **DELAYFSTOF** in particular has no effect in subroutine 2.

---

#### **Note**

REPOSA is an end of subroutine command and **DELAYFSTON** is always deselected.

If a "hard" stop event coincides with the "stop delay section", the entire "stop delay section" is deselected! Thus, if any other stop occurs in this program section, it will be stopped immediately. A new program setting (new **DELAYFSTON**) must be made in order to start a new stop delay section.

If the Stop key is pressed before the stop delay section and the NCK must travel into the stop delay section for braking, the NCK will stop in the stop delay section and the stop delay section will remain deselected!

A stop delay section entered with an override of 0% will **not** be accepted!

This applies to all "soft" stop events.

**STOPALL** can be used to decelerate in the stop delay section. A **STOPALL**, however, immediately activates all other stop events that were previously delayed.

---

### **System variables**

A stop delay section can be detected in the parts program with \$P\_DELAYFST. If bit 0 of the system variables is set to 1, parts program processing is now in a stop delay section.

A stop delay section can be detected in synchronized actions with \$AC\_DELAYFST. If bit 0 of the system variables is set to 1, parts program processing is now in a stop delay section.

### **Compatibility**

Default of machine data MD 11550: STOP\_MODE\_MASK Bit 0 = 0 triggers implicit stop delay section during a G code group G331/G332 and when a path movement or G4 is programmed.

Bit 0 = 1 permits a stop during a G code group G331/G332 and when a path movement or G4 has been programmed (behavior until SW 6). The DELAYFSTON/DELAYFSTOF commands must be used to define a stop delay section.

## **8.5 Preventing program position for SERUPRO (IPTRLOCK, IPTRUNLOCK)**

### **Function**

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 operator interface.

### **Syntax**

IPTRLOCK  
IPTRUNLOCK

The commands are located in a part program line and allow a programmable interruption pointer

## Significance

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 section 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.

## 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 features regulate the interaction between NC commands `IPTRLOCK` and `IPTRUNLOCK` with nesting and end of subroutine:

1. `IPTRLOCK` is activated implicitly at the end of the subroutine in which `IPTRUNLOCK` is called.
2. `IPTRLOCK` in an untraceable section has no effect.
3. If subroutine 1 calls subroutine 2 in an untraceable section, the whole of subroutine 2 remains untraceable. `IPTRUNLOCK` in particular has no effect in subroutine 2.

For more information, see

/FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1).

## System variables

An untraceable section can be detected in the parts 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 gearbox with `EGON`
- axial leading value coupling with `LEADON`

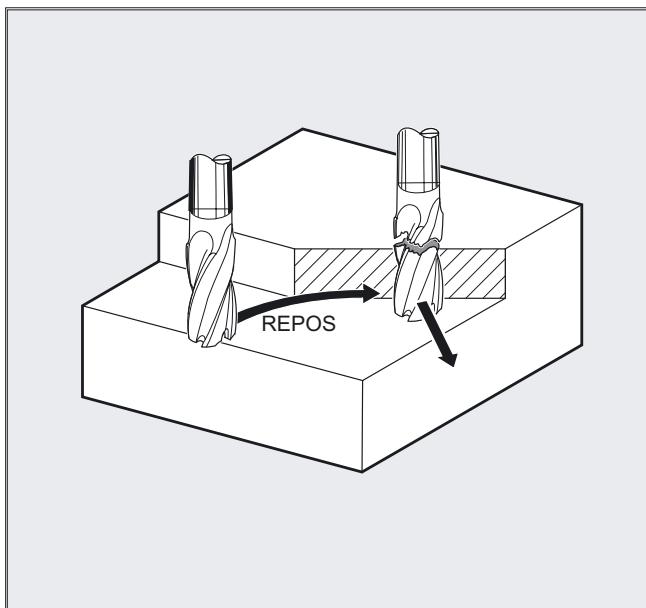
are used to activate the automatic interrupt pointer. If the programmed interrupt pointer and interrupt pointer activated with automatic interrupt pointers overlap, the largest possible untraceable section will be generated.

## **8.6 Repositioning to a contour (REPOSA, REPOS<sub>L</sub>, REPOSQ, REPOSQA, REPOSH, REPOSHA, DISR, DISPR, RMI, RMB, RME, RMN)**

### **Function**

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 check a measurement, you can reposition at any selected point on the contour under control by the program.

The REPOS command acts in the same way as a subprogram return jump (e.g., via M17). Blocks programmed after the command in the interrupt routine are not executed.



For information about interrupting program runs, see also Section "Flexible NC programming", Chapter "Interrupt routine" in this Programming Manual.

## Syntax

```

REPOSA RMI DISPR=...
REPOSA RMB
REPOSA RME
REPOSA RMN
REPOSL RMI DISPR=...
REPOSL RMB
REPOSL RME
REPOSL RMN
REPOSQ RMI DISPR=... DISR=...
REPOSQ RMB DISR=...
REPOSQ RME DISR=...
REPOSQA DISR=...
REPOSH RMI DISPR=... DISR=...
REPOSH RMB DISR=...
REPOSH RME DISR=...
REPOSHA DISR=...
  
```

## Significance

### Approach path

REPOSA	Approach along line on all axes
REPOS <sub>L</sub>	Approach along line
REPOS <sub>Q</sub> DISR=...	Approach along quadrant with radius DISR
REPOSQA DISR=...	Approach on all axes along quadrant with radius DISR
REPOS <sub>H</sub> DISR=...	Approach along semi-circle with diameter DISR
REPOSHA DISR=...	Approach on all axes along semi-circle with diameter DISR

### Reapproach point

RMI	Approach interruption point
RMI DISPR=...	Entry point at distance DISPR in mm/inch in front of interruption point
RMB	Approach block start point
RME	Approach end of block
RME DISPR=...	Approach block end point at distance DISPR in front of end point
RMN	Approach at nearest path point
A0 B0 C0	Axes in which approach is to be made

## *Path traversing behavior*

### **8.6 Repositioning to a contour (REPOSA, REPOS<sub>L</sub>, REPOS<sub>Q</sub>, REPOSQA, REPOS<sub>H</sub>, REPOS<sub>HA</sub>, DISR, DISPR, RMI, RMB, RME, RMN)**

#### **Example: Approach along a straight line, REPOSA, REPOS<sub>L</sub>**

The tool approaches the repositioning point along a straight line.

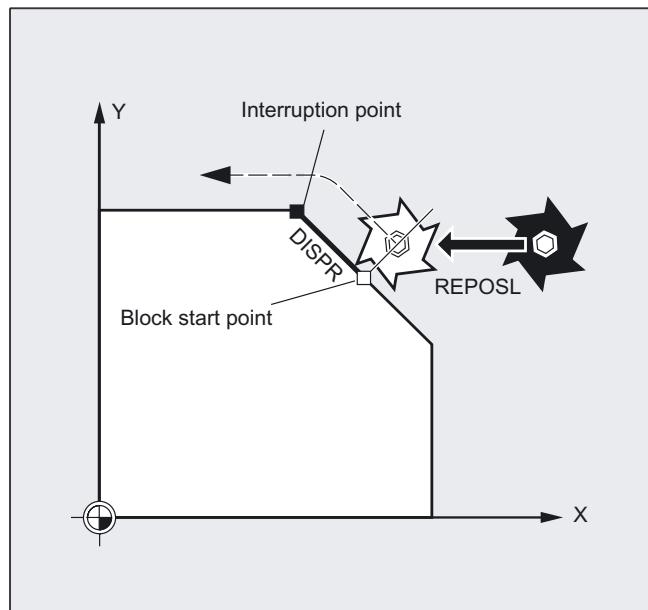
All axes are automatically traversed with command REPOSA. With REPOS<sub>L</sub> you can specify which axes are to be moved.

**Example:**

REPOS<sub>L</sub> RMI DISPR=6 F400

or

REPOSA RMI DISPR=6 F400

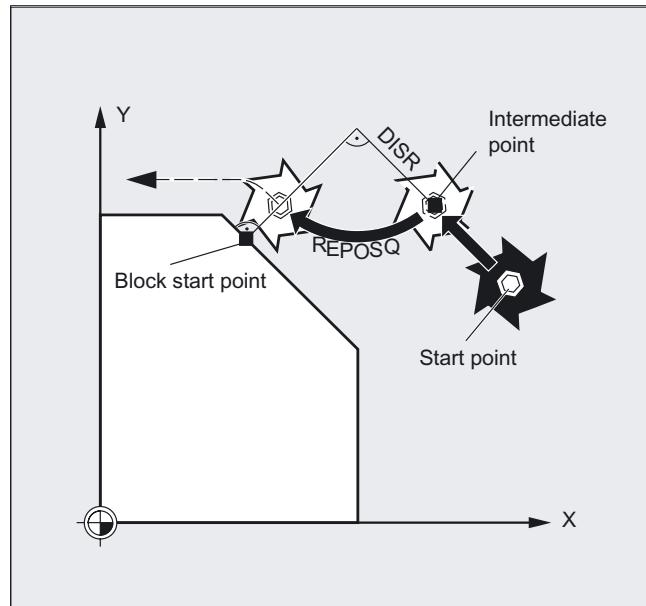


**Example: Approach in circle quadrant, REPOSQ, REPOSQA**

The tool approaches the repositioning point along a quadrant with a radius of  $\text{DISR}=\dots$ . The control system automatically calculates the intermediate point between the start and repositioning points.

**Example:**

REPOSQ RMI DISR=10 F400



## *Path traversing behavior*

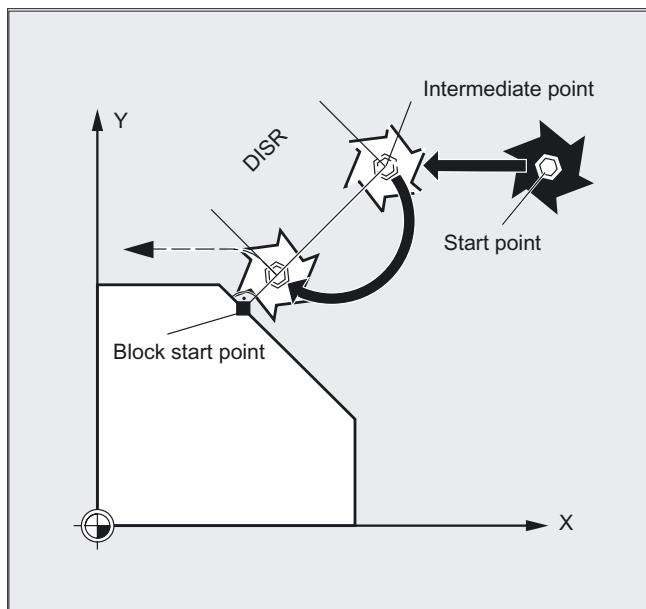
### **8.6 Repositioning to a contour (REPOSA, REPOS<sub>L</sub>, REPOS<sub>Q</sub>, REPOSQA, REPOS<sub>H</sub>, REPOSHA, DISR, DISPR, RMI, RMB, RME, RMN)**

#### **Example: Approach tool in a semicircle, REPOS<sub>H</sub>, 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:**

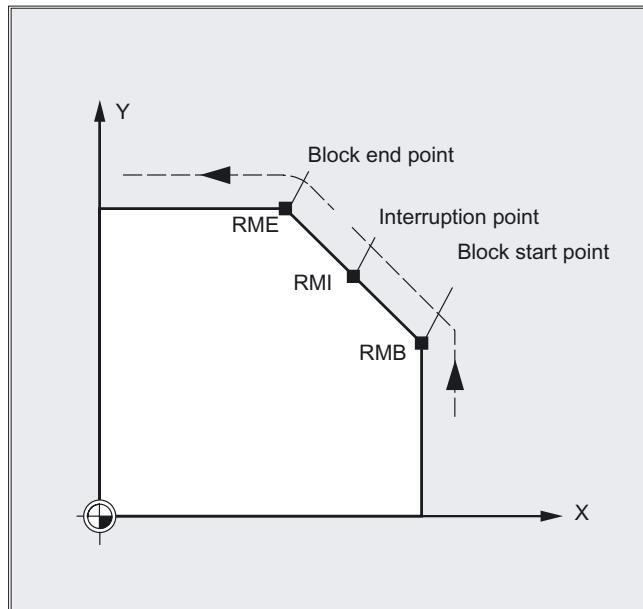
REPOS<sub>H</sub> RMI DISR=20 F400



### Specifying the repositioning point (not for SERUPRO approaching with RMN)

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:

- RMI, interruption point
- RMB, block start point or last end point
- RME, block end point



RMI DISPR=... or RME DISPR=... allows you to select a repositioning point which sits 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 RMI) or the block end point (with RME).

## *Path traversing behavior*

### *8.6 Repositioning to a contour (REPOSA, REPOS1, REPOSQ, REPOSQA, REPOS1H, REPOS1HA, DISR, DISPR, RMI, RMB, RME, RMN)*

#### **DISPR sign**

The sign **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 **RMB**, 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.

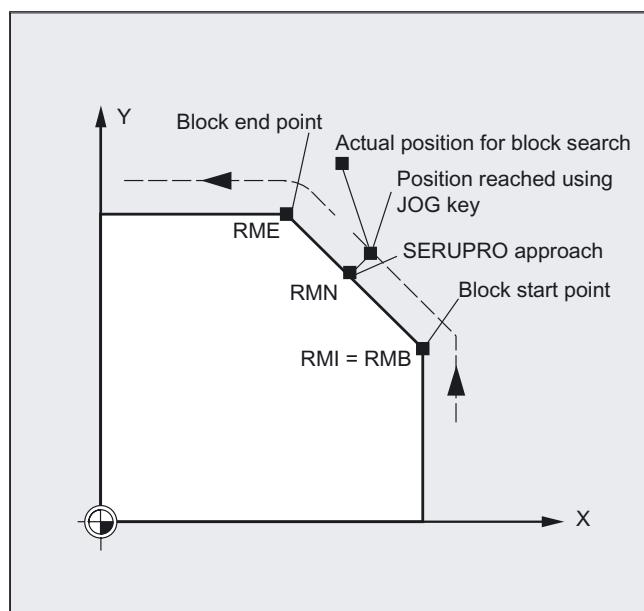
#### **Sample application:**

A sensor will recognize the approach to a clamp. An **ASUP** 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 RMN**

If abort is forced during machining at any position, the shortest path from the abort point is approached with SERUPRO approach and **RMN** 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.

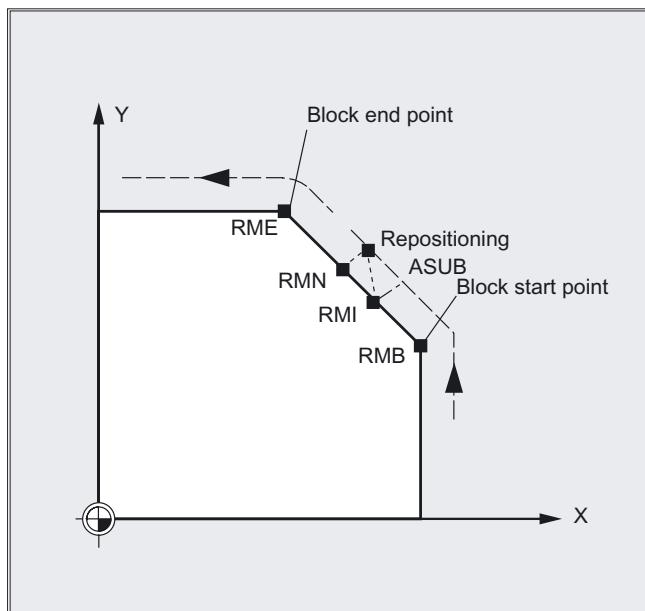


**Note****SERUPRO**

For SERUPRO , RMI and RMB are identical. RMN is not limited to SERUPRO but is generally applicable.

**Approach from the nearest path point RMN**

When REPOSA is interpreted, the repositioning block with RMN 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 RMB: Approach to beginning
- 2 RMI: Approach to point of interruption
- 3 RME: Approach to end of block
- 4 RMN: Approaching to next path point of the interrupted block

## *Path traversing behavior*

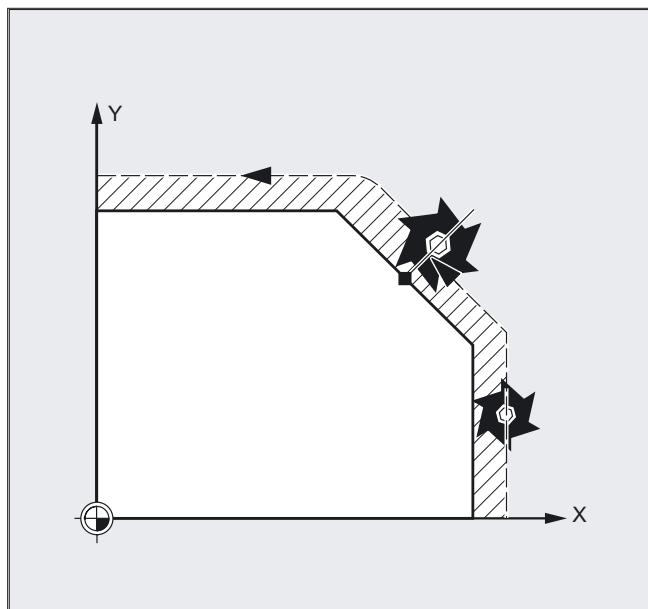
### **8.6 Repositioning to a contour (REPOSA, REPOS<sub>L</sub>, REPOS<sub>Q</sub>, REPOSQA, REPOS<sub>H</sub>, REPOS<sub>HA</sub>, DISR, DISPR, RMI, RMB, RME, RMN)**

#### **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 REPOS<sub>HA</sub> commands automatically reposition all axes. Individual axis names need not be specified.

When the commands REPOS<sub>L</sub>, REPOS<sub>Q</sub> and REPOS<sub>H</sub> are programmed, all geometry axes are traversed automatically, i.e. they need not be named in the command. All other axes must be specified in the commands.

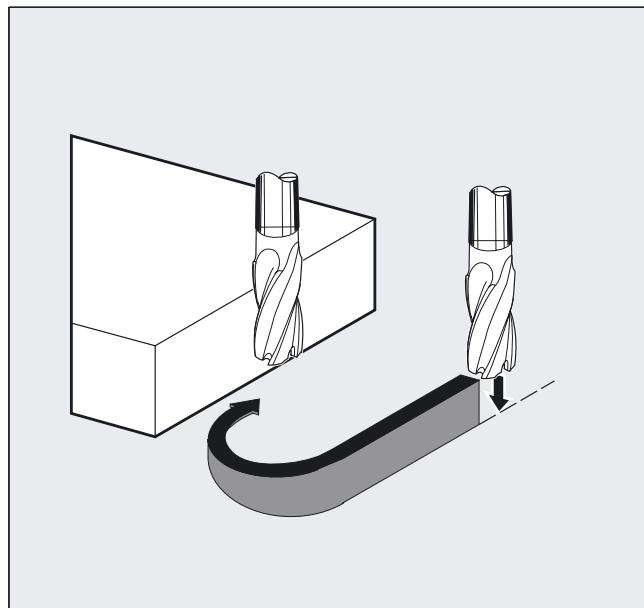
**The following applies to the REPOSH and REPOSQ circular motions:**

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, the control automatically switches over to linear approach REPOS<sub>L</sub>:

- 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.



## **8.7 Influencing the motion control**

### **8.7.1 Percentage jerk correction (JERKLIM)**

#### **Function**

In critical program sections, it may be necessary to limit the jerk to below the maximum possible value - e.g. to reduce the level of stress on the machine. The acceleration mode SOFT must be active. The function only effects path axes.

#### **Syntax**

JERKLIM[<axis>]=<value>

#### **Significance**

JERKLIM	Command for jerk correction
<axis>	Machine axis whose jerk limit value is to be adapted.
<value>	Change as a percentage for the highest permissible jerk referred to the value set for the axis in the machine data. Range of values: 1 ... 200 Value 100 does not influence the jerk. This setting is effective after a reset and part program start.

#### **Example**

In the AUTOMATIC operating modes, for the programmed axis, the jerk limit value is limited to the specified percentage of the jerk limit value saved in the machine data:

N60 JERKLIM[X]=75

Meaning: The axis slide in the X direction should only be accelerated/decelerated with 75% of the jerk permissible for the axis.

## 8.7.2 Percentage velocity correction (VELOLIM)

### Function

In critical program sections, it may be necessary to limit the velocity to below the maximum possible value - e.g. for instance to reduce the level of stress on the machine or to improve the machining quality. The function only effects path and positioning axes.

### Syntax

VELOLIM[<axis>]=<value>

### Significance

VELOLIM	Command for velocity correction
<axis>	Machine axis whose velocity limit value should be adapted.
<value>	Change as a percentage for the highest permissible velocity referred to the value set for the axis in the machine data. Range of values: 1 ... 100 The velocity is not influenced for a value of 100. This setting is effective after a reset and part program start.

### Example

In the AUTOMATIC operating modes, the velocity limit value for the programmed axis is limited to the specified percentage of the velocity limit value saved in the machine data:  
N70 VELOLIM[X]=80

Meaning: The axis slide in the X direction should only be traversed with 80% of the velocity permissible for the axis.

### **8.7.3 Program example for JERKLIM and VELOLIM**

The following program presents an application example for the percentage jerk and velocity limit:

<b>Program code</b>	<b>Comments</b>
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	

# Axis couplings

## 9.1 Coupled motion (TRAILON, TRAILOF)

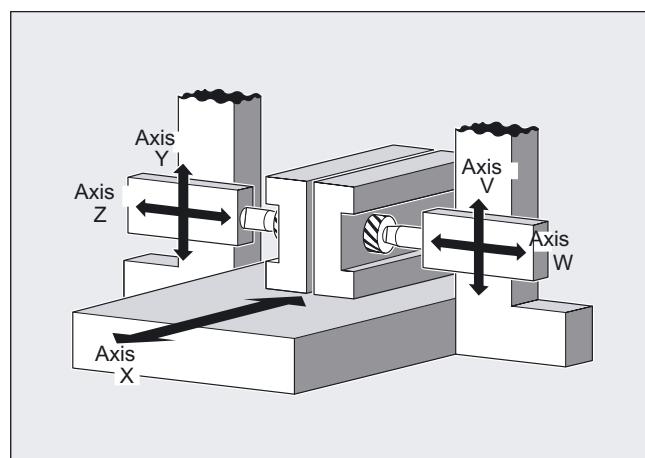
### Function

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 2 coupled motion groups:
  1. Leading axis Y, coupled motion axis V
  2. Leading axis Z, coupled motion axis W



## *Axis couplings*

---

### *9.1 Coupled motion (TRAILON, TRAILOF)*

#### **Syntax**

```
TRAILON(Faxis,Laxis,couple)
TRAILOF(Faxis,Laxis,Laxis2)
TRAILOF(FAxis)
```

#### **Significance**

TRAILON	Activating and defining a coupled-axis grouping
	Active: modal
TRAILOF	Deactivate coupled axes
	Active: modal
	TRAILOF with 2 parameters deactivates the coupling to only 1 leading axis. If a coupled motion axis has 2 leading axes, e.g. V = coupled motion axis and X, Y=leading axes, then TRAILOF with 3 parameters can be called to switch-out the coupling: TRAILOF (V, X, Y)
	TRAILOF (V) : Deactivate the coupling without details of leading axis. If the trailing axis has 2 leading axes, both couplings are deactivated.
Faxis	Axis name of trailing axis
	A coupled axis can also act as the leading axis for other coupled axes. In this way, it is possible to create a range of different coupled axis groupings.
Laxis	Axis name of trailing axis
Couple	Coupling factor = Path of coupled-motion axis/path of trailing axis
	Default = 1

---

#### **Note**

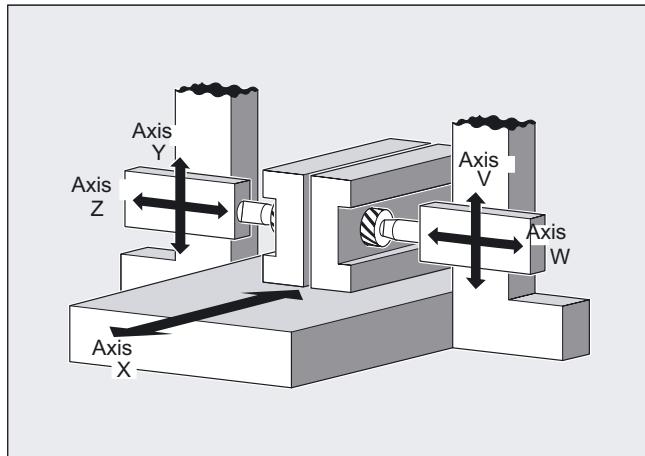
Coupled axis motion is always executed in the base coordinate system (BCS).

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	Comments
...	
N100 TRAILON(V,Y)	; Activation of 1st coupled axis group
N110 TRAILON(W,Z,-1)	; Switching-in the 2nd coupled axis group negative coupling factor: ;Coupled motion axis traverses ;in the opposite direction from leading axis
N120 G0 Z10	; Infeed Z and W axes in opposite axis direction
N130 G0 Y20	; Infeed of Y and V axes in same axis direction
...	
N200 G1 Y22 V25 F200	; Superimpose dependent and independent movement of ;coupled motion axis ;"V"
...	
TRAILOF(V,Y)	; Deactivate 1st coupled axis group
TRAILOF(W,Z)	; Deactivate 2nd coupled axis group

### Coupled 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 axis can be programmed with the full range of available motion commands (G0, G1, G2, G3, ...). The coupled axis not only traverses the independently defined paths, but also those derived from its leading axes on the basis of coupling factors.

### Coupling factor

The coupling factor specifies the desired relationship between the paths of the coupled axis and the leading axis.

**Formula:** Coupling factor = Path of coupled-motion axis/path of trailing axis

If a coupling factor is not programmed, then coupling factor 1 automatically applies.

The factor is entered as a fraction with decimal point (of type REAL). The input of a negative value causes the master and coupled axes to traverse in opposition.

### Acceleration and velocity

The acceleration and velocity limits of the combined axes are determined by the "weakest axis" in the combined axis pair.

### 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

8: Coupled motion active

## 9.2 Curve tables (CTAB)

### 9.2.1 Curve tables: general relationships

#### Function

The Curve tables section contains the program commands that can be used to program the relationships between two axes (leading and following axis).

A following variable can be assigned uniquely to each master value within a defined master value range. If the master value is outside the definition range, the behavior at the edge of the curve table can be programmed for periodic and non-periodic curve tables.

#### Description

The mechanical cams are replaced by curve tables that can be used to define

- the specific curve traces in a definition range
- individual sections, known as curve segments
- the edges of the curve for periodic and non-periodic curve tables
- the curve segment positions concerned

In a defined value range of

- the associated table positions and
- the start and end values of a table segment

the corresponding slave value for a master value and similarly the master value for a slave value can be read.

All other forms are shown and optional parameters can be assigned to the associated program commands. The resulting possibilities to influence specific individual or several curve tables in the corresponding memory type provide a flexible programming for further applications. This also provides comprehensive possibilities for programming the diagnosis of axis couplings.

Typical program examples are provided for the definition of curve tables and the access to curve table positions .

### 9.2.2 Curve tables: Principal functions (CTABDEF, CATBEND, CTABDEL)

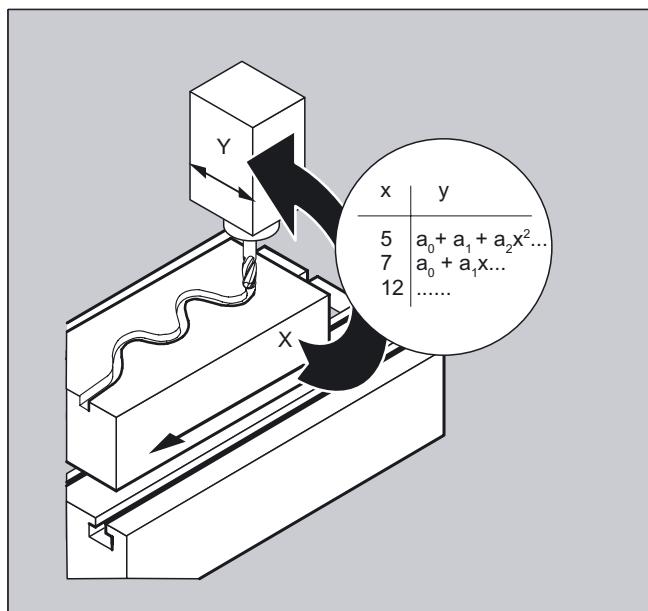
#### Function

You can use curve tables to program position and velocity relationships between two axes. Curve tables are defined in a part program.

**Example** of substitution of mechanical cam:

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.



#### Syntax

Modal language commands with curve tables

```
CTABDEF(Faxis,Laxis,n,applim,memType)
CTABEND()
CTABDEL()
CTABDEL(, ,memType)
```

## Significance

### Principal functions

CTABDEF ( )	Define beginning of curve table.
CTABEND ()	Define end of curve table.
CTABDEL ()	Deleting all curve tables, <b>irrespective of the memory type</b> .
Faxis	Following axis
	Axis that is programmed via the curve table.
Laxis	Leading axis
	Axis that is programmed with the master value.
n, m	Number of curve table; n < m, e.g., in CTABDEL(n, m) The number of the curve table is unique and not dependent on the memory type. Tables with the same number can be in the SRAM and DRAM.
applim	Identifier for table periodicity: Table is not periodic Table is periodic with regard to the leading axis Table is periodic with regard to leading axis and following axis
memType	Optional specification of memory type of the NC: "DRAM" / "SRAM" If no parameter is programmed for this value, the standard memory type set with MD 20905: CTAB_DEFAULT_MEMORY_TYPE is used.

### Machine manufacturer

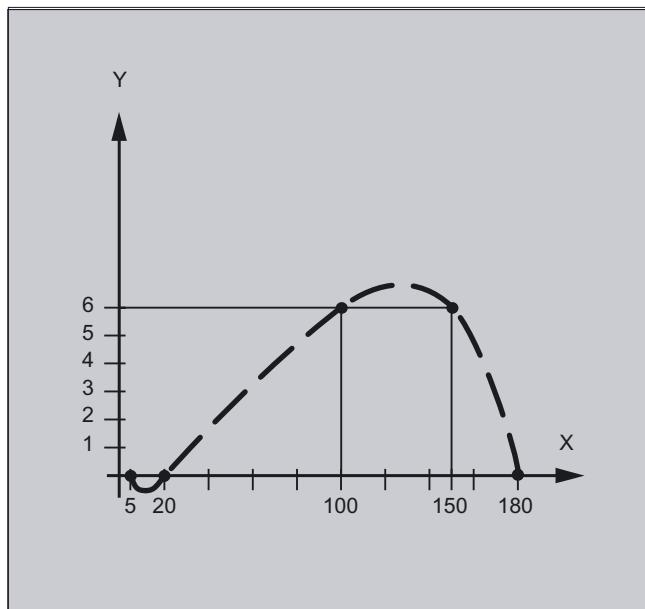
To create curve tables the memory space must be reserved by setting the machine data.

**Example: Using CTABDEF and CTABEND**

A program section is to be used unchanged for defining a curve table. The command for preprocess stop STOPRE can remain and is active again immediately as soon as the program section is not used for table definition and CTABDEF and CTABEND have been removed:

Program code	Comments
...	
CTABDEF (Y,X,1,1)	; Definition of a curve table.
...	
...	
IF NOT (\$P_CTABDEF)	
STOPRE	
ENDIF	
...	
...	
CTABEND	

### Example: Definition of a curve table



Program code	Comments
N100 CTABDEF(Y,X,3,0)	; Beginning of the definition of a ;non-periodic curve table with number 3.
N110 X0 Y0	; 1. Traverse statement, defines the starting values and 1st intermediate point: Master value: 0, Following value: 0
N120 X20 Y0	; 2. Intermediate point: Master value: 0..20, Following value: starting value..0
N130 X100 Y6	; 3. Intermediate point: Master value: 20..100, Following value: 0..6
N140 X150 Y6	; 4. Intermediate point: Master value: 100..150, Following value: 6..6
N150 X180 Y0	; 5. Intermediate point: Master value: 150..180, Following value: 6..0
N200 CTABEND	; End of definition; in its internal representation, the curve table is generated as a polynomial, maximum 5th degree. The calculation of the curve with the specified intermediate points is dependent on the selected interpolation type (circular, linear, spline interpolation); the part program state before starting the definition is restored.

**Example: 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:

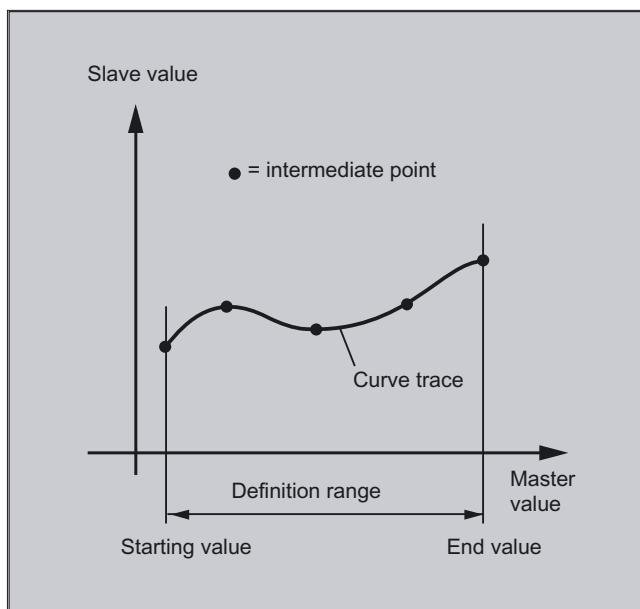
Program code	Comments
N10 DEF REAL DEPPOS	
N20 DEF REAL GRADIENT	
N30 CTABDEF(Y,X,2,1)	; Start of definition
N40 G1 X=0 Y=0	
N50 POLY	
N60 PO[X]=(45.0)	
N70 PO[X]=(90.0) PO[Y]=(45.0,135.0,-90)	
N80 PO[X]=(270.0)	
N90 PO[X]=(315.0) PO[Y]=(0.0,-135.0,90)	
N100 PO[X]=(360.0)	
N110 CTABEND	; End of definition
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 master value 75.0.
N180 G0 X75 Y=DEPPOS	; Positioning leading and following 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	

## Definition of a curve table

CTABDEF, CTABEND

A curve table represents a part program or a section of a part program, which is enclosed by **CTABDEF** at the beginning and **CTABEND** at the end.

Within this part program section, unique trailing axis positions are assigned to individual positions of the leading axis by traverse statements and used as intermediate positions in calculating the curve definition in the form of a polynomial up to the 5th order.



## 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.

Within the definition of the curve table, you have use of the entire NC language.

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.

---

**Note**

The following are not permissible:

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.

---

### Activating ASPLINE, BSPLINE, CSPLINE

If an ASPLINE, BSPLINE or CSPLINE is activated within a curve table CTABDEF( ) . . . CTABEND, at least a start point should be programmed before this spline activation. An immediate activation after CTABDEF must be avoided as otherwise the spline will depend on the current axis position before the curve table definition.

Example:

Program code	Comments
... CTABDEF (Y,X,1,0) X0 Y0 ASPLINE X=5 Y=10 X10 Y40 ... CTABEND	

Depending on 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. The other restrictions given in the notice still apply.

When creating and deleting tables you can use the definitions of the memory type of the NC.

## Deleting curve tables, CTABDEL

CTABDEL can be used to delete the curve tables. Curve tables that are active in an axis coupling cannot be deleted. If at least one curve table of a multiple delete command CTABDEL() or CTABDEL(n, m) is active in a coupling, **none** of the addressed curve tables will be deleted. The curve tables of a specific memory type can be deleted by the optional specification of a memory type. See chapter "Curve table forms (CTABDEL, ... CTABUNLOCK)".

## 9.2.3 Curve tables: Forms (CTABNOMEM, CTABFNO, CTABID, CTABLOCK, CTABDEL, CTABUNLOCK, CTABISLOCK, CTABEXISTS, CTABMEMTYP, CTABPERIOD, CTABSEGID, CTABSEG, CTABFSEG, CTABMSEG, CTABPOLID, CTABFPOL, CTABMPOL)

### Function

Other applications of curve tables are:

- Delete in a specific SRAM or DRAM memory type.
- Specify the number of **defined** and still **possible** curve tables in the memory type.
- **Lock** or **remove** the lock to prevent curve tables from being deleted or overwritten.
- Optional details for selections, such as the deletion of **one** curve table, deletion of **one** curve table area, of **all** curve tables in the specified memory, and **lock** or **unlock** overwrite protection.
- Supply, return and check details for the diagnosis of axis couplings such as specific curve table properties  
Determine the number of curve tables, curve segments and curve polynomials.

## Syntax

Modal language commands with curve tables

```
CTABDEL(n, m, memType)
CTABNOMEM (memType)
CTABFNO (memType)
CTABID(n, memType)
CTABLOCK(n, m, memType) or CTABUNLOCK(n, m, memType)
CTABDEL(n) or CTABDEL(n, m)
CTABLOCK(n) or CTABLOCK(n, m) or CTABLOCK() or
CTABLOCK(, , memType)
CTABUNLOCK(n) or CTABUNLOCK(n, m) or CTABUNLOCK() or CTABUNLOCK(, ,
memType)
CTABID(n) or CTABID(n, memType) or CTABID(p, memType)
CTABISLOCK(n)
CTABEXISTS(n)
CTABMEMTYP(n)
TABPERIOD(n)
CTABSEGID(n, segType)
CTABSEG(memType, segType) or CTABFSEG(memType, segType) or
CTABMSEG(memType, segType)
CTABPOLID(n) or CTABMPOL(memType)
```

## Significance

**General form** in static or dynamic NC memory:

CTABDEL(n, m, memType)	Deletion of the curve tables of the curve table range that are stored in memType.
CTABNOMEM (memType)	Number of <b>defined</b> curve tables.
CTABFNO (memType)	Number of <b>possible</b> tables.
CTABID(n, memType)	Outputs table number entered in memory type as the nth curve table.
CTABLOCK(n, m, memType)	<b>Enable</b> deletion and overwrite <b>lock</b> .
CTABUNLOCK(n, m, memType)	<b>Cancel</b> deletion and overwrite <b>lock</b> . CTABUNLOCK releases the tables locked with CTABLOCK. Tables, which are involved in an active coupling, remain locked and cannot be deleted. Lock with CTABLOCK is canceled as soon as locking with active coupling is canceled with deactivation of coupling. This table can therefore be deleted. It is not necessary to call CTABUNLOCK again.

**Uses of other forms** Optional details for selections:

CTABDEL (n)	Delete <b>one</b> curve table.
CTABDEL (, , memType)	Delete <b>one</b> curve table range.
CTABLOCK (n)	Delete <b>all</b> curve tables in the specified memory.
CTABLOCK (n, m)	<b>Lock</b> the delete <b>and</b> overwrite: Curve table with number n.
CTABLOCK ()	Lock curve tables in the number range n to m.
CTABLOCK (, , memType)	All existing curve tables.
CTABUNLOCK (n)	All curve tables <b>in the</b> specified memory type.
CTABUNLOCK (n, m)	<b>Remove</b> lock for the delete <b>and</b> overwrite: Curve table with number n.
CTABUNLOCK ()	Re-enable curve tables in the number range n to m.
CTABUNLOCK (, , memType)	All curve tables that already exist.

**Uses of other forms** for the diagnosis of axis couplings:

CTABID (n, memType)	Outputs table number of the nth/pth curve table <b>with memory type</b> memType.
CTABID (p, memType)	
CTABID (n)	Outputs table number of the nth curve table with memory type defined in MD 20905: CTAB_DEFAULT_MEMORY_TYPE <b>specified</b> memory type.
CTABISLOCK (n)	Returns the lock status of the curve table <b>with number n</b> .
CTABEXISTS (n)	<b>Checks</b> curve table with number n.
CTABMEMTYP (n)	<b>Returns the memory</b> in which curve table no. n is stored.
CTABPERIOD (n)	Returns the <b>table periodicity</b> .
CTABSEG (memType)	Number of <b>curve segments already used</b> in the specified memory type.

CTABSEGID (n)	Number of <b>curve segments</b> used in <b>curve table</b> number n
CTABFSEG (memType)	Number of <b>possible</b> curve segments.
CTABMSEG (memType)	<b>Maximum</b> possible number of curve segments.
CTABPOLID (n)	Number used by <b>curve table</b> number n. <b>Curve table polynomials</b>
CTABSEG (memTyp, segType)	Number of type "L" or "P" <b>curve segments</b> used in the memory type.
CTABFSEGID (n, segType)	Number of type "L" or "P" <b>curve segments</b> used in <b>curve table</b> number n
CTABFSEG (memTyp, segType)	Number of type "L" or "P" <b>curve segments still possible</b> in the memory type
CTABMSEG (memTyp, segType)	<b>Maximum possible</b> number of type "L" or "P" <b>curve segments</b> in the memory type
CTABFPOL (memType)	Number of <b>curve polynomials still possible</b> in the specified memory type
CTABMPOL (memType)	<b>Maximum possible</b> number of <b>curve polynomials</b> in the specified memory type
n, m	Number of curve table; n < m, e.g., in CTABDEL(n, m)
p	The number of the curve table is unique and not dependent on the memory type. It is not possible for there to be tables with the same number in the static and dynamic NC memory.
memType	Entry location (in memType memory area) Optional specification of NC memory type: Both the "dynamic memory" and the "static memory" are possible If no parameter is programmed for this value, the standard memory type set with MD 20905: CTAB_DEFAULT_MEMORY_TYPE is used.
segType	Optional details for segment type. Possible settings are: segType "L" linear segments segType "P" polynomial segments

## Description

### Loading curve tables using "Execution from external source"

If curve tables are executed externally, the size of the reload buffer (DRAM) must be selected via MD18360 \$MN\_MM\_EXT\_PROG\_BUFFER\_SIZE in such a way that the entire curve table definition can be simultaneously stored in the reload buffer. Otherwise parts program processing is canceled with alarm 15150.

### Repeated use of curve tables

The functional relation between the leading axis and the following axis calculated using the curve table is retained under the table number selected beyond the end of the part program and power-off, if the table has been saved to the static NC memory (SRAM).

A table that was created in the dynamic memory (DRAM) will be deleted on power-on and may have to be regenerated.

The curve table created can be applied to any axis combinations of leading and trailing 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 you overwrite curve tables!

With the system variable \$P\_CTABDEF it is possible to query from inside a parts program whether a curve table definition is active.

The parts program section can be used as a curve table definition after excluding the statements and therefore as a real parts program again.

---

### 9.2.4 Curve tables: Behavior at the edges (CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABTMIN, CTABTMAX)

#### Function

If the master value lies outside the definition range, the value at the start and the end of the curve table can be read for a following axis.

CTABTSV can read for a **following axis** the value at the **beginning** of the curve table.  
CTABTEV can read for a **following axis** the value at the **end** of the curve table.

The start and end values of a curve table do not depend on whether the table is defined with increasing or decreasing master values. The start value is always defined by the lower interval limit, and the end value by the upper interval limit.

The **minimum** and **maximum values** of a curve table can be defined for a whole range or a defined interval with CTABMIN and CTABTMAX. Two limits are specified for the interval of the master value.

#### Syntax

**Start and end value slave value for following axis:**

CTABTSV(n, degrees, Faxis), CTABTEV(n, degrees, Faxis)

**Start and end value master value for leading axis:**

CTABTSP(n, degrees, Faxis), CTABTEP(n, degrees, Faxis)

**Min and max value ranges:**

CTABTMIN(n, Faxis)

CTABTMAX(n, Faxis)

#### Significance

CTABTSV	Read the start value of the curve table from a following axis.
CTABTEV ()	Read the end value of the curve table from a following axis.
CTABTSP ()	Read the start value of the curve table from a leading axis.
CTABTEP ()	Read the end value of the curve table from a leading axis.
CTABTMIN ()	Determine the minimum value of a curve table in the complete area or in a defined interval.
CTABTMAX ()	Determine the maximum value of a curve table in the complete area or in a defined interval.

Faxis	Following axis
Laxis	Axis that is programmed via the curve table.
n, m	Leading axis
	Axis that is programmed with the master value.
degrees	Number of curve tables
	Curve table numbers can be freely assigned. They are used exclusively for the unique identification.
	Gradient for incline at start or end of segment in curve table

### Values and value range

Values of the trailing and leading axis located at the beginning and end of a curve table  
**CTABTSV, CTABTEV, CTABTSP, CTABTEP**

R10=CTABTSV(n, degrees, Faxis)	Trailing value at beginning of curve table
R10=CTABTEV(n, degrees, Faxis).	Trailing value at beginning of curve table
R10=CTABTSP(n, degrees, Laxis).	Master value at beginning of curve table
R10=CTABTEP(n, degrees, Laxis).	Master value at end of curve table

### Value range of curve table of following value CTABTMIN, CTABTMAX

R10=CTABTMIN(n, Faxis).	Minimum following value of curve table over entire interval
R10=CTABTMAX(n, Faxis).	Maximum following value of curve table over entire interval
R10=CTABTMIN(n, a, b, Faxis, Laxis)	Minimum following value of curve table in interval a...b of master value
R10=CTABTMAX(n, a, b, Faxis, Laxis)	Maximum following value of curve table in interval a...b of master value

---

### Note

R parameter assignments in the table definition are reset.

---

### Example of the assignments to R parameters

Program code	Comments
<pre> ... R10=5 R11=20 ... CTABDEF G1 X=10 Y=20 F1000 R10=R11+5 X=R10 CTABEND ... </pre>	<p>R10=25 ; R10=5</p>

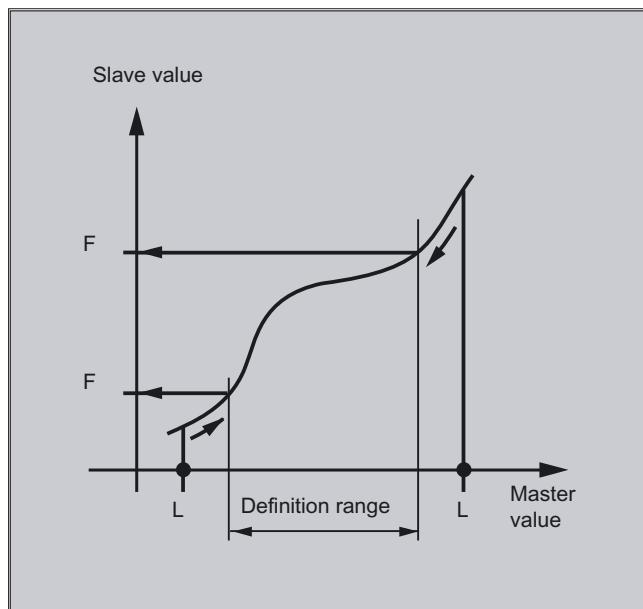
### Example: Using CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABTMIN, CTABTMAX

Determining the minimum and maximum value of a curve table.

Program code	Comments
N10 DEF REAL STARTVAL	;
N20 DEF REAL ENDVAL	
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)	; Beginning of table definition
N110 X0 Y10	; Start value of the 1st table segment
N120 X30 Y40	; End value of the 1st table segment =
N130 X60 Y5	; Start value of the 2nd table segment ...
N140 X70 Y30	
N150 X80 Y20	
N160 CTABEND	; End of table definition
...	
N200 STARTPOS = CTABTSV(1, GRADIENT)	; Start position STARTPOS = 10,
N210 ENDPOS = CTABTEV(1, GRADIENT)	; End position ENDPOS = 20 of the table, and
N220 SRARTPARA = CTABTSP(1, GRADIENT)	; STARTPARA = 10,
N230 ENDPARA = CTABTEP(1, GRADIENT)	; ENDPARA = 80 read the value range of the following axis.
...	
N240 MINVAL = CTABTMIN(1)	; Minimum value when Y = 5 and
N250 MAXVAL = CTABTMAX(1)	; Maximum value when Y = 40

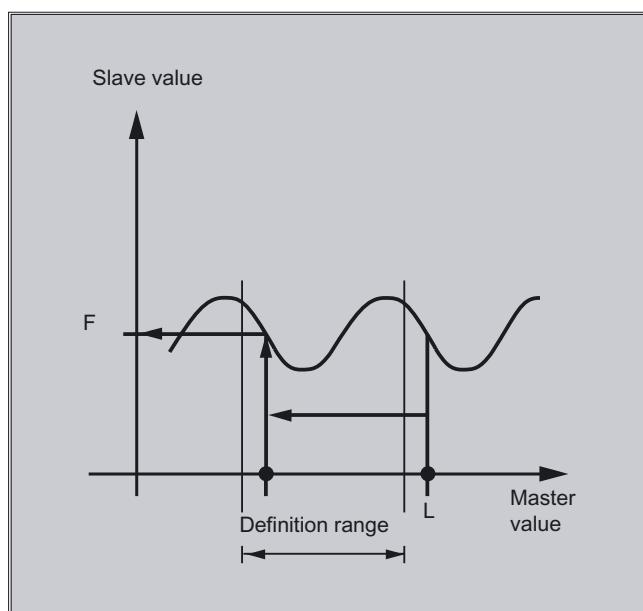
### Non-periodic curve table

If the master value is outside the definition range, the following value output is the upper or lower limit.



### Periodic curve table

If the master value is outside the definition range, the master value is evaluated modulo of the definition range and the corresponding following value is output.



**Note****CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABTMIN, CTABTMAX**

These language commands can be used directly from the part program or synchronized actions.

<b>Dependency of the function's internal execution time on the number of table segments:</b>	
CTABINV( )	dependent
CTABTSV, CTABTEV, CTABTSP, CTABTEP (CTABTMIN, CTABTMAX only if no interval of the master value is specified)	Independent

**Read in synchronized actions**

When using commands CTABINV( ) or CTABTMIN( ) and CTABTMAX( ) in synchronized actions, the user must ensure that at the instant of execution

- either sufficient NC power is available or
- the number of segments in the curve table must be queried before it is called up in case it is necessary to subdivide the table.

Additional related information about programming synchronized actions is given in chapter, "Motion synchronous actions".

### 9.2.5 Curve tables: Access to table positions and table segments (CTAB, CTABINV, CTABSSV, CATBSEV)

**Function****Reading table positions: CTAB, CTABINV**

With CTAB you can read the following value for a master value directly from the part program or from synchronized actions.

With CTABINV, you can read the master value for a following value. This assignment does not always have to be unique. CTABINV therefore requires an approximate value for the expected master value.

## Syntax

### Reading the following value for a leading value:

CTAB(master value, n, degrees, [following axis, leading axis])

### Reading the leading value for a following value:

CTABINV(following value, approx. leading value, n, degrees, [following axis, leading axis])

### Reading the start and end values of a table segment:

CTABSSV(leading value, n, degrees, [Faxis])

CTABSEV(leading value, n, degrees, [ Faxis])

## Significance

CTAB	Read a following value directly from a master value.
CTABINV	Read the master value for a following value.
CTABSSV	Read the start value of the curve segment for a following axis.
CTABSEV	Read the end value of the curve segment for a following axis.
Faxis	Following axis
Laxis	Axis that is programmed via the curve table.
n, m	Leading axis
	Axis that is programmed with the master value.
	Numbers for curve tables.
Degrees	Curve table numbers can be freely assigned. They are used exclusively for the unique identification.
ApproxLeadingValue	Gradient for incline at start or end of segment in curve table
e	The position value of the expected approximation value that can be used to determine a unique master value.

- **CTABSSV, CTABSEV**

CTABSSV can be used to read the **starting value** of the curve segment that belongs to the specified master value. CTABSSV can be used to read the **end value** of the curve segment that belongs to the specified master value.

- Trailing or leading position derived from curve table with CTAB, CTABINV

R10 = CTAB(LV, n, degree, Faxis, Laxis)	Following value for a master value
R10=CTABINV(FV, approxLV, n, degrees, Faxis, Laxis)	Master value to a following value

- Determining the segments of the curve table by specifying a master value with CTABSSV, CTABSEV

R10 = CTABSSV(LV, n, degrees, Faxis, Laxis)	Starting value of the following axis in the segment belonging to the LV
R10 = CTABSEV(LV, n, degrees, Faxis, Laxis)	End value of the following axis in the segment belonging to the LV

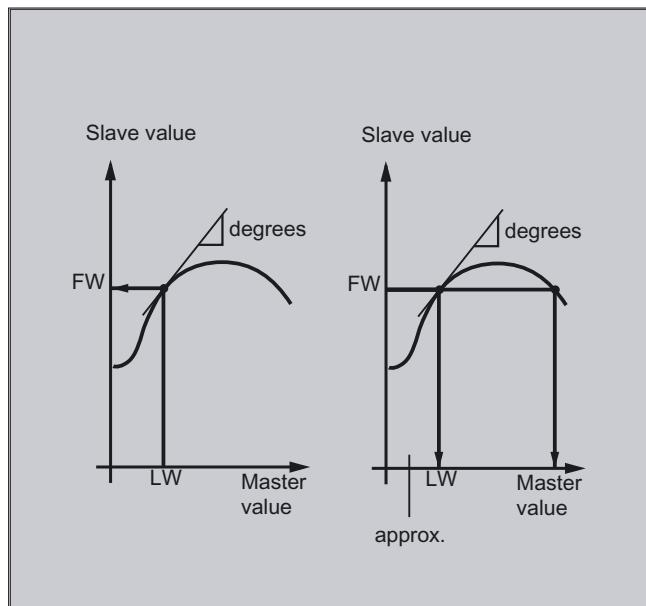
### Example of the use of CTABSSV and CTABSEV

Determining the curve segment belonging to master value X = 30.

Program code	Comments
N10 DEF REAL STARTPOS	
N20 DEF REAL ENDPOS	
N30 DEF REAL GRADIENT	
...	
N100 CTABDEF(Y,X,1,0)	; Beginning of table definition
N110 X0 Y0	; Starting position 1st table segment
N120 X20 Y10	; End position 1st table segment = start position 2nd table segment
...	
N130 X40 Y40	
N140 X60 Y10	
N150 X80 Y0	
N160 CTABEND	; End of table definition
...	
N200 STARTPOS = CTABSSV(30.0,1,GRADIENT)	; Start position Y in segment 2 = 10
...	
N210 ENDPOS = CTABSEV(30.0,1,GRADIENT)	; End position Y in segment 2 = 40 ; Segment 2 belongs to LV X = 30.0.

## Reading table positions, CTAB, CTABINV

CTABINV therefore requires an approximate value (`approxLV`) for the expected leading value. CTABINV returns the leading value that is closest to the approximate value. The approximate value can be the leading value from the previous interpolation cycle.



Both functions also output the gradient of the table function at the correct position to the gradient parameter (`degrees`). In this way, you can calculate the speed of the leading or following axis at the corresponding position.

### Note

#### CTAB, CTABINV, CTABSSV and CTABSEV

The language commands `CTAB`, `CTABINV` and `CTABSSV`, `CTABSEV` can be used directly from the part program or synchronized actions. Additional related information about programming synchronized actions is given in chapter, "Motion synchronous actions".

The optional specification of the leading or following axis for `CTAB/CTABINV/CTABSSV/CTABSEV` is important if the leading and following axes are configured in different length units.

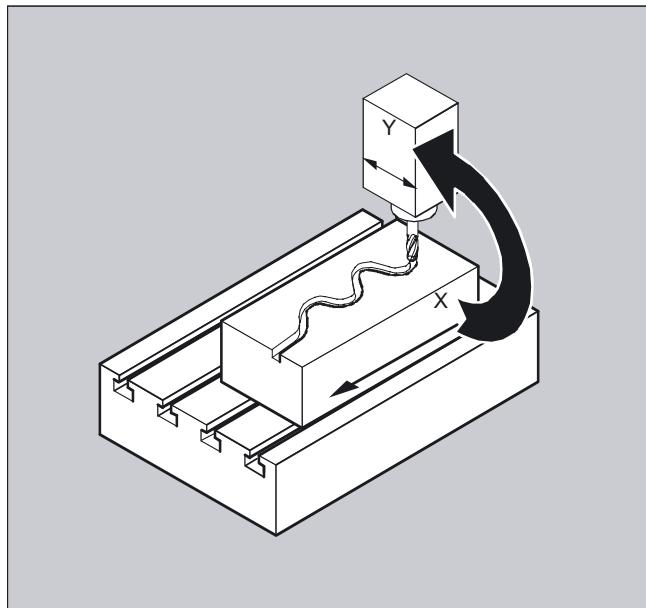
The language commands `CTABSSV` and `CTABSEV` are **not suitable** in the following cases to query programmed segments:

- Circles or involutes are programmed.
- Chamfer or rounding with `CHE`, `RND` is active.
- Corner rounding with `G643` is active.
- Compressor is, for example, active with `COMPON`, `COMPCURV`, `COMPCAD`.

## 9.3 Axial leading value coupling (LEADON, LEADOF)

### Function

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 the creation of curve tables, see section "Curve tables".

For the master value coupling, see /FB/, M3, Coupled-axis motion and master value coupling.

## Syntax

LEADON (FAxis, LAxis, n)  
LEADOF (FAxis, LAxis)

or deactivation without specifying the leading axis:

LEADOF (FAxis)

The master value coupling can be activated and deactivated both from the part program and during the movement from synchronized actions, see section "Motion synchronous actions".

## Significance

LEADON	Activate master value coupling
LEADOF	Deactivate master value coupling
Faxis	Following axis
Laxis	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 master value 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. To the control, everything that follows a line number constitutes a single line.

---

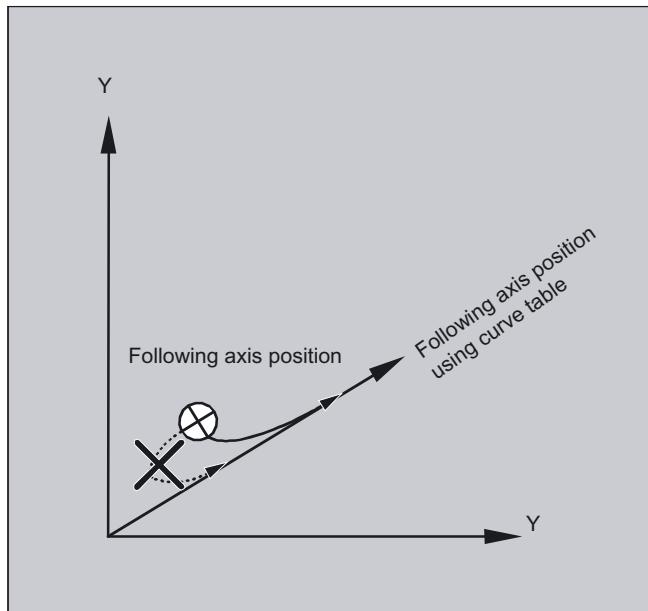
Program code	Comments
	; Defines all static synchronized actions.
	; *****Reset marker
N2 \$AC_MARKER[0]=0 \$AC_MARKER[1]=0 \$AC_MARKER[2]=0 \$AC_MARKER[3]=0 \$AC_MARKER[4]=0 \$AC_MARKER[5]=0 \$AC_MARKER[6]=0 \$AC_MARKER[7]=0	
	; ***** E1 0=>1 transfer ON
N10 IDS=1 EVERY (\$A_IN[1]==1) AND (\$A_IN[16]==1) 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).

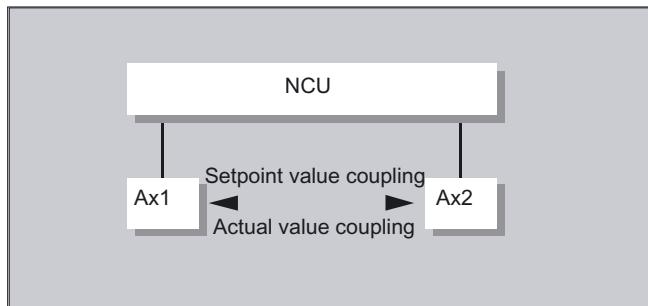


#### No synchronism

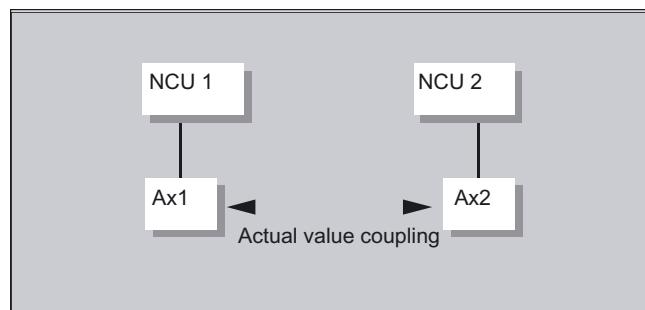
If the following axis position calculated moves away from the current following axis position when the master value coupling is activated, it is not possible to establish synchronization.

#### 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

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

## **9.4      Electronic gear (EG)**

### **Function**

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.

### 9.4.1 Defining an electronic gear (EGDEF)

#### Function

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.

#### Requirements

Prerequisites 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,...)
```

#### Significance

EGDEF	Definition of an electronic gear
Following axis	Axis that is influenced by the leading axes
Leading axis1	Axes that influence the following axis
‘...’	Leading axis5
Coupling type1	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.
<b>Value: Significance:</b>	
0	The following axis is influenced by the <b>actual value</b> of the corresponding leading axis.
1	The following axis is influenced by the <b>setpoint</b> 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**

<b>Program code</b>	<b>Comments</b>
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.

## **9.4.2 Switch-in the electronic gearbox (EGON, EGONSYN, EGONSYNE)**

**Function**

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])

## Significance

### Version 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	Counter for coupling factor i
N1, ... N5	Denominator for coupling factor i Coupling factor i = Counter 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.

### Version 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.

**Version 3:**

The parameters correspond to those of version 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****Version 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.

**Version 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.

**Version 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 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.) Nominator 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

### **9.4.3 Switching-in the electronic gearbox (EGOFS, EGOFC)**

#### **Function**

There are 3 different ways to switch-out an active EG axis group.

#### **Programming**

##### **Version 1:**

<b>Syntax</b>	<b>Description</b>
EGOFS (following axis)	The electronic gear is deactivated. The following axis is braked to a standstill. This call triggers a preprocessing stop.

##### **Version 2:**

<b>Syntax</b>	<b>Description</b>
EGOFS (following axis, leading axis1,...,leading axis5)	This command parameter setting made it possible to <b>selectively</b> 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.

##### **Version 3:**

<b>Syntax</b>	<b>Description</b>
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 version is only permitted for spindles.

---

#### 9.4.4 Deleting the definition of an electronic gear (EGDEL)

##### Function

An EG axis group must be switched-out before its definition can be deleted.

##### Programming

Syntax	Significance
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.

#### 9.4.5 Rotational feedrate (G95) / electronic gear (FPR)

##### Function

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.

## **9.5      Synchronous spindle**

### **Function**

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 parts program. Up to two synchronized spindle pairs can be operated simultaneously on each NC channel.

Refer to the parts program for the following coupling actions

- defined or changed
- activated
- deactivated
- deleted

from the parts program.

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.

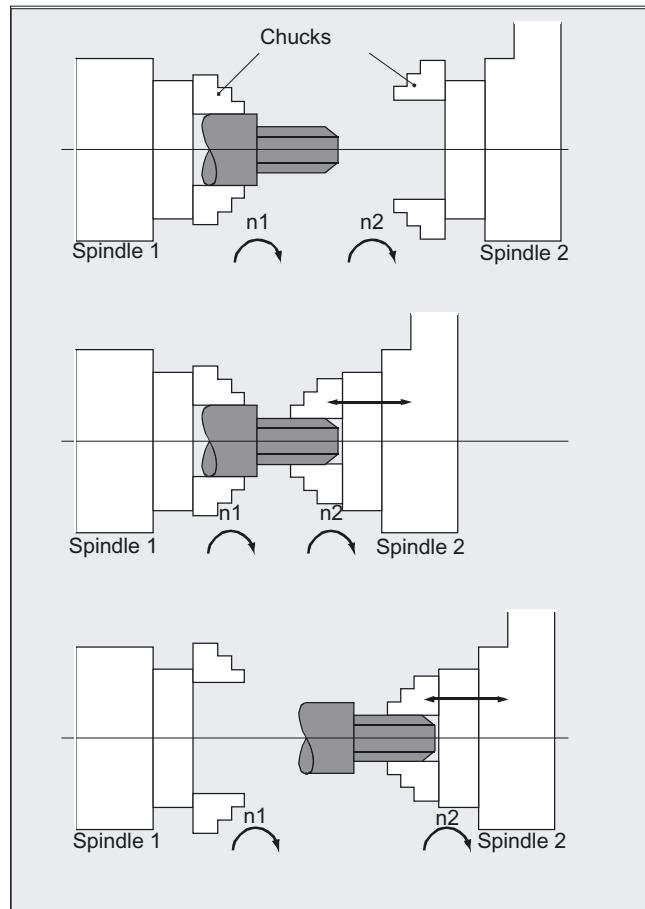
### 9.5.1 Synchronous spindle: Programming (COUPDEF, COUPDEL, COUPON, COUPONC, COUPOF, COUPOFS, COUPRES, WAITC)

#### Function

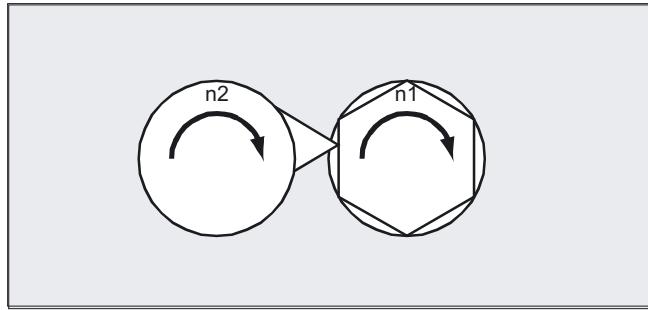
For lathes, the synchronous spindle function allows a flying workpiece transfer while changing from spindle 1 to spindle 2, e.g. for final machining – this avoids idle times as a result of re-clamping.

The transfer of the workpiece can be performed with:

- speed synchronism ( $n_{FS} = n_{LS}$ )
- position synchronism ( $\phi_{FS} = \phi_{LS}$ )
- position synchronism with angular offset ( $\phi_{FS} = \phi_{LS} + \Delta\phi$ )



Specification of a speed ratio  $SR_T$  between the main spindle and a "tool spindle" provides the prerequisite conditions for multi-edge machining (polygon turning).



## Syntax

```
COUPDEF(FS,LS,ÜFS,ÜLS,block behavior,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 behavior,LS,block behavior)
```

The reduced specification without the main spindle is also possible for:

```
COUPOF(FS), COUPOFS(FS), COUPRES(FS), COUPDEL(FS)
```

---

### Note

The following spindle and leading spindle must be programmed for each COUPDEF, COUPON and COUPONC instruction so that alarm messages are not triggered.

The other coupling parameters must only be programmed when they need to be changed. The last status remains applicable for non-specified parameters.

---

## Significance

COUPDEF	Define/change user coupling
COUPON	Activate coupling. The following spindle and main spindle are synchronized based on the current speed
COUPONC	Transfer 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. Block change as quickly as possible with immediate block change: COUPOF (S2, S1) Block change only after passing the: <ul style="list-style-type: none"><li>• Switch-off position: COUPOF (S2, S1, POSFS)</li><li>• Switch-off positions: COUPOF (S2, S1, POSFS, POSLS)</li></ul>
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
ÜFS, ÜLS	Speed ratio parameters for FS = numerator and LS = denominator Default setting = 1.0; specification of denominator optional
Block behavior	Block change behavior The block change is: <ul style="list-style-type: none"><li>"NOC"              Immediately</li><li>"FINE"             At "Synchronism fine"</li><li>"COARSE"           At "Synchronism coarse"</li><li>"IPOSTOP"          in response to IPOSTOP (e.g. after setpoint-based synchronism) (presetting)</li></ul>
Coupling type	The block change behavior is effective modally. Coupling type: Coupling between FS and LS <ul style="list-style-type: none"><li>"DV"              Setpoint linkage (default)</li><li>"AV"              Actual value coupling</li><li>"VV"              Speed coupling</li></ul>
POSFS	The coupling type is modal. Angle offset between leading and following spindles
POSFS, POSLS	Switch-off positions of the following and leading spindles "The block change is enabled once POS <sub>FS</sub> , POS <sub>LS</sub> has been passed"

**Examples****Example 1: Machining with leading and following spindles**

<b>Programming</b>	<b>Comments</b>
	; Leading spindle = master spindle = spindle 1
	; Following spindle = spindle 2
N05 M3 S3000 M2=4 S2=500	; ; Master spindle rotates at 3000 rpm; following spindle at 500 rpm
N10 COUPDEF(S2,S1,1,1,"NOC","Dv")	; Def. 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° overlay in negative direction
N210 WAITC(S2,"Fine")	; Wait for "fine" synchronism
N212 G1 X... Y... F...	; Machining
...	
N215 SPOS[2]=IC(180)	; Traverse with 180° 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 With configured velocity in the negative direction
...	
N350 COUPOF(S2,S1)	; Couple-out on-the-fly, S=S2=3000
N355 SPOS[2]=0	; Stop FS at zero degrees
N360 G0 X0 Y0	;
N365 WAITS(2)	; Wait for spindle 2
N370 M5	; Stop FS
N375 M30	

**Example 2: Programming the difference in speed**

Programming	Comments
	; Leading spindle = master spindle = spindle 1
N01 M3 S500	; Following spindle = spindle 2
N02 M2=3 S2=300	; Leading spindle rotates at 500 rpm
	; 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 main spindle and coupling factor
...	
N26 M2=3 S2=100	; Programming the difference in speed

**Example 3: Examples of transfer of a movement for difference in speed**

## 1. Activate coupling during previous programming of following spindle with COUPON

Programming	Comments
	; 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)	; Ratio of following spindle to leading spindle is 1.0 (presetting)
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**

<b>Programming</b>	<b>Comments</b>
	; 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)	; Ratio of following spindle to leading spindle is 1.0 (presetting)
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**

<b>Programming</b>	<b>Comments</b>
	; 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)	; Ratio of following spindle to leading spindle is 1.0 (presetting)
N20 COUPON(S2,S1)	; On-the-fly coupling to the leading spindle
N10 G4 F1	; Coupling is closed, ; S2 remains 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).

## Define synchronized spindle pair

Fixed definition of coupling:

The leading and following spindle are defined in machine data. With this coupling, the machine axes defined for the `LS` and `FS` cannot be changed from the NC parts program. The coupling can nevertheless be parameterized in the NC parts program by means of `COUPDEF` (on condition that no write protection is valid).

User-defined coupling:

The statement `COUPDEF` can be used to create new couplings and change existing ones in the NC parts programs. If a new coupling relationship is to be defined, any existing user-defined coupling must be deleted with `COUPDEL`.

## Define a new coupling COUPDEF

The following paragraphs define the parameters for the predefined subroutine:

```
COUPDEF(FS, LS, TFS, TLS, block behavior, coupling)
```

## Following and leading spindles, FS and LS

The axis names `FS` and `LS` are used to identify the coupling uniquely. They must be programmed for each `COUP` statement. Further coupling parameters only need to be defined if they are to be changed (modal scope).

Example:

```
N ... COUPDEF(S2, S1, TFS, TLS)
```

Meaning:

`S2` = following spindle, `S1` = leading spindle

### Speed ratio $SR_T$

The speed ratio is defined with parameters for  $FS$  (numerator) and  $LS$  (denominator).

Options:

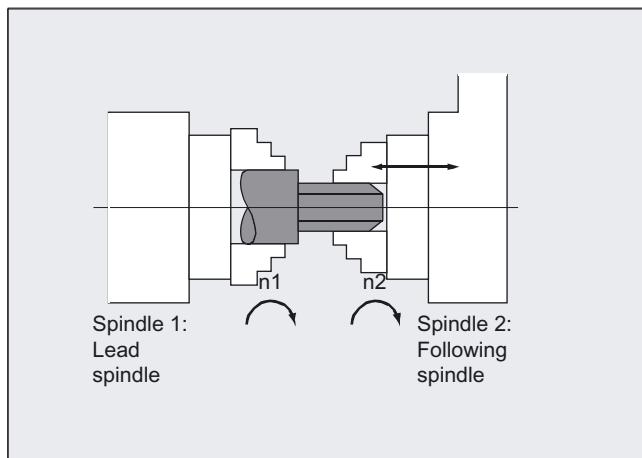
- Following and leading spindle rotate at the same speed ( $n_{FS} = n_{LS}$ ;  $SR_T$  positive)
- Rotation in the same or opposite direction ( $SR_T$  negative) between  $LS$  and  $FS$
- Following and leading spindles rotate at different speeds  
( $n_{FS} = SR_T \cdot n_{LS}$ ;  $SR_T \neq 1$ )

Application: Polygonal turning

Example:

```
N ... COUPDEF (S2, S1, 1.0, 4.0)
```

Meaning: the following spindle  $S_2$  and the leading spindle  $S_1$  rotate at a speed ratio of 0.25.



---

#### Note

The numerator must always be programmed. If no numerator is programmed, "1" is taken as the default.

---

The speed ratio can also be changed on-the-fly, when the coupling is active.

---

## Block change behavior NOC, FINE, COARSE, IPOSTOP

The following options can be selected during definition of the coupling to determine when the block change takes place:

"**NOC**" immediate (default)

"**FINE**" for "fine synchronism"

"**COARSE**" for "coarse synchronism"

"**IPOSTOP**" for **IPOSTOP** (i.e., after setpoint-based synchronism)

The block change response can be specified simply by writing the letters in bold print.

## Type of coupling DV, AV

Options:

"**DV**" setpoint coupling between **FS** and **LS** (default)

"**AV**" actual value coupling between **FS** and **LS**



### CAUTION

The coupling type may be changed only when the coupling is deactivated!

## Activate synchronized mode COUPON, POS<sub>FS</sub>

- Fastest possible activation of coupling with any angle reference between **LS** and **FS**:

N ... COUPON(S2, S1) or  
N ... COUPON(S2, S1, POS<sub>FS</sub>) or  
N ... COUPON(S2)

- Activation with angular offset **POS<sub>FS</sub>**

Position-synchronized coupling for profiled workpieces.

**POS<sub>FS</sub>** refers to the 0° position of the lead spindle in the positive direction of rotation  
**POS<sub>FS</sub>** value range: 0° ... 359,999°:

COUPON(S2, S1, 30)

You can use this method to change the angle offset even when the coupling is already active.

### **Position the following spindle**

When the synchronized spindle coupling is active, following spindles can also be positioned within the  $\pm 180^\circ$  range independently of the motion initiated by the master spindle.

### **Positioning SPOS**

The following spindle can be interpolated with **SPOS=...**. Please refer to Programming Manual "Fundamentals" for more information about **SPOS**.

**Example:**

N30 SPOS[2]=IC(-90)

### **Difference in speed M3 S... or M4 S...**

A difference in speed results from signed superimposition of two sources of speed and is programmed again for the following spindle e.g. where  $S_n=...$  or  $M_n=3$ ,  $M_n=4$  in speed control mode during an active synchronized spindle coupling. During the process, this speed component is derived from the main spindle using the coupling factor and the following spindle added to this with the correct prefix.

---

#### **Note**

When the direction of rotation is M3 or M4, the speed S... also has to be reprogrammed because otherwise an alarm is triggered to report missing programming.

For more information on difference in speed, see

**References:** /FB2/ Function Manual, Extension Functions; Synchronized Spindle (S3).

---

### **Difference in speed for COUPONC**

#### **Transfer of a movement for difference in speed**

The previous programming of M3 S... or M4 S... of the following spindle is superimposed by activating a synchronized coupling with **COUPONC**. The spindle speed previously programmed into a separate block is then retained when the coupling is activated. The difference in speed is transferred immediately.

---

**Note****Enabling difference in speed**

The difference in speed produced is only transferred when superimposition of the movement is also enabled. Otherwise a self-canceling alarm signals this impermissible superimposition.

---

**Dynamic response distribution on the available motor dynamic response**

The dynamic response to be limited for the main spindle must be limited by programming such that another movement component does not restrict the dynamic response of the following spindle to an impermissible extent e.g. as a result of the difference in speed.

**FA, ACC, OVRA, VELOLIMA: Velocity, acceleration**

FA[SPI] (Sn) or FA[Sn], ACC[SPI(Sn)] or ACC[Sn] and OVRA[SPI(n)] or OVRA[Sn] as well as VELOLIMA[Sn] can be used to program the positioning speeds and acceleration values for following spindles (refer to the Programming Manual, Fundamentals). "n" stands for spindle number 1...n.

The programmable ranges of values for the dynamic response offset of the following spindle Sn act on

- the feed for positioning axles or spindles in position mode  
FA[Sn] = ... to 999 999.999 mm/min or degrees/min
  - the percentage acceleration correction ACC[Sn] = 1 to 200%
  - the percentage feed correction OVRA[Sn] = ... to 200%
  - the speed component VELOLIMA[Sn] = percentage speed correction of maximum speed of between 1 and 100%
- 

**Note****Acceleration component JERKLIMA[Sn]**

The jerk offset may be specified but does not impact on spindles.

For further information on configuring the dynamic response programming using machine data, see **Reference: /FB2/ Function Manual, Extension Functions; Round Axes (R2)**.

---

### **Programmable block change WAITC**

WAITC can be used to define the block change behavior with various synchronism conditions (coarse, fine, IPOSTOP) for continuation of the program, e.g., after changes to coupling parameters or positioning operations. WAITC causes a delay in the insertion of new blocks until the appropriate synchronism condition is fulfilled, thereby allowing the synchronized state to be processed faster. If no synchronism conditions are specified, then the block change behavior programmed/configured for the relevant coupling applies.

Examples:

N200 WAITC

Wait for synchronism conditions for all active slave spindles without specification of these conditions.

N300 WAITC(S2, "FINE", S4, "COARSE")

Wait for the specified "Coarse" synchronism conditions for slave spindles S2 and S4.

### **Deactivate synchronous mode COUPOF**

Three variants are possible:

- For the fast possible activation of the coupling and immediate enabling of the block change:  
COUPOF(S2, S1) or  
COUPOF(S2); without specification of the main spindle
- After the deactivation positions have been crossed; the block change is not enabled until the deactivation positions POS<sub>FS</sub> and, where appropriate, POS<sub>LS</sub> have been crossed.  
Value range 0° ... 359.999°:  
COUPOF(S2, S1, 150)  
COUPOF(S2, S1, 150, 30)

### **Deactivating a coupling with stop of following spindle COUPOFS**

Two versions are possible:

- For fastest possible activation of the coupling and stop without position data, and immediate enabling of the block change:  
COUPOFS(S2, S1)
- After the programmed following axis deactivation position that is relative to the machine coordinate system has been crossed, the block change is not enabled until the deactivation positions POS<sub>FS</sub> have been crossed.  
Value range 0° ... 359.999°:  
COUPOFS(S2, S1, POS<sub>FS</sub>)

## Delete couplings COUPDEL

N ... COUPDEL{S2}, S1} or  
N ... COUPDEL{S2}; without specification of the main spindle

impacts on an active synchronized spindle coupling, deactivates the coupling and deletes the coupling data. The following spindle takes over the last speed and its behavior is the same as that of the COUPOF(FS, LS) previously.

## Reset coupling parameters, COUPRES

Statement "COUPRES" is used to

- activate the parameters stored in the machine data and setting data (permanently defined coupling) and
- activate the presets (user-defined coupling).

The parameters programmed with COUPDEF (including the transformation ratio) are subsequently deleted.

N ... COUPRES{S2}, S1} or  
N ... COUPRES{S2}; without specification of the main spindle

S2 = following spindle, S1 = leading spindle

## System variables

### Current coupling status following spindle

The current coupling status of the following spindle can be read in the NC parts program with the following axial system variable:

\$AA\_COUP\_ACT[FS]

FS = axis name of the following spindle with spindle number, e.g., S2.

The value read has the following significance for the following spindle:

0: No coupling active

4: Synchronous spindle coupling active

#### Current angular offset

The setpoint of the current position offset of the **FS** to the **LS** can be read in the parts program with the following axial system variable:

**\$AA\_COUP\_OFFSET [S2]**

The actual value for the current position offset can be read with:

**\$VA\_COUP\_OFFSET [S2]**

**FS** = axis name of the following spindle with spindle number, e.g., **S2**.

---

#### Note

When the controller has been disabled and subsequently re-enabled during active coupling and follow-up mode, the position offset when the controller is re-enabled is different to the original programmed value. In this case, the new position offset can be read and, if necessary, corrected in the NC parts program.

---

## 9.6 Master/slave group (**MASLDEF**, **MASLDEL**, **MASLON**, **MASLOF**, **MASLOFS**)

### Function

The master/slave coupling in SW 6.4 and lower permitted coupling of the slave axes to their master axis only while the axes involved are stopped.

Extension of SW 6.5 permits coupling and uncoupling of **rotating**, speed-controlled spindles and dynamic configuration.

### Syntax

<b>MASLON(Slv1,Slv2,..., )</b>	
<b>MASLOF(Slv1,Slv2,..., )</b>	
<b>MASLDEF(Slv1,Slv2,..., master axis)</b>	Extension for dynamic configuration
<b>MASLDEL(Slv1,Slv2,..., )</b>	Extension for dynamic configuring extension
<b>MASLOFS(Slv1, Slv2, ..., )</b>	Extension for slave spindle

**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.

**Significance****General**

<code>MASLON</code>	Close (switch-in) a temporary coupling.
<code>MASLOF</code>	Open an active coupling. The extensions for spindles must be observed.

**Dynamic configuration extension**

<code>MASLDEF</code>	Coupling user-defined using machine data or also create/change from the part program.
<code>MASLOFS</code>	Open the coupling analog to <code>MASLOF</code> and automatically brake the slave spindle.
<code>MASLDEL</code>	Uncouple master/slave axis group and delete group definition.
<code>Slv1, Slv2, ...</code>	Slave axes led by a master axis.
<code>Master axis</code>	Axis, that controls defined slave axes in a master/slave group.

**Examples****Example 1: Dynamic configuration of a master/slave coupling**

Dynamic configuration of a master/slave coupling from the part program:

The axis relevant after axis container rotation must become the master axis.

Program code	Comments
<code>MASLDEF(AUX,S3)</code>	; S3 master for AUX
<code>MASLON(AUX)</code>	; Close coupling for AUX
<code>M3=3 S3=4000</code>	; Clockwise direction of rotation
<code>MASLDEL(AUX)</code>	; Delete configuration and open the coupling
<code>AXCTSWE(CT1)</code>	; Container rotation

**Examples****Example 2: Actual-value coupling of a slave axis**

Actual value coupling of a slave axis to the same value of the master axis using PRESETON.

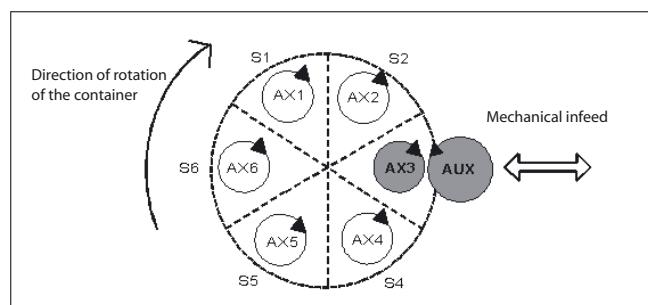
For a permanent master/slave coupling, the actual value at the SLAVE axis is to be changed using PRESETON.

Program code	Comments
N37262 \$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=0	; Briefly switch-out the permanent coupling.
N37263 NEWCONF	
N37264 STOPRE	
MASLOF(Y1)	; Temporary coupling open.
N5 PRESETON(Y1,0,Z1,0,B1,0,C1,0,U1,0)	; Set the actual value of the non-referenced slave axes as these are activated with power on.
N37262 \$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=1	; Activate permanent coupling.
N37263 NEWCONF	

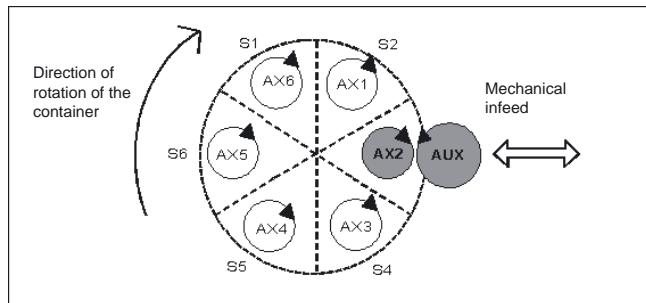
**Example 3: Coupling sequence, position 3/container CT1**

To enable coupling with another spindle after container rotation, the previous coupling must be uncoupled, the configuration cleared, and a new coupling configured.

Initial situation:



After rotation by one slot:



### References:

Function Manual, Extension Functions; Several Operator Panel Fronts and NCUs (B3), Chapter "Axis container" "Axis container"

## Further Information

### General

**MASLOF** This instruction is executed directly for spindles in speed control mode. The slave spindles rotating at this instant keep their speeds until a new speed is programmed.

### Dynamic configuration extension

**MASLDEF** Definition of a master/slave group from the part program. Beforehand, the definition was made exclusively using machine data.

**MASLDEL** The instruction cancels assignment of the slave axes to the master axis and simultaneously opens the coupling, like MASLOF.

The master/slave definitions specified in the machine data are kept.

**MASLOFS** MASLOFS can be used to automatically brake slave spindles when the coupling is opened.

For axes and spindles in the positioning mode, the coupling is only closed and opened at standstill (zero speed).

---

**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-out 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 using the MD setting MD37262 \$MA\_MS\_COUPLING\_ALWAYS\_ACTIVE = 1 – and this has no effect on the language commands of the temporary coupling.

---

**Coupling behavior for spindles!**

For spindles in the open-loop speed controlled mode, the coupling behavior of MASLON, MASLOF, MASLOFS and MASLDEL are explicitly defined using 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. The slave spindles rotating at this instant in time are for MASLOFS automatically braked and for MASLOF keep their speed until a new speed is programmed.

# 10

## Motion synchronous actions

### 10.1 Basics

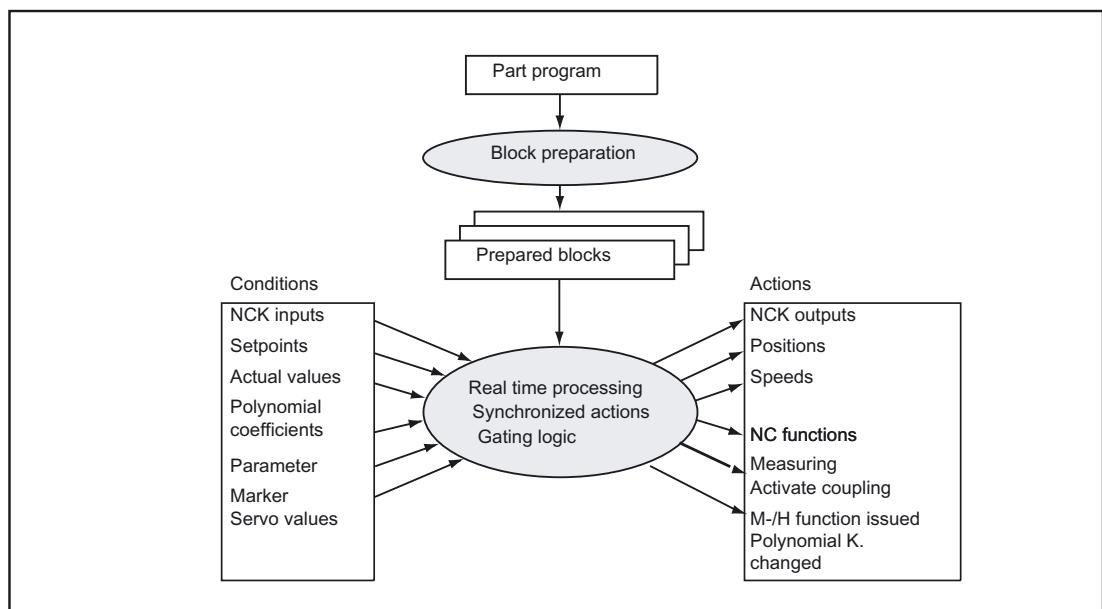
#### Function

Synchronized actions allow actions to be executed such that they are synchronized to machining blocks.

The time at which the actions are executed can be defined by conditions. The conditions are monitored in the interpolation cycle. The actions are therefore responses to real-time events, their execution is not limited by block boundaries.

A synchronized action also contains information about its service life and about the frequency with which the programmed main run variables are scanned and therefore about the frequency with which the actions are started. In this way, an action can be triggered just once or cyclically in interpolation cycles.

#### Possible applications

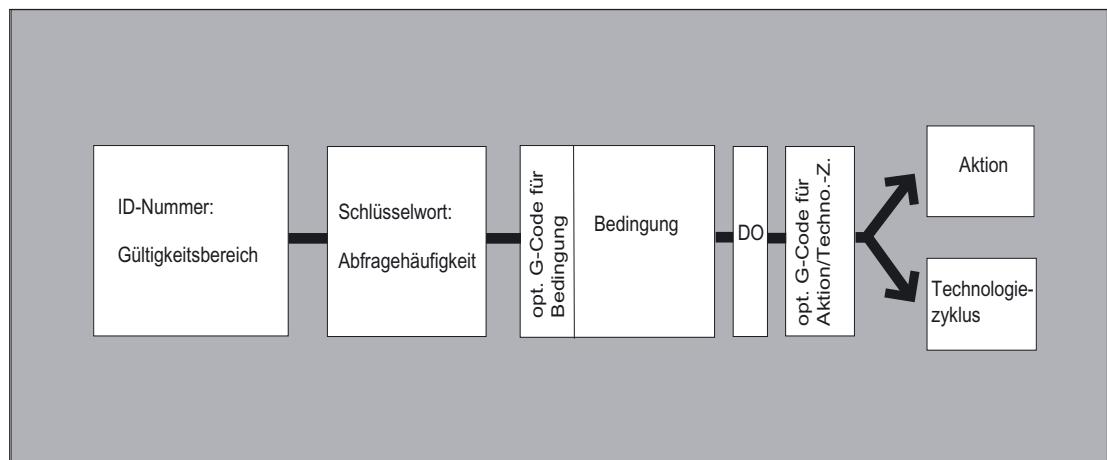


- Optimizing runtime-critical applications (e.g. a tool change)
- Fast response to an external event
- Programming AC controls
- Setting up safety functions
- ....

## Programming

A synchronized action is programmed on its own in a separate block and triggers a machine function as of the next executable block (e.g. traversing movement with G0, G1, G2, G3).

Synchronized actions comprise up to 5 command elements with different tasks:



### Syntax:

```
DO <action1> <action2> ...
<KEYWORD> <condition> DO <action1> <action2> ...
ID=<n> <KEYWORD> <condition> DO <action1> <action2> ...
IDS=<n> <KEYWORD> <condition> DO <action1> <action2> ...
```

**Significance:**

DO	Instruction to initiate the programmed action(s) Only effective if the <condition> is fulfilled (if programmed). → See " Actions "
<action1> <action2> ...	Action(s) to be started Examples: <ul style="list-style-type: none"><li>• Assign variable</li><li>• Start technology cycle</li></ul>
<KEYWORD>	The cyclic check of the <condition> of a synchronized action is defined using the keyword (WHEN, WHENEVER, FROM or EVERY). → See " Cyclic checking of the condition "
<condition>	Gating logic for main run variables
ID=<n> or IDS=<n>	The condition is checked in the IPO clock cycle Identification number The area of validity and the position within the machining sequence is defined using the identification number. → See " area of validity and machining sequence "

**Coordinating synchronized actions/technology cycles**

The following commands are available to coordinate synchronized actions/technology cycles:

Command	Significance
CANCEL (<n>)	Cancel synchronized actions → See " Cancel synchronized actions "
LOCK (<n>)	Disable synchronized actions
UNLOCK (<n>)	Unlock synchronized actions
RESET	Reset technology cycle Regarding LOCK, UNLOCK and RESET: → see " Inhibiting, enabling, resetting "

## Example

Program code	Comments
WHEN \$AA_IW[Q1]>5 DO M172 H510	; If the actual value of axis Q1 exceeds 5 mm, auxiliary functions M172 and H510 are output at the PLC interface.

### 10.1.1 Area of validity and machining sequence (ID, IDS)

#### Function

##### Area of validity

The area of validity of a synchronized action is defined using the identification ID or IDS:

- |              |   |
|--------------|---|
| No modal ID: | Non-modal synchronized actions in automatic mode  |
| ID:          | ID=n modal synchronized actions in the automatic mode up to the end of program                            |
| IDS:         | Static synchronized action, effective modally in every operating mode, also beyond the end of the program |

##### Applications

- AC loops in JOG mode
- Logic operations for Safety Integrated
- Monitoring functions, responses to machine states in all modes

##### Sequence of execution

Modal synchronized actions that are statically effective are processed in the interpolation clock cycle in the sequence of their ID or IDS number (ID=<n> or IDS=<n>).

Non-modal synchronized actions (without ID number) are executed in the programmed sequence after execution of the modal synchronized actions.

---

##### Note

Modal synchronized actions can be protected from modifications or deletions by machine data settings (→ machinery construction OEM!).

---

## Programming

Syntax	Significance
No modal ID	<p>The synchronized action is only effective in the <b>automatic mode</b>. It applies only to the next executable block (block with motion statement or other machine action), is <b>non-modal</b>.</p> <p><b>Example:</b>  <code>WHEN \$A_IN[3]==TRUE DO \$A_OUTA[4]=10</code></p>
ID=<n> ...	<p>The synchronized action applies <b>modally</b> in the following blocks and can be deactivated by <code>CANCEL(&lt;n&gt;)</code> or can be overwritten by programming a new synchronized action with the same ID.</p> <p>The synchronized actions active in the <code>M30</code> block delay the program end.</p> <p>ID synchronized actions are only effective in the <b>automatic mode</b>.</p> <p>Value range for &lt;n&gt;: 1 ... 255</p> <p><b>Example:</b>  <code>ID=2 EVERY \$A_IN[1]==1 DO POS[X]=0</code></p>
IDS=<n>	<p>The static synchronized actions act <b>modally in all modes</b>. They even remain active beyond the end of the program and can be activated directly after Power On using an ASUB. This means that it is possible to activate actions that should run in the NC independent of the selected operating mode.</p> <p>Value range for &lt;n&gt;: 1 ... 255</p> <p><b>Example:</b>  <code>IDS=1 EVERY \$A_IN[1]==1 DO POS[X]=100</code></p>

**10.1.2 Cyclically checking the condition (WHEN, WHENEVER, FROM, EVERY)****Function**

A keyword is used to define cyclic checking of the condition of a synchronized action. If no keyword is programmed, the actions of the synchronized action is performed once in every IPO cycle.

**Keywords**

No keyword	Execution of the action is not subject to any condition. The action is executed cyclically in any interpolation cycles.
WHEN	The condition is scanned in each interpolation cycle until it is fulfilled once, whereupon the associated action is executed once.
WHENEVER	The condition is checked in cycles in each interpolation cycle. The associated action is executed in each interpolation cycle while the condition is fulfilled.
FROM	The condition is checked in each interpolation cycle until it is fulfilled once. The action is then executed while the synchronous action is active, i.e. even if the condition is no longer fulfilled.
EVERY	The condition is scanned in each interpolation cycle. The action is executed once when the condition is fulfilled. Edge triggering: the action is executed again when the condition changes from the FALSE state to the TRUE state.

**Main run variables**

The variables used are evaluated in the interpolation cycle. Main run variables in synchronized actions do not trigger a preprocessing stop.

Analysis:

If main run variables occur in a part program (e.g. actual value, position of a digital input or output etc.), preprocessing is stopped until the previous block has been executed and the values of the main run variables obtained.

## Examples

### Example 1: No keyword

Program code	Comments
DO \$A_OUTA[1]==\$AA_IN[X]	; Actual value output to analog output.

### Example 2: WHENEVER

Program code	Comments
WHENEVER \$AA_IM[X] > 10.5*SIN(45) DO ...	; Comparison with an expression calculated during preprocessing
WHENEVER \$AA_IM[X] > \$AA_IM[X1] DO ...	; Comparison with other main run variable.
WHENEVER (\$A_IN[1]==1) OR (\$A_IN[3]==0) DO ...	; Two comparison operations that are gated with one another.

### Example 3: EVERY

Program code	Comments
ID=1 EVERY \$AA_IM[B]>75 DO POS[U]=IC(10) FA[U]=900	; Always when the actual value of axis B exceeds the value 75 in machine coordinates, the U axis should move forwards by 10 with an axial feed.

## Further Information

### Conditions

The condition is a logical expression which can be built up in any way using Boolean operators. Boolean expressions should always be given in brackets.

The condition is checked in the interpolation cycle.

A G code can be given before the condition. This allows defined settings to exist for the evaluation of the condition and the action/technology cycle to be executed, independent of the current part program status. It is necessary to separate the synchronized actions from the program environment, because synchronized actions are required to execute their actions at any time from a defined initial state as a result of fulfilled trigger conditions.

### Applications

Definition of the systems of measurement for condition evaluation and action through G codes G70, G71, G700, G710.

A G code specified for the condition is valid for the evaluation of the condition and for the action if no separate G code is specified for the action.

Only **one G code** of the G code group may be programmed for each part of the condition.

### Possible conditions

- Comparison of main run variables (analog/digital inputs/outputs, etc.)
- Boolean gating of comparison results
- Computation of real-time expressions
- Time/distance from beginning of block
- Distance from block end
- Measured values, measurement results
- Servo values
- Velocities, axis status

### 10.1.3 Actions (DO)

#### Function

In synchronized actions, you can program one or more actions. All actions programmed in a block are active in the same interpolation cycle.

#### Syntax

```
DO <action1> <action2> ...
```

#### Significance

DO	Initiates an action or a technology cycle when the condition is satisfied.
<action>	Action started if the condition is fulfilled, e.g., assign variable, activate axis coupling, set NCK outputs, output M, S and H functions, specify the programmed G code, ...

The **G codes** can be programmed in synchronized actions for the actions/technology cycles. The G code may specify a different G code from the condition for all actions in the block and technology cycles. If technology cycles are contained in the action part, the G code remains modally active for all actions until the next G code, even after the technology cycle has been completed.

Only **one G code** of the G code group (G70, G71, G700, G710) may be programmed per action section.

#### Example: Synchronized action with two actions

Program code	Comments
WHEN \$AA_IM[Y]>=35.7 DO M135 \$AC_PARAM=50	; If the condition is fulfilled, M135 is output at the PLC and the override is set to 50%.

## 10.2 Operators for conditions and actions

Comparison (==, <>, <, >, <=, >=)	Variables or partial expressions can be compared in conditions. The result is always of data type BOOL. All the usual comparison operators are permissible.
Boolean operators (NOT, AND, OR, XOR)	Variables, constants or comparisons can be linked with each other with the usual Boolean operators.
Bit-by-bit operators (B_NOT, B_AND, B_OR, B_XOR)	The bit operators B_NOT, B_AND, B_OR, B_XOR can be used.
Basic arithmetic operations (+, -, *, /, DIV, MOD)	Main run variables can be linked to one another or to constants by forms of basic computation.
Mathematical functions (SIN, COS, TAN, ASIN, ACOS, ABS, TRUNC, ROUND, LN, EXP, ATAN2, POT, SQRT, CTAB, CTABINV).	Mathematical functions cannot be applied to variables of data type REAL .
Indexing	Indexing can be undertaken using main run expressions.

### Example

- Basic arithmetic operations used together

Multiplication and division are performed before addition and subtraction and bracketing of expressions is permissible. The operators DIV and MOD are permissible for the data type REAL.

Programming	Comments
DO \$AC_PARAM[3] = \$A_INA[1]-\$AA_IM[Z1]	; ;Subtraction of two ;Main run variables
WHENEVER \$AA_IM[x2] < \$AA_IM[x1]-1.9 DO \$A_OUT[5] = 1	; ;Subtraction of a constant from variables
DO \$AC_PARAM[3] = \$INA[1]-4*SIN(45.7 \$P_EP[Y])*R4	;Constant expression, calculated during preprocessing

- Mathematical functions

Programming	Comments
DO \$AC_PARAM[3] = COS(\$AC_PARAM[1]) ;	;

- Real-time expressions

Programming	Comments
ID=1 WHENEVER (\$AA_IM[Y]>30) AND (\$AA_IM[Y]<40)	; Selecting a position window
DO \$AA_OVR[S1]=80	
ID=67 DO \$A_OUT[1]=\$A_IN[2] XOR \$AN_MARKER[1]	; Evaluate 2 boolean signals
ID=89 DO \$A_OUT[4]=\$A_IN[1] OR (\$AA_IM[Y]>10)	; Output the result of a comparison

- Main run variable indexed

Programming	Comments
WHEN...DO \$AC_PARAM[\$AC_MARKER[1]] = 3	;
Illegal	;
\$AC_PARAM[1] = \$P_EP[\$AC_MARKER]	;

## 10.3 Main run variables for synchronized actions

### 10.3.1 System variables

#### Function

NC data can be read and written with the help of system variables. A distinction is made between preprocessing and main run system variables. Preprocessing variables are always executed at the preprocessing time. Main run variables always calculate their value with reference to the current main run status.

**Name**

Generally, the system variable names start with a "\$" character:

**Preprocessing variables:**

\$M...	Machine data
\$S...	Setting data, protection zones
\$T...	Tool management data
\$P...	Programmed values, preprocessing data
\$C...	Cycle variables of the ISO envelope cycles
\$O...	Option data
R ...	R parameters

**Main run variables:**

\$\$A...	Actual main run data
\$\$V...	Servo data
\$R...	R parameters

A 2nd letter describes options for accessing the variable:

N...	NCK global value (generally valid value)
C...	Channel-specific value
A...	Axis-specific value

The 2nd letter is usually only used for main run variables. Preprocessing variables, such as e.g. \$P\_, are usually executed without the 2nd letter.

An underscore and the subsequent variable name (usually an English designation or abbreviation) follow the prefix (\$ followed by one or two letters).

## Data types

Main run variables can feature the following data types:

INT	Integer for whole values with prefix signs
REAL	Real for rational counting
BOOL	Boolean TRUE and FALSE
CHAR	ASCII character
STRING	Character string with alpha-numerical characters
AXIS	Axis addresses and spindles

Preprocessing variables can also feature the following data types:

FRAME	Coordinate transformations
-------	----------------------------

## Variable arrays

System variables can be set-up as 1 to 3-dimensional arrays.

The following data types are supported: BOOL, CHAR, INT, REAL, STRING, AXIS

The data type of the indices can be either type INT or AXIS, whereby this can be sorted as required.

STRING variables can only be set-up in 2-dimensions.

Example for array definitions:

```
DEF BOOL $AA_NEVAR[x,y,2]
DEF CHAR $AC_NEVAR[2,2,2]
DEF INT $AC_NEVAR[2,10,3]
DEF REAL $AA_VECTOR[x,y,z]
DEF STRING $AC_NEVSTRING[3,3]
DEF AXIS $AA_NEWAX[x,3,y]
```

---

### Note

3-dimensional system variables can be displayed without any restrictions if there is a OPI variable for the system variables

---

### 10.3.2 Implicit type conversion

#### Function

During value assignments and parameter transfers, variables of different data types are assigned or transferred.

The implicit type conversion triggers an internal type conversion of values.

#### Possible type conversions

To	REAL	INT	BOOL	CHAR	STRING	AXIS	FRAME
from							
REAL	Yes	yes*	Yes <sup>1)</sup>	-	-	-	-
INT	Yes	Yes	Yes <sup>1)</sup>	-	-	-	-
BOOL	Yes	Yes	Yes	-	-	-	-

#### Explanations

- \* At type conversion from REAL to INT, fractional values that are  $\geq 0.5$  are rounded up, others are rounded down (cf. ROUND function).  
An alarm is output if values are exceeded.
- 1) Value  $\leftrightarrow 0$  is equivalent to TRUE; value  $\neq 0$  is equivalent to FALSE

#### Results

```
Type conversion from REAL or INTEGER to BOOL
Result BOOL = TRUE           if the REAL or INTEGER value does not equal zero
Result BOOL = FALSE          if the REAL or INTEGER value equals zero

Type conversion from BOOL to REAL or INTEGER
Result REAL TRUE             if the BOOL value = TRUE (1)
Result INTEGER = TRUE         if the BOOL value = TRUE (1)

Type conversion from BOOL to REAL or INTEGER
Result REAL FALSE            if the BOOL value = FALSE (0)
Result INTEGER = FALSE        if the BOOL value = FALSE (0)
```

## Examples of implicit type conversions

```

Type conversion from INTEGER to BOOL
$AC_MARKER[1]=561
ID=1 WHEN $A_IN[1] == TRUE DO $A_OUT[0]=$AC_MARKER[1]

Type conversion from REAL to BOOL
R401 = 100.542
WHEN $A_IN[0] == TRUE DO $A_OUT[2]=$R401

Type conversion from BOOL to INTEGER
ID=1 WHEN $A_IN[2] == TRUE DO $AC_MARKER[4] = $A_OUT[1]

Type conversion from BOOL to REAL
R401 = 100.542
WHEN $A_IN[3] == TRUE DO $R10 = $A_OUT[3]

```

### 10.3.3 GUD variables

#### Function

In addition to the system variables, the programmer can use special GUD variables in synchronized actions. The variables are displayed on HMI in the operating area parameter and can be used in Wizard as well as in the variable view and in the variable protocol.

#### Configurable parameter areas

##### Machine manufacturer

Machine data can be used to extend the individual GUD blocks by additional channel-specific parameter areas of data types REAL, INT, BOOL, AXIS, CHAR and STRING; these can be read and written to from the part program as well as also via synchronized actions.

The parameters are available during the next control power up once the corresponding machine data has been set.

To configure the related machine data, refer to the machine manufacturer's specifications.

## Default variables

### Note

Even if no GUD definition files are active, machine data can be used to read defined new parameters in the relevant HMI GUD module.

List of predefined variable names			
Name of the Synact GUD			
of data type <b>REAL</b>	of data type <b>INT</b>	of data type <b>BOOL</b>	in module
SYG_RS[ ]	SYG_IS[ ]	SYG_BS[ ]	SGUD module
SYG_RM[ ]	SYG_IM[ ]	SYG_BM[ ]	MGUD module
SYG_RU[ ]	SYG_IU[ ]	SYG_BU[ ]	UGUD module
SYG_R4[ ]	SYG_I4[ ]	SYG_B4[ ]	GUD4 module
SYG_R5[ ]	SYG_I5[ ]	SYG_B5[ ]	GUD5 module
SYG_R6[ ]	SYG_I6[ ]	SYG_B6[ ]	GUD6 module
SYG_R7[ ]	SYG_I7[ ]	SYG_B7[ ]	GUD7 module
SYG_R8[ ]	SYG_I8[ ]	SYG_B8[ ]	GUD8 module
SYG_R9[ ]	SYG_I9[ ]	SYG_B9[ ]	GUD9 module

List of predefined variable names			
Name of the synact GUD			
of data type <b>AXIS</b>	of data type <b>CHAR</b>	of data type <b>STRING</b>	in the block
SYG_AS[ ]	SYG_CS[ ]	SYG_SS[ ]	SGUD block
SYG_AM[ ]	SYG_CM[ ]	SYG_SM[ ]	MGUD block
SYG_AU[ ]	SYG_CU[ ]	SYG_SU[ ]	UGUD block
SYG_A4[ ]	SYG_C4[ ]	SYG_S4[ ]	GUD4 block
SYG_A5[ ]	SYG_C5[ ]	SYG_S5[ ]	GUD5 block
SYG_A6[ ]	SYG_C6[ ]	SYG_S6[ ]	GUD6 block
SYG_A7[ ]	SYG_C7[ ]	SYG_S7[ ]	GUD7 block
SYG_A8[ ]	SYG_C8[ ]	SYG_S8[ ]	GUD8 block
SYG_A9[ ]	SYG_C9[ ]	SYG_S9[ ]	GUD9 block

### Note

STRING type variables in synchronized actions have a fixed length of 32 characters.

- Array size corresponding to <value> of the machine data.
- Predefined names in accordance with previous list of predefined variable names.
- Access via HMI in the same way as access to the GUDs created using the definition file.
- The protection level assignments which are already possible in a GUD definition file using keywords APR and APW remain valid and only relate to the GUDs defined in these GUD definition files.
- Deletion behavior: If the content of a particular GUD definition file is re-activated, the old GUD data block in the memory of the active file system is deleted first. The new parameters are also reset at the same time. This process is also possible using the HMI in the operator area "Services" > "Define and activated user data (GUD)".

## Variable arrays

GUD or LUD variables can be set-up as 1 to 3-dimensional arrays.

The following data types are supported: BOOL, CHAR, INT, REAL, STRING, AXIS, FRAME

The data type of the indices is restricted to INT for GUD and LUD variables.

STRING variables can only be set-up in 2-dimensions.

Example for array definitions:

```
DEF BOOL bname3[3,4,3]
DEF CHAR cname3[3,3,3]
DEF INT iname3[5,5,5]
DEF REAL rname3[5,5,5]
DEF STRING[10] sname2[3,3]
DEF AXIS aname3[5,5,5]
DEF FRAME fname3[3,2,2]
```

---

### Note

It is only possible to display array elements of 2 and 3-dimensional variables with some restrictions.

---

### 10.3.4 Default axis identifier (NO\_AXIS)

#### Function

AXIS type variables or parameters which have not been initialized by a value can be provided with defined default axis identifiers. Undefined axis variables are initialized with this default value.

Non-initialized valid axis names are recognized in synchronized actions by querying the "NO\_AXIS" variable. This non-initialized axis identifier is assigned the configured default axis identifier by machine data.

#### Machine manufacturer

At least one valid existing axis identifier must be defined and pre-assigned using machine data. All existing valid axis identifiers can however be pre-assigned. Please refer to the machine manufacturer's instructions.

---

#### Note

During definition, newly created variables are now automatically given the value saved in the machine data for default axis names.

For additional information on a definition applicable via machine data, see:

#### References:

/FBSY/ Function Manual Synchronized Actions

---

#### Syntax

```
PROC UP(AXIS PAR1=NO_AXIS, AXIS PAR2=NO_AXIS)  
IF PAR1 <>NO_AXIS...
```

#### Significance

PROC	Subprogram definition
SR	Subprogram name for recognition
PARn	Parameter n
NO_AXIS	Initialization of formula parameter with default axis identifier

### Example: Definition of axis variables in the main program

```
Program code
DEF AXIS AXVAR
UP( , AXVAR)
```

### 10.3.5 Synchronized action marker (\$AC\_MARKER[n])

#### Function

The array variable \$AC\_MARKER[n] can be read and written in synchronized actions. These variables can either be saved in the memory of the active or passive file system.

#### Synchronized action variable: Data type INT

\$AC_MARKER[n]	Channel-specific marker/counter, INTEGER data type
\$MC_MM_NUM_AC_MARKER	Machine data for setting the number of channel-specific markers for movement synchronized actions
n	Array index of variables 0-n

#### Example of reading and writing marker variables

<b>Program code</b>	<pre>WHEN ... DO \$AC_MARKER[0] = 2 WHEN ... DO \$AC_MARKER[0] = 3 WHENEVER \$AC_MARKER[0] == 3 DO \$AC_OVR=50</pre>
---------------------	--

### 10.3.6 Synchronized action parameters (\$AC\_PARAM[n])

#### Function

The synchronized action parameter \$AC\_PARAM[n] is used for calculations and as intermediate memory in synchronized actions. These variables can either be saved in the memory of the active or passive file system.

#### Synchronized action variable: Data type:REAL

These parameters exist once in each channel under the same name.

\$AC_PARAM[n]	Arithmetic variable for motion synchronized actions (REAL)
\$MC_MM_NUM_AC_PARAM	Machine data for setting the number of parameters for movement synchronized actions up to a maximum of 20000.
n	Array index of parameter On

#### Example of synchronized action parameter \$AC\_PARAM[n]

Program code
\$AC_PARAM[0]=1.5 \$AC_MARKER[0]=1 ID=1 WHEN \$AA_IW[X]>100 DO \$AC_PARAM[1]=\$AA_IW[X] ID=2 WHEN \$AA_IW[X]>100 DO \$AC_MARKER[1]=\$AC_MARKER[2]

### 10.3.7 Arithmetic parameter (\$R[n])

#### Function

This static array variable is used for calculations in the part program and synchronized actions.

#### Syntax

Programming in part program:

```
REAL R[n]
REAL Rn
```

Programming in synchronized actions:

```
REAL $R[n]
REAL $Rn
```

#### Arithmetic parameters

Using arithmetic parameters allows for:

- storage of values that you want to retain beyond the end of program, NC reset, and Power On
- display of stored value in the R parameter display.

#### Examples

Program code	Comments
<pre>WHEN \$AA_IM[X]&gt;=40.5 DO \$R10=\$AA_MM[Y] G01 X500 Y70 F1000 STOPRE IF R10&gt;20</pre>	<p>; Using R10 in synchronized actions.</p> <p>; Preprocessing stop</p> <p>; Evaluation of the arithmetic variable.</p>

Program code
<pre>SYG_AS[2]=X SYG_IS[1]=1 WHEN \$AA_IM[SGY_AS[2]]&gt;10 DO \$R3=\$AA_EG_DENOM[SYG_AS[1]], SYG_AS[2]] WHEN \$AA_IM[SGY_AS[2]]&gt;12 DO \$AA_SCTRACE[SYG_AS[2]]=1 SYG_AS[1]=X SYG_IS[0]=1 WHEN \$AA_IM[SGY_AS[1]]&gt;10 DO \$R3=\$\$MA_POSCTRL_GAIN[SYG_IS[0]], SYG_AS[1]] WHEN \$AA_IM[SGY_AS[1]]&gt;10 DO \$R3=\$\$MA_POSCTRL_GAIN[SYG_AS[1]] WHEN \$AA_IM[SGY_AS[1]]&gt;15 DO \$\$MA_POSCTRL_GAIN[SYG_AS[0]], SYG_AS[1]]=\$R3</pre>

### 10.3.8 Read and write NC machine and NC setting data

#### Function

It is also possible to read and write NC machine / setting data of synchronized actions. When reading and writing machine data array elements, an index can be left out during programming. If this happens in the part program, all of the array's elements are described with the value when reading the **first** array element and when writing.

In synchronized actions, only the **first** element is read or written in such cases.

#### Definition

MD, SD with

**\$**: Read the value at the interpretation time of the synchronized actions

**\$\$**: Read the value in the main run

#### Read MD and SD values at the preprocessing time

They are addressed from within the synchronized action using the **\$** characters and evaluated by the preprocessing time.

```
| ID=2 WHENEVER $AA_IM[z]<$SA_OSCILL_REVERSE_POS2[Z]-6 DO $AA_OVR[X]=0  
| ;Here, reversal range 2, assumed to remain static during operation, is addressed for  
| oscillation.
```

#### Read MD and SD values at the main run time

They are addressed from within the synchronized action using the **\$** characters and evaluated by the main run time.

```
| ID=1 WHENEVER $AA_IM[z]<$$SA_OSCILL_REVERSE_POS2[Z]-6 DO $AA_OVR[X]=0  
| ;It is assumed here that the reverse position can be modified by a command during  
| the machining
```

## Write MD and SD at the main run time

The currently set access authorization level must allow write access. The active states are listed for all MD and SD in **References: /LIS/, Lists (Book 1)**.

The MD and SD to be written must be addressed preceded by **\$\$**.

## Example

Program code	Comments
<pre>ID=1 WHEN \$AA_IW[X]&gt;10 DO \$\$SN_SW_CAM_PLUS_POS_TAB_1[0]=20 \$\$SN_SW_CAM_MINUS_POS_TAB_1[0]=30</pre>	<p>; Changing the switching position of software cams. Note: The switching positions must be changed two to three interpolation cycles before they reach their position.</p>

## 10.3.9 Timer variable (\$AC\_Timer[n])

### Function

System variable **\$AC\_TIMER[n]** permits actions to be started after defined periods of delay.

### Timer variable: Data type:REAL

<b>\$AC_TIMER[n]</b>	Channel-specific timer of data type REAL
<b>s</b>	Unit in seconds
<b>n</b>	Index of timer variable

### Setting timers

Incrementation of a timer variable is started by means of value assignment:  
**\$AC\_TIMER[n]=value**

<b>n :</b>	Number of time variables
<b>value:</b>	Starting value (generally "0")

### **Stopping timers**

Incrementation of a timer variable can be stopped by assigning a negative value:

$\$AC\_TIMER[n]=-1$

### **Reading timers**

The current timer value can be read whether the timer variable is running or has been stopped. After a timer variable has been stopped through the assignment of -1, the current time value remains stored and can be read.

## **Example**

Output of an actual value via analog output 500 ms after detection of a digital input:

<b>Program code</b>	<b>Comments</b>
WHEN \$A_IN[1] == 1 DO \$AC_TIMER[1]=0	; Reset and start timer
WHEN \$AC_TIMER[1]>=0.5 DO \$A_OUTA[3]=\$AA_IM[X] \$AC_TIMER[1]=-1	

## **10.3.10 FIFO variables (\$AC\_FIFO1[n] ... \$AC\_FIFO10[n])**

### **Function**

10 FIFO variables (circulating buffer store) are available to store associated data sequences.  
Data type: REAL

Application:

- Cyclical measurement
- Pass execution

Each element can be accessed in read or write

## FIFO variables

The number of available FIFO variables is defined in machine data MD28260 \$MC\_NUM\_AC\_FIFO.

The number of values that can be written into a FIFO variable is defined using machine data MD28264 \$MC\_LEN\_AC\_FIFO. All FIFO variables are equal in length.

The sum of all FIFO elements is only generated if bit 0 is set in MD28266 \$MC\_MODE\_AC\_FIFO.

Indices **0 to 5** have a special significance:

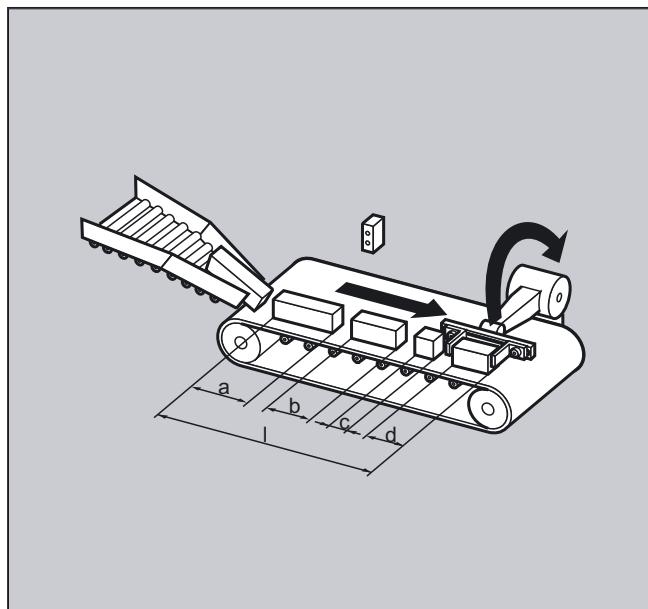
Index	Significance	
0	While writing:	New value is saved in the FIFO.
	While reading:	The oldest element is read and removed from the FIFO.
1	Access to oldest stored element	
2	Access to latest stored element	
3	Sum of all FIFO elements	
4	Number of elements available in FIFO. Read and write access can be assigned to each element of the FIFO. FIFO variables are reset by resetting the number of elements, e.g. for the first FIFO variable: \$AC_FIFO1[4]=0	
5	Current write index relative to beginning of FIFO	
6 to n <sub>max</sub>	Access to nth FIFO element	

## Example: Circulating memory

During a production run, a conveyor belt is used to transport products of different lengths (a, b, c, d). The conveyor belt of transport length therefore carries a varying number of products depending on the lengths of individual products involved in the process. With a constant speed of transport, the function for removing the products from the belt must be adapted to the variable arrival times of the products.

## Motion synchronous actions

### 10.3 Main run variables for synchronized actions



Program code	Comments
DEF REAL INTV=2.5	; Constant distance between products that are placed down.
DEF REAL TOTAL=270	; Distance between the length measurement and removal position.
EVERY \$A_IN[1]==1 DO \$AC_FIFO1[4]=0	; At the start of the process, reset FIFO
EVERY \$A_IN[2]==1 DO \$AC_TIMER[0]=0	; If a product interrupts the light barrier, start the time measurement.
EVERY \$A_IN[2]==0 DO \$AC_FIFO1[0] = \$AC_TIMER[0]*\$AA_VACTM[B]	; If the product leaves the light barrier, calculate the product length from the measured time and transport speed and save in FIFO.
EVERY \$AC_FIFO1[3]+\$AC_FIFO1[4]*ZWI>=GESAMT DO POS[Y]=-30 \$R1=\$AC_FIFO1[0]	; As soon as the sum of all product lengths and intermediate distances are greater than/equal to the length between the position where the product is placed down and where it is removed, remove the product from the conveyor belt at the removal position, read-out the associated product length from FIFO.

### 10.3.11 Information about block types in the interpolator (\$AC\_BLOCKTYPE, \$AC\_BLOCKTYPEINFO, \$AC\_SPLITBLOCK)

#### Function

The following system variables are available for synchronized actions to provide information about a block current executing in the main run:

- \$AC\_BLOCKTYPE
- \$AC\_BLOCKTYPEINFO
- \$AC\_SPLITBLOCK

#### Block type and block type info variables

\$AC_BLOCKTYPE		\$AC_BLOCKTYPEINFO				
Value:		Value:				
0	Not equal to 0	T	H	Z	E	Meaning:
Original block	Intermediate block					Trigger for intermediate block:
	1	1	0	0	0	Internally generated block, no further information
	2	2	0	0	1	Chamfer/rounding: Straight
	2	2	0	0	2	Chamfer/rounding: Circle
	3	3	0	0	1	WAB: Approach with straight line
	3	3	0	0	2	WAB: Approach with quadrant
	3	3	0	0	3	WAB: Approach with semicircle
						Tool compensation:
	4	4	0	0	1	Approach block after STOPRE
	4	4	0	0	2	Connection blocks if intersection point not found
	4	4	0	0	3	Point-type circle on inner corners (on TRACYL only)
	4	4	0	0	4	Bypass circle (or conical cut) at outer corners
	4	4	0	0	5	Approach blocks for offset suppression
	4	4	0	0	6	Approach blocks on repeated WRC activation
	4	4	0	0	7	Block split due to excessive curvature
	4	4	0	0	8	Compensation blocks on 3D face milling (tool vector    area vector)

## Motion synchronous actions

### 10.3 Main run variables for synchronized actions

\$AC_BLOCKTYPE		\$AC_BLOCKTYPEINFO				
Value:		Value:				
0	Not equal to 0	T	H	Z	E	Meaning:
Original block	Intermediate block					Trigger for intermediate block:
						Corner rounding with:
	5	5	0	0	1	G641
	5	5	0	0	2	G642
	5	5	0	0	3	G643
	5	5	0	0	4	G644
						TLIFT block with:
	6	6	0	0	1	linear movement of tangential axis and without lift motion
	6	6	0	0	2	nonlinear movement of tangential axis (polynomial) and without lift motion
	6	6	0	0	3	lift movement, tangential axis movement and lift movement start simultaneously
	6	6	0	0	4	lift movement, tangential axis does not start until certain lift position is reached.
						Path segmentation:
	7	7	0	0	1	programmed path segmentation is active without punching or nibbling
	7	7	0	0	2	programmed path segmentation with active punching or nibbling
	7	7	0	0	3	automatically, internally generated path segmentation
						Compile cycles:
	8	ID application			ID of the compile cycle application that generated the block	

T: Thousands digit  
H: Hundreds digit  
Z: Tens digit  
E: Units digit

#### Note

\$AC\_BLOCKTYPEINFO also always includes the value for the block type in the thousands position (T) in the case that there is an intermediate block. The thousands position is not accepted in \$AC\_BLOCKTYPE not equal to 0.

\$AC_SPLITBLOCK	
Value:	Significance:
0	Unchanged programmed block (a block generated by the compressor is also dealt with as a programmed block)
1	There is an internally generated block or a shortened original block
3	The last block in a chain of internally generated blocks or shortened original blocks is available

### Example: Counting blending blocks

Program code	Comments
<pre>\$AC_MARKER[0]=0 \$AC_MARKER[1]=0 \$AC_MARKER[2]=0 ... ID=1 WHENEVER (\$AC_TIMEC==0) AND (\$AC_BLOCKTYPE==5) DO \$AC_MARKER[0]=\$AC_MARKER[0]+1 ... ID=2 WHENEVER (\$AC_TIMEC==0) AND (\$AC_BLOCKTYPEINFO==5001) DO \$AC_MARKER[1]=\$AC_MARKER[1]+1 ; Blending blocks generated with G641 are counted in \$AC_MARKER[1]: ; ID=3 WHENEVER (\$AC_TIMEC==0) AND (\$AC_BLOCKTYPEINFO==5002) DO \$AC_MARKER[2]=\$AC_MARKER[2]+1 ...</pre>	<p>; Definition of synchronized actions with which blending blocks are counted.</p> <p>; All blending blocks are counted in \$AC_MARKER[0]:</p> <p>; Blending blocks generated with G641 are counted in \$AC_MARKER[1]:</p> <p>; Blending blocks generated with G642 are counted in \$AC_MARKER[2]:</p>

## 10.4 Actions in synchronized actions

### 10.4.1 Overview of possible actions in synchronized actions

Actions in synchronized actions consist of value assignments, function or parameter calls, keywords or technology cycles. Complex executions are possible using operators.

Possible applications include:

- Calculations of complex expressions in the IPO cycle
- Axis movements and spindle controls
- Setting data from synchronized actions can be changed and evaluated online (e.g. output positions and times of software cams at the PLC or NC I/O)
- Output of auxiliary functions to PLC
- Set-up additional safety functions
- Set superimposed movement, online tool offset and clearance control
- Execute actions in all operating modes
- Influence synchronized actions from PLC
- Run technology cycles
- Output of digital and analog signals
- Record performance recording of the synchronized actions at the interpolation cycle and the computation time of the position controller for the loading report
- Diagnostic capabilities in the user interface

Synchronized action	Description
DO \$V...=	Assign variable (servo-values)
DO \$A...=	Assign variable (main-run variable)
DO \$AC...[n]=	Special main run variable
DO \$AC_MARKER[n]=	Read or write synchronized action markers
DO \$AC_PARAM[n]=	Read or write synchronized action parameters
DO \$R[n]=	Read or write arithmetic variable
DO \$MD...=	Read MD value at the interpolation instant in time
DO \$\$SD...=	Writing SD value in the main run

Synchronized action	Description
DO \$AC_TIMER[n]=Start value	Timers
DO \$AC_FIFO1[n] ... FIFO10[n]=	FIFO variables
DO \$AC_BLOCKTYPE=	Interpret the current block (main run variable)
DO \$AC_BLOCKTYPEINFO=	
DO \$AC_SPLITBLOCK=	
DO M-, S and H e.g. M07	Output of M, S and H auxiliary functions
DO RDISABLE	Set read-in disable
DO STOPREOF	Cancel preprocessing stop
DO DELDTG	Fast deletion of distance-to-go without preprocessing stop
FTCDEF(polynomial, LL, UL , coefficient)	Definition of polynomials
DO SYNFCFT(Polyn., Output, Input)	Activation of synchronized functions: adaptive control
DO FTOC	Online tool offset
DO G70/G71/G700/G710	Define dimension system for positioning tasks (dimensions either in inches or metric)
DO POS[axis]= / DO MOV[axis]=	Start/position/stop command axes
DO SPOS[spindle]=	Start/position/stop spindles
DO MOV[Axis]=value	Start/position infinite movements of a command axis
DO POS[Axis]= FA [Axis]=	Axial feed FA
ID=1 ... DO POS[axis]= FA [axis]=	Position from synchronized actions
ID=2 ... DO POS[axis]=	
\$AA_IM[axis] FA [axis]=	
DO PRESETON(axis, value)	Set actual value (preset from synchronized actions)
ID=1 EVERY \$A_IN[1]=1 DO M3 S...	Start/position/stop spindles
ID=2 EVERY \$A_IN[2]=1 DO SPOS=	
DO TRAILON(FA,LA,coupling factor)	Activate coupled-axis motion
DO LEADON(FA,LA,NRCTAB,OVW)	Activate leading value coupling
DO MEAWA(axis)=	Activate axial measurement
DO MEAC(axis)=	Activate continuous measurement
DO [array n, m]=SET(value, value, ...)	Initialization of array variables with value lists
DO [array n, m]=REP(value, value, ...)	Initialization of array variables with the same values
DO SETM(Marker No.)	Set wait markers
DO CLEARM(Marker No.)	Delete wait markers
DO SETAL(alarm no.)	Set cycle alarm (additional safety function)

## Motion synchronous actions

### 10.4 Actions in synchronized actions

Synchronized action	Description
DO FXS[axis]= DO FXST[axis]= DO FXSW[axis]= DO FOCON[axis]= DO FOCOF[axis]=	Select travel to fixed stop Change clamping torque Change monitoring window Activate travel with limited torque/force (modal) FOC Deactivate travel with limited torque/force (synchronized actions act on specific blocks)
ID=2 EVERY \$AC_BLOCKTYPE==0 DO \$R1=\$AC_TANE <sup>B</sup>	The angle between the path tangent at the end of the current block and the path tangent at the start of the programmed following block
DO \$AA_OVR= DO \$AC_OVR= DO \$AA_PLC_OVR DO \$AC_PLC_OVR DO \$AA_TOTAL_OVR DO \$AC_TOTAL_OVR	Axial override Path override of the axial override specified from the PLC of the path override specified from the PLC resulting axial override resulting path override
\$AN_IPO_ACT_LOAD= \$AN_IPO_MAX_LOAD= \$AN_IPO_MIN_LOAD= \$AN_IPO_LOAD_PERCENT= \$AN_SYNC_ACT_LOAD=  \$AN_SYNC_MAX_LOAD=  \$AN_SYNC_TO_IPO=	actual IPO computation time longest IPO computation time shortest IPO computation time actual IPO computation time in the ratio to the IPO clock cycle actual computation time for synchronized actions over all channels longest computation time for synchronized actions over all channels percentage component of the total synchronized action
DO TECCYCLE	Run technology cycle
DO LOCK(n, n, ...) DO UNLOCK(n, n, ...) DO RESET(n, n, ...)	Inhibit Enable RESET of a technology cycle
CANCEL(n, n, ...)	Delete modal synchronized actions with the designation ID(S) in the part program

## 10.4.2 Output of auxiliary functions

### Function

Auxiliary functions are output directly in the synchronized action at the output time of the action. The output timing defined in the machine data for auxiliary functions is not active.

The output timing is given when the condition is fulfilled.

#### Example:

Switch on coolant at a specific axis position:

```
WHEN $AA_IM[X]>=15 DO M07 POS[X]=20 FA[X]=250
```

### Permitted key words in non-modal synchronized actions (no modal ID)

Auxiliary functions can only be programmed with the WHEN or EVERY key words.

---

#### Note

The following auxiliary functions are not permitted in synchronized actions:

- M0, M1, M2, M17, M30: Program halt/end (M2, M17, M30 possible for technology cycle)
  - M70: Spindle functions
  - M functions for tool change set with M6 or via machine data
  - M40, M41, M42, M43, M44, M45: Gear change
- 

### Example

Program code	Comments
WHEN \$AA_IW[Q1]>5 DO M172 H510	; If the actual value of the Q1 axis exceeds 5 mm, output help functions M172 and H510 at the PLC

### 10.4.3 Set read-in disable (RDISABLE)

#### Function

Using RDISABLE, when the condition is fulfilled, the additional block processing is held in the main program. Programmed synchronized motion actions are still executed, the following blocks are still prepared.

In path control mode, an exact stop is always triggered at the beginning of the block with RDISABLE in synchronized actions, regardless of whether RDISABLE is active or not.

#### Example

Start the program in interpolation cycles dependent on external inputs.

Program code	Comments
...	
WHENEVER \$A_INA[2]<7000 DO RDISABLE	; If the voltage 7V is not reached at input 2, program execution is stopped (1000= 1V).
N10 G1 X10	; If the condition is fulfilled, the read-in inhibit at the end of N10 is effective
N20 G1 X10 Y20	
...	

### 10.4.4 Cancel preprocessing stop (STOPREOF)

#### Function

In the case of an explicitly programmed preprocessing stop STOPRE or a preprocessing stop implicitly activated by an active synchronized action, STOPREOF cancels the preprocessing stop after the next machining block as soon as the condition is fulfilled.

#### Note

STOPREOF must be programmed with the keyword WHEN and non-modally (without ID number).

## Example

Fast program branch at end of block.

Program code	Comments
WHEN \$AC_DTEB<5 DO STOPREOF	; If the distance to the end of the block is less than 5 mm, withdraw preprocessing stop.
G01 X100	; After executing linear interpolation, the preprocessing stop is withdrawn.
IF \$A_INA[7]>500 GOTOF MARKEL=X100	; If the voltage of 5V at input 7 is exceeded, ;jump to label 1.

## 10.4.5 Delete distance-to-go (DELDTG)

### Function

Delete distance-to-go can be triggered for a path and for specified axes depending on a condition.

The possibilities are:

- Fast, prepared delete distance-to-go
- Unprepared delete distance-to-go

Prepared delete distance-to-go with DELDTG permits a fast response to the triggering event and is therefore used for time-critical applications, e.g. if

- the time between delete distance-to-go and the start of the next block must be very short.
- the condition for delete distance-to-go will very probably be fulfilled.

---

### Note

The axis designation contained in brackets behind DELDTG is only valid for **one** positioning axis.

---

## Syntax

**Delete distance-to-go for the path**  
DO DELDTG  
**Axial delete distance-to-go**  
DO DELDTG(axis1) DELDTG(axis2) ...

### Example of fast deletion of distance-to-go path

Program code	Comments
WHEN \$A_IN[1]==1 DO DELDTG	
N100 G01 X100 Y100 F1000	; If the input is set, motion is interrupted
N110 G01 X... IF \$AA_DELT>50...	

### Example of fast axial deletion of distance-to-go

Program code	Comments
Cancelation of a positioning movement: ID=1 WHEN \$A_IN[1]==1 DO MOV[V]=3 FA[V]=700	; Start axis
WHEN \$A_IN[2]==1 DO DELDTG(V)	; Delete distance-to-go, the axis is stopped using MOV=0
Delete distance-to-go depending on the input voltage: WHEN \$A_INA[5]>8000 DO DELDTG(X1)	; As soon as the voltage at input 5 exceeds 8V, delete distance-to-go for axis X1. Path motion continues.
POS[X1]=100 FA[X1]=10 G1 Z100 F1000	

## Further Information

At the end of a traversing block in which a prepared delete distance-to-go was triggered, preprocess stop is activated implicitly.

Continuous path mode or positioning axis movements are therefore interrupted or stopped at the end of the block with fast delete distance-to-go.

---

#### Note

Prepared delete distance-to-go:

- cannot be used with active tool radius correction.
  - the action must only be programmed in non modal synchronized actions (without ID number).
- 

## 10.4.6 Polynomial definition (FCTDEF)

### Function

FCTDEF can be used to define 3rd order polynomials in the form  $y=a_0+a_1x+a_2x^2+a_3x^3$ . These polynomials are used by the online tool offset (FTOC) and the evaluation function (SYNFCT).

### Syntax

```
FCTDEF(Polynomial_No.,LLIMIT,ULIMIT,a0,a1,a2,a3)
```

### Significance

Polynomial_No.	Number of the 3rd order polynomial
LLIMIT	Lower limit for function value
ULIMIT	Upper limit for function value
a0, a1, a2, a3	Polynomial coefficient

These values can also be accessed via system variables

\$AC_FCTL[n]	Lower limit for function value
\$AC_FCTUL[n]	Upper limit for function value
\$AC_FCT0[n]	a0
\$AC_FCT1[n]	a1
\$AC_FCT2[n]	a2
\$AC_FCT3[n]	a3

---

### Note

#### Writing system variables

- The system variables can be written from the part program or from a synchronized action. When writing from part programs, program STOPRE to ensure that writing is block synchronized.
- The `$AC_FCTLL[n]`, `$AC_FCTUL[n]`, `$AC_FCT0[n]` to `$AC_FCTn[n]` system variables can be changed from synchronized actions

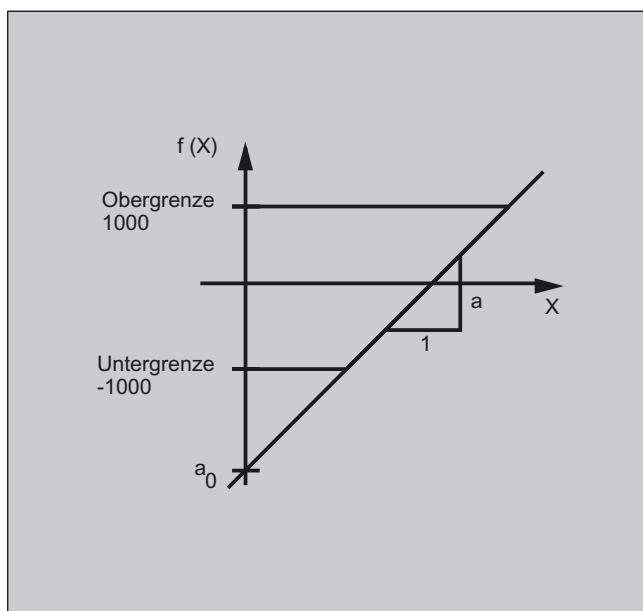
When writing from synchronized actions, the polynomial coefficients and function value limits are active immediately.

---

#### Example of a polynomial for straight section:

With upper limit 1000, lower limit -1000, ordinate section  $a_0 = \$AA\_IM[X]$  and linear gradient 1 the polynomial is:

`FCTDEF(1, -1000, 1000, $AA_IM[X], 1)`

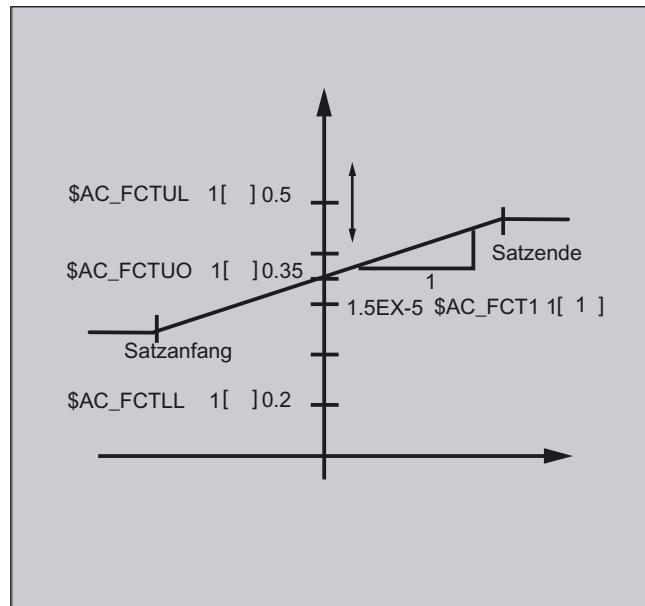


### Example of laser output control

One of the possible applications of polynomial definition is the laser output control.

Laser output control means:

Influencing the analog output in dependence on, for example, the path velocity.



Program code	Comments
\$AC_FCTLL[1]=0.2	; Definition of a polynomial coefficients
\$AC_FCTUL[1]=0.5	
\$AC_FCT0[1]=0.35	
\$AC_FCT1[1]=1.5EX-5	
STOPRE	
ID=1 DO \$AC_FCTUL[1] = \$A_INA[2]*0.1 + 0.35	; Change upper limit online.
ID=2 DO SYNFC(1,\$A_OUTA[1],\$AC_VACTW)	; Depending on the path velocity (stored in \$AC_VACTW) the laser power control is controlled via analog output 1

#### Note

The polynomial defined above is used with SYNFC.

### **10.4.7 Synchronized function (SYNFCT)**

#### **Function**

SYNFCT calculates the output value of a polynomial 3 grade weighted using the input variables. The result is in the output variables and has maximum and minimum limits.

The evaluation function is used

- in AC control (adaptive control),
- in laser output control,
- with position feed-forward

#### **Syntax**

```
SYNFCT (Polynomial_No., main run variable output, main run variable  
input)
```

#### **Significance**

For the output variable, it is possible to select variables that

- with additive influencing
- with multiplicative influencing
- as a position offset or
- directly

affect the machining process.

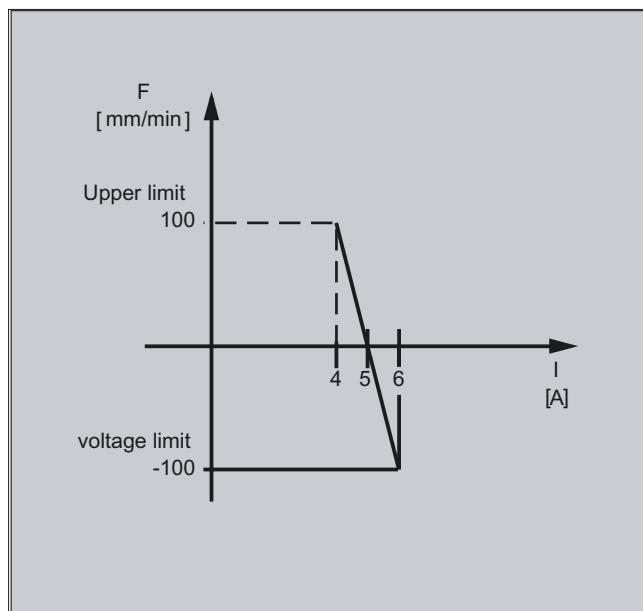
DO SYNFCT	Activation of the evaluation function
Polynomial No.	With polynomial defined with FCTDEF (see Subsection "Polynomial definition").
Main run variable output	Write main run variable
Main run variable input	Read main run variable

### Example of adaptive control (additive)

#### Additive influence on the programmed feedrate

A programmed feedrate is to be controlled additive using the current of the X axis (infeed axis):

The feedrate should only vary by +/- 100 mm/min and the current fluctuates by +/-1A around the working point of 5A.



#### 1. Polynomial definition

Determination of the coefficients

$$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

$$a_1 = -100 \text{ mm/1 min A}$$

$$a_0 = -(-100)*5 = 500$$

$$a_2 = a_3 = 0 \text{ (no square and cubic component)}$$

$$\text{Upper limit} = 100$$

$$\text{Lower limit} = -100$$

This means:

FCTDEF(1,-100,100,500,-100,0,0)

#### 2. Activate AC control

ID=1 DO SYNFCT(1,\$AC\_VC,\$AA\_LOAD[x])

;Read the current axis load (% of the max. drive current) via \$AA\_LOAD[x], calculate the path feedrate override with the polynomial defined above.

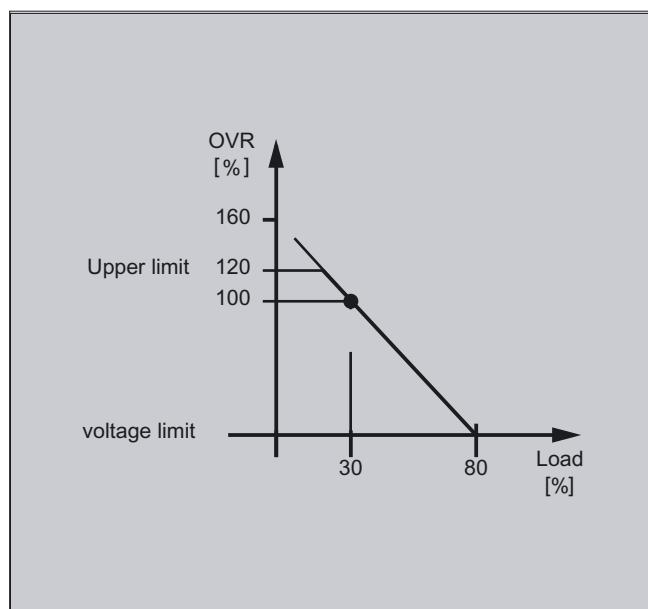
### Example of adaptive control (multiplicative)

Influence the programmed feedrate by multiplication

The aim is to influence the programmed feedrate by multiplication. The feedrate must not exceed certain limits – depending on the load on the drive:

- The feedrate is to be stopped at a drive load of 80%: override = 0
- At a drive load of 30% it is possible to traverse at programmed feedrate: override = 100%.

The feedrate can be exceeded by 20%:  
Max. override = 120%.



#### 1st polynomial definition

Determination of coefficients

$$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

$$a_1 = -100\%/(80-30)\% = -2$$

$$a_0 = 100 + (2 \cdot 30) = 160$$

$$a_2 = a_3 = 0 \text{ (neither a square nor cubic element)}$$

$$\text{Upper limit} = 120$$

$$\text{Lower limit} = 0$$

This means:

FCTDEF (2, 0, 120, 160, -2, 0, 0)

## 2. Switch-in AC control

ID=1 DO SYNFCFT(2,\$AC\_OVR,\$AA\_LOAD[x])

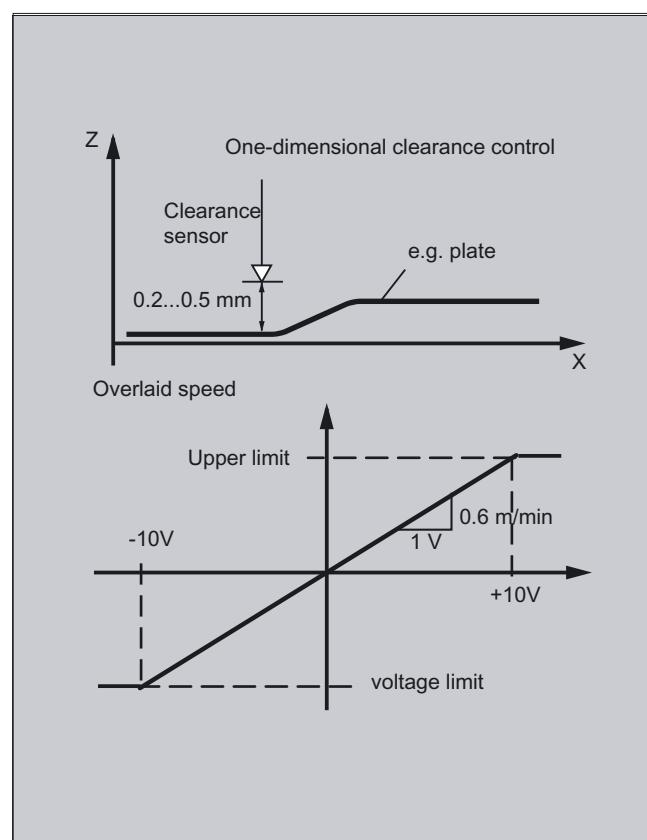
;Read the current axis load (% of the max. drive current) via \$AA\_LOAD[x], calculate the feedrate override with the polynomial defined above.

### 10.4.8 Closed-loop clearance control with limited correction (\$AA\_OFF\_MODE)

#### Function

The integrating calculation of the clearance values is realized with a limit range check:

\$AA\_OFF\_MODE = 1



#### NOTICE

The loop gain of the higher-level control loop depends on the setting of the Ipo clock cycle.  
Remedy: Read-in the MD for IPO clock cycle and take this into account.

---

**Note**

Limitation of the speed of the superimposed interpolator using MD 32020: JOG\_VEL0 with Ipo clock cycle 12 ms. Formula for the speed:

$$\frac{0.120\text{mm}}{0.6\text{ms}} / \text{mV} = 0.6 \frac{\text{m}}{\text{min}} / \text{V}$$

---

**Example**

**Subprogram "AON": Clearance control on**

Program code	Comments
PROC AON	
\$AA_OFF_LIMIT[Z]=1	; Specifies limit value.
FCTDEF(1, -10, +10, 0, 0.6, 0.12)	; Polynomial definition
ID=1 DO SYNFCT(1,\$AA_OFF[Z],\$A_INA[3])	; Clearance control active.
ID=2 WHENEVER \$AA_OFF_LIMIT[Z]<>0	; Inhibit axis X when the limit range is exceeded.
DO \$AA_OVR[X] = 0	
RET	
ENDPROC	

**Subprogram "AOFF": Clearance control off**

Program code	Comments
PROC AOFF	
CANCEL(1)	; Delete synchronized action, clearance control
CANCEL(2)	; Cancel limit range check
RET	
ENDPROC	

**Main program "MAIN"**

Program code	Comments
AON	; Clearance control on
...	
G1 X100 F1000	
AOFF	; Clearance control off
M30	

## Further Information

### Position offset in the basic coordinate system

Using the system variable \$AA\_OFF[axis] motion can be superimposed on every axis in the channel. It acts as a position offset in the basic coordinate system.

The position offset programmed in this way is overlaid immediately in the axis concerned, whether the axis is being moved by the program or not.

Limit main run variable output:

It is possible to limit the absolute value to be corrected (main run variable output) to the value saved in the setting data SD43350 \$SA\_AA\_OFF\_LIMIT.

The type of superimposition of the clearance is defined using machine data MD36750 \$MA\_AA\_OFF\_MODE:

Value	Significance
0	Proportional evaluation
1	Integrating evaluation

Using the system variable \$AA\_OFF\_LIMIT[axis] it can be interrogated, dependent on the direction, whether the correction value is within the limit range. The system variable can be interrupted from synchronized actions and when a limit value is reached, can be used to e.g. stop the axis or set an alarm.

- 0: Offset value not in range
- 1 Limit of offset value reached in the positive direction
- 1: Limit of offset value reached in the negative direction

### 10.4.9 Online tool offset (FTOC)

#### Function

FTOC permits overlaid movement for a geometry axis after a polynomial programmed with FCTDEF depending on a reference value that might, for example, be the actual value of an axis.

Coefficient  $a_0$  of the function definition FCTDEF( ) is evaluated with FTOC.  
The maximum and minimum limits are determined by  $a_0$ .

This means that you can also program modal, online tool offsets or clearance controls as synchronized actions.

This function is used for the machining of a workpiece and dressing of a grinding wheel in the same channel or in different channels (machining and dressing channel).

The supplementary conditions and specifications for dressing grinding wheels apply to FTOC in the same way that they apply to tool offsets using PUTFTOCF. For further information, please refer to "Tool Offsets" section.

#### Syntax

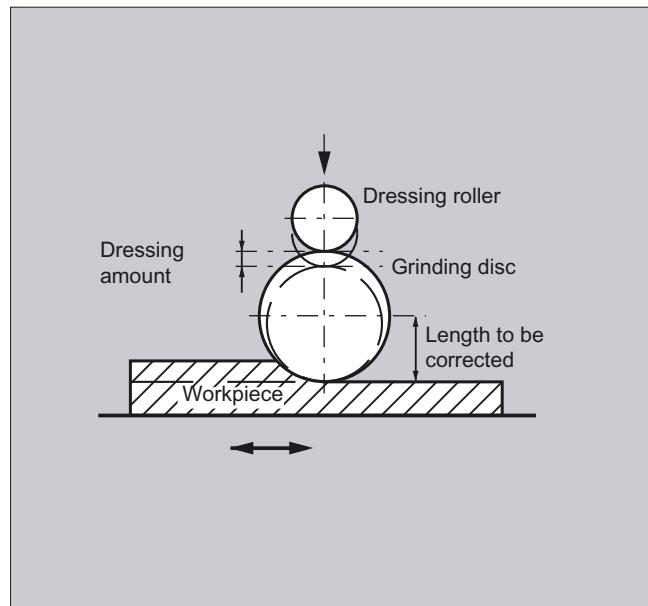
```
FTOC (polynomial No., EV, length1_2_3 or Radius4, channel, spindle)
```

#### Significance

DO FTOC	Perform online tool offsets
Polynomial No.	For polynomial defined with FCTDEF, see Subsection "Polynomial definition" in this Section.
RV	Main run variable for which a function value for the specified polynomial is to be calculated.
Length1_2_3 Radius4	Length offset (\$TC_DP1 to 3) or radius offset to which the calculated function value is added.
Channel	Number of the channel in which the offset is active. No specification is made here for an offset in the active channel. FTOCON must be activated in the target channel.
Spindle	Only specified if it is not the active spindle, which is to be compensated.

## Example

In this example, we want to compensate for the length of the active grinding wheel.



Program code	Comments
FCTDEF(1,-1000,1000,-\$AA_IW[V],1)	; Function definition.
ID=1 DO FTOC(1,\$AA_IW[V],3,1)	; Select online tool offset: Actual value of the V axis is the input value for polynomial 1.Result is added in channel 1 as compensation value to length 3 of the active grinding disk.
WAITM(1,1,2)	; Synchronization with machining channel.
G1 V-0.05 F0.01 G91	; Feed motion to dress.
G1 V-0.05 F0.02	
...	
CANCEL(1)	; Deselect online offset
...	

### 10.4.10 Online tool length compensation (\$AA\_TOFF)

#### Function

Use the system variable \$AA\_TOFF[ ] 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 the index. Thus, the number of active directions of offset is determined by the geometry axes that are active at the same time.

All offsets can be active at the same time.

#### Syntax

```
N.. TRAORI  
N.. TOFFON(X, 25)  
N.. WHEN TRUE DO $AA_TOFF[X]  
N.. TOFFON(Y, 25)  
N.. WHEN TRUE DO $AA_TOFF[Y]  
N.. TOFFON(Z, 25)  
N.. WHEN TRUE DO $AA_TOFF[Z]
```

#### Significance

TOFFON	Tool Offset ON (activate online tool length offset) On activation, an offset value can be specified for the relevant direction of offset and this is immediately recovered.
TOFFOF	Tool Offset OF (reset online tool length offset) The relevant offset values are reset and a preprocessing stop is initiated.
X, Y, Z	Direction of compensation for the offset value indicated for TOFFON
\$AA_TOFF[X]=value \$AA_TOFF[Y]=value \$AA_TOFF[Z]=value	Superimposition in X direction Superimposition in Y direction Superimposition in Z direction

## Examples

### Example 1: Selecting the tool length compensation

Program code	Comments
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.
N40 TOFFON(X)	; Activation of online tool length compensation for the X tool direction.
N50 ID=1 DO \$AA_TOFF[X] = \$AA_IW[X2] G4 F5	; For the X tool direction compensation is implemented depending on the position of axis X2.
...	
	; Assigns actual compensation in the X direction. For the X tool direction, the TLC is reduced back to 0:
N100 XOFFSET=\$AA_TOFF_VAL[X] N120 TOFFON(X, -XOFFSET) G4 F5	

### Example 2: Deselect the tool length compensation

Program code	Comments
N10 TRAORI(1)	; Transformation on.
N20 TOFFON(X)	; Activating the Z 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 traversed, the n position offset corresponding to the actual orientation is added to the actual position in the WCS.

### **10.4.11 Positioning movements**

#### **Function**

Axes can be positioned completely unsynchronized with respect to the parts program from synchronized actions. Programming positioning axes from synchronized actions is advisable for cyclic sequences or operations that are strongly dependent on events. Axes programmed from synchronized actions are called **command axes**.

#### **Programming**

##### **References:**

/PG/ Programming Guide Fundamentals; "Path details" Section  
/FBSY/ Function Description, Synchronized Actions; "Starting command axes"

#### **Parameters**

The measuring system for positioning tasks in synchronized actions is specified with the G codes `G70/G71/G700/G710`.

By programming the G functions in the synchronized action, the INCH/METRIC evaluation for the synchronized action can be defined independently of the parts program context.

### **10.4.12 Position axis (POS)**

#### **Function**

Unlike programming from the part program, the positioning axis movement has no effect on execution of the part program.

#### **Syntax**

`POS [axis]=value`

#### **Significance**

DO POS	Start/position command axis
Axis	Name of the axis to be traversed
Value	The value to traverse by (depending on traverse mode)

## Examples

### Example 1:

Program code	Comments
ID=1 EVERY \$AA_IM[B]>75 DO POS[U]=100	; Axis U traverses, dependent on the traversing mode, incrementally through 100 (inch/mm) or to position 100 (inch/mm) from the control zero.
	; Traverse axis U by the distance calculated from the main run variables:
ID=1 EVERY \$AA_IM[B]>75 DO POS[U]=\$AA_MW[V]-\$AA_IM[W]+13.5	

### Example 2:

Program environment influences the positioning travel of the positioning axis (no G function in the action component of the synchronized action):

Program code	Comments
N100 R1=0	
N110 G0 X0 Z0	
N120 WAITP(X)	
N130 ID=1 WHENEVER \$R==1 DO POS[X]=10	
N140 R1=1	
N150 G71 Z10 F10	; Z=10mm X=10mm
N160 G70 Z10 F10	; Z=254mm X=254mm
N170 G71 Z10 F10	; Z=10mm X=10mm
N180 M30	

G71 in the action component of the synchronized action clearly defines the positioning travel of the positioning axis (metric) independent of the program environment:

Program code	Comments
N100 R1=0	
N110 G0 X0 Z0	
N120 WAITP(X)	
N130 ID=1 WHENEVER \$R==1 DO G71 POS[X]=10	
N140 R1=1	
N150 G71 Z10 F10	; Z=10mm X=10mm
N160 G70 Z10 F10	; Z=254mm X=10mm (X always positions to 10mm)
N170 G71 Z10 F10	; Z=10mm X=10mm
N180 M30	

---

#### 10.4 Actions in synchronized actions

If axis motion is not to be started at the beginning of the block, the override for the axis can be held at 0 from a synchronized action up to the required starting instant:

Program code	Comments
WHENEVER \$A_IN[1]==0 DO \$AA_OVR[W]=0 G01 X10 Y25 F750 POS[W]=1500 FA=1000 ;	The positioning axis is held as long as digital input 1=0.

#### 10.4.13 Position in specified reference range (POS RANGE)

##### Function

The POSRANGE( ) function can be used to determine whether the current interpolated setpoint position of an axis is in a window around a specified reference position. The position specifications can refer to coordinates systems which can be specified.

The module offset is taken into account when interrogating the actual axis position of a module axis.

---

##### Note

The function can only be called up from the synchronized action. If called up from the part program, the alarm 14091 %1 block %2 is triggered, function not permitted, index: %3 with index 5 called up.

---

##### Syntax

BOOL POSRANGE(Axis, Refpos, Winlimit, [Coord])

## Significance

BOOL POSRANGE	Current position of command axis is in window of specified reference position.
AXIS <axis>	Axis identifier of machine-, channel- or geometry axis
REAL Refpos	Reference position in Coord coordinate system
REAL Winlimit	Amount resulting in limit for position window
INT Coord	MCS is active (option). The following are possible: 0 for MCS (machine coordinates system) 1 for BCS (basic coordinates system) 2 for Szs (settable zero system) 3 for WCS (workpiece coordinate system)

## Function value

Current setpoint depending on position details in specified coordinates system

Function value: TRUE	if Refpos(Coord) - abs(Winlimit) $\leq$ Actpos(Coord) $\leq$ Refpos(Coord) + abs(Winlimit)
Function value: FALSE	otherwise

#### 10.4.14 Start/stop axis (MOV)

##### Function

With MOV[axis]=value it is possible to start a command axis without specifying an end position. The axis is moved in the programmed direction until another movement is set by another motion or positioning command or until the axis is stopped with a stop command.

##### Syntax

MOV[axis] = value

##### Significance

DO MOV	Start command axis motion
Axis	Name of the axis to be started
Value	Start command for traverse/stop motion. The sign determines the direction of motion. The data type for the value is INTEGER.
Value >0 (usually +1)	Positive direction
Value <0 (usually -1)	Negative direction
Value ==0	Stop axis motion

---

##### Note

If an indexing axis is stopped with MOV[Axis]=0, the axis is halted at the next indexing position.

---

##### Example

Program code	Comments
... DO MOV[U]=0	; Axis U is stopped

### 10.4.15 Axis replacement (RELEASE, GET)

#### Function

For a tool change, the corresponding command axes can be requested as an action of a synchronized action using GET(axis). The axis type assigned to this channel and the interpolation right thus linked to this time can be queried using the \$AA\_AXCHANGE\_TYPE system variable. Different processes are possible depending on the actual status and on the channel having the current interpolation right for this axis.

Once the tool change is complete, this command axis can then be released for the channel as an action of a synchronized action using RELEASE(axis).

#### Machine manufacturer

The axis concerned must be assigned to the channel via machine data. Please refer to the machine manufacturer's specifications.

#### Syntax

```
GET (axis[,axis{,...}]) Get axis  
RELEASE (axis[,axis{,...}]) Release axis
```

#### Significance

DO RELEASE	Release axis as neutral axis
DO GET	Get axis for axis replacement
Axis	Name of the axis that is to be started

**Example: Program sequence for axis replacement, two channels**

The Z axis has been declared in the first and second channels.

**Program sequence in the first channel:**

<b>Program code</b>	<b>Comments</b>
WHEN TRUE DO RELEASE(Z)	; Z axis becomes the neutral axis
WHENEVER(\$AA_TYP[Z]==1) DO RDISABLE	; Read-in inhibit as long as Z axis is program axis
N110 G4 F0.1	
WHEN TRUE DO GET(Z)	; Z axis again becomes the NC program axis
WHENEVER(\$AA_TYP[Z]<>1) DO RDISABLE	; Read-in inhibit until Z axis is the program axis
N120 G4 F0.1	
WHEN TRUE DO RELEASE(Z)	; Z axis becomes the neutral axis
WHENEVER(\$AA_TYP[Z]==1) DO RDISABLE	; Read-in inhibit as long as Z axis is the program axis
N130 G4 F0.1	
N140 START(2)	; Start the second channel

**Program sequence in the second channel:**

<b>Program code</b>	<b>Comments</b>
WHEN TRUE DO GET(Z)	; ;Move Z axis to second channel
WHENEVER(\$AA_TYP[Z]==0) DO RDISABLE	; ;Read-in disable as long as Z axis is in other channel ;
N210 G4 F0.1	
WHEN TRUE DO GET(Z)	; ;Z axis is NC program axis
WHENEVER(\$AA_TYP[Z]<>1) DO RDISABLE	; ;Read-in disable until Z axis is program axis
N220 G4 F0.1	
WHEN TRUE DO RELEASE(Z)	; ;Z axis in second channel is neutral axis
WHENEVER(\$AA_TYP[Z]==1) DO RDISABLE	; ;Read-in disable as long as Z axis is program axis
N230 G4 F0.1	
N250 WAITM(10, 1, 2)	; Synchronize with channel 1

**Program sequence in the first channel continues:**

Program code	Comments
N150 WAIM(10, 1, 2)	; Synchronize with channel 2
WHEN TRUE DO GET(Z)	; Move Z axis to this channel
WHENEVER(\$AA_TYP[Z]==0) DO RDISABLE	; Read-in inhibit as long as Z axis is in another channel
N160 G4 F0.1	
N199 WAITE(2)	
N999 M30	; Wait for end of program in channel 2

**Example: Axis replacement in technology cycle**

The U axis U (\$MA\_AUTO\_GET\_TYPE=2) has been declared in the first and second channel and channel 1 currently has the interpolation right. The following technology cycle is started in channel 2:

Program code	Comments
GET(U)	; Fetch U axis to channel
POS[U]=100	; U axis is to be moved to position 100

The command-axis-movement line POS[U] is not executed until the U axis has been moved to channel 2.

**Sequence**

The axis that is requested at the time the action GET (axis) is activated can be read with respect to axis type for an axis replacement via the system variable (\$AA\_AXCHANGE\_TYP[<axis>]):

- 0: Axis assigned to NC program
- 1: Axis assigned to PLC or active as command axis or oscillating axis
- 2: Another channel has the interpolation right
- 3: Axis is neutral axis
- 4: Neutral axis is controlled by PLC

- 5: Another channel has the interpolation right, axis is requested for NC program
- 6: Another channel has the interpolation right, axis is requested as neutral axis
- 7: Axis active for PLC or as command or oscillating axis, axis is requested for PLC program
- 8: Axis active for PLC or as command or oscillating axis, axis is requested as neutral axis

**Boundary conditions**

The axis concerned must be assigned to the channel via machine data.

An axis controlled exclusively by the PLC cannot be assigned to the NC program.

**References:**

/FB2/ Function Manual, Extended Functions; Positioning Axes (P2)

**Using GET to request an axis from another channel**

If, when the `GET` action is activated, **another channel is authorized to write** (has the interpolation right) to the axis (`$AA_AXCHANGE_TYP[<axis>] == 2`), axis replacement is used to get the axis from this channel (`$AA_AXCHANGE_TYP[<axis>]==6`) and assign it to the requesting channel as soon as possible.

The axis then becomes the neutral axis (`$AA_AXCHANGE_TYP[<axis>]==3`).

There is no reorganize in the requesting channel.

**Assignment as NC program axis with reorganize:**

If an attempt to make the axis the neutral axis is already in progress when the `GET` action is activated (`$AA_AXCHANGE_TYP[<axis>]==6`), the axis is requested for the NC program (`$AA_AXCHANGE_TYP[<axis>]==5`) and assigned to the NC program on the channel as soon as possible (`$AA_AXCHANGE_TYP[<axis>]==0`).

### Axis already assigned to requested channel

Assignment as NC program axis with reorganize:

If the requested axis has already been assigned to the requesting channel at the point of activation, and its status is that of a neutral axis (not controlled by the PLC) (\$AA\_AXCHANGE\_TYP[<axis>]==3), it is assigned to the NC program (\$AA\_AXCHANGE\_TYP[<axis>]==0).

### Axis in neutral axis status controlled by the PLC

If the axis is in neutral axis status controlled by the PLC (\$AA\_AXCHANGE\_TYP[<axis>]==4), the axis is requested as a neutral axis (\$AA\_AXCHANGE\_TYP[<axis>] == 8). This locks the axis for automatic axis replacement between channels in accordance with the value of bit 0 in MD 10722: AXCHANGE\_MASK (bit 0 == 0). This corresponds to (\$AA\_AXCHANGE\_STAT[<axis>] == 1).

### Axis is active as neutral command axis/oscillating axis or assigned to PLC

If the axis is active as the command axis/oscillating axis or assigned to the PLC for travel, PLC axis == concurrent positioning axis, (\$AA\_AXCHANGE\_TYP[<axis>]==1), the axis is requested as a neutral axis (\$AA\_AXCHANGE\_TYP[<axis>] == 8). This locks the axis for automatic axis replacement between channels in accordance with the value of bit 0 in MD 10722: AXCHANGE\_MASK (bit 0 == 0). This corresponds to (\$AA\_AXCHANGE\_STAT[<axis>] == 1).

A new GET action will request the axis for the NC program (\$AA\_AXCHANGE\_TYP[<axis>] changes to == 7).

### Axis already assigned to NC program

If the axis is already assigned to the NC program (\$AA\_AXCHANGE\_TYP[<axis>]==0) or if this assignment is requested, e.g., axis replacement triggered by NC program (\$AA\_AXCHANGE\_TYP[<axis>]==5 or \$AA\_AXCHANGE\_TYP[<axis>] == 7), there will be no change in state.

### 10.4.16 Axial feed (FA)

#### Function

The axial feed for command axes acts modal.

#### Syntax

FA[<axis>]=<value>

#### Example

Program code	Comments
ID=1 EVERY \$AA_IM[B]>75 DO POS[U]=100 FA[U]=990	; Enter fixed feedrate value. ; Generate feedrate value from main run variables:
ID=1 EVERY \$AA_IM[B]>75 DO POS[U]=100 FA[U]=\$AA_VACTM[W]+100	

### 10.4.17 Software limit switch

#### Function

The working area limitation programmed with G25/G26 is taken into account for the command axes depending on the setting data \$SA\_WORKAREA\_PLUS\_ENABLE.

Switching the working area limitation on and off with G functions WALIMON/WALIMOF in the parts program has no effect on the command axes.

### 10.4.18 Axis coordination

#### Function

Typically, an axis is either moved from the part program or as a positioning axis from a synchronized action.

If the same axis is to be traversed alternately from the part program as a path or positioning axis and from synchronized actions, however, a coordinated transfer takes place between both axis movements.

If a command axis is subsequently traversed from the part program, preprocessing must be reorganized. This, in turn, causes an interruption in the part program processing comparable to a preprocessing stop.

#### Example for traversing X axis alternately from part program and from synchronized actions

Program code	Comments
N10 G01 <b>X100</b> Y200 F1000	; X axis programmed in part program
...	
N20 ID=1 WHEN \$A_IN[1]==1 DO <b>POS[X]=150 FA[X]=200</b>	; Start positioning from synchronized action, if ;digital input is present
...	
CANCEL(1)	; Select synchronized action
...	
N100 G01 <b>X240</b> Y200 F1000	; X becomes path axis; before motion there is a delay time due to axis transfer, if the digital input was 1 and X was positioned from synchronized action.

#### Example of changing traverse command for the same axis:

Program code	Comments
ID=1 EVERY \$A_IN[1]>=1 DO POS[V]=100 FA[V]=560	; Start positioning from synchronized action, if digital >= 1
ID=2 EVERY \$A_IN[2]>=1 DO POS[V]=\$AA_IM[V] FA[V]=790	; Axis tracks, the 2nd input is set, i.e. end position and feedrate for axis V, for two synchronized actions simultaneously active, is tracked on-the-fly while moving.

### 10.4.19 Set actual values (PRESETON)

#### Function

When PRESETON (axis, value) is executed, the current axis position is not changed but a new value is assigned to it.

PRESETON from synchronized actions can be programmed for

- modulo rotary axes that have been started from the part program and
- all command axes that have been started from a synchronized action

#### Syntax

```
DO PRESETON(axis, value)
```

#### Significance

DO PRESETON	Setting actual values in synchronized actions
Axis	Axis of which the control zero is to be changed
Value	The value by which the control zero is to be changed

#### Restrictions for axes

PRESETON cannot be programmed for axes, which are involved in a transformation.

One and the same axis can be moved from the part program and from a synchronized action, only at different times. For this reason, delays can occur in the programming of an axis from the part program if the same axis has been programmed in a synchronized action first.

If the same axis is used alternately, transfer between the two axis movements is coordinated. Part program execution must be interrupted for that.

#### Example

Moving the control zero of an axis

Program code	Comments
WHEN \$AA_IM[a] >= 89.5 DO PRESETON(a4,10.5)	; Shift the control zero of the axis a through 10.5 length units (inch or mm) in the positive axis direction

### 10.4.20 Spindle motions

#### Function

Spindles can be positioned completely unsynchronized with respect to the part program from synchronized actions. This type of programming is advisable for cyclic sequences or operations that are strongly dependent on events.

If conflicting commands are issued for a spindle via simultaneously active synchronized actions, the most recent spindle command takes priority.

#### Example of starting/stopping/positioning spindles

Program code	Comments
ID=1 EVERY \$A_IN[1]==1 DO M3 S1000	; Set the direction of rotation and speed
ID=2 EVERY \$A_IN[2]==1 DO SPOS=270	; Position spindle

#### Example of setting the direction and speed of rotation/ positioning the spindle

Program code	Comments
ID=1 EVERY \$A_IN[1]==1 DO M3 S300	; Set the direction of rotation and speed
ID=2 EVERY \$A_IN[2]==1 DO M4 S500	; Specify a new direction of rotation and a new speed
ID=3 EVERY \$A_IN[3]==1 DO S1000	; Specify a new speed
ID=4 EVERY (\$A_IN[4]==1) AND (\$A_IN[1]==0) DO SPOS=0	; Position spindle

### 10.4.21 Coupled motion (TRAILON, TRAILOF)

#### Function

When the coupling is activated from the synchronized action, the leading axis can be in motion. In this case the following axis is accelerated up to the set velocity. The position of the leading axis at the time of synchronization of the velocity is the starting position for coupled-axis motion. The functionality of coupled-axis motion is described in the Section "Path traversing behavior".

#### Syntax

Switch-in coupled motion

DO TRAILON(following axis, leading axis, coupling factor)

Deactivate coupled-axis motion

DO TRAILOF (following axis, leading axis, leading axis 2)

#### Significance

Activate unsynchronized coupled motion:

... DO TRAILON(FA,  
LA, Kf)      with:  
                  FA: Following axis  
                  LA: Leading axis  
                  Kf: Coupling factor

Deactivate unsynchronized coupled motion:

... DO TRAILOF(FA,  
LA, LA2)      with:  
                  FA: Following axis  
                  LA: Leading axis, optional  
                  LA2: Leading axis 2, option  
... DO TRAILOF(FA)      All couplings to the following axis are disengaged.

## Example

Program code	Comments
\$A_IN[1]==0 DO TRAILON(Y,V,1)	; Activation of the 1st coupled motion group if the digital input is 1
\$A_IN[2]==0 DO TRAILON(Z,W,-1)	; Activation of the 2nd coupled axis group
G0 Z10	; Infeed Z and W axes in the opposite ;axis direction
G0 Y20	; Infeed of Y and V axes in same axis direction
...	
G1 Y22 V25	; Superimpose dependent and independent ;movement of coupled motion axis "V"
...	
TRAILOF (Y,V)	; Deactivate 1st coupled axis group
TRAILOF (Z,W)	; Deactivate 2nd coupled axis group

## Example of conflict avoidance with TRAILOF

The coupled axis is released again for access as a channel axis by invoking the TRAILOF function for the axis. It must ensure that TRAILOF is executed before the channel requests the axis involved. However, this is not the case in this example

...  
N50 WHEN TRUE DO TRAILOF(Y,X)  
N60 Y100

...  
In this case, the axis is not released early enough because the non-modal synchronized action becomes active synchronously with N60 with TRAILOF, see section, Motion-synchronous action, "Structure, basic information".  
To avoid conflict situations the following procedure

should be followed.

...  
N50 WHEN TRUE DO TRAILOF(Y,X)  
N55 WAITP(Y)  
N60 Y100

### 10.4.22 Leading value coupling (LEADON, LEADOF)

#### Function

The axial leading value coupling can be programmed in synchronized actions without restriction. The changing of a curve table for an existing coupling without a previous resynchronization is optionally possible only in synchronized actions.

#### Syntax

Activate master value coupling

DO LEADON (following axis, leading axis, curve table no., OVW)

Deactivate leading value coupling

DO LEADOF(following axis, leading axis, leading axis 2)

#### Significance

Activate axial leading value coupling:

...DO LEADON(FA, LA, with:

NR, OVW)

FA: Following axis

LA: Leading axis

NR: Number of the stored curve table

OVW: Permit overwriting an existing coupling with changed  
curve table

Deactivate axial leading value coupling:

...DO LEADOF(FA, LA) with:

FA: Following axis

LA: Leading axis, optional

... DO LEADOF(FA)

Shortened form without specification of leading axis

### Activate access with synchronized actions RELEASE

The axis to be coupled is released for synchronized action access by invoking the RELEASE function for the axis.

Example:

```
RELEASE (XKAN)
ID=1 every SR1==1 to LEADON(CACH,XKAN,1)
OVW=0 (default value)
```

Without a resynchronization, no new curve table can be specified for an existing coupling. A change of the curve table requires the previous deactivation of the existing coupling and a reactivation with the changed curve table number. This causes a resynchronization of the coupling.

### Changing the curve table for an existing coupling using OVW=1

OVW=1 can be used to specify a new curve table to an existing coupling. No resynchronization is performed. The following axis attempts as fast as possible to follow the position values specified by the new curve table.

## Example of on-the-fly parting

An extruded material which passes continuously through the operating area of a cutting tool must be cut into parts of equal length.

X axis: Axis in which the extruded material moves, WCS

X1 axis: Machine axis of extruded material, MCS

Y axis: Axis in which cutting tool "tracks" the extruded material

It is assumed that the infeed and control of the cutting tool are controlled via the PLC. The signals at the PLC interface can be evaluated to determine whether the extruded material and cutting tool are synchronized.

Actions

Activate coupling, LEADON

Deactivate coupling, LEADOF

Set actual values, PRESETON

---

**10.4 Actions in synchronized actions**

<b>Program code</b>	<b>Comments</b>
N100 R3=1500	; Length of a part to be cut off
N200 R2=100000 R13=R2/300	
N300 R4=100000	
N400 R6=30	; Start position Y axis
N500 R1=1	; Start condition for conveyor axis
N600 LEADOF(Y,X)	; Delete any existing coupling
N700 CTABDEF(Y,X,1,0)	; Table definition
N800 X=30 Y=30	; Value pairs
N900 X=R13 Y=R13	
N1000 X=2*R13 Y=30	
N1100 CTABEND	; End of table definition
N1200 PRESETON(X1,0)	; PRESET at beginning
N1300 Y=R6 G0	; Start position Y axis, axis is linear
N1400 ID=1 WHENEVER \$AA_IW[X]>\$R3 DO PESETON(X1,0)	; PRESET after length R3, new start after disconnecting
N1500 RELEASE(Y)	
N1800 ID=6 EVERY \$AA_IM[X]<10 DO LEADON(Y,X,1)	; For X < 10, couple Y to X via table 1
N1900 ID=10 EVERY \$AA_IM[X]>\$R3-30 DO EADOF(Y,X)	; > 30 before traversed parting distance, deactivate coupling
N2000 WAITP(X)	
N2100 ID=7 WHEN \$R1==1 DO MOV[X]=1 FA[X]=\$R4	; Set extruded material axis continuously in motion
N2200 M30	

### 10.4.23 Measuring (MEAWA, MEAC)

#### Function

Compared with use in traverse blocks of the part program, the measuring function can be activated and deactivated as required.

For further information concerning measuring, see special motion commands "Extended measuring function"

#### Syntax

Axial measurement without deletion of distance-to-go

`MEAWA[axis]=(mode, trigger_event_1, ..._4)`

Continuous measurement without deleting distance-to-go

`MEAC[axis]=(mode, measurement_memory, trigger_event_1, ..._4)`

#### Significance

Program code	Comments	
<code>DO MEAWA</code>	; Switch-in axial measurement	
<code>DO MEAC</code>	; Switch-in continuous measurement	
<code>Axis</code>	; The name of the axis for which measurement is taken	
<code>Mode</code>	<p>; Specification of the <b>tens</b> decade</p> <p>0: active measuring system</p> <p>Number of the measuring systems (depending on the mode)</p> <p>1: 1. Measuring system</p> <p>2: 2. Measuring system</p> <p>3: both measuring systems</p>	Specification of the <b>units</b> decade
		0: Cancel measuring job
		up to 4 trigger events can be activated
		1: concurrently
		2: successively
		3: as for 2, however no monitoring of trigger event1 at the start
<code>Trigger_event_1 to trigger_event_4</code>	<p>; : rising edge, probe</p> <p>-1: falling edge, probe 1 optional</p> <p>2: rising edge, probe 2 optional</p> <p>-2: falling edge, probe 2 optional</p>	
<code>Measurement memory</code>	; Number of the FIFO circulating storage	

## 10.4.24 Initialization of array variables (SET, REP)

### Function

Array variables can be initialized or described with particular values in synchronized actions.

---

#### Note

Only variables that can be described in synchronized actions are possible. Machine data cannot therefore be initialized. Axis variables cannot be specified using the NO\_AXIS value.

---

### Syntax

```
DO ARRAY [n,m]=SET (<value1>,<value2>,...)  
DO ARRAY [n,m]=REP (<value>)
```

### Significance

ARRAY [n,m]	Programmed array indices set Initialization starts at the programmed array indexes. For 2D arrays, the second index is incremented first. This is not done with axis indices.
SET (<value1>,<value2>,...)	Initialization with value lists The array is described from the programmed array indices onwards using the SET parameters. As many array elements are assigned as values are programmed. If more values than exist in the remaining array elements are programmed, an alarm is triggered.
REP (<value>)	Initialization with the same values The array is described from the programmed array indices to the end of the array and repeated using the parameter <value> of REP.

## Example

Program code	Comments
WHEN TRUE DO SYG_IS[0]=REP(0)	; Result: ; SYG_IS[0]=0
WHEN TRUE DO SYG_IS[1]=SET(3,4,5)	SYG_IS[1]=3 SYG_IS[2]=4 SYG_IS[3]=5 SYG_IS[4]=0

### 10.4.25 Set/delete wait markers (SETM, CLEARM)

#### Function

Wait markers can be set or deleted in synchronized actions in order to e.g. coordinate channels with one another.

#### Syntax

```
DO SETM(<marker number>)
DO CLEARM(<marker number>)
```

#### Significance

SETM	Command to set the wait marker for the channel  The command SETM can be programmed in the part program and in the action component of a synchronized action. It sets the marker (<marker number>) for the channel in which the command is applied.
CLEARM	Command to delete the wait marker for the channel  The command CLEARM can be programmed in the part program and in the action component of a synchronized action. It deletes the marker (<marker number>) for the channel in which the command is applied.
<marker number>	Wait marker

## 10.4.26 Fault responses (SETAL)

### Function

Fault responses can be programmed using synchronized actions. Status variables are interrogated and the corresponding actions initiated.

Some possible responses to error conditions are:

- Stop axis (override=0)
- Set alarm

SETAL can set cycle alarms from synchronized actions.

- Set output
- All possible actions in synchronized actions

### Syntax

#### **Set cycle alarm:**

DO SETAL(<alarm number>)

### Significance

SETAL	Command to set a cycle alarm
<alarm number>	Alarm number
	Cycle alarm range for users: 65000 ... 69999

### Example

Program code	Comments
ID=67 WHENEVER (\$AA_IM[X1]-\$AA_IM[X2])<4.567 DO \$AA_OVR[X2]=0	; Stop axis X2 if the safety clearance between axes X1 and X2 is too small.
ID=67 WHENEVER (\$AA_IM[X1]-\$AA_IM[X2])<4.567 DO SETAL(61000)	; Set an alarm if the safety clearance between axes X1 and X2 is too small.

### 10.4.27 Travel to fixed stop (FXS, FXST, FXSW, FOCON, FOCOF)

#### Function

The commands for the function "travel to fixed stop" are programmed using the part program commands FXS, FXST and FXSW in synchronized actions/technology cycles.

Activation can take place without movement, the torque is immediately limited. As soon as the axis is moved via a setpoint, the limit stop monitor is activated.

#### Travel with limited torque/force (FOC)

The function allows torque/force to be changed any time via synchronized actions and can be modally activated or block-for-block.

#### Syntax

```
FXS [<axis>]  
FXST [<axis>]  
FXSW [<axis>]  
FOCON [<axis>]  
FOCOF [<axis>]
```

#### Significance

FXS	Can only be selected in systems with digital drives (FDD, MSD, HLA)
FXST	Changes the clamping torque FXST
FXSW	Changes the monitoring window FXSW
FOCON	Activates the modally effective torque/force limiting
FOCOF	Disables the torque/force limiting
<axis>	Axis identifier The following is permissible: <ul style="list-style-type: none"><li>• Geometry axis identifier</li><li>• Channel axis identifier</li><li>• Machine axis identifier</li></ul>

---

#### Note

A selection may only be carried out once.

---

## Examples

### Example 1: Travel to fixed stop (FXS), initiated using a synchronized action

Program code	Comments
Y axis:	; Static synchronized actions
Activate:	
N10 IDS=1 WHENEVER ((\$R1==1) AND \$AA_FXS[Y]==0) D \$R1=0 FXS[Y]=1 FXST[Y]=10 FA[Y]=200 POS[Y]=150	; FXS is activated for axis Y by setting \$R1=1, the effective torque is reduced to 10% and travel is started towards the fixed stop.
N11 IDS=2 WHENEVER (\$AA_FXS[Y]==4) DO FXST[Y]=30	; As soon as the fixed stop was detected (\$AA_FXS[Y]==4), the torque was reduced to 30%.
N12 IDS=3 WHENEVER (\$AA_FXS[Y]==1) DO FXST[Y]=\$R0	; After the fixed stop is reached the torque is controlled depending on R0.
N13 IDS=4 WHENEVER ((\$R3==1) AND \$AA_FXS[Y]==1) DO FXS[Y]=0 FA[Y]=1000 POS[Y]=0	; De-selection depending on R3 and retract.
N20 FXS[Y]=0 G0 G90 X0 Y0	; Normal program execution:
N30 RELEASE(Y)	; Release axis Y for motion in the synchronized action.
N40 G1 F1000 X100	; Motion of another axis.
N50 ...	
N60 GET(Y)	; Include axis Y back into the path group

### Example 2: Activating torque/force limiting (FOC)

Program code	Comments
N10 FOCON[X]	; Modal activation of limiting.
N20 X100 Y200 FXST[X]=15	; X travels with reduced torque (15%).
N30 FXST[X]=75 X20	; The torque is changed to 75%, X travels with this limited torque.
N40 FOCOF[X]	; The torque limiting is switched-out.

## Further Information

### Multiple selection

If the function is called once more due to incorrect programming after activating (`FXS[axis] = 1`) the following alarm is output:

Alarm 20092 "Travel to fixed stop is still active"

Programming, that either interrogates `$AA_FXS[ ]` or a dedicated bit memory (here R1) in the condition, avoids activating the function "Part program fragment" a multiple number of times:

```
Program code
N10 R1=0
N20 IDS=1 WHENEVER ($R1==0 AND
$AA_IW[AX3] > 7) DO R1=1 FXST[AX1]=12
```

### Block-related synchronized actions

By programming a block-related synchronized action, travel to fixed stop can be connected during an approach motion.

Example:

Program code	Comments
N10 G0 G90 X0 Y0 N20 WHEN \$AA_IW[X] > 17 DO FXS[X]=1 N30 G1 F200 X100 Y110	; FXS is activated if X reaches a position greater than 17 mm.

### Static and block-related synchronized actions

In static and block-related synchronized actions, the same commands `FXS`, `FXST` and `FXSW` can be used as in a normal part program execution. The values assigned can be resulted from a calculation.

## 10.4.28 Determining the path tangent in synchronized actions

### Function

The system variable `$AC_TANE` (Tangent ANgle at End of Block), which can be read in synchronized actions, calculates the angle between the path tangent at the end of the current block and the path tangent at the start of the programmed following block.

## Parameters

The tangent angle is always output positive in the range 0.0 to 180.0 degrees. If there is no following block in the main run, the angle -180.0 degrees is output.

The system variable \$AC\_TANE<sub>B</sub> should not be read for blocks generated by the system (intermediate blocks). The system variable \$AC\_BLOCKTYPE is used to tell whether it is a programmed block (main block).

## Example

```
ID=2 EVERY $AC_BLOCKTYPE==0 DO $SR1 = $AC_TANEB
```

### 10.4.29 Determining the current override

#### Function

##### **The current override**

(NC component) can be read and written with system variables:

\$AA\_OVR Axial override

\$AC\_OVR Path override

in synchronized actions.

The override defined by the PLC is provided for synchronized actions to read in the system variables:

\$AA\_PL<sub>C</sub>\_OVR Axial override

\$AC\_PL<sub>C</sub>\_OVR Path override

##### **The resulting override**

is provided for synchronized actions to read in the system variables:

\$AA\_TOTAL\_OVR Axial override

\$AC\_TOTAL\_OVR Path override

##### **The resulting override can be calculated as:**

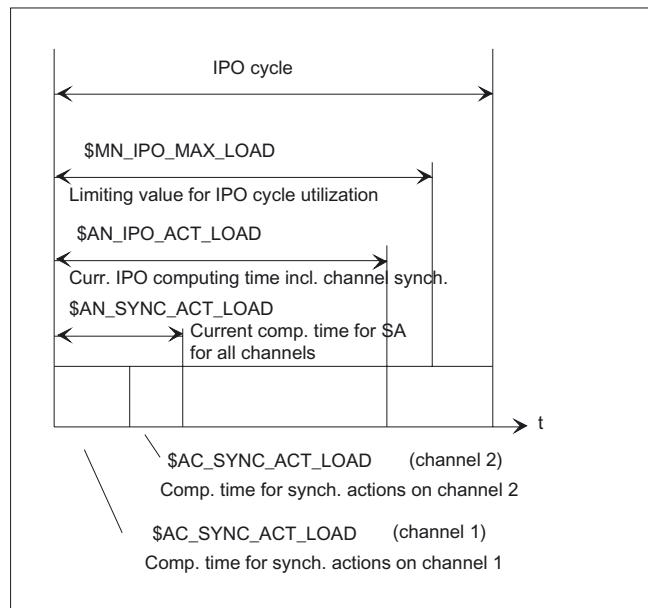
\$AA\_OVR \* \$AA\_PL<sub>C</sub>\_OVR or

\$AC\_OVR \* \$AC\_PL<sub>C</sub>\_OVR

### 10.4.30 Time use evaluation of synchronized actions

#### Function

In a interpolation cycle, synchronized actions have to be both interpreted and motions calculated by the NC. The system variables presented below provide synchronized actions with information about the current time shares that synchronized actions have of the interpolation cycle and about the computation time of the position controllers.



#### Significance

The variables only have valid values if machine data `$MN_IPO_MAX_LOAD` is greater than 0. Otherwise the variables for both SINUMERIK powerline and solution line systems always specify the net computing time during which the interrupts caused by HMI are no longer taken into account. The net computing time results from:

- synchronized action time,
- position control time and
- remaining IPO computing time without interrupts caused by HMI

The system variables always contain the values of the previous IPO cycle.

\$AN_IPO_ACT_LOAD	current IPO computing time (incl. synchronized actions of all channels)
\$AN_IPO_MAX_LOAD	longest IPO computing time (incl. synchronized actions of all channels)
\$AN_IPO_MIN_LOAD	shortest IPO computing time (incl. synchronized actions of all channels)
\$AN_IPO_LOAD_PERCENT	current IPO computing time as percentage of IPO cycle (%)
\$AN_SYNC_ACT_LOAD	current computing time for synchronized actions over all channels
\$AN_SYNC_MAX_LOAD	longest computing time for synchronized actions over all channels
\$AN_SYNC_TO_IPO	percentage share that the synchronized actions have of the complete IPO computer time (over all channels)
\$AC_SYNC_ACT_LOAD	current computing time for synchronized actions in the channel
\$AC_SYNC_MAX_LOAD	longest computing time for synchronized actions in the channel
\$AC_SYNC_AVERAGE_LOAD	average computing time for synchronized actions in the channel
\$AN_SERVO_ACT_LOAD	current computing time of the position controller
\$AN_SERVO_MAX_LOAD	longest computing time of the position controller
\$AN_SERVO_MIN_LOAD	shortest computing time of the position controller

**Variable for the overload notification:**

The machine data \$MN\_IPO\_MAX\_LOAD is used to set the net IPO computing time (as % of IPO cycle) from which the system variable

\$AN\_IPO\_LOAD\_LIMIT will be set to TRUE. If the current load falls below this limit, the variable is again set to FALSE. If the machine data is 0, the entire diagnostic function is deactivated.

The evaluation of \$AN\_IPO\_LOAD\_LIMIT allows the user to define a strategy for avoiding a level overflow.

## 10.5 Technology cycles

### Function

As an action in synchronized actions, you can invoke programs. These must consist only of functions that are permissible as actions in synchronized actions. Programs structured in this way are called technology cycles.

Technology cycles are stored in the control as subprograms.

It is possible to process several technology cycles or actions in parallel in one channel.

### Programming

The following rules apply when programming technology cycles:

- End of program is programmed with M02/M17/M30/RET.
- All actions specified in ICYCOF can be processed in one cycle without waiting cycles within one program level.
- Up to 8 technology cycles can be queried one after another per synchronized action.
- Technology cycles are also possible in synchronized actions that are effective block-for-block (non-modal).
- Both IF check structures and GOTO, GOTOF and GOTOB jump instructions can be programmed.
- The following applies to blocks with DEF and DEFINE instructions:
  - DEF and DEFINE instructions are read over (ignored) into technology cycles.
  - These result in alarm messages if the syntax is incorrect or incomplete.
  - They can be read over without an alarm message without being set-up themselves.
  - They are taken into full consideration with value assignments as part program cycle.

## Parameter transfer

Parameter transfer to technology cycles is possible. Both simple data types which are transferred as formal "Call by Value" parameters and default settings which take effect when technology cycles are called up are taken into account. These are:

- Programmed default values when no transfer parameters are programmed.
- Standard parameter with initial values.
- Non-initialized actual parameters with a default value are transferred.

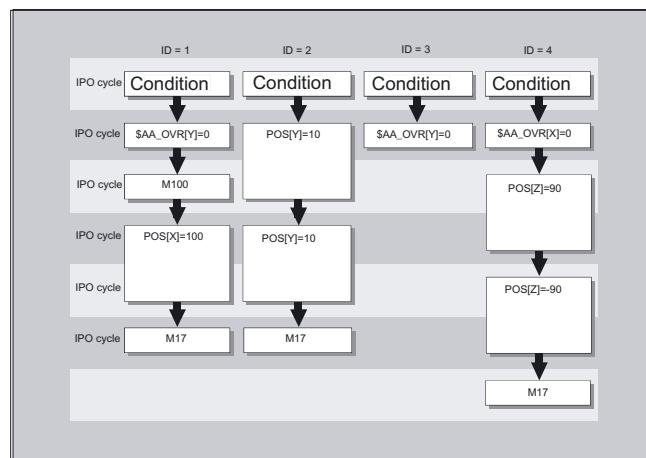
## Sequence

Technology cycles are started as soon as their conditions have been fulfilled. Each line in a technology cycle is processed in a separate IPO cycle. Several IPO cycles are required to execute positioning axes. Other functions are executed in one cycle. Blocks are sequentially executed in the technology cycle.

If actions that are mutually exclusive are called up in the same interpolation cycle, the action that is called up from the synchronized action with the higher ID number becomes active.

## Examples

### Example 1: Axis programs are started by setting digital inputs.



Main program:

Program code	Comments
ID=1 EVERY \$A_IN[1]==1 DO AXIS_X	; If input 1 is 1, start axis program X.
ID=2 EVERY \$A_IN[2]==1 DO AXIS_Y	; If input 2 is 1, start axis program Y.
ID=3 EVERY \$A_IN[3]==1 DO \$AA_OVR[Y]=0	If input 3 is 1, set override of axis Y to 0.
ID=4 EVERY \$A_IN[4]==1 DO AXIS_Z	; If input 4 is 1, start axis program Z
M30	

Technology cycle AXIS\_X:

```
Program code
$AA_OVR[Y]=0
M100
POS[X]=100 FA[X]=300
M17
```

Technology cycle AXIS\_Y:

```
Program code
POS[Y]=10 FA[Y]=200
POS[Y]=-10
M17
```

Technology cycle AXIS\_Z:

```
Program code
$AA_OVR[X]=0
POS[Z]=90 FA[Z]=250
POS[Z]=-90
M17
```

**Example 2: Various program sequences in the technology cycle**

**Program code**

```
PROC CYCLE  
N10 DEF REAL "value"=12.3  
N15 DEFINE ABC AS G01
```

Both blocks are read over without the variables and/or macros being set-up.

**Program code**

```
PROC CYCLE  
N10 DEF REAL  
N15 DEFINE ABC G01
```

Both blocks still result in the NC alarm because the syntax is not written correctly.

**Program code**

```
PROC CYCLE  
N10 DEF AXIS "axis1"=XX2
```

If axis XX2 is not known, alarm 12080 is output. Otherwise the block is read over without alarms and without the variables being set-up.

**Program code**

```
PROC CYCLE  
N10 DEF AXIS "axis1"  
N15 G01 X100 F1000  
N20 DEF REAL"valu1"
```

Block N20 always results in alarm 14500 because the DEF instruction is not permitted after the first program line.

### 10.5.1 Context variable (\$P\_TECCYCLE)

#### Function

The \$P\_TECCYCLE variables can be used to divide programs into synchronized action programs and preprocessing programs. It is then possible to process blocks or program sequences that are written correctly (in terms of syntax) or alternatively process them as the part program cycle.

#### Interpreting context variable

The \$P\_TECCYCLE system variable allows context-specific interpretation of program sections to be controlled in technology cycles, if:

IF \$P\_TECCYCLE==TRUE Program sequence for technology cycle in synchronized action

otherwise

ELSE Program sequence for part program cycle

---

#### Note

A block with incorrect or unauthorized program syntax as well as unknown value assignments also result in an alarm message in the part program cycle.

---

#### Example

Program sequence with interrogation of \$P\_TECCYCLE in the technology cycle:

Program code	Comments
<pre> PROC CYCLE N10 DEF REAL "value1" N15 G01 X100 F1000 N20 IF \$P_TECCYCLE==TRUE N25 "Program sequence for technology cycle (without variable value1)" N30 ELSE N35 "Program sequence for part program cycle (variable value1 is present)" ENDIF </pre>	<p>; Is read over in the technology cycle.</p>

### 10.5.2 Call by value parameters

#### Function

Technology cycles can be defined using call by value parameters. Simple data types such as INT, REAL, CHAR, STRING, AXIS and BOOL can be used as parameters.

---

#### Note

Formal parameters that are transferred to call by values cannot be arrays.

The current parameters can also consist of default parameters,  
see Section "Initializing Default Parameters".

---

#### Syntax

```
ID=1 WHEN $AA_IW[X]>50 DO TEC(IVAL, RVAL, , SVAL, AVAL)  
A default value is transferred for non-initialized actual parameters  
ID=1 WHE $AA_IW[X]>50 DO TEC(IVAL, RVAL, , SYG_SS[0], AVAL)
```

### 10.5.3 Default parameter initialization

#### Function

Default parameters can also be provided with an initial value in the PROC instructions.

#### Syntax

Assign default parameters in the technology cycle:

```
PROC TEC (INT IVAL=1, REAL RVAL=1.0, CHAR CVAL='A', STRING[10]  
SVAL="ABC", AXIS AVAL=X, BOOL BVAL=TRUE)
```

If a current parameter consists of a default parameter, the initial value is transferred from the PROC instruction. This applies both in the part program and in synchronized actions.

## Example

Program code	Comments
TEC (IVAL, RVAL, SVAL, AVAL)	; For CVAL and BVAL, the initial value applies

## 10.5.4 Control processing of technology cycles (ICYCOF, ICYCON)

### Function

The ICYCOF and ICYCON language commands are used to control the time processing of technology cycles.

All blocks of a technology cycle are processed in just one interpolation cycle using ICYCOF. All actions which require several cycles result in parallel processes with ICYCOF.

### Application

With ICYCON, command axis movements can result in a delay to the processing of a technology cycle. If this is not wanted, then all actions can be processed with ICYCOF in one interpolation cycle without waiting times.

### Syntax

The following applies to the cyclic processing of technology cycles:

ICYCON each block of a technology cycle is processed in a separate IPO cycle following ICYCON.

ICYCOF all subsequent blocks of a technology cycle are processed in one interpolation cycle following ICYCOF.

---

### Note

The two ICYCON and ICYCOF language commands are only effective within the program level. Both commands are easily overlooked without a response in the part program.

---

### **Example of ICYCOF processing mode**

<b>Program code</b>	<b>Comments</b>
IPO cycle	; PROC TECHNOCYC
1.	; \$R1=1
2.25	; POS[X]=100
26.	; ICYCOF
26.	; \$R1=2
26.	; \$R2=\$R1+1
26.	; POS[X]=110
26.	; \$R3=3
26.	; RET

### **10.5.5 Cascading technology cycles**

#### **Function**

Up to 8 technology cycles can be processed switched in line. Several technology cycles can then be programmed in one synchronized action.

#### **Syntax**

```
ID=1 WHEN $AA_IW[X]>50 DO TEC1($R1) TEC2 TEC3(X)
```

#### **Sequence of execution**

The technology cycles are processed in order (in a cascade) working from left to right in accordance with the aforementioned programming. If a cycle is to be processed in ICYCON mode, this delays all the subsequent processing actions. An alarm aborts all subsequent actions.

### 10.5.6 Technology cycles in non-modal synchronized actions

#### Function

Technology cycles are also possible in non-modal synchronized actions.

If the processing time of a technology cycle is longer than the processing time of the associated block, the technology cycle is aborted when the block is changed.

---

#### Note

A technology cycle does not prevent the block change.

---

### 10.5.7 Check structures (IF)

#### Function

IF check structures can be used in synchronized actions for branches in the processing sequence of technology cycles.

#### Syntax

```
IF <condition>
$R1=1
[ELSE] optional
$R1=0
ENDIF
```

### **10.5.8 Jump instructions (GOTO, GOTOF, GOTOB)**

#### **Function**

Jump instructions (GOTO, GOTOF, GOTOB) are possible in technology cycles. The specified labels must be present in the subprograms to prevent alarms from being triggered.

---

#### **Note**

Labels and block numbers may only be constants.

---

#### **Syntax**

##### **Unconditional jumps**

GOTO Label, block number

GOTOF Label, block number

GOTOB Label, block number

#### **Jump instructions and jump destinations**

GOTO	Firstly jump forwards and then backwards
GOTOF	Jump forwards
GOTOB	Jump backwards
Label:	Jump marker
Block number	Jump destination for this block
N100	Block number is subblock
:100	Block number is main block

## 10.5.9 Lock, unlock, reset (LOCK, UNLOCK, RESET)

### Function

The sequence of a technology cycle can be locked, unlocked or reset using another modal synchronized action.

### Syntax

```
LOCK(<n1>,<n2>,...)
UNLOCK(<n1>,<n2>,...)
RESET(<n1>,<n2>,...)
```

### Significance

LOCK	Command to lock synchronized actions The active action is interrupted.
UNLOCK	Command to unlock synchronized actions
RESET	Command to reset technology cycles
<n1>,<n2>,...	Identification numbers of synchronized actions or technology cycles that are to be locked, unlocked or reset.

### Interlocking synchronized actions

Modal synchronized actions with ID numbers  $<n> = 1 \dots 64$  can be interlocked from the PLC. The associated condition is no longer evaluated and execution of the associated function is locked in the NCK.

All synchronized actions can be locked indiscriminately with one signal in the PLC interface.

---

### Note

A programmed synchronized action is active as standard and can be protected against overwriting/locking by a machine data setting.

It should not be possible for end users to modify synchronized actions defined by the machine manufacturer.

---

## Examples

### Example 1: Lock synchronized actions (LOCK)

```
Program code
N100 ID=1 WHENEVER $A_IN[1]==1 DO M130
...
N200 ID=2 WHENEVER $A_IN[2]==1 DO LOCK(1)
```

### Example 2: Unlock synchronized actions (UNLOCK)

```
Program code
N100 ID=1 WHENEVER $A_IN[1]==1 DO M130
...
N200 ID=2 WHENEVER $A_IN[2]==1 DO LOCK(1)
...
N250 ID=3 WHENEVER $A_IN[3]==1 DO UNLOCK(1)
```

### Example 3: Interrupt technology cycle (RESET)

```
Program code
N100 ID=1 WHENEVER $A_IN[1]==1 DO M130
...
N200 ID=2 WHENEVER $A_IN[2]==1 DO RESET(1)
```

## 10.6 Delete synchronized action (CANCEL)

### Function

Modal synchronized actions (`ID=<n>` or `IDS=<n>`) can only be deleted with CANCEL directly from the part program.

---

#### Note

Incomplete movements originating from a canceled synchronized action are completed as programmed.

---

### Syntax

`CANCEL(<n1>,<n2>,...)`

### Significance

<code>CANCEL</code>	Command to delete programmed synchronized actions
<code>&lt;n1&gt;,&lt;n2&gt;,...</code>	Identification numbers of the synchronized actions to be deleted

### Example

Program code	Comments
<code>N100 ID=2 WHENEVER \$A_IN[1]==1 DO M130</code>	
<code>...</code>	
<code>N200 CANCEL(2)</code>	<code>; Deletes synchronized action No. 2.</code>

## **10.7      Restrictions**

Boundary conditions apply for when the following events arise:

- Power on
- Mode change
- Reset
- NC Stop
- End of program
- Block search
- Program interruption by the asynchronous subprogram ASUB
- Repositioning REPOS
- Deselection with CANCEL

### **Power on**

No synchronized actions are ever active during POWER ON. Static synchronized actions can be activated by an asynchronous subprogram (ASUB) started by the PLC.

### **Mode change**

Synchronized actions activated by keyword `IDS` remain active after a change in operating mode. All other synchronized actions become inactive following operating mode changeover (e.g., axis positioning) and become active again following repositioning and a return to automatic mode.

## Reset

All non-modal and modal synchronized actions are ended by a NC reset. Static synchronized actions remain active. They can start new actions. If a command axis movement is active during RESET, this is aborted. Completed synchronized actions of the WHEN type are not processed again after RESET.

Response following RESET		
Synchronized action/ technology cycle	Modal/non-modal	Static (IDS)
	Active action is aborted, synchronized actions are canceled	Active action is aborted, technology cycle is reset
Axis/ positioning spindle	Motion is aborted	Motion is aborted
Speed-controlled spindle	\$MA_SPIND_ACTIVE_AFTER_RESET==1: Spindle remains active  \$MA_SPIND_ACTIVE_AFTER_RESET==0: Spindle stops.	
Master value coupling	\$MC_RESET_MODE_MASK, bit13 == 1: Master value coupling remains active  \$MC_RESET_MODE_MASK, Bit13 == 0: Master value coupling is separated	
Measuring operations	Measuring operations started from synchronized actions are aborted	Measuring operations started from static synchronized actions are aborted

## NC stop

**Static** synchronized actions remain active for NC stop. Movements started from static synchronized actions are not canceled. Synchronized actions that are **local to the program** and belong to the active block remain active, movements started from them are stopped.

## End of program

End of program and synchronized action do not influence one another. Current synchronized actions are completed even after end of program. Synchronized actions active in the M30 block remain active. If this is undesirable, then you must cancel the synchronized actions using `CANCEL` before the end of the program.

Response following end of program		
Synchronized action/ technology cycle	Modal and non-modal actions are aborted	Static actions (IDS) remain active
<b>Axis/ positioning spindle</b>	M30 is delayed until the axis / spindle is stationary.	Motion continues
<b>Speed-controlled spindle</b>	End of program: <code>\$MA_SPIND_ACTIVE_AFTER_RESET==1</code> : Spindle remains active <code>\$MA_SPIND_ACTIVE_AFTER_RESET==0</code> : Spindle stops  The spindle remains active if the operating mode changes	Spindle remains active
<b>Master value coupling</b>	<code>\$MC_RESET_MODE_MASK</code> , bit13 == 1: Master value coupling remains active <code>\$MC_RESET_MODE_MASK</code> , Bit13 == 0: Master value coupling is separated	A coupling started from a static synchronized action remains active
<b>Measuring operations</b>	Measuring operations started from synchronized actions are aborted	Measuring operations started from static synchronized actions remain active

## Block search

Synchronized actions are collected during a block search and evaluated on NC Start; the associated actions are then started if necessary. Static synchronized actions are active during block search. If polynomial coefficients programmed with `FCTDEF` are found during a block search, they are written directly to the setting data.

## Program interruption using an asynchronous subprogram ASUB

ASUB start:

Modal and static motion-synchronous actions remain active and are also operative in the asynchronous subprogram.

ASUB end:

If the synchronized subprogram is not resumed with REPOS modal and static motion-synchronized actions that were modified in the synchronized subprogram remain active in the main program.

## Repositioning (REPOS)

After repositioning (REPOS), the synchronized actions effective in the interrupted block become active again. Modal synchronized actions, changed from the asynchronous subprogram are no longer effective after REPOS when processing the remainder of the block.

Polynomial coefficients programmed with FCTDEF are not influenced by asynchronous subprograms and REPOS. No matter where they were programmed, they can be used at any time in the synchronized subprogram and in the main program after execution of REPOS.

## Deselecting with CANCEL

If an active synchronized action is deselected with CANCEL, this does not affect the active action. Positioning motions are completed as programmed.

The CANCEL command is used to interrupt a modally or statically active synchronized action. If a synchronized action is canceled while the positioning axis movement that was activated from it is still active, the positioning axis movement is interrupted. If this is not desired, axis motion can be braked with axial delete distance to go before the CANCEL command.

Example:

Program code	Comments
ID=17 EVERY \$A_IN[3]==1 DO POS[X]=15 FA[X]=1500	; Start positioning axis motion.
...	
WHEN ... DO DELDTG(X)	; End positioning axis motion.
CANCEL(1)	



# 11

## Oscillation

### 11.1 Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE)

#### Function

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 synchronized 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 addresses allow synchronized oscillation to be activated and controlled 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.

#### Activate, deactivate oscillation: OS

```
OS[axis] = 1: resistor  
OS[axis] = 0: switch off
```

## *Oscillation*

---

### 11.1 Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE)

#### Significance

OSP1 [axis]=	Position of reversal point 1 (oscillating: left reversal point)
OSP2 [axis]=	Position of reversal point 2 (oscillating: right reversal point)
OST1 [axis]=	Stopping time at reversal points in seconds
OST2 [axis]=	
FA[axis]=	Feed for oscillating axis
OSCTRL [axis]=	(Set, reset options)
OSNSC [axis]=	Number of sparking-out strokes
OSE [axis]=	End position
OS [axis]=	1 = activate oscillation; 0 = deactivate oscillation

#### Stopping times at reversal points: OST1, OST2

Stop time	Movement in exact stop area at reversal point
-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 stopping time

The unit for the stopping time is identical to the stopping time programmed with G4.

## 11.1 Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE)

**Example of an oscillating axis that should oscillate between two reversal points**

The oscillation axis Z must oscillate between 10 and 100. Approach reversal point 1 with exact stop fine, reversal point 2 with exact stop coarse. Machining is performed with feedrate 250 for the oscillating axis. Three sparking-out strokes must be executed at the end of the machining operation followed by approach by oscillation axis to end position 200. The feedrate for the infeed axis is 1, end of the infeed in X direction is at 15.

<b>Program code</b>	<b>Comments</b>
WAITP(X, Y, Z)	; Initial setting
G0 X100 Y100 Z100	; Changeover into positioning axis operation
N40 WAITP(X, Z)	
N50 OSP1[Z]=10 OSP2[Z]=100 ->	; Reversal point 1, reversal point 2
-> OSE[Z]=200 ->	
-> OST1[Z]=0 OST2[Z]=-1 ->	End position
-> FA[Z]=250 FA[X]=1 ->	Stopping time at U1: Exact stop fine;
-> OSCTRL[Z]=(4, 0) ->	Stopping time at U2: Exact stop coarse
-> OSNSC[Z]=3 ->	Feedrate for oscillating axis, infeed axis
N60 OS[Z]=1	Setting options
	Three spark-out strokes
	Start oscillation
N70 WHEN \$A_IN[3]==TRUE ->	; Deletion of distance-to-go
-> DO DELDTG(X)	
N80 POS[X]=15	; Starting position, X axis
N90 POS[X]=50	
N100 OS[Z]=0	; Stop oscillation
M30	

-> can be programmed in a single block.

**Example of oscillation with online change of the reversal position****Setting data**

The setting data necessary for asynchronous oscillation can be set in the part program.

If the setting data are described directly in the program, the change takes effect during preprocessing. A synchronized response can be achieved by means of a STOPRE preprocessing stop.

Program code	Comments
\$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 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	

**Description**

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 G1 is 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 a dynamic transformation
- act as a guide axis for gantry and combined-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 1`

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:

`N10 OSP1[Z]=100 OSP2[Z]=110`

.

.

`N40 OSP1[Z]=IC (3)`

### Note

`WAITP (axis):`

- If oscillation is to be performed with a geometry axis, you must enable this axis for oscillation with `WAITP`.
- When oscillation has finished, this command is used to enter the oscillating axis as a positioning axis again for normal use.

## Oscillation with motion-synchronous actions and stop times, OST1/OST2

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 motional 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 the case.**

## Setting feed, FA

The feedrate is the defined feedrate of the positioning axis. If no feedrate is defined, the value stored in the machine data applies.

## Oscillation

### 11.1 Asynchronous oscillation (OS, OSP1, OSP2, OST1, OST2, OSCTRL, OSNSC, OSE)

#### Defining the sequence of motions, OSCTRL

The control settings for the movement are set with enable and reset options.

OSCTRL[oscillating axis] = (set-option, reset-option)

The set options are defined as follows (the reset options deselect the settings):

#### Reset options

These options are deactivated (only if they have previously been activated as setting options).

#### Setting options

These options are switched over. When OSE (end position) is programmed, option 4 is implicitly activated.

Option value	Meaning
0	When the oscillation is deactivated, stop at the next reversal point (default) only possible by resetting values 1 and 2
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 the oscillation movement is canceled by deletion of the distance-to-go: then execute spark-out strokes and approach end position if appropriate
16	If the oscillation movement is canceled by deletion of the distance-to-go: reversal position is approached as with deactivation
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	0=The sparking out stroke is a dual stroke.(default) 1=single stroke.

Several options are appended with plus characters.

**Example:**

The oscillating motion for the Z axis should stop at the reversal point 1 when switched off.  
Where

- an end position is approached,
- a changed feed acts immediately and should immediately stop the axis after the deletion of distance-to-go.

OSCTRL [Z] = (1+4, 16+32+64)

## 11.2 Control oscillation via synchronized actions

### Function

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

**Significance**

OSP1 [OscillationAxis]=	Position of reversal point 1
OSP2 [OscillationAxis]=	Position of reversal point 2
OST1 [OscillationAxis]=	Stopping time at reversal point 1 in seconds
OST2 [OscillationAxis]=	Stopping time at reversal point 2 in seconds
FA [OscillationAxis]=	Feed for oscillating axis
OSCTRL [OscillationAx is]=	Set or reset options
OSNSC [OscillationAxis]=	Number of sparking-out strokes
OSE [OscillationAxis] =	end position
WAITP (oscillation axis)	Enable axis for oscillation

**Axis assignment, infeed**

OSCILL[oscillation axis] = (infeed axis1, infeed axis2, infeed axis3)

POSP[InfeedAxis] = (Endpos, Partial length, Mode)

OSCILL	Assign infeed axis or axes for oscillating axis
POSP	Define complete and partial infeeds (see the "File and Program Management" chapter)
Endpos	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 0 = Two residual steps of equal size (default); 1 = All partial infeeds of equal size

**Motion-synchronous actions**

WHEN... ... DO

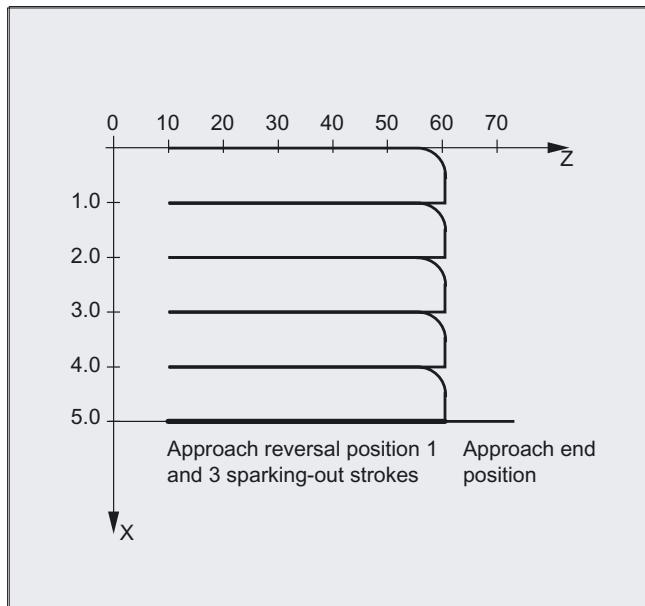
when ... , do ...

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	Comments
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	Comments
WHENEVER \$AA_IM[Z]<\$SA_OSCILL_REVERSE_POS2[Z] DO -> -> \$AA_OVR[X]=0 \$AC_MARKER[0]=0	; If the actual position of the 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.
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 is the same as the partial infeed, 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	; If the bit memory with the index 0 is the same, then always set the axial override of the infeed axis X to 100% - this prevents premature infeed (oscillating axis Z has still not again exited the reversal range 2, however, infeed axis X is ready for a new infeed), always set the axial override of the oscillating axis Z to 100% (this cancels the 2nd synchronized action).

-> must be programmed in a single block

## 3. Start oscillation

Program code	Comments
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

## Description

### 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 pos, 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 ...	if..., then...
WHENEVER ... DO	always if..., then...

### 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 point 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 point 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 obtained under the given assumptions (refer above):

**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 obtained under the given assumptions (refer above):

#### 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 obtained under the given assumptions (refer above):

<pre>WHENEVER \$AA_DTEPW[X]==0 DO \$AA_OVR[Z]= 100</pre>	<p>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%.</p>
--	--

### 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%.

# 12

## Punching and nibbling

### 12.1 Activation, deactivation

#### 12.1.1 Punching and nibbling on or off (SPOF, SON, PON, SONS, PONS, PDELAYON, PDELAYOF)

##### Function

###### **Punching and nibbling activate, deactivate, PON/SON**

Activate the punch or nibble function using **PON** and **SON**. **SPOF** ends all punching and nibbling-specific functions. Modal commands **PON** and **SON** are mutually exclusive, i.e., **PON** deactivates **SON** and vice versa.

###### **Punching and nibbling with leader, PONS/SONS**

The functions **SONS** and **PONS** also switch-in the punching or nibbling functions.

Contrary to **SON/PON** – stroke control at the interpolation level – with these functions, the signal-related control of the stroke initiation is at the servo level. This means that you can work with higher stroke frequencies and thus with an increased punching capacity.

While signals are evaluated in the leader, all functions that cause the nibbling or punching axes to change position are inhibited.

Example: Handwheel mode, changes to frames via PLC, measuring functions.

###### **Punching with delay, PDELAYON/PDELAYOF**

**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**.

## Syntax

```
PONS G... X... Y... Z...
SON G... X... Y... Z...
SONS G... X... Y... Z...
SPOF
PDELAYON
PDELAYOF
PUNCHACC (<Smin>, <Amin>, <Smax>, <Amax>)
```

## Significance

PON	Punching ON
PONS	Punching with leader on
SON	Nibbling ON
SONS	Nibbling with leader on
SPOF	Punching, nibbling off
PDELAYON	Punching with delay ON
PDELAYOF	Punching with delay OFF
PUNCHACC	Travel-dependent acceleration
<Smin>	Minimum hole spacing
<Smax>	Maximum hole spacing
<Amin>	Initial acceleration
<Amax>	<Amin> can be greater than <Amax>. Final acceleration
<Amax>	<Amax> can be greater than <Amin>.

## Use of M commands

By using macro technology, you can also use M commands instead of language commands:

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

## Punching and nibbling with leader, PONS/SONS

Punching and nibbling with a leader is not possible in more than one channel simultaneously. PONS or SONS can only be activated in one channel at a time.

If PONS or SONS is activated in more than one channel at a time, alarm 2200 "Channel %1 fast punching/nibbling not possible in several channels" detects this impermissible action.

Otherwise, PONS and SONS work in exactly the same way as PON and SON.

## Travel-dependent acceleration PUNCHACC

The PUNCHACC ( $S_{min}, A_{min}, S_{max}, A_{max}$ ) language command defines an acceleration curve that can define different accelerations (A) depending on the hole spacing (S).

Example for PUNCHACC (2, 50, 10, 100):

*Distance between holes less than 2 mm:*

Traversal acceleration is 50% of maximum acceleration.

*Distance between holes from 2 mm to 10 mm:*

Acceleration is increased to 100%, proportional to the spacing.

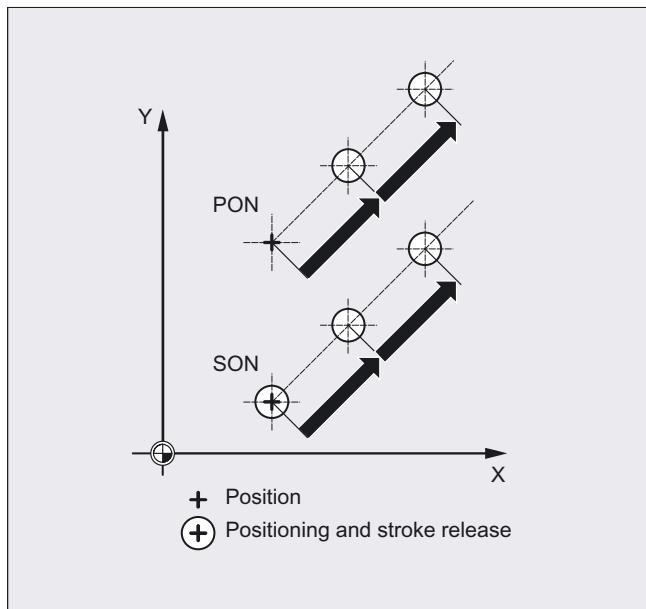
*Distance between holes more than 10 mm:*

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, if you wish to initiate a stroke at the same position, you can program one of the punching/nibbling axes with a traversing path of 0.

---

#### Note

#### Machining with rotatable tools

Use the tangential control function if you wish to position rotatable tools at a tangent to the programmed path.

---

## 12.2 Automatic path segmentation

### Function

#### Segmentation into path segments

When punching or nibbling is activated, both SPP as well as also 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; on the other hand 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=

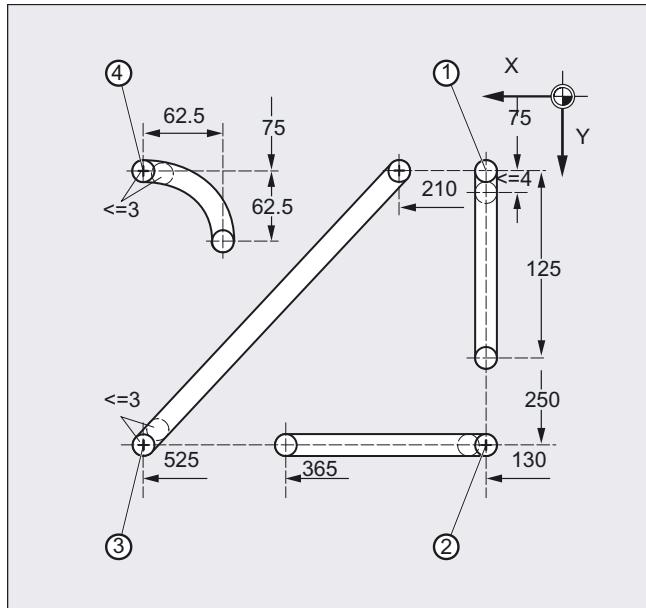
SPN=

### Significance

SPP	Size of path segment (maximum distance between strokes); modal
SPN	Number of path segments per block; modally effective

### Example 1

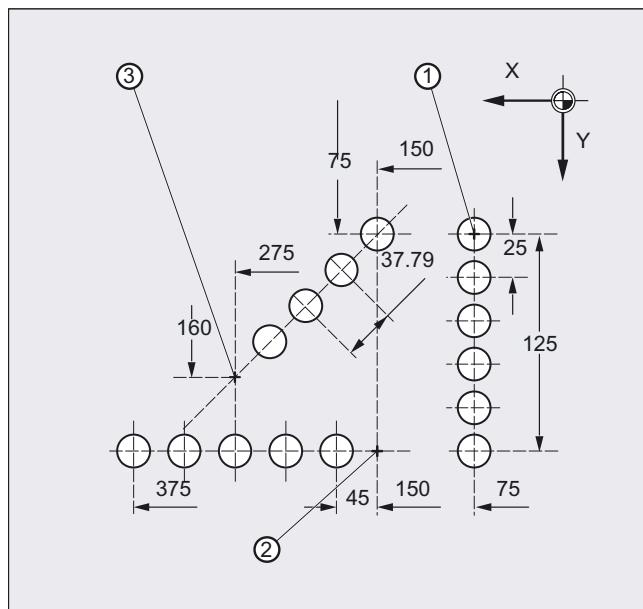
The programmed nibbling segments should be automatically split-up into path segments.



Program code	Comments
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 (3.81 lb)
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 (3.81 lb)
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 (3.81 lb)
N140 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 (3.81 lb)
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	Comments
N100 G90 X75 Y75 F60 PON	; Position to starting point 1; punch an individual hole
N110 G91 Y125 SPP=25	; Maximum path segment length for automatic path segmentation: 25 mm (3.81 lb)
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 (3.81 lb)
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

### 12.2.1 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:

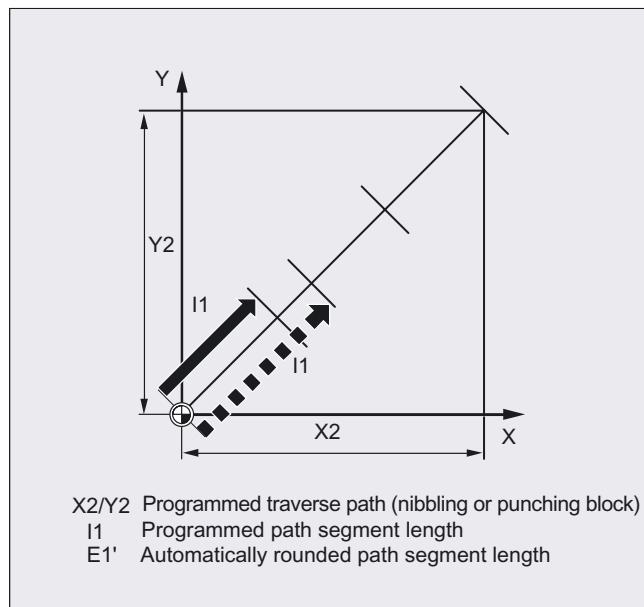
N10 SON X0 Y0

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.



Example:

```
N10 G1 G91 SON X10 Y10  
N20 SPP=3.5 X15 Y15
```

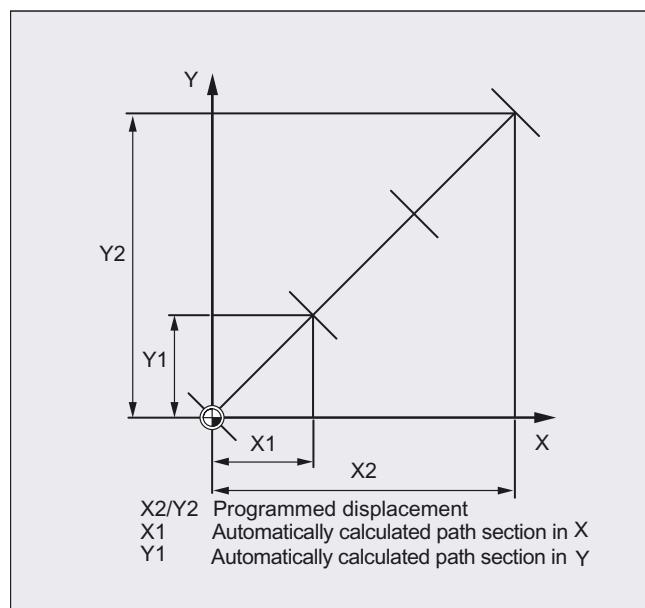
When the total traversing distance is 15 mm and the path segment length 3.5 mm, the quotient is not an integer value (4.28). In this case, the SPP value is reduced down to the next possible integer quotient. The result in this example would be a path segment length of 3 mm.

### 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 control, then it is possible to program the automatic path segmentation function with **SPN** or **SPP** even independent of this technology.

---

### 12.2.2 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

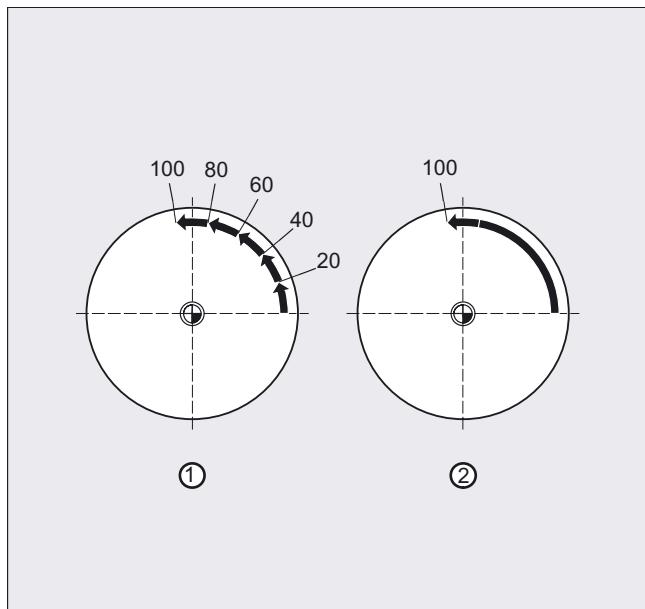
The path traversed by the single axis is distributed evenly among the intermediate blocks generated by **SPP**.

##### 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

### 13.1 Grinding-specific tool monitoring in the part program (TMON, TMOF)

#### Function

With the **TMON** command, you can activate geometry and speed monitoring for grinding tools (type 400 - 499) in the NC part program. Monitoring remains active until deactivated in the part program using the **TMOF** command.

---

#### Note

Please follow the machine manufacturer's instructions!

---

#### Requirements

The grinding-specific tool parameters \$TC\_TPG1 to \$TC\_TPG9 must be set.

#### Syntax

TMON (<T-No.>)  
TMOF (<T-No.>)

#### Description

TMON	Command to <b>switch-in</b> the grinding-specific tool monitoring
TMOF	Command to <b>switch-out</b> the grinding-specific tool monitoring
<T-No.>	Specifies the T number
<b>Note:</b>	
	Only necessary if the tool with this T number is not active.
TMOF (0)	Deactivate monitoring for all tools

## *Grinding*

### *13.1 Grinding-specific tool monitoring in the part program (TMON, TMOF)*

#### **Further Information**

##### **Grinding-specific tool parameters**

Parameters	Significance	Data type
\$TC_TPG1	Spindle number	INT
\$TC_TPG2	Chaining rule The parameters are automatically kept identical for the lefthand and righthand grinding wheel side.	INT
\$TC_TPG3	Minimum wheel radius	REAL
\$TC_TPG4	Minimum wheel width	REAL
\$TC_TPG5	Current wheel width	REAL
\$TC_TPG6	Maximum speed	REAL
\$TC_TPG7	Maximum peripheral speed	REAL
\$TC_TPG8	Angle of the inclined wheel	REAL
\$TC_TPG9	Parameter number for radius calculation	INT

##### References:

Function Manual Basic Functions; Tool Offset (W1)

##### **Switch-in tool monitoring by selecting a tool**

According to the machine data settings, tool monitoring for the grinding tools (types 400-499) can be automatically activated when the tool selection is activated.

Only **one** monitoring routine can be active at any one time for each spindle.

##### **Geometry monitoring**

The current wheel radius and the current width are monitored.

The set speed is monitored against the speed limitation cyclically with allowance for the spindle override.

The speed limit is the smaller value resulting from a comparison of the maximum speed with the speed calculated from the maximum wheel peripheral speed and the current wheel radius.

##### **Working without a T or D number**

A standard **T** number and standard **D** number can be set per machine data, which do not have to be reprogrammed and are effective after power on/reset.

Example: All machining is performed with the same grinding wheel.

The machine data can be used to set that the active tool remains for a reset (see " Free D number assignment, cutting edge number (Page 454) ").

# 14

## Additional functions

### 14.1 Axis functions (AXNAME, AX, SPI, AXTOSPI, ISAXIS, AXSTRING, MODAXVAL)

#### Function

**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.

This **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>)
```

## Significance

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 reverse 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	Comments
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	Comments
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	Comments
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	Comments
R11=MODAXVAL(X,372.55)	; X is a linear axis; R11 = 372.55.

## **14.2 Replaceable geometry axes (GEOAX)**

### **Function**

The "Replaceable geometry axes" function allows the geometry axis grouping configured via machine data to be modified from the part program. Here any geometry axis can be replaced by a channel axis defined as a synchronous special axis.

### **Syntax**

```
GEOAX(<n>,<channel axis>,<n>,<channel axis>,<n>,<channel axis>)  
GEOAX()
```

### **Significance**

GEOAX( . . . )

Command to change over (replace) the geometry axes

**Note:**

GEOAX() without any parameter calls the basic configuration of the geometry axes.

<n>

This parameter is used to specify the number of the geometry axis that should be assigned to the subsequently specified channel axis.

Range of values: 1, 2, or 3

**Note:**

With <n>=0, the subsequently specified channel axis can be completely removed – without any replacement – from the geometry axis group.

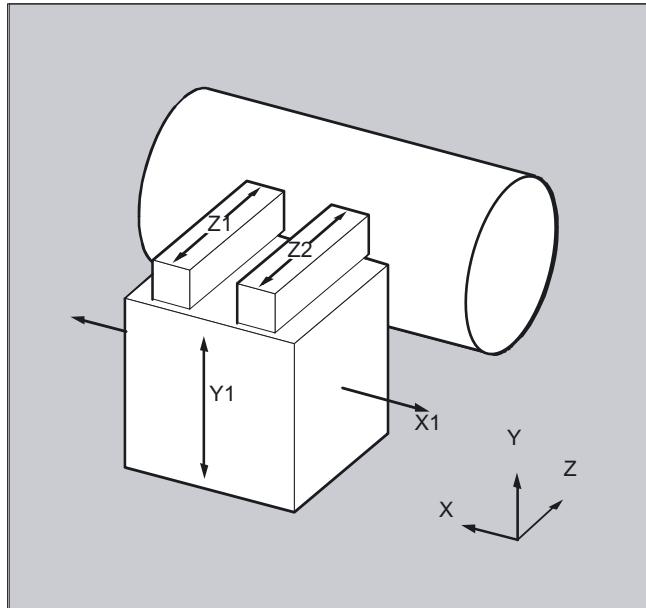
<channel axis>

The parameter is used to specify the name of the channel axis that should be included in 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	Comments
...	
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).
...	

---

14.2 Replaceable geometry axes (GEOAX)**Example 2: Changing over the geometry axes for 6 channel axes**

A machine has 6 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	Comments
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).
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 (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 supplementary 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 overwrite W and this is therefore taken out of the geometry axis group. The second geometry axis (Y) is, as before, unassigned.

---

**Note****Axis configuration**

The machine data below are used to assign the geometry axes, special axes, channel axes and machine axes as well as the names of the individual axis types:

MD20050 \$MC\_AXCONF\_GEOAX\_ASIGN\_TAB (assignment of geometry axis to channel axis)

MD20060 \$MC\_AXCONF\_GEOAX\_NAME\_TAB (name of the geometry axis in the channel)

MD20070 \$MC\_AXCONF\_MACHAX\_USED (machine axis number valid in channel)

MD20080 \$MC\_AXCONF\_CHANAX\_NAME\_TAB (name of the channel axis in the channel)

MD10000 \$MN\_AXCONF\_MACHAX\_NAME\_TAB (machine axis name)

MD35000 \$MA\_SPIND\_ASSIGN\_TO\_MACHAX (assignment of spindle to machine axis)

**References:**

Function Manual Basic Functions; Axes, Coordinate Systems, Frames (K2)

---

**Restrictions**

- It is not possible to switch the geometry axes over during:
  - Active transformation
  - Active spline interpolation
  - Active tool radius compensation
  - Active fine tool compensation
- If the geometry axis and the channel axis have the same name, it is not possible to change the particular geometry axis.
- None of the axes involved in the changeover may be involved in an action that can go beyond the block limits – e.g. type A positioning axes or for following axes.
- Geometry axes can only be replaced using the command **GEOAX** if they were already available when the system was powered-up (this means that no new ones can be defined).
- An alarm is output if an attempt is made to replace an axis with **GEOAX** while executing the contour table (**CONTPRON**, **CONTDCON**).

## **Supplementary conditions**

### **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 zones, working area limits**

All frames, protection zones 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, ZO**

A possible handwheel offset (DRF) or an external zero offset (ZO) 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 applicable in an active transformation (defined using machine data) cannot be changed using the function "replaceable geometry axes".

Should it be necessary to modify the geometry axis configuration in conjunction with transformations, then this is only possible using an additional transformation.

A geometry axis configuration changed using `GEOAX` is deleted by activating a transformation.

If the machine data settings for the transformation do not match those for changing over geometry axes, then the settings in the transformation have priority.

Example:

One transformation active. According to the machine data, the transformation should be kept for a reset; however, at the same time, for a reset, the basic configuration of geometry axes should be established. In this particular case, the geometry axis configuration that was defined with the transformation is kept.

## 14.3 Link communication

### Function

The NCU link, the link between several NCU units of an installation, is used in distributed system configurations. When there is a high demand for axes and channels, e.g. with revolving machines and multi-spindle machines, computing capacity, configuration options and memory areas can reach their limits when only one NCU is used.

Several NCUs interconnected with an NCU link module provide a scalable solution which fully meets the requirements of this type of machine tools. The NCU link module (hardware) realizes a fast NCU-to-NCU communication by providing read and write access to system variables.

### Requirements

Options providing this functionality can be ordered separately.

## Link variables

Link variables are **global system data** that can be addressed by the connected NCUs as **system variables**.

The user (in this case, normally the machine manufacturer) specifies:

- the **contents** of these variables,
- their **data type**,
- their **use**,
- their position (**access index**) in the link memory.

**Applications for link variables:**

- global machine states,
- workpiece clamping open/closed
- etc.

## Time behavior for accessing applications

The various NCU applications that access the link memory jointly **at any one time** must use the link memory **in a uniform way**. The link memory can have different assignments for processes that are completely separated in time.



### WARNING

A link variable write process is only then completed when the written information is also available to all the other NCUs. Approximately two interpolation cycles are necessary for this process. Local writing to the link memory is delayed by the same time for purposes of consistency.

For more information, see  
*/FB2/ Function Manual Extension Functions; Multiple Operator Panels and NCUs (B3)*.

### 14.3.1 Access to a global NCU memory area

#### Function

Several NCUs linked via link modules can have read and write access to a global NCU memory area via the system variables described in the following.

- Each NCU linked via a link module can **use global link variables**. These link variables are addressed in the same way by all connected NCUs.
- Link variables can be programmed in the same was as system variables. As a rule, the machine manufacturer defines and documents the meaning of these variables.
- Applications for link variables
- Data volume comparatively small
- Very high transfer speed, therefore: Use is intended for time-critical information.
- These system variables can be accessed from the **part program** and from **synchronized actions**. The size of the memory area for global NCU system variables configurable.

When a value is written in a global system variable, it can be read by all the NCUs connected after one interpolation cycle.

#### Significance

**Link variables** are stored in the link memory. After power-up, the link memory is initialized with 0.

The following link variables can be addressed within the link memory:

INT \$A_DLW[i]	Data byte (8 bits)
INT \$A_DLW[i]	Data word (16 bits)
INT \$A_DLW[i]	Data double word (32 bits)
REAL \$A_DLW[i]	Real data (64 bits)

---

#### 14.4 Axis container (AXCTSWE, AXCTSWED)

According to the data type, 1, 2, 4, 8 bytes are addressed when reading/writing the link variables.

Index **i** defines the start of the respective variable in relation to the start of the configured link memory. The index is counted from 0.

##### Ranges of values

The data types have the following value ranges:

BYTE: 0 to 255

WORD: -32768 to 32767

DWORD: -2147483646 to +2147483647

REAL:  $\pm(2,2 \cdot 10^{-308} \dots 1,8 \cdot 10^{+308})$

#### Example

Program code	Comments
\$A_DL8[5]=21	; The 5th byte in the shared link memory is assigned value 21.

## 14.4 Axis container (AXCTSWE, AXCTSWED)

#### Function

On rotary indexing machines/multi-spindle machines, the axes holding the workpiece move from one machining unit to the next. Since the machining units are subject to different NCU channels, the axes holding the workpiece must be dynamically reassigned to the corresponding NCU channel if there is a change in station/position. Axis containers are used for this purpose.

Only one workpiece clamping axis/spindle is active on the local machining unit at a time. The axis container combines the possible connections to all clamping axes/spindles, of which only one is active at a time for the machining unit.

The available axes that are defined in the axis container can be changed by switching the entries in the axis container ("axis container rotation") through an increment that can be entered via setting data (number of slots).

The command AXCTSWE or AXCTSWED is used to call the axis container rotation from the part program.

## Syntax

```
AXCTSWE(<axis container>)
AXCTSWED(<axis container>)
```

## Significance

AXCTSWE	Command to rotate an axis container If all of the enable signals of all channels for the axes of the container are available in the control, the container is rotated with the container-specific increment saved in the SD41700 \$SN_AXCT_SWWIDTH[<container number>].
AXCTSWED	Command to rotate an axis container where just the active channel is effective (command version for commissioning!) <b>Note:</b> The axes entered in the container are only released if the remaining channels, which have axes in the container, are in the reset state.
<axis container>	Identifier of the axis container that should be moved. Possible data include: CT<container number> The number of the axis container is attached to the CT letter combination. Example: CT3
<container name>	Individual name of the axis container set using MD12750 \$MN_AXCT_NAME_TAB. Example: A_CONT3

## Further Information

### Axis container

Can be assigned via the axis container.

- Local axes and/or
- Link axes

Axis containers with link axes are a NCU-cross device (NCU-global) that is coordinated via the control. It is also possible to have axis containers that are only used for managing local axes.

### References:

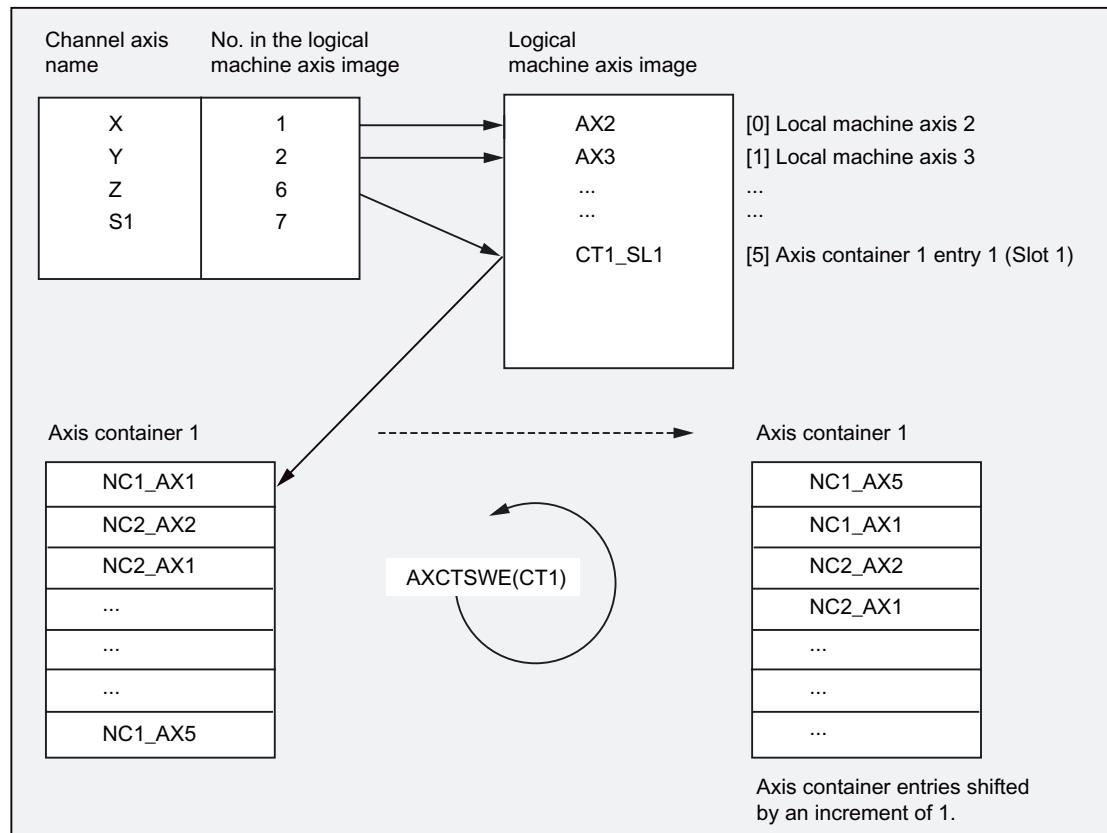
Detailed information on configuring axis containers, see:  
Function Manual, Extension Functions; Several Control Panels on Multiple NCUs,  
Decentralized Systems (B3)

### Enable criteria

AXCTSWE( )

Each channel whose axes are entered in the specified container issues an enable for a container rotation if it has finished machining the position/station. If the enable signals of **all** channels for the axes of the container have been received in the control, the container is rotated with the container-specific increment saved in the SD41700 \$SN\_AXCT\_SWWIDTH[<container number>].

Example:



After the axis container is rotated through 1, channel axis Z is assigned to axis AX5 on NCU 1 instead of axis AX1 on NCU.

#### AXCTSWED()

The command version `AXCTSWED()` can be used to simplify commissioning. The axis container rotates with just the active channel being effective through the container-specific increment saved in `SD41700 $SN_AXCT_SWWIDTH[<container number>]`. This call may only be used if the remaining channels, that have axes in the container, are in the **reset** state.

### **Effectiveness**

After an axis container rotation, all NCUs are involved in the new axis assignment whose channels refer to the rotated axis container via the logical machine axis image.

### **Axis container revolution with implicit GET/GETD**

When an axis container revolution is enabled, all axis container axes assigned to the channel are assigned to the channel with GET or GETD. Axes cannot be released until the axis container has been rotated.

---

#### **Note**

This behavior can be set using machine data. Please refer to the machine manufacturer's instructions.

---

---

#### **Note**

Axis container rotation with implicit GET / GETD**cannot** be used for an axis in the state main run axis, (e.g. for a PLC axis) as this axis would have to exit the main run status for the purpose of axis container rotation.

---

## 14.5 Extended stop and retract

### Function

The "Extended stop and retract" function ESR provides a means to react flexibly to selective error sources while preventing damage to the workpiece.

#### Available part reactions

"Extended stop and retract" provides the following part reactions:

- **"Extended stop"** (drive-independent) is a defined, time-delayed stop.
- **""Retract""** (drive-independent)  
means "escaping" from the machining plane to a safe retracted position. This means any risk of collision between the tool and the workpiece is avoided.
- **""Generator operation""**(drive-independent)  
Generator operation is possible in the event that the DC link power is insufficient for safe retraction. As a separate drive operating mode, it provides the necessary power to the drive DC link for carrying out an orderly "Stop" and "Retract" in the event of a power outage or similar failure.

### Additional extensions

- **Extended stop** (NC-controlled)  
is a defined, time-delayed, contour-friendly shut down controlled by the NC.
- **Retract** (NC-controlled)  
means "escaping" from the machining plane to a safe retracted position under the control of the NC. This means any risk of collision between the tool and the workpiece is avoided. With gear cutting, for example, retract will cause a retraction from tooth gaps that are currently being machined.

All reactions can be used independently from one another. For further information refer to /FB3/ Function Manual, Special Functions; Coupled axis and ESR (M3).

## Possible initiation sources

The following error sources are possible for starting "Extended stop and retract": **General sources** (NC-external/global or mode group/channel-specific):

- Digital inputs (e.g. on NCU module or terminal box) or the readback digital output image within the control (`$A_IN`, `$A_OUT`)
- Channel status `$AC_STAT`
- VDI signals (`$A_DBB`)
- Group messages of a number of alarms (`$AC_ALARM_STAT`)

## Axial sources

- Emergency retraction threshold of the following axis (synchronization of electronic coupling, `$VC_EG_SYNCDIFF`[following axis])
- Drive: DC link warning threshold (pending undervoltage), `$AA_ESR_STAT`[axis]
- Drive: Generator minimum velocity threshold (no more regenerative rotation energy available), `$AA_ESR_STAT`[axis].

## Gating logic for the static synchronized actions: Source/reaction logic operation

The static synchronized actions' flexible gating possibilities are used to trigger specific reactions relatively quickly according to the sources.

The operator has several options for gating all relevant sources by means of static synchronized actions. 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.

For a detailed description of how to use synchronized actions, please see:

**References:** /FBSY/ Description of Functions, Synchronized Actions

## Activation

Enabling functions:

`$AA_ESR_ENABLE`

The generator operation, stop and retract functions are enabled by setting the associated control signal (`$AA_ESR_ENABLE`). This control signal can be modified by the synchronized actions.

Function initiation (general triggering of all released axes)

`$AN_ESR_TRIGGER`

Generator operation "automatically" becomes active in the drive when the risk of DC link undervoltage is detected.

Drive-independent stop and/or retract are activated when communication failure is detected (between NC and drive) as well as when DC link undervoltage is detected in the drive (providing they are configured and enabled).

Drive-independent stop and/or retract can also be triggered from the NC side by setting the corresponding control signal `$AN_ESR_TRIGGER` (broadcast command to all drives).

### 14.5.1 Drive-independent responses to ESR

#### Function

Independent drive reactions are defined axially, that is, if activated each drive processes its stop and retract request independently. There is no interpolatory coupling of axes or coupling adhering to the path at stop/retract, the reference to the axes is time-controlled.

During and after execution of drive-independent reactions, the respective drive no longer follows the NC enables or NC travel commands. Power OFF/Power ON is necessary. Alarm "26110: Drive-independent stop/retract triggered" indicates this.

## Parameters

### Generator operation

The generator operation is

- configured: via MD 37500: **10**
- enabled: system variable `$AA_ESR_ENABLE`
- activated: depending on the setting of the drive machine data when the voltage in the DC link falls below the value.

### Retract (drive-independent)

The drive-independent retract is

- configured: via MD 37500: **11**; time specification and retract velocity are set in MD; see "Example: Using the drive-independent reaction" at the end of this chapter,
- enabled: system variable `$AA_ESR_ENABLE`
- triggered: system variable `$AN_ESR_TRIGGER`.

### Stop (independent drive)

Independent drive stop is

- configured: via MD 37500: **12** and time specified via MD;
- enabled (`$AA_ESR_ENABLE`) and
- started: system variable `$AN_ESR_TRIGGER`.

## Example of the use of drive-independent response

### Example configuration

- Axis A is to operate as generator drive,
- in the event of an error, axis X must retract by 10 mm at maximum speed, and
- axes Y and Z must stop after a 100 ms delay to give the retraction axis time to cancel the mechanical coupling.

### Example execution

1. Activate options "Ext. Stop and retract" and "Mode-independent actions" (includes "Static synchronized actions IDS ...").
2. Function assignment:  
`$MA_ESRREACTION[X] = 11,  
$MA_ESRREACTION[Y] = 12,  
$MA_ESRREACTION[Z] = 12,  
$MA_ESRREACTION[A] = 10;`
3. Drive configuration:  
`MD 1639: RETRACT_SPEED[X] = 400000H in pos. direction (max. speed),  
= FFC0000H in neg. direction,  
MD 1638: RETRACT_TIME[X] = 10ms (retraction time),  
MD 1637: GEN_STOP_DELAY[Y] = 100ms,  
MD 1637: GEN_STOP_DELAY[Z] = 100ms,  
MD 1635: GEN_AXIS_MIN_SPEED[A] = generator min. speed (rpm).`
4. Function enable (from parts program or synchronous actions) by setting the system variables:  
`$AA_ESR_ENABLE[X] = 1,  
$AA_ESR_ENABLE[Y] = 1,  
$AA_ESR_ENABLE[Z] = 1,  
$AA_ESR_ENABLE[A] = 1.`
5. Accelerate generator drive to "momentum" speed (e.g. in spindle operation M03 S1000)
6. Formulate trigger condition as static synchronous action(s), e.g.:
  - dependent on intervention of generator axis: IDS = 01 WHENEVER  
`$AA_ESR_STAT[A]>0 DO $AN_ESR_TRIGGER = 1`
  - and/or dependent on alarms that trigger follow-up mode (bit13=2000H): IDS = 02  
`WHENEVER ($AC_ALARM_STAT_B_AND 'H2000')>0  
DO $AN_ESR_TRIGGER = 1`
  - and also dependent on EG synchronized operation (if, for example, Y is defined as the EG following axis and if the max. permissible synchronized operation deviation is to be 100 µm):  
`IDS = 03 WHENEVER ABS($VA_E_SYNCDIFF[Y])>0.1  
DO $AN_ESR_TRIGGER = 1`

### **14.5.2 NC-controlled responses to retraction (POLF, POLFA, POLFMASK, POLFMLIN)**

#### **Function**

NC-controlled reactions require certain initial conditions listed below as restrictions. If these prerequisites for retraction are satisfied, fast retraction will be activated.

The retraction position POLF must be programmed in the part program. The activate signals must be set for the retraction movement and remain set.

#### **Syntax**

POLF[geo mach]=,=value	Target position of retracting axis
POLFA(axis,type,value)	Retraction position of single axes
	The following abbreviated forms are permitted:
POLFA(axis,type)	Abbreviated form for single axis retraction
POLFA(axis,0/1/2) type)	high-speed deactivation / activation
POLFA(axis,0,\$AA_POLFA[axis])	causes a preprocessing stop
POLFA(axis,0)	does not cause a preprocessing stop
POLFMASK(axis name1,axis name2,...)	Axis selection for the retraction unconnected axes
POLFMLIN(axis name1,axis name2,...)	Axis selection for retraction linearly connected axes

#### **NOTICE**

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.

## Significance

geo   mach	Geometry axis or channel/machine axis that retracts.
Axis	Axis designations of the valid single axes.
Type	Position values of the single axes of the type: Invalidate the position value Position value is absolute Position value is incremental (distance)
Value	Retract position, WCS is valid for geometry axis, otherwise MCS. If the identifiers for the geo axis and channel/machine axis are <b>identical</b> , retraction is carried out in the workpiece coordinate system. Incremental programming is permissible. Retraction position with type=1 for single axes Retraction position with type=2 for single axes The value is also accepted with type=0: Only this value is marked as invalid and has to be reprogrammed for retraction.
POLF	The POLF command is modally effective.
POLFA	If an axis is no a single axis, or if the type is missing or type=0, the relevant alarms 26080 and 26081 are output.
POLFMASK	The specified axes for the retraction – without any interrelationship between the axes – is enabled using the POLFMASK command. The POLFMASK() command deactivates fast retraction for all axes without specifying any one axis that were retracted without any interrelationship between the axes.
POLFMLIN	The specified axes for the retraction – with linear interrelationship between the axes – is enabled using the POLFMLIN command. The POLFMLIN() command deactivates the fast retraction for all axes without specifying any one axis, that were retracted with a linear interrelationship.

**axisnamei** Name of the axes that should traverse for LIFTFAST to their positions defined using POLF. All the axes specified must be in the same coordinate system. Before fast retraction to a fixed position can be enabled via POLFMASK or POLFMLIN a position must have been programmed with POLF for the selected axes. There is no machine data for pre-assigning the values of POLF.  
During interpretation of POLFMASK or POLFMLIN, alarm 16016 is issued if POLF has not been programmed.

---

**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.

**! CAUTION**

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.

For more information on changing the coordinate system, the effect on modulo rotary axes, etc. see

**References:**

Function Manual, Special Functions; Coupled axes and ESR (M3)

**Example**

Retracting an individual axis:

<b>Programming</b>	<b>Comments</b>
MD 37500: ESRREACTION[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.

## Requirements

### Retract

- the axes selected with POLFMASK or POLFMLIN,
- the axis-specific positions defined with POLF,
- the retraction positions of a single axis defined with POLFA ,
- the time window in  
MD 21380: ESR\_DELAY\_TIME1 and  
MD 21381: ESR\_DELAY\_TIME2,
- the trigger via system variable \$AC\_ESR\_TRIGGER  
\$AA\_ESR\_TRIGGER for single axes,
- the agreed ESR  
MD 37500: ESRREACTION = 21
- LFPOS from the modal 46. G code group.

## Enable and start NC-controlled reactions

If system variable \$AC\_ESR\_TRIGGER = 1 is set, and if a retract axis is configured in this channel (i.e. MD 37500: ESRREACTION = 21) and \$AA\_ESR\_ENABLE = 1 is set for this axis, then LIFTFAST becomes active in this channel.

The retraction position POLF must have been programmed in the parts program. On single axis retraction with POLFA(axis, type, value), the value must have been programmed and the following conditions met:

- \$AA\_ESR\_ENABLE = 1 set.
- POLFA(axis) must be a single axis at the time of triggering.
- POLFA(type) either type=1 or type=2.

The activate signals must be set for the retraction movement and remain set.

- The retracting movement configured with LFPOS, POLF for the axes selected with POLFMASK or POLFMLIN replaces the path motion defined for these axes in the parts program.
- The extended retraction (i.e. LIFTFAST/LFPOS initiated through \$AC\_ESR\_TRIGGER ) cannot be interrupted and can only be terminated prematurely via an EMERGENCY STOP.

The maximum time available for retraction is the sum of the times MD 21380: ESR\_DELAY\_TIME1 and MD 21381: ESR\_DELAY\_TIME2. When this time has expired, rapid deceleration with follow-up is also initiated for the retraction axis.

### **Direction of withdrawal during rapid lifting and axis replacement**

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`. The NC-controlled retraction is

- configured: via MD 37500: **21** and 2 times specified via MD see above;
  - enabled (`$AA_ESR_ENABLE`) and
  - started: System variable `$AC_ESR_TRIGGER` with `$AA_ESR_TRIGGER` for single axes.
- 

During NC-controlled retraction, `LIFTFAST/LFPOS` is used as with thread cutting, and the retraction axis configured in the channel is enabled for rapid lifting using system variable `$AC_ESR_TRIGGER`. Retraction initiated via `$AC_ESR_TRIGGER` is locked to prevent multiple retractions.

Retraction axes must always be assigned to exactly one NC channel and may not be switched among the channels. Attempts to change a retraction axis to another channel will be indicated by alarm 26122.

Only once this axis has been deactivated again using `$AA_ESR_ENABLE[AX] = 0`, can it be changed in a new channel. Once the axis has been changed, axes can be acted upon again with `$AA_ESR_ENABLE[AX] = 1`.

Neutral axes cannot undertake NC-controlled ESR.

When `$AA_ESR_ENABLE[AX] = 1` and when the axis is changed in neutral, the suppressible ShowAlarm 26121 is triggered.

### 14.5.3 NC-controlled reactions to stoppage

#### Function

##### Stopping

The sequence for extended stopping (NC-controlled) is defined using the two machine data

MD21380 \$MC\_ESR\_DELAY\_TIME1 and  
MD21381 \$MC\_ESR\_DELAY\_TIME2.

This axis continues interpolating as programmed for the time duration set in MD21380: After the time delay specified in MD21380 has expired, controlled braking (ramp stop) is initiated: The maximum time available for interpolatory controlled braking is specified in MD21381; after this time has expired, rapid deceleration with subsequent tracking is initiated.

##### Enabling and starting NC controlled stopping

The NC-controlled stopping is

configured: using MD37500: **22** as well as 2 times using the two MDs, refer above;

Enabled (\$AA\_ESR\_ENABLE) and

started: System variable \$AC\_ESR\_TRIGGER for individual axes with \$AA\_ESR\_TRIGGER.

#### Example, stopping an individual axis

Program code	Comments
MD37500 \$MC_ESRREACTION[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.

#### 14.5.4 Generator operation/DC link backup

##### Function

By configuring drive MD and carrying out the required programming via static synchronized actions (\$AA\_ESR\_ENABLE), temporary DC link voltage drops can be compensated. The time that can be bridged depends on how much energy the generator that is used as DC link backup has stored, as well as how much energy is required to maintain the active movements (DC link backup and monitoring for generator speed limit).

When the value falls below the DC link voltage lower limit, the axis/spindle concerned switches from position or speed-controlled operation to generator operation. By braking the drive (default speed setpoint = 0), regenerative feedback to the DC link takes place.

For more information, see  
/FB3/ Function Manual Special Functions; Coupled Axes and ESR (M3).

#### 14.5.5 Drive-independent stopping

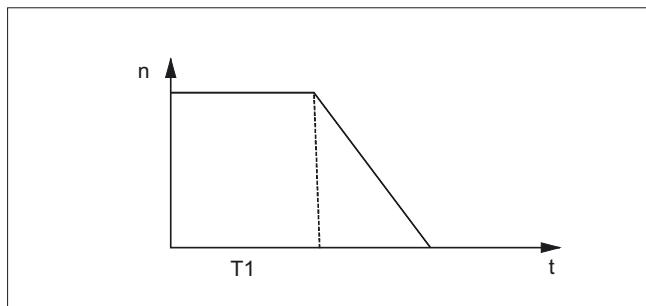
##### Function

The drives of a previously coupled grouping can be stopped by means of time-controlled cutout delay with minimum deviations from each other, if this cannot be performed by the control.

Drive-independent stop is configured and enabled via MD (delay time T1 in MD) and is enabled by system variable \$AA\_ESR\_ENABLE and started with \$AN\_ESR\_TRIGGER.

## Responses

The speed setpoint currently active as the error occurred will continue to be output for time period T1. This is an attempt to maintain the motion that was active before the failure, until the physical contact is annulled or the retraction movement initiated in other drives is completed. This can be useful for all leading/following drives or for the drives that are coupled or in a group.



After time T1, all axes with speed setpoint feedforward zero are stopped at the current limit, and the pulses are deleted when zero speed is reached or when the time has expired (+drive MD).

## 14.5.6 Drive-independent retraction

### Function

Axes with digital SIMODRIVE 611Digital drives can (if configured and enabled)

- when the control fails (sign-of-life failure detection),
- when the DC link voltage drops below a warning threshold,
- when triggered by system variable \$AN\_ESR\_TRIGGER

execute a retraction movement independently. The retraction movement is performed independently by the SIMODRIVE 611Digital drive. After the beginning of the retraction phase the drive independently maintains its enables at the previously valid values.

For more information, see

/FB3/ Function Manual Special Functions; Axis Functions and ESR (M3).

## **14.6 Check scope of NC language present (STRINGIS)**

### **Function**

The scope of NC language generated by a SINUMERIK 840D sl, including the active GUD/macro definitions and the installed and active cycle programs, can be checked for actual availability and their program-specific characteristics using the STRINGIS command. For example, at the start of program interpretation, you can check the effectiveness of non-activated functions.

The return values are output with coding by the HMI user interface and include basic information as well as detailed information with additional coding.

### **Syntax**

STRINGIS (STRING name) = return value with coding

In the current expansion stage, the (STRING name) to be checked is identified as follows:

**000** as not known.

**100** as NC language command which cannot however be programmed.

All programmable NC language commands which are active as options or function are identified using

**2xx**. Associated detailed information is explained in more detailed under the value ranges.

### **Significance**

The machine manufacturer uses machine data to define how to proceed and which NC language commands should be used.

If language commands are programmed and their functions are not active or they are not known in the current scope, an alarm message will be issued. Please refer to the machine manufacturer's specifications in such cases.

STRINGIS

Checks the existing NC language scope and NC cycle names, user variables, macros and label names that specifically belong to this command: whether these exist, are valid, defined or active. The STRINGIS NC language command is an integer type variable.

Especially for STRINGIS	<b>NC cycle names</b> (an active cycle) <b>GUD variables</b> <b>LUD variables</b> <b>Macros</b> <b>Label names</b>
STRING name	Variable identifier of the scope of NC language to be checked and transfer parameter of recognized STRING type values.

The **ISVAR** language command is a subset of the **STRINGIS** command and can still be used for certain checks.

#### Scope of NC language

All available language commands and in particular all those not needed and active language commands are still known for SINUMERIK powerline. The scope of language to be checked for SINUMERIK solution line depends on the pre-configured machine data and either includes all known / just the approved options or active functions in the current scope of NC language.

NC language scope	Scope of NC language includes: <b>G codes</b> of all existing G code groups such as G0, G1, G2, INV CW, POLY, ROT, KONT, SOFT, CUT2D, CDON, RMB, S PATH <b>DIN or NC addresses</b> such as ADIS, RNDM, SPN, SR, MEAS <b>NC language functions</b> such as predefined subprograms TANG(Faxis1..n, Laxis1..n, coupling factor). <b>NC language procedures</b> (pre-defined procedures with return value) such as subprogram call with parameter transfer GETMDACT. <b>NC language procedures</b> (pre-defined procedures without return value) such as deactivate single block suppression SBLOF. <b>NC key words</b> such as ACN, ACP, AP, RP, DEFINE, SETMS <b>Machine data</b> \$MN general, \$MA axial, \$MC channel-specific as well as all setting data \$S... and options data \$O... <b>NC system variable</b> \$ in the part program and synchronized actions as well as <b>NC computing parameters</b> R.
-------------------	---

**Return values**

Basic information STRINGIS	The return value is coded. The basic information included is sub-divided into y and existing detailed information into x.
Coding:	<b>Test result.</b> whether, in the actual expansion stage:
000	The NCK is not aware of the STRING name.
100	The STRING name is a language command but <b>cannot be programmed</b> , i.e. this function is inactive.
2xx	The STRING name is a <b>programmable</b> language command , i.e. this function is active.
y00	Assignment not possible
y01 to y11	Value ranges for existing detailed information known.
400	For NC addresses which do not have xx=01 or xx=10 and are not G code G or computing parameter R, see comments (1).

**Note**

During a check with, STRINGIS should **no other** coding be found, then the corresponding NC language command can be programmed and 2xx coding applies.

**2xx value ranges of the detailed information**

<b>Detailed Information</b>	<b>Significance of the test result:</b>
200	Interpretation not possible
201	A DIN address or NC address is defined, i.e. whether names have recognized the address letters from this, see comments (1)
202	G codes from the existing groups of G code have been recognized.
203	NC language functions with return value and parameter transfer are present.
204	NC language functions with return value and parameter transfer are present.
205	NC key words are present.

---

206	General, axial or channel-specific machine data (\$M...), setting data (\$S...) or option data(\$O...) are present.
207	User variables, such as NC system variables beginning with \$... or computing parameters beginning with R are present.
208	The cycle names have been loaded in NCK and cycle programs are also activated, see comment (2).
209	The defined name has been recognized and activated GUD variable found by global user variables (GUD variables) .
210	The macro names along with the names defined and macros activated in the macro definition files have been found, see comment (3).
211	Of local user variables (LUD variables) whose name is contained in the current program.

---

### Note

#### Comments on the individual return values

(1) Fixed, standardized addresses are recognized as DIN addresses. The following definitions for geometry axes apply for NC addresses with adjustable identifiers:

A, B, C for specified rotary axes, E is reserved for extensions and I, J, K, Q, U, V, W, X, Y, Z for specified linear axes.

The axle identifiers can be programmed with an address extension and can be written for the test, e.g. 201 = STRINGIS("A1").

The following addresses cannot be written with an address extension for the test and always deliver the fixed value of 400.

Example 400 = STRINGIS("D") or specification of an address expansion where 0 = STRINGIS("M02") results in 400 = STRINGIS("M").

(2) Cycle parameter names cannot be checked with STRINGIS.

(3) NC address letters G, H, L, M defined as macros are identified as macros.

---

---

#### 14.6 Check scope of NC language present (STRINGIS)

##### Valid NC addresses without address extension with the fixed value of 400

NC addressed D, F, G, H, R and L, M, N, O, P, S, T are valid. Then

400

D as tool correction, cutting edge number (D function)  
 F as feed (F function)  
 G is defined as G code (not the path condition in this case)  
 H stands for auxiliary function (H function)  
 R is defined as system parameter and  
 L stands for sub-routine call-up, M stands for additional  
 function, N stands for sub-block,  
 O is free for extensions,  
 P stands for number of program executions,  
 S stands for spindle speed (S function),  
 T stands for tool number (T function).

#### Example of programmable auxiliary function T

<b>Programming</b>	<b>Comments</b>
T is defined as auxiliary function and can always be programmed	
400 = STRINGIS("T")	;
0 = STRINGIS("T3")	;Return value without address extension
	;Return value with address extension

#### Examples of other checks for the programmable scope of NC language 2xx

<b>Programming</b>	<b>Comments</b>
X is defined as axis	; Axis is a linear axis X
201 = STRINGIS("X")	; Return value of linear axis X
201 = STRINGIS("X1")	Return value of linear axis X1
A2 is an NC address with extension	NC address A2 with extension
201 = STRINGIS("A")	Return value for NC address A
201 = STRINGIS("A2")	with extended NC address A2
NVCW is a defined G code	INVCW is G code evolvent clockwise interpolation.
202 = STRINGIS("INVCW")	Return value of known G code
GETMDACT is an NC language function	the NC language function GETMDACT is present.
203 = STRINGIS("GETMDACT")	GETMDACT is an NC language function

## 14.6 Check scope of NC language present (STRINGIS)

Programming	Comments
DEFINE is an NC key word	the DEFINE key word exists for identification of macros.
205 = STRINGIS("DEFINE") the \$MC_GCODES_RESET_VALUES is channel-specific machine data	DEFINE is present as a key word the machine data ;\$MC_GCODE_RESET_VALUES exists.
206 = STRINGIS("\$MC_GCODE_RESET_VALUES")	\$MC_GCODE_RESET_VALUES has been recognized as machine data
\$TC_DP3 is a system variable for the tool length components	NC system variable \$TC_DP3 exists for tool length components.
207 = STRINGIS("\$TC_DP3")	\$TC_DP3 recognized as system variable.
\$TC_TP4 is a system variable for a tool size	NC system variable \$TC_TP4 exists for tool size.
207 = STRINGIS("\$TC_TP4")	\$TC_TP4 recognized as system variable.
\$TC_MPP4 is a system variable for the magazine space status	Check magazine management for
207 = STRINGIS("\$TC_MPP4")	Magazine management is active
0 = STRINGIS("\$TC_MPP4")	Magazine management is not available (4)
MACHINERY_NAME is defined as GUD variable	Global user variable is defined as MACHINERY_NAME.
209 = STRINGIS("MACHINERY_NAME")	MACHINERY_NAME found as GUD
LONGMACRO is defined as macro	Macro name is LONGMACRO
210 = STRINGIS("LONGMACRO")	Macro identified as LONGMACRO
MYVAR is defined as LUD variable	Local user variable has been named MYVAR
211 = STRINGIS("MYVAR")	LUD variable is included in current program as the MYVAR name
X, Y, Z is a command not known in the NC	X,Y,Z is an unknown language command and is also not a GUD/macro/cycle name
0 = STRINGIS("XYZ")	STRING name X, Y, Z is not known

**(4)** For the system parameters of magazine management, the following characteristic applies in particular: if the function is not active, then STRINGIS always supplies the result value of 0 regardless of the value set for machine data for configuring the scope of NC language.

## **14.7      Function call ISVAR and read machine data array index**

### **Function**

The ISVAR command is a function as defined in the NC language that has a

- Function value of type BOOL
- transfer parameter of type STRING

The ISVAR command returns TRUE if the transfer parameter contains a variable known in the NC (machine data, setting data, system variable, general variables such as GUDs).

### **Syntax**

```
ISVAR(<variable identifier>)
ISVAR(<identifier>, [<value>, <value>])
```

### **Significance**

<Variable identifier>	Transfer parameter of type string can be undimensioned, 1-dimensional, or 2-dimensional.
<identifier>	Identifier with a known variable with or without an array index as machine data, setting data, system variable, or general variable.
	<b>Extension:</b>
	For general and channel-specific machine data, the first element of the array will be read even when no index is specified.

<value>                  Function value of type BOOL

### **Checks**

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

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

Only if all these checks have a positive result will TRUE be returned. If a check has a negative result or if a syntax error has occurred, it will return FALSE. Axial variables are accepted as an index for the axis names but not checked.

**Extension:** Read machine data and setting data array without index.

If there is no index for **general and channel-specific** machine data, alarm 12400 "channel % 1 block % 2 array % 3 element not present" is **no longer** output.

**At least the axis index** must still be programmed for **axis-specific** machine data. Otherwise alarm 12400 will be issued.

### Example: Function call ISVAR

Program code	Comments
DEF INT VAR1	
DEF BOOL IS_VAR=FALSE	; Transfer parameter is a general variable
N10 IS_VAR=ISVAR("VAR1")	; IS_VAR is in this case, TRUE
DEF REAL VARARRAY[10,10]	
DEF BOOL IS_VAR=FALSE	; Various syntax versions
N20 IS_VAR=ISVAR("VARARRAY[, ]")	; IS_VAR is TRUE with a two-dimensional array
N30 IS_VAR=ISVAR("VARARRAY")	; IS_VAR is TRUE, variable exists
N40 IS_VAR=ISVAR("VARARRAY[8,11]")	; IS_VAR is FALSE, array index is not permitted
N50 IS_VAR=ISVAR("VARARRAY[8,8")	; IS_VAR is FALSE, syntax error for missing "]"
N60 IS_VAR=ISVAR("VARARRAY[,8]")	; IS_VAR is TRUE, array index is permitted
N70 IS_VAR=ISVAR("VARARRAY[8,]")	; IS_VAR is TRUE
DEF BOOL IS_VAR=FALSE	; Transfer parameter is a machine data
N100 IS_VAR=ISVAR("\$MC_GCODE_RESET_VALUES[1]"	; IS_VAR is TRUE
DEF BOOL IS_VAR=FALSE	; Transfer parameter is a system variable
N10 IS_VAR=ISVAR("\$P_EP")	; IS_VAR is in this case TRUE
N10 IS_VAR=ISVAR("\$P_EP[X] ")	; IS_VAR is in this case TRUE

**Example: Read machine data array with and without index.**

The first element will be read for

R1=\$MC\_EXTERN\_GCODE\_RESET\_VALUES

as previous, this corresponds to

R1=\$MC\_EXTERN\_GCODE\_RESET\_VALUES[0]

or the first element will be read

R1=\$MA\_POSTCTRL\_GAIN[X1]

The corresponds to as before

R1=\$MA\_POSTCTRL\_GAIN[0, X1]

The first element in synchronized actions is also read for

WHEN TRUE DO \$R1 = \$MC\_EXTERN\_GCODE\_RESET\_VALUES

The corresponds to as before

WHEN TRUE DO \$R1 = \$MC\_EXTERN\_GCODE\_RESET\_VALUES[0]

and would previously **not be read** with alarm 12400.

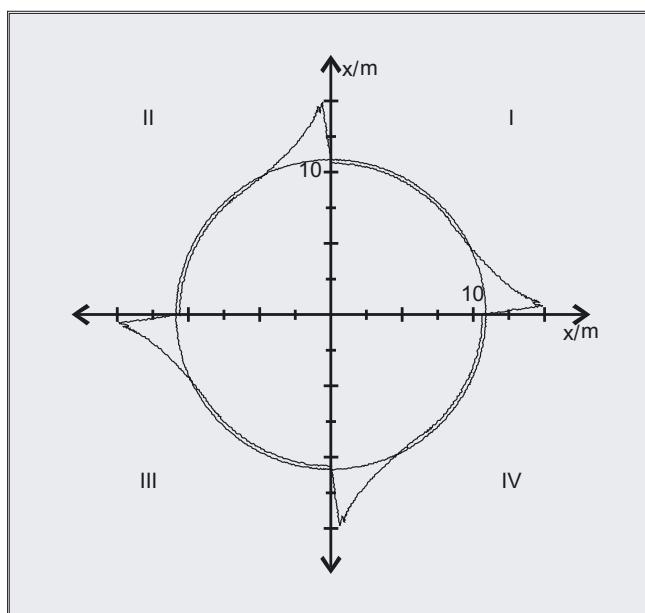
The alarm 12400 will still be issued for

R1=\$MA\_POSTCTRL\_GAIN

## 14.8 Learn compensation characteristics (QECLRNON, QECLRNOF)

**Function error! Bookmark not defined..**

Quadrant error compensation (QEC) reduces contour errors that occur on reversal of the traversing direction due to mechanical non-linearities (e.g. friction, backlash) or torsion. On the basis of a neural network, the optimum compensation data can be adapted by the control during a learning phase in order to determine the compensation characteristics automatically. Learning can take place simultaneously for up to four axes.



### Syntax

QECLRNON  
QECLRNOF

#### Activate learning operation: QECLRNON

The actual learning operation is activated in the NC program using the command `QECLRNON` specifying the axes:

`QECLRNON (X1, Y1, Z1, Q)`

The characteristics are only changed if the command is active.

#### Deactivating learning: QECLRNOF

After the learning motions of the required axes have been completed, the learning process is simultaneously deactivated for all axes using `QECLRNOF`.

## **Significance**

QECLRNON (axis 1,...4)	Activate "Learn quadrant error compensation" function
QECLRNO	Deactivate "Learn quadrant error compensation" function
QECLRN.SPF	Learning cycle
QECDAT.MPF	Sample NC program for assigning system variables and for parameterizing the learning cycle
QECTEST.MPF	Sample NC program for the circularity test

## **Description**

The traversing movements of the axes required for the learning process are generated with the aid of an NC program. The learning movements are stored in the program in the form of a learning cycle.

### **First teach-in**

Sample NC programs contained on the disk of the standard PLC program are used to teach the movements and assign the QEC system variables in the initial learning phase during startup of the control:

### **Relearning**

The learnt characteristics can be optimized with subsequent learning. The data stored in the user memory are used as the basis for optimization. Optimization is performed by adapting the sample NC programs to your needs.

The parameters for the learning cycle (e.g. QECLRN.SPF) might have to be changed for "relearning".

- Set "Learn mode" = 1
- Reduce "Number of learn passes" if required
- Activate "Modular learning" if required and define area limits.

## 14.9 Interactively call the window from the part program (MMC)

### Function

You can use the **MMC** command to display user-defined dialog windows (dialog displays) on the HMI from the part 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 (CYCLES, PICTURE_ON, T_SK.COM, BILD, MGUD.DEF, BILD_3.AWB, TEST_1, A1",
      "S")
```

### Significance

MMC	Calling the dialog window interactively from the part program on the HMI.
CYCLES	Operating area in which the configured user dialog boxes are implemented.
PICTURE_ON or PICTURE_OFF T_SK.COM	Command: Display selection or display deselection.
DISPLAY	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.
MGUD.DEF	Name of dialog display: The individual displays are selected via the names of the dialog displays.
PICTURE_3.AWB	User data definition file, which is addressed while reading/writing variables.
TEST_1	Graphics file
A1	Display time or acknowledgement variable.
"S"	Text variables..." ,
	Acknowledgement mode: synchronous, acknowledgement via "OK" soft key.

### References

Refer to the Commissioning Manual for detailed information on programming the MMC command (incl. programming examples).

## **14.10 Program runtime/part counter**

### **14.10.1 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.

### **14.10.2 Program runtime**

#### **Function**

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 variables

system variables	Significance	Activity
<b>NC-specific</b>		
\$AN_SETUP_TIME	Time since the control last booted with standard values ("cold start") in minutes; each time the control boots with standard values, this is automatically reset to "0".	Always active
\$AN_POWERON_TIME	Time since the control normally booted the last time ("warm start") in minutes; this is automatically reset to "0" when the control boots normally.	
<b>Channel-specific</b>		
\$AC_OPERATING_TIME	Total runtime of NC programs in seconds in the automatic operating mode; is automatically set to "0" each time that the control boots.	<ul style="list-style-type: none"> <li>• Activated via MD27860</li> <li>• Only AUTOMATIC mode</li> </ul>
\$AC_CYCLE_TIME	Runtime of the actual NC program in seconds; this is automatically reset to "0" when an NC program starts.	
\$AC_CUTTING_TIME	Tool operation time in seconds; this is automatically reset to "0" each time the control boots with the standard values.	
\$AC_ACT_PROG_NET_TIME	Actual net runtime of the actual NC program in seconds; this is automatically reset to "0" when an NC program starts.	<ul style="list-style-type: none"> <li>• Always active</li> <li>• Only AUTOMATIC mode</li> </ul>
\$AC_OLD_PROG_NET_TIME	Net runtime in seconds of the program that has just been 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.	

**14.10 Program runtime/part counter**

system variables	Significance		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!			

**Literatur:**

Function Manual Basic Functions; BAG, Channel, Program Operation, Reset Response (K1),  
Chapter: Program runtime

**NOTICE**

**Using STOPRE**

The system variables \$AC\_OLD\_PROG\_NET\_TIME and \$AC\_OLD\_PROG\_NET\_TIME\_CTR 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.

## Examples

### Example 1: Measuring the duration of "mySubProgrammA"

<b>Program code</b>
... 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"

<b>Program code</b>
... 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

### 14.10.3 Workpiece counter

#### Function

The "Workpiece counter" function can be used to prepare counters, e.g., for internal counting of workpieces on the control. These counters exist as channel-specific system variables with read and write access within a value range from 0 to 999 999 999.

Machine data can be used to control counter activation, counter reset timing and the counting algorithm.

#### System variables

The following counters are available:

System variable	Significance
\$AC_REQUIRED_PARTS	Number of workpieces required (workpiece setpoint) In this counter you can define the number of workpieces at which the actual workpiece counter \$AC_ACTUAL_PARTS is reset to zero. The generation of the display alarm workpiece setpoint reached and the channel VDI signal workpiece setpoint reached can be activated via MD.
\$AC_TOTAL_PARTS	Total number of workpieces produced (total actual) The counter specifies the total number of all workpieces produced since the start time. The counter is automatically reset with default values only when the control is powered up.
\$AC_ACTUAL_PARTS	Number of actual workpieces (actual) This counter registers the total number of all workpieces produced since the start time. The counter is automatically reset to zero (on condition that \$AC_REQUIRED_PARTS is not equal to 0) when the required number of workpieces (\$AC_REQUIRED_PARTS) has been reached.
\$AC_SPECIAL_PARTS	Number of workpieces specified by the user This counter allows users to make a workpiece counting in accordance with their own definition. Alarm output can be defined for the case of identity with \$AC_REQUIRED_PARTS (workpiece target). Users must reset the counter themselves.

#### Note

The "workpiece counter" function is independent of the tool management functions. All counters can be read and written from the HMI.

All counters are set to zero when the control boots with standard values and can be read and written independent of their activation.

## Example

Program code	Comments
\$MC_PART_COUNTER='H3'	; Activate workpiece counter \$AC_REQUIRED_PARTS: ; \$AC_REQUIRED_PARTS is active, alarm display for \$AC_REQUIRED_PARTS==\$AC_SPECIAL_PARTS.
\$MC_PART_COUNTER='H10' \$MC_PART_COUNTER_MCODE[0]=80	; Activate workpiece counter \$AC_TOTAL_PARTS: ; \$AC_TOTAL_PARTS is active; the counter is incremented by the value 1 with each M02. \$MC_PART_COUNTER_MCODE[0] has no significance.
\$MC_PART_COUNTER='H300' \$MC_PART_COUNTER_MCODE[1]=17	; Activate workpiece counter \$AC_ACTUAL_PARTS: ; \$AC_TOTAL_PARTS is active. The counter is incremented by the value 1 with each M17.
\$MC_PART_COUNTER='H3000' \$MC_PART_COUNTER_MCODE[2]=77	; Activate workpiece counter \$AC_SPECIAL_PARTS: ; \$AC_SPECIAL_PARTS is active. The counter is incremented by the value 1 with each M77.
\$MC_PART_COUNTER='H200' \$MC_PART_COUNTER_MCODE[1]=50	; Deactivate workpiece counter \$AC_ACTUAL_PARTS: ; \$AC_TOTAL_PARTS is not active, rest has no significance.
\$MC_PART_COUNTER='H3313' \$MC_PART_COUNTER_MCODE[0]=80 \$MC_PART_COUNTER_MCODE[1]=17 \$MC_PART_COUNTER_MCODE[2]=77	; Activating all counters in examples 1-4: ; \$AC_REQUIRED_PARTS is active, alarm display for \$AC_REQUIRED_PARTS==\$AC_SPECIAL_PARTS. \$AC_TOTAL_PARTS is active; the counter is incremented by the value 1 with each M02. \$MC_PART_COUNTER_MCODE[0] has no significance. \$AC_ACTUAL_PARTS is active; the counter is incremented by the value 1 with each M17. \$AC_SPECIAL_PARTS is active; the counter is incremented by the value 1 with each M77.

## **14.11 Alarms (SETAL)**

### **Function**

Alarms can be set in an NC program. Alarms are displayed in a separate field at the operator interface. An alarm always goes hand in hand with a response from the controller according to the alarm category.

#### **References:**

Further information on alarm responses, refer to the Commissioning Manual.

### **Syntax**

```
SETAL(<alarm number>)
SETAL(<alarm number>,<character string>)
```

### **Significance**

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 4 parameters can be specified. Variable user texts can be defined in these parameters. However, the following predefined parameters are available: <table><thead><tr><th>Parameter</th><th>Significance</th></tr></thead><tbody><tr><td>%1</td><td>Channel number</td></tr><tr><td>%2</td><td>Block number, label</td></tr><tr><td>%3</td><td>Text index for cycle alarms</td></tr><tr><td>%4</td><td>Additional alarm parameters</td></tr></tbody></table>	Parameter	Significance	%1	Channel number	%2	Block number, label	%3	Text index for cycle alarms	%4	Additional alarm parameters
Parameter	Significance										
%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 operator interface.

---

**Example**

Program code	Comments
... N100 SETAL (65000) ...	; Setting alarm No. 65000



## User stock removal programs

### 15.1 Supporting functions for stock removal

#### Functions

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)
- Executing 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.

---

#### Prerequisites

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.

## 15.2 Generate contour table (CONTPRON)

### Function

Contour preparation is activated using the command CONTPRON. 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) "

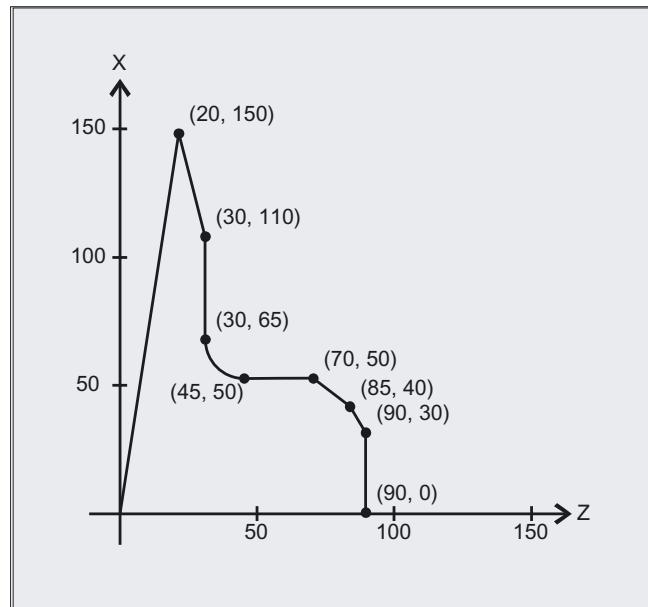
### Significance

CONTPRON	Command to activate contour preparation to generate a contour table
<contour table>	Names of contour table
<machining type>	Parameter for the machining type Type: CHAR Value: "G" Longitudinal turning: Inside machining "L" Longitudinal turning: External machining "N" Face turning: Inside 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, forwards (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 fault messages



**NC program:**

<b>Program code</b>	<b>Comments</b>
N10 DEF REAL KTAB[30,11]	; Contour table with the KTAB name 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 fault feedback signal (0=no fault, 1=fault).
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, change over to normal program operation.
N140 ...	; Continue to process the table.

**Contour table KTAB:**

<b>Index Line</b>	<b>Column</b>										
(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	

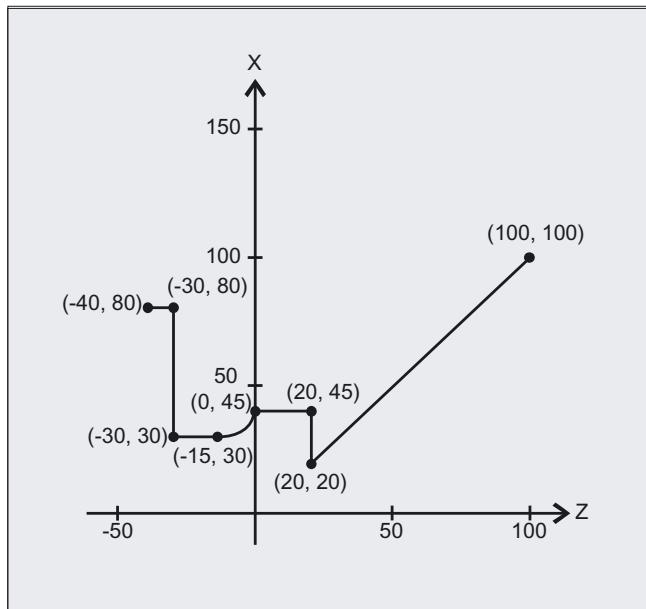
**Explanation of the column contents:**

- (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<sup>2</sup>      G90 = 0      G91 = 1
    - b = 10<sup>1</sup>      G70 = 0      G71 = 1
    - c = 10<sup>0</sup>      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)
- Mode: Longitudinal turning, outer machining
- Preparation, forwards and backwards

**NC program:**

Program code	Comments
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, outer machining
N30 DEF INT HE=0	; Number of relief cut elements=0
N40 DEF INT MODE=1	; Preparation, forwards and backwards
N50 DEF INT ERR=0	; Fault 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, change over to normal program operation.
...	

**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 <sup>1)</sup>	7 <sup>2)</sup>	11	100	100	20	20	0	45	0	0
1	0 <sup>3)</sup>	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 <sup>4)</sup>	11	-30	80	-40	80	0	0	0	0
7	5	6	11	-30	45	-30	80	0	90	0	0
8	1 <sup>5)</sup>	2 <sup>6)</sup>	0	0	0	0	0	0	0	0	0
	...										
83	84	0 <sup>7)</sup>	11	20	45	20	80	0	90	0	0
84	90	83	11	20	20	20	45	-1111	90	0	0
85	0 <sup>8)</sup>	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 <sup>9)</sup>	85 <sup>10)</sup>	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 (forwards)
- 2) Successor: Line n is the contour table end (forwards)

**Once each within the contour elements forwards:**

- 3) Predecessor: Contour start (forwards)
- 4) Successor: Contour end (forwards)

**Always in line contour table end (forwards) +1:**

- 5) Predecessor: Number of relief cuts (forwards)
- 6) Successor: Number of relief cuts (backwards)

**Once each within the contour elements backwards:**

- 7) Successor: Contour end (backwards)
- 8) Predecessor: Contour start (backwards)

**Always in last line of table:**

- 9) Predecessor: Line n is the contour table start (backwards)
- 10) Successor: Line n contains the contour start (backwards)

## 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.

## 15.3 Generate coded contour table (CONTDCON)

### Function

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 are 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) "

### Significance

CONTDCON	Command to activate the contour preparation to generate a coded contour table
<contour table>	Names of 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

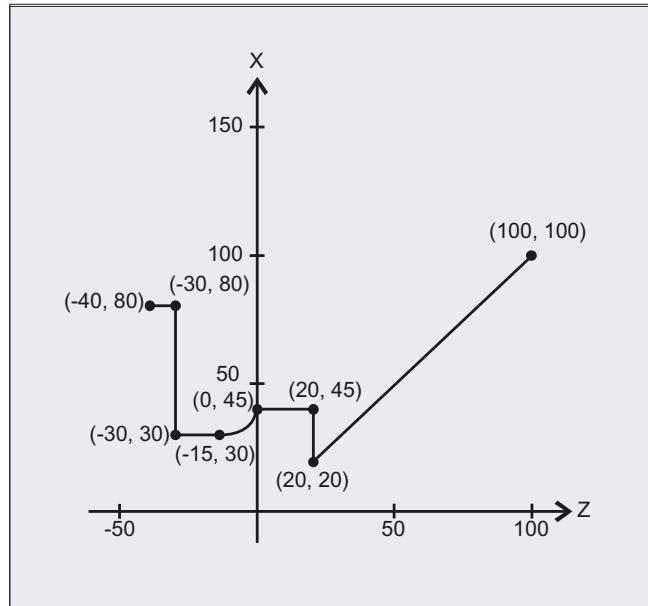
The G codes permitted for CONTDCON in the program section to be included in the table are more comprehensive than for CONTPRON. Further, feed rates and feed rate type are saved for each contour section.

---

### Example

Generating a contour table with:

- Name "KTAB"
- Contour elements (circles, straight lines)
- Mode: Turning
- Machining direction: Upwards



**NC program:**

Program code	Comments
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 direction. Standard value 0: Only in the programmed direction of the contour.
N30 DEF INT ERROR = 0	; Variable for the fault feedback signal.
...	
N100 G18 G64 G90 G94 G710	
N101 G1 Z100 X100 F1000	
N105 CONTDCON (KTAB, MODE)	; Call contour preparation (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, change over to normal program operation.
...	

**Contour table KTAB:**

Line index	Column index					
	0	1	2	3	4	5
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<br>$10^1$ (tens position): G70 = 0, G71 = 1, G700 = 2, G710 = 3 |
| Column 1:   | starting point of abscissa   |
| Column 2:   | starting point of 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<br>$10^1$ (tens position): G70 = 0, G71 = 1, G700 = 2, G710 = 3<br>$10^2$ (hundreds digit): G90 = 0, G91 = 1<br>$10^3$ (thousands digit): G93 = 0, G94 = 1, G95 = 2, G96 = 3<br>$10^4$ (ten thousands digit): G60 = 0, G44 = 1, G641 = 2, G642 = 3<br>$10^5$ (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.

## 15.4 Determine point of intersection between two contour elements (INTERSEC)

### Function

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>)
```

### Significance

INTERSEC	Key word 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	Significance
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	Comments
DEF REAL TABNAME1[12,11]	; Contour table 1
DEF REAL TABNAME2[10,11]	; Contour table 2
DEF REAL ISCOORD [2]	; Variable for the point of intersection coordinates.
DEF BOOL ISPOINT	; Variable for the intersection status.
DEF INT MODE	; Variable for machining type.
...	
MODE=1	; Calculation independent of the active plane.
N10 ISPOINT=INTERSEC(TABNAME1[3],TABNAME2[7],ISCOORD,MODE)	; Call point of intersection of the contour elements.
N20 IF ISPOINT==FALSE GOTO NOCUT	; Jump to NOCUT
...	

## 15.5 Execute the contour elements of a table block-by-block (EXECTAB)

### Function

Using the command EXECTAB, you can execute the contour elements of a table – that were generated e.g. using the command CONTPRON – block-by-block.

### Syntax

```
EXECTAB(<contour table>[<contour element>])
```

### Significance

EXECTAB	Command to execute a contour element
<contour table>	Names of 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	Comments
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.

## 15.6 Calculate circle data (CALCDAT)

### Function

Using the CALCDAT command, 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 taken for the circle center point and the radius.

### Syntax

```
<Status>=CALCDAT (<circle points>[<number>,<type>],<number>,<result>)
```

### Significance

CALCDAT	Command to calculate the radius and center point coordinates of a circle from 3 or 4 points
<status>	Variable for the circle calculation status
	Type:    BOOL
	Value:    TRUE      The specified points lie on a circle.
	FALSE     The specified points <b>do not</b> 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

---

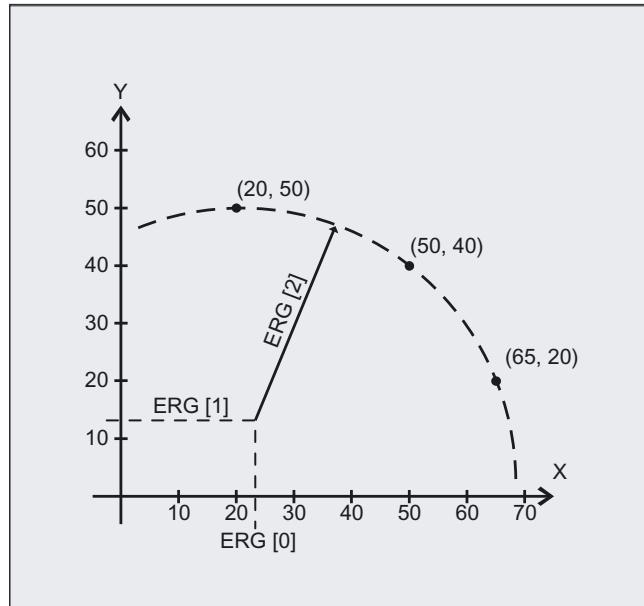
### Note

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	Comments
N10 DEF REAL PT[3,2]=(20,50,50,40,65,20)	; Variable to specify the points along a circle
N20 DEF REAL RES[3]	; Variable for result
N30 DEF BOOL STATUS	; Variable for status
N40 STATUS=CALCDAT(PKT,3,ERG)	; Call the determined circle data.
N50 IF STATUS == FALSE GOTO ERROR	; Jump to error

## 15.7 Deactivate contour preparation (EXECUTE)

### Function

The command EXECUTE is used to deactivate the contour preparation and at the same time the system returns to the normal execution mode.

### Syntax

EXECUTE (<ERROR>)

### Significance

EXECUTE	Command to terminate contour preparation
<FAULT>	Variable for the fault feedback signal
	Type: INT
	The value of the variable indicates whether the contour was able to be prepared error-free:
0	Errors
1	No error

### Example

```
Program code
...
N30 CONTPRON(....)
N40 G1 X... Z...
...
N100 EXECUTE(....)
...
```



# 16

## Tables

### 16.1 Statements A - G

#### Legend:

- <sup>1</sup> Default setting at beginning of program (factory settings of the control, if nothing else programmed)
- <sup>2</sup> The groups are numbered according to the table in section "List of G functions / preparatory functions".
- <sup>3</sup> Absolute end points: modal (m);  
incremental end points: non-modal (n);  
otherwise: modal/non-modal depending on syntax of G function
- <sup>4</sup> As circle center points, IPO parameters act incrementally. They can be programmed in absolute mode with AC. The address modification is ignored when the parameters have other meanings (e.g. thread lead).
- <sup>5</sup> The OEM can add two extra interpolation types. The names can be changed by the OEM.
- <sup>6</sup> Extended address notation cannot be used for these functions.

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
:	Block number - main block (see N)	0 ... 99 999 999 integers only, without signs	Special identification of blocks, instead of N... ;this block should contain all statements required for a complete subsequent machining section	e.g. 20		
A	Axis	Real			m/n	

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
A2	Tool orientation: Euler angles	Real			n	
A3	Tool orientation: Direction-vector component	Real			n	
A4	Tool orientation for start of block	Real			n	
A5	Tool orientation for end of block: Normal-vector component	Real			n	
ABS	Absolute value	Real				
AC	Input of absolute dimensions	0, ..., 359.9999°		X=AC(100)	n	
ACC	Axial acceleration	Real, without sign			m	
ACCLIMA	Reduction or overshoot of maximum axial acceleration	0, ..., 200	Valid range is 1% to 200%	ACCLIMA[X]= ...[%]	m	
ACN	Absolute dimensions for rotary axes, approach position in negative direction			A=ACN(...) B=ACN(...) C=ACN(...)	n	
ACOS	Arc cosine (trigon. function)	Real				
ACP	Absolute dimensions for rotary axes, approach position in positive direction			A=ACP(...) B=ACP(...) C=ACP(...)	n	
ACTBLOCNO	With suppressed current block display (DISPLOF), the number of the current block should be output when an alarm is issued.					
ADIS	Rounding clearance for path functions G1, G2, G3, ...	Real, without sign			m	
ADISPOS	Rounding clearance for rapid traverse G0	Real, without sign			m	

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
ADISPOSA	Size of the tolerance window for IPOBRKA	Integer, Real		ADISPOSA=... or ADISPOSA(<axis>[,REAL])	m	
ALF	Angle tilt fast	Integer, without sign			m	
AMIRROR	Programmable mirroring (additive mirror)			AMIRROR X0 Y0 Z0AMIRROR ;separate block	n	3
AND	Logical AND					
ANG	Contour angle	Real			n	
AP	Angle polar	0, ..., ± 360°			m/n	
APR	Read/display access protection (access protection read)	Integer, without sign				
APW	Write access protection (access protection write)	Integer, without sign				
AR	Opening angle (angle circular)	0, ..., 360°			m/n	
AROT	Programmable rotation (additive rotation)	Rotation about: 1st geometry axis: -180° ... +180° 2nd geometry axis: -90° ... +90° 3rd geometry axis: -180° ... +180°		AROT X... Y... Z... AROT RPL= ;separate block	n	3
AROTS	Programmable frame rotations with solid angles (additive rotation)			AROTS X... Y... AROTS Z... X... AROTS Y... Z... AROTS RPL= ;separate block	n	3
SL	Macro definition	String				
ASCALE	Programmable scaling (additive scale)			ASCALE X... Y... Z... ;separate block	n	3
ASPLINE	Akima spline				m	1
ATAN2	Arc tangent 2	Real				
ATRANS	Additive programmable offset (additive translation)			ATRANS X... Y... Z... ;separate block	n	3

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
AX	Variable axis identifier	Real			m/n	
AXCSWAP	Advance container axis			AXCSWAP(CT <sub>n</sub> , CT <sub>n+1</sub> ,...)		25
AXCTSWE	Advance container axis			AXCTSWE(CT <sub>i</sub> )		25
AXIS	Data type: Axis identifier		Name of file can be added			
AXNAME	Converts the input string to an axis name (get axname)	String	An alarm is generated if the input string does not contain a valid axis name			
AXSTRING	Converts the spindle-number string (get string)	String	Name of file can be added	AXSTRING[ SPI(n) ]		
AXTOCHAN	Request axis for a specific channel. Possible from NC program and synchronized action.			AXTOCHAN(axis, channel number[,axis, channel number,...])		
B	Axis	Real			m/n	
B_AND	Bit AND					
B_OR	Bit OR					
B_NOT	Bit negation					
B_XOR	Bit exclusive OR					
B2	Tool orientation: Euler angles	Real			n	
B3	Tool orientation: Direction vector component	Real			n	
B4	Tool orientation for start of block	Real			n	
B5	Tool orientation for end of block: Normal vector component	Real			n	

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
BAUTO	Definition of first spline segment by the following 3 points (begin not a knot)				m	19
BLSYNC	Processing of interrupt routine is only to start with the next block change					
BNAT <sup>1</sup>	Natural transition to first spline block (begin natural)				m	19
BOOL	Data type: Boolean value TRUE/FALSE or 1/0					
BOUND	Tests whether the value falls within the defined value range. If the values are equal, the test value is returned.	Real	Var1 : Varmin Var2: Varmax Var3: Varcheck	RetVar =		
BRISK <sup>1</sup>	Fast non-smoothed path acceleration				m	21
BRISKA	Switch on brisk path acceleration for the programmed axes					
BSPLINE	B-spline				m	1
BTAN	Tangential transition to first spline block (begin tangential)				m	19
C	Axis	Real			m/n	
C2	Tool orientation: Euler angles	Real			n	
C3	Tool orientation: Direction vector component	Real			n	
C4	Tool orientation for start of block	Real			n	
C5	Tool orientation for end of block; normal vector component	Real			n	
CAC	Absolute approach of position (coded position: absolute coordinate)		Coded value is table index; table value is approached			

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CACN	Absolute approach in negative direction of value stored in table (coded position absolute negative)		Permissible for the programming of rotary axes as positioning axes			
CACP	Absolute approach in positive direction of value stored in table (coded position absolute positive)					
CALCDAT	Calculate radius and center point or circle from 3 or 4 points (calculate circle data)	VAR Real [3]	The points must be different			
CALL	Indirect subroutine call			CALL PROGVAR		
CALLPATH	Programmable search path for subroutine calls		A path can be programmed to the existing NCK file system with CALLPATH	CALLPATH (/_N_WKS_DIR/_N_MY WPD/ subroutine_identifier_S PF)		
CANCEL	Cancel modal synchronized action	INT	Cancel with specified ID Without parameters: All modal synchronized actions are deselected			
CASE	Conditional program branch					
CDC	Direct approach of position (coded position: direct coordinate)		See CAC			
CDOF <sup>1</sup>	Collision detection OFF				m	23
CDON	Collision detection ON				m	23

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CDOF2	Collision detection OFF		For CUT3DC only		m	23
CFC <sup>1</sup>	Constant feed at contour				m	16
CFIN	Constant feed at internal radius only, not at external radius				m	16
CFTCP	Constant feed in tool-center- point (center-point path)				m	16
CHAN	Specify validity range for data		Once per channel			
CHANDATA	Set channel number for channel data access	INT	Only permissible in the initialization block			
CHAR	Data type: ASCII character	0, ..., 255				
CHECKSUM	Forms the checksum over an array as a fixed-length STRING	Max. length 32	Returns string of 16 hex digits	ERROR= CHECKSUM		
CHF CHR	Chamfer; value = length of chamfer  Chamfer; value = width of chamfer in direction of movement (chamfer)	Real, w/o signs			n	
CHKDNO	Check for unique D numbers					
CIC	Incremental approach of position (coded position: incremental coordinate)	See CAC				
CIP	Circular interpolation through intermediate point			CIP X... Y... Z... I1=... J1=... K1=...	m	1

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CLEARM	Reset one/several markers for channel coordination	INT, 1 - n	Does not influence machining in own channel			
CLRINT	Deselect interrupt:	INT	Parameter: Interrupt number			
CMIRROR	Mirror on a coordinate axis	FRAME				
COARSEA	Motion end when "Exact stop coarse" reached			COARSEA=... or COARSEA[n]=...	m	
COMPOF <sup>1</sup>	Compressor OFF				m	30
COMPON	Compressor ON				m	30
COMPCURV	Compressor ON: Polynomials with constant curvature				m	30
COMPCAD	Compressor ON: Optimized surface quality CAD program				m	30
CONTDCON	Tabular contour decoding ON					
CONTPRON	Activate contour preparation (contour preparation ON)					
COS	Cosine (trigon. function)	Real				
COUPDEF	Definition ELG group/synchronous spindle group (couple definition)	String	Block change (software) response: NOC: no block-change control FINE/COARSE: block change on "synchronism fine/coarse" IPOSTOP: block change in setpoint-dependent termination of overlaid movement	COUPDEF(FS, ...)		

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
COUPDEL	Delete ELG group (couple delete)			COUPDEL(FS,LS)		
COUPOF	ELG group/synchronous spindle pair OFF (couple OFF)			COUPOF(FS,LS, POS <sub>FS</sub> ,POS <sub>LS</sub> )		
COUPOFS	Deactivating ELG group/synchronized spindle pair with stop of following spindle			COUPOFS(FS,LS,POS <sub>FS</sub> )		
COUPON	ELG group/synchronous spindle pair ON (couple ON)			COUPON(FS,LS, POS <sub>FS</sub> )		
COUPONC	Transfer activation of ELG group/synchronized spindle pair with previous programming			COUPONC(FS,LS)		
COUPRES	Reset ELG group (couple reset)		Programmed values invalid; machine data values valid	COUPRES(FS,LS)		
CP	Path movement (continuous path)				m	49
CPRECOF <sup>1</sup>	Programmable contour precision OFF				m	39
CPRECON	Programmable contour precision ON				m	39
C PROT	Channel-specific protection zone ON/OFF					
C PROTDEF	Channel specific protection area definition					
CR	Circle radius	Real, without sign			n	
CROT	Rotation of the current coordinate system	FRAME	Max. parameter count: 6			
CROTS	Programmable frame rotations with solid angles (rotation in the specified axes)			CROTS X... Y... CROTS Z... X... CROTS Y... Z... CROTS RPL= ;separate block	n	

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CSCALE	Scale factor for multiple axes	FRAME	Max. parameter count: 2 * axis count <sub>max</sub>			
CSPLINE	Cubic spline				m	1
CT	Circle with tangential transition			CT X... Y.... Z...	m	1
CTAB	Define following axis position according to leading axis position from curve table	Real	If parameter 4/5 not programmed: Standard scaling			
CTABDEF	Table definition ON					
CTABDEL	Clear curve table					
CTABEND	Table definition OFF					
CTABEXISTS	Checks the curve table with number n		Parameter n			
CTABFNO	Number of curve tables still possible in the memory		memType			
CTABFPOL	Number of polynomials still possible in the memory		memType			
CTABFSEG	Number of curve segments still possible in the memory		memType			
CTABID	Returns table number of the nth curve table		parameter n and memType			
CTABINV	Define leading axis position according to following axis position from curve table	Real	See CTAB			
CTABIS LOCK	Returns the lock state of the curve table with number n		Parameter n			
CTABLOCK	Set lock against deletion and overwriting		Parameters n, m, and memType			
CTABMEMTYP	Returns the memory in which curve table number n is created		Parameter n			
CTABMPOL	Max. number of polynomials still possible in the memory		memType			

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CTABMSEG	Max. number of curve segments still possible in the memory	memType				
CTABNO	Number of defined curve tables irrespective of mem. type	No parameters				
CTABNOMEM	Number of defined curve tables in SRAM or DRAM memory	memType				
CTABPERIOD	Returns the table periodicity with number n	Parameter n				
CTABPOL	Number of polynomials already used in the memory	memType				
CTABPOLID	Number of the curve polynomials used by the curve table with number n	Parameter n				
CTABSEG	Number of curve segments already used in the memory	memType				
CTABSEGID	Number of the curve segments used by the curve table with number n	Parameter n				
CTABSEV	Returns the final value of the following axis of a segment of the curve table	Segment is determined by LW	R10 = CTABSEV(LW, n, degree, Faxis, Laxis)			
CTABSSV	Returns the initial value of the following axis of a segment of the curve table	Segment is determined by LW	R10 = CTABSSV(LW, n, degree, Faxis, Laxis)			
CTABTEP	Returns the value of the leading axis at curve table end	Master value at end of curve table	R10 = CTABTEP(n, degree, Laxis)			
CTABTEV	Returns the value of the the following axis at curve table end	Following value at end of curve table	R10 = CTABTEV(n, degree, Faxis)			
CTABTMAX	Returns the maximum value of the following axis of the curve table	Following value of the curve table	R10 = CTABTMAX(n, Faxis)			
CTABTMIN	Returns the minimum value of the following axis of the curve table	Following value of the curve table	R10 = CTABTMIN(n, Faxis)			
CTABTSP	Returns the value of the leading axis at curve table start	Master value at start of curve table	R10 = CTABTSP(n, degree, Laxis)			

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CTABTSV	Returns the value of the following axis at curve table start		Following value at start of curve table	R10 = CTABTSV(n, degree, Faxis)		
CTABUNLOCK	Cancel locking against deletion and overwriting		Parameters n, m, and memType			
CTRANS	Zero offset for multiple axes	FRAME	Max. eight axes			
CUT2D <sup>1</sup>	2D cutter compensation type 2-dimensional				m	22
CUT2DF	2D cutter compensation type 2-dimensional frame. The cutter compensation acts relative to the current frame (inclined plane).				m	22
CUT3DC	3D cutter compensation type 3-dimensional circumference milling				m	22
CUT3DCC	3D cutter compensation type 3-dimensional circumference milling with limitation surfaces				m	22
CUT3DCCD	3D cutter compensation type 3-dimensional circumference milling with limitation surfaces with differential tool				m	22
CUT3DF	3D cutter compensation type 3-dimensional face milling				m	22
CUT3DFF	3D cutter compensation type 3-dimensional face milling with constant tool orientation dependent on the current frame				m	22
CUT3DFS	3D cutter compensation type 3-dimensional face milling with constant tool orientation independent of the current frame				m	22

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
CUTCONOF <sup>1</sup>	Constant radius compensation OFF				m	40
CUTCONON	Constant radius compensation ON				m	40
CUTMOD	Activate "Modification of the offset data for rotatable tools"					
D	Tool offset number	1, ..., 32 000	Contains offset data for a particular tool T... ;D0 → offset values for a tool	D...		
DAC	Absolute, non-modal, axis-specific diameter programming		Diameter programming	DAC(50)	n	
DC	Absolute dimensions for rotary axes, approach position directly			A=DC(...) B=DC(...) C=DC(...) SPOS=DC(...)	n	
DEF	Variable definition	Integer, without sign				
DEFAULT	Branch in CASE branch		Jump to if expression does not fulfill any of the specified values			
DELAYFSTON	Define start of a stop delay range (DELAY feed stop ON)		Implied if G331/G332 active		m	
DELAYFSTOF	Define end of a stop delay range (DELAY feed stop OFF)				m	
DELDTG	Delete distance-to-go					
DELETE	Delete the specified file. The file name can be specified with path and file identifier		Can delete all files			

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
DELT	Delete tool	Duplo number can be omitted				
DIACYCOFA	Axis-specific, modal diametral programming: OFF in cycles	Radius programming, last active G code	DIACYCOFA[axis]	m		
DIAM90	Diameter programming for G90, radius programming for G91			m	29	
DIAM90A	Axis-specific, modal diameter programming for G90 and AC, radius programming for G91 and IC			m		
DIAMCHAN	Acceptance of all axes from MD axis functions in diameter-programming channel status	Accept diameter programming from MD	DIAMCHAN			
DIAMCHANA	Acceptance of the diameter-programming channel status	Channel status	DIAMCHANA[axis]			
DIAMCYCOF	Radius programming for G90/G91: ON. The G code of this group that was last active remains active for display	Radius programming, last active G code		m	29	
DIAMOF <sup>1</sup>	Diameter programming: OFF (Diameter programming OFF) For default setting, see machine manufacturer.	Radius programming for G90/G91		m	29	
DIAMOFA	Axis-specific, modal diameter programming: ON For default setting, see machine manufacturer	Radius progr. for G90/G91 and AC, IC	DIAMOFA[axis]	m		
DIAMON	Diameter programming: ON (Diameter programming ON)	Diameter programming for G90/G91.		m	29	
DIAMONA	Axis-specific, modal diameter programming: ON For activation, see machine manufacturer	Diameter programming for G90/G91 and AC, IC	DIAMONA[axis]	m		

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
DIC	Relative, non-modal, axis-specific diameter programming		Diameter programming	DIC(50)	n	
DILF	Length for lift fast				m	
DISABLE	Interrupt OFF					
DISC	Transition circle overshoot - radius compensation	0, ..., 100			m	
DISPLOF	Suppress current block display (display OFF)					
DISPR	Distance for repositioning	Real, without sign			n	
DISR	Distance for repositioning	Real, without sign			n	
DITE	Thread run-out path	Real			m	
DITS	Thread run-in path	Real			m	
DIV	Integer division					
DL	Total tool offset	INT			m	
DRFOF	Deactivate the handwheel offsets (DRF)				m	
DRIVE <sup>6</sup>	Velocity-dependent path acceleration				m	21
DRIVEA	Switch on bent acceleration characteristic curve for the programmed axes					
DYNFINISH	Dynamic response for smooth finishing		Technology G group	DYNFINISH G1 X10 Y20 Z30 F1000	m	59
DYNNORM	Standard dynamic, as previously			DYNNORM G1 X10	m	59
DYNPOS	Dynamics for positioning mode, tapping			DYNPOS G1 X10 Y20 Z30 F...	m	59
DYNROUGH	Dynamic for roughing			DYNROUGH G1 X10 Y20 Z30 F10000	m	59
DYNSEMFIN	Dynamic for finishing			DYNSEMFIN G1 X10 Y20 Z30 F2000	m	59

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
EAUTO	Definition of last spline segment by the last three points (end not a knot)				m	20
EGDEF	Definition of an electronic gear (electronic gear define)		For one following axis with up to five leading axes			
EGDEL	Delete coupling definition for the following axis (electronic gear delete)		Stops the preprocessing			
EGOFC	Switch off electronic gear continuously (electronic gear OFF continuous)					
EGOFS	Switch off electronic gear selectively (electronic gear OFF selective)					
EGON	Switch on electronic gear (Electronic gear ON)		Without synchronization			
EGONSYN	Switch on electronic gear (electronic gear ON synchronized)		With synchronization			
EGONSYNE	Switch on electronic gear, stating approach mode (electronic gear ON synchronized)		With synchronization			
ELSE	Program branch, if IF condition not fulfilled					
ENABLE	Interrupt ON					
ENAT <sup>1</sup>	Natural transition to next traversing block (end natural)				m	20
ENDFOR	End line of FOR counter loop					
ENDIF	End line of IF branch					
ENDLOOP	End line of endless program loop LOOP					
ENDPROC	End line of program with start line PROC					
ENDWHILE	End line of WHILE loop					

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
ETAN	Tangential transition to next traversing block at spline end (end tangential)				m	20
EVERY	Execute synchronized action if condition changes from FALSE to TRUE					
EXECSTRING	Transfer of a string variable with the part program line to run	Indirect part program line	EXECSTRING(MFCT1 << M4711)			
EXECTAB	Execute an element from a motion table (execute table)					
EXECUTE	Program execution ON	Return from the reference-point edit mode or after building a protection area to normal program processing				
EXP	Exponential function ( $e^x$ )	Real				
EXTCALL	Execute external subroutine		Reload program from HMI in "Execution from external source" mode			
EXTERN	Broadcast a subroutine with parameter passing					
F	Feed value (in conjunction with G4 the dwell time is also programmed in F)	0.001, ..., 99999.999	Path veloc. of a tool/workpiece; unit: mm/min or mm/revolution depending on G94 or G95	F=100 G1 ...		

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
FA	Axial feed (feed axial)	0.001, ..., 999999.999 mm/min, degrees/min; 0.001, ..., 39999.9999 inch/min		FA[X]=100	m	
FAD	Infeed feed for smooth approach and retraction (feed approach/depart)	Real, without sign				
FALSE	Logical constant: Incorrect	BOOL	Can be replaced with integer constant 0			
FCTDEF	Define polynomial function		Is evaluated in SYNFCT or PUTFTOCF			
FCUB	Feedrate variable according to cubic spline (feed cubic)		Acts on feed with G93 and G94		m	37
FD	Path feed for handwheel override (feed DRF)	Real, w/o signs			n	
FDA	Axial feed for handwheel override (feed DRF axial)	Real, w/o signs			n	
FENDNORM	Corner deceleration OFF				m	57
FFWOF <sup>1</sup>	Feedforward control OFF (feed forward OFF)				m	24
FFWON	Feedforward control ON (feed forward ON)				m	24
FGREF	Reference radius of rotary axis or path reference factors of orientation axes (vector interpolation)		Reference size effective value		m	
FGROUP	Definition of axis/axes with path feed		F applies to all axes specified under FGROUP	FGROUP (axis1, [axis2], ...)		
FIFOCTRL	Control of preprocessing buffer				m	4

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
FIFOLEN	Programmable preprocessing depth					
FILEDATE	Delivers date when file was last accessed and written	STRING, length 8	Format is "dd.mm.yy"			
FILEINFO	Delivers sum of FILEDATE, FILESIZE, FILESTAT and FILETIME	STRING, length 32	Format "rwxsd nnnnnnnn dd. hh:mm:ss"			
FILESIZE	Delivers current file size	Type: INT	In BYTES			
FILESTAT	Delivers file status of rights for read, write, execute, display, delete (rwxsd)	STRING, length 5	Format is "rwxsd"			
FILETIME	Delivers time when file was last accessed and written	STRING, length 8	Format is "dd:mm:yy"			
FINEA	Motion end when "Exact stop fine" reached			FINEA=... or FINEA[n]=...	m	
FL	Speed limit for synchronized axes (feed limit)	Real, without sign	The unit set with G93, G94, G95 is applicable (max. rapid traverse)	FL[axis] =...	m	
FLIN	Feed linear variable (feed linear)		Acts on feed with G93 and G94		m	37
FMA	Feed multiple axial	Real, without sign			m	
FNORM <sup>1</sup>	Feed normal to DIN 66025				m	37
FOCOF	Deactivate travel with limited moment/force				m	
FOCON	Activate travel with limited moment/force				m	
FOR	Counter loop with fixed number of passes					

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
FP	Fixed point: Number of fixed point to be approached	Integer, without sign		G75 FP=1	n	
FPO	Feed characteristic programmed via a polynomial (feed polynomial)	Real	Quadratic, cubic polynomial coefficient			
FPR	Identification for rotary axis	0.001, ..., 999999.999		FPR (rotary axis)		
FPRAOF	Deactivate revolutionary feedrate					
FPRAON	Activate revolutionary feedrate					
FRAME	Data type to define the coordinate system		Contains for each geometry axis: Offset, rotation, angle of shear, scaling, mirroring; For each special axis: offset, scaling, mirroring			
FRC	Feed for radius and chamfer				n	
FRCM	Feed for radius and chamfer, modal				m	
FTOC	Change fine tool offset		As a function of a 3rd-order polynomial defined with FCTDEF			
FTOCOF <sup>1</sup>	Online fine tool offset OFF				m	33
FTOCON	Online fine tool offset ON				m	33

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
FXS	Travel to fixed stop ON	Integer, without sign	1 = select, 0 = deselect		m	
FXST	Torque limit for travel to fixed stop (fixed stop torque)	%	parameter optional		m	
FXSW	Monitoring window for travel to fixed stop (fixed stop window)	mm, inch or degrees	parameter optional			
G	G function (preparatory function)  The G functions are divided into G 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)	Only specified, integer values		G...		
G0	Linear interpolation with rapid traverse (rapid traverse motion)	Motion commands		G0 X... Z...	m	1
G1 <sup>1</sup>	Linear interpolation with feedrate (linear interpolation)			G1 X... Z... F...	m	1
G2	Circular interpolation clockwise			G2 X... Z... I... K... F... ;Center point and end point G2 X... Z... CR=... F... ;radius and end point G2 AR=... I... K... F... ;opening angle and center point G2 AR=... X... Z... F... ;opening angle and end point	m	1
G3	Circular interpolation counterclockwise			G3 ...; otherwise as for G2	m	1

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
G4	Dwell time preset		Special motion	G4 F... ; dwell time in seconds or G4 S... ; dwell time in spindle revolutions. ;separate block	n	2
G5	Oblique plunge-cut grinding		Oblique plunge-cutting		n	2
G7	Compensatory motion during oblique plunge-cut grinding		Start position		n	2
G9	Exact stop - deceleration				n	11
G17 <sup>1</sup>	Selection of working plane X/Y		Infeed direction Z		m	6
G18	Selection of working plane Z/X		Infeed direction Y		m	6
G19	Selection of working plane Y/Z		Infeed direction X		m	6
G25	Lower working area limitation		Value assignment in channel axes	G25 X... Y... Z... ;separate block	n	3
G26	Upper working area limitation			G26 X... Y... Z... ;separate block	n	3
G33	Thread interpolation with constant lead	0.001, ..., 2000.00 mm/rev	Motion command	G33 Z... K... SF=... ;cylindrical thread G33 X... I... SF=... ;face thread G33 Z... X... K... SF=... ;taper thread (path longer in Z axis than in X axis) G33 Z... X... I... SF=... ;taper thread (path longer in X axis than in Z axis)	m	1
G34	Linear progressive speed change [mm/rev <sup>2</sup> ]		Motion command	G34 X... Y... Z... I... J... K... F...	m	1
G35	Linear degressive speed change [mm/rev <sup>2</sup> ]		Motion command	G35 X... Y... Z... I... J... K... F...	m	1

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
G40 <sup>1</sup>	Tool radius compensation OFF				m	7
G41	Tool radius compensation left of contour				m	7
G42	Tool radius compensation right of contour				m	7
G53	Suppression of current zero offset (non-modal)		Incl. programmed offsets		n	9
G54	1st settable zero offset				m	8
G55	2nd settable zero offset				m	8
G56	3rd settable zero offset				m	8
G57	4th settable zero offset				m	8
G58	Axial programmable zero offset, absolute				n	3
G59	Axial programmable zero offset, additive				n	3
G60 <sup>1</sup>	Exact stop - deceleration				m	10
G62	Corner deceleration at inside corners when tool radius offset is active (G41, G42)		Together with continuous-path mode only	G62 Z... G1	m	57
G63	Tapping with compensating chuck			G63 Z... G1	n	2
G64	Exact stop - continuous-path mode				m	10
G70	Dimension in inches (lengths)				m	13
G71 <sup>1</sup>	Metric dimension (lengths)				m	13
G74	Reference point approach			G74 X... Z... ;separate block	n	2
G75	Fixed point approach	Machine axes		G75 FP=.. X1=... Z1=... ;separate block	n	2
G90 <sup>1</sup>	Absolute dimensions			G90 X... Y... Z...(...) Y=AC(...) or X=AC Z=AC(...)	m n	14
G91	Incremental dimension input			G91 X... Y... Z... or X=IC(...) Y=IC(...) Z=IC(...)	m n	14

## Tables

### 16.1 Statements A - G

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
G93	Inverse-time feedrate 1/rpm		Execution of a block: Time	G93 G01 X... F...	m	15
G94 <sup>1</sup>	Linear feedrate F in mm/min or inch/min and °/min				m	15
G95	Revolutional feedrate F in mm/rev or inches/rev				m	15
G96	Constant cutting speed (as for G95) ON			G96 S... LIMS=... F...	m	15
G97	Constant cutting speed (as for G95) OFF				m	15
G110	Pole programming relative to the last programmed setpoint position			G110 X... Y... Z...	n	3
G111	Polar programming relative to origin of current workpiece coordinate system			G110 X... Y... Z...	n	3
G112	Pole programming relative to the last valid pole			G110 X... Y... Z...	n	3
G140 <sup>1</sup>	SAR approach direction defined by G41/G42				m	43
G141	SAR approach direction to left of contour				m	43
G142	SAR approach direction to right of contour				m	43
G143	SAR approach direction tangent-dependent				m	43
G147	Soft approach with straight line				n	2
G148	Soft retraction with straight line				n	2
G153	Suppress current frames including base frame		Incl. system frame		n	9
G247	Soft approach with quadrant				n	2
G248	Soft retraction with quadrant				n	2
G290	Switch to SINUMERIK mode ON				m	47
G291	Switch to ISO2/3 mode ON				m	47
G331	Tapping	±0.001, ..., 2000.00 mm/rev	Motion commands		m	1
G332	Retraction (tapping)				m	1

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
G340 <sup>1</sup>	Spatial approach block (depth and in plane (helix))		Effective during soft approach/retraction		m	44
G341	Initial infeed on perpendicular axis (z), then approach in plane		Effective during soft approach/retraction		m	44
G347	Soft approach with semicircle				n	2
G348	Soft retraction with semicircle				n	2
G450 <sup>1</sup>	Transition circle		Corner behavior with tool radius compensation		m	18
G451	Intersection of equidistances				m	18
G460 <sup>1</sup>	Collision monitoring for approach and retraction block ON				m	48
G461	Extend border block with arc if ...	... no intersection in TRC block			m	48
G462	Extend border block with line if ...				m	48
G500 <sup>1</sup>	Deactivation of all settable frames if G500 does not contain a value				m	8
G505 ...G599	5 ... 99. Settable zero offset				m	8
G601 <sup>1</sup>	Block change at exact stop fine	Only effective: - with act. G60 or - with G9 with programmable transition rounding			m	12
G602	Block change at exact stop coarse				m	12
G603	Block change at IPO - end of block				m	12
G641	Exact stop - continuous-path mode		G641 AIDS=...		m	10
G642	Corner rounding with axial precision				m	10
G643	Block-internal corner rounding				m	10
G644	Corner rounding with specified axis dynamics				m	10
G621	Corner deceleration at all corners	Together with continuous-path mode only	G621 AIDS=...		m	57

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
G700	Dimensions in inches and inch/min (lengths + velocities + system variable)				m	13
G710 <sup>1</sup>	Metric dimension in mm and mm/min (lengths + velocities + system variable)				m	13
G810 <sup>1</sup> , ..., G819	G group reserved for the OEM					31
G820 <sup>1</sup> , ..., G829	G group reserved for the OEM					32
G931	Feedrate specified by travel time	Travel time			m	15
G942	Freeze linear feedrate and constant cutting rate or spindle speed				m	15
G952	Freeze revolutionary feedrate and constant cutting rate or spindle speed				m	15
G961	Constant cutting rate and linear feed	Feed type as for G94	G961 S... LIMS=... F...		m	15
G962	Linear or revolutionary feedrate and constant cutting rate				m	15
G971	Freeze spindle speed and linear feed	Feed type as for G94			m	15
G972	Freeze linear or revolutionary feedrate and constant spindle speed				m	15
G973	Revolutionary feedrate without spindle speed limitation	G97 without LIMS for ISO mode			m	15
GEOAX	Assign new channel axes to geometry axes 1 - 3	Without parameter: MD settings effective				
GET	Assign machine axis/axes	Axis must be released in the other channel with RELEASE				

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
GETD	Assign machine axis/axes directly		See GET			
GETACTT	Get active tool from a group of tools with the same name					
GETSELT	Get selected T number					
GETT50	Get T number for tool name					
GOTO	Jump statement first forward then backward (direction initially to end of program and then to start of program)		Can be applied in part program and technology cycles	GOTO (label, block no.) Labels must exist in the subroutine.		
GOTOC	As with GOTO + suppress alarm 14080 "Destination not found"					
GOTOB	Jump backwards (toward the start of the program)			GOTOB (Label, block no.)		
GOTOF	Jump forwards (toward the end of the program)			GOTOF (Label, block no.)		
GOTOS	Jump back to start of program			GOTOS		
BP	Keyword for the indirect programming of position attributes			E.g. X=GP(...)		
GWPSOF	Deselect constant grinding wheel peripheral speed (GWPS)			GWPSOF(T No.)	n	
GWPSON	Select constant grinding wheel peripheral speed (GWPS)			GWPSON(T No.)	n	
H...	Auxiliary function output to the PLC	Real/INT progr.: REAL: 0 ...+/- 3.4028 exp38 INT: -2147483646 ... +2147483647 Display: ± 999,999,999.9 999	Can be set for each MD (machine manufacturer)	H100 or H2=100		

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
I <sup>4</sup>	Interpolation parameters	Real			n	
I1	Intermediate point coordinate	Real			n	
IC	Incremental dimensioning	0, ..., ±99999.999°		X=IC(10)	n	
ICYCOF	All blocks of a technology cycle are processed in one IPO cycle following ICYCOF		Within the program level only			
ICYCON	Each block of a technology cycle is processed in a separate IPO cycle following ICYCON		Within the program level only			
IDS	Identification of static synchronized actions					
IF	Introduction of a conditional jump in the part program/technology cycle		Structure: IF-ELSE-ENDIF	IF (condition)		
INCCW	Travel on a circle involute in CCW direction with interpolation of involute by G17/G18/G19	Real	End point: Center point: Radius with CR > 0:	INCW/INCCW X... Y... Z... INCW/INCCW I... J... K...	m	1
INCW	Travel on a circle involute in CW direction with interpolation of involute by G17/G18/G19	Real	Angle of rotation in degrees between start and end vectors	INCW/INCCW CR=... AR... Direct programming: INCW/INCCW I... J... K... CR=... AR=...	m	1
INDEX	Define index of character in input string	0, ..., INT	String: 1st parameter, character: 2nd parameter			
INIT	Select module for execution in a channel		Channel numbers 1-10 or \$MC_CHAN_NAME	INIT(1,1,2) or INIT(CH_X, CH_Y)		
INT	Data type: Integer with sign	- (2 <sup>31</sup> -1), ..., 2 <sup>31</sup> -1				

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
INTERSEC	Calculate intersection between two contour elements and specify TRUE intersection status in ISPOINT	VAR REAL [2]	ISPOINT error status: BOOL FALSE	ISPOINTS=INTERSEC (TABNAME1[n1], TABNAME2[n2], ISTCOORD, MODE)		
IP	Variable interpolation parameter	Real				
IPOBRKA	Motion criterion from braking ramp activation		Braking ramp at 100% to 0%	IPOBRKA=.. or IPOBRKA(<axis>[,REAL])	m	
IPOENDA	End of motion when "IPO stop" is reached			IPOENDA=.. or IPOENDA[n]..	m	
IPTRLOCK	Freeze start of the untraceable program section at next machine function block		Freeze interrupt pointer		m	
IPTRUNLOCK	Set end of untraceable program section at current block at time of interruption		Set interrupt pointer		m	
ISAXIS	Check if geometry axis 1 specified as parameter	BOOL				
ISD	Insertion depth	Real			m	
ISFILE	Check whether the file exists in the NCK user memory	BOOL	Returns results of type BOOL	RESULT=ISFILE("Testfile") IF (RESULT==FALSE)		
ISNUMBER	Check whether the input string can be converted to a number	BOOL	Convert input string to a number			
ISPOINTS	Possible intersections calculated by ISTAB between two contours on the current plane	INT	Machining type MODE (optional)	STATE=ISPOINTS (KTAB1[n1], KTAB2[n2], ISTAB, [MODE])		

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
ISVAR	Check whether the transfer parameter contains a variable known in the NC	BOOL	Machine data, setting data and variables such as GUDs			
J <sup>4</sup>	Interpolation parameters	Real			n	
J1	Intermediate point coordinate	Real			n	
JERKA	Activate acceleration response set via MD for programmed axes					
JERKLIMA	Reduction or overshoot of maximum jerk (jerk axial)	1, ..., 200	Valid range is 1 to 200%	JERKLIMA[X]= ...[%]	m	
K <sup>4</sup>	Interpolation parameters	Real			n	
K1	Intermediate point coordinate	Real			n	
KONT	Travel round contour on tool offset				m	17
KONTC	Approach/retract with continuous-curvature polynomial				m	17
KONTT	Approach/retract with continuous-tangent polynomial				m	17
L	Subroutine number	Integer, up to 7 places		L10	n	
LEAD	Lead angle	Real			m	
LEADOF	Master value coupling OFF (lead off)					
LEADON	Master value coupling ON (lead on)					
LFOF <sup>1</sup>	Interrupt thread cutting OFF				m	41
LFON	Interrupt thread cutting ON				m	41
LFPOS	Axial retraction to a position				m	46
LFTXT <sup>1</sup>	Tangential tool direction on retraction				m	46
LFWP	Non-tangential tool direction on retraction				m	46

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
LIFTFAST	Rapid lift before interrupt routine call					
LIMS	Spindle speed limitation with G96/G961 and G97 (limit spindle speed)	0.001, ..., 99 999. 999			m	
LN	Natural logarithm	Real				
LOCK	Disable synchronized action with ID (stop technology cycle)					
LOG	(Common) logarithm	Real				
LOOP	Introduction of an endless loop		Structure: LOOP- ENDLOOP			
M...	Switching operations	INT Display: 0, ..., 999 999 999 Program: 0,..., 2147483647	Up to 5 unassigned M functions can be assigned by the machine manufacturer			
M0 <sup>6</sup>	Programmed stop					
M1 <sup>6</sup>	Optional stop					
M2 <sup>6</sup>	End of main program with return to beginning of program					
M3	Direction of spindle rotation clockwise for master spindle					
M4	Direction of spindle rotation counterclockwise for master spindle					
M5	Spindle stop for master spindle					
M6	Tool change					
M17 <sup>6</sup>	End of subroutine					
M19	For SSL accumulated spindle programming					
M30 <sup>6</sup>	End of program, same effect as M2					
M40	Automatic gear change					
M41... M45	Gear stage 1, ..., 5					
M70	Transition to axis mode					

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
MASLDEF	Define master/slave axis grouping					
MASLDEL	Uncouple master/slave axis grouping and clear grouping definition					
MASLOF	Disable a temporary coupling					
MASLOFS	Deactivate a temporary coupling with automatic slave axis stop					
MASLON	Enable a temporary coupling					
MAXVAL	Larger value of two variables (arithm. function)	Real	If values are the same, the same value is returned	ValMax = MAXVAL(Var1, Var2)		
MCALL	Modal subroutine call		Without subroutine name: Deselection			
MEAC	Continuous measurement without deleting distance-to-go	Integer, without sign			n	
MEAFRAME	Frame calculation from measuring points	FRAME				
MEAS	Measure with touch-trigger probe	Integer, without sign			n	
MEASA	Measurement with deletion of distance-to-go				n	
MEAW	Measure with touch-trigger probe without deleting distance-to-go	Integer, without sign			n	
MEAWA	Measurement without deletion of distance-to-go				n	
MI	Access to frame data: Mirroring				MI	
MINDEX	Define index of character in input string	0, ..., INT	String: 1st parameter, character: 2nd parameter			

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
MINVAL	Smaller value of two variables (arithm. function)	Real	If values are the same, the same value is returned	ValMin = MINVAL(Var1, Var2)		
MIRROR	Programmable Mirroring			MIRROR X0 Y0 Z0 ;separate block	n	3
MMC	Call the dialog window interactively from the part program on the HMI	STRING				
MOD	Modulo division					
MODAXVAL	Determine modulo position of a modulo rotary axis	Real				
MOV	Start positioning axis (start moving positioning axis)	Real				
MSG	Programmable messages			MSG("message")	m	
N	Block number - subblock	0, ..., 9999 9999 integers only, without signs	Can be used for assigning a number to a block; located at beginning of block	e.g. N20		
NCK	Specify validity range for data		Once per NCK			
NEWCONF	Accept modified machine data Corresponds to set machine data active		Also possible via HMI			
NEWT	Create new tool		Duplo number can be omitted			
NORM <sup>1</sup>	Standard setting in starting point and end point with tool offset				m	17
NOT	Logical NOT (negation)					
NPROT	Machine-specific protection zone ON/OFF					

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
NPROTDEF	Machine-specific protection area definition (NCK-specific protection area definition)					
NUMBER	Convert input string to number	Real				
OEMIPO1 <sup>5</sup>	OEM interpolation 1				m	1
OEMIPO2 <sup>5</sup>	OEM interpolation 2				m	1
OF	Keyword in CASE branch					
OFFN	Allowance on the programmed contour			OFFN=5		
OMA1	OEM address 1	Real			m	
OMA2	OEM address 2	Real			m	
OMA3	OEM address 3	Real			m	
OMA4	OEM address 4	Real			m	
OMA5	OEM address 5	Real			m	
OFFN	Offset - normal	Real			m	
OR	Logical OR					
ORIC <sup>1</sup>	Orientation changes at outside corners are superimposed on the circle block to be inserted (orientation change continuously)				m	27
ORID	Orientation changes are performed before the circle block (orientation change discontinuously)				m	27
ORIAXPOS	Orientation angle via virtual orientation axes with rotary axis positions				m	50
ORIEULER	Orientation angle via Euler angle				m	50
ORIAxes	Linear interpolation of machine axes or orientation axes	Final orientation: Vector specification A3, B3, C3, or Euler/RPY angle A2_B2	Parameter settings as follows:  Direction vectors normalized A6=0 B6=0 C6=1 Opening angle		m	51
ORICONCW	Interpolation on a circular peripheral surface in CW direction				m	51
ORICONCCW	Interpolation on a circular peripheral surface in CCW direction				m	51

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
ORICONIO	Interpolation on a circular peripheral surface with intermediate orientation setting				m	51
ORICONTO	Interpolation on circular peripheral surface in tangential transition (final orientation)				m	51
ORICURVE	Interpolation of orientation with specification of motion of two contact points of tool				m	51
ORIPLANE	Interpolation in a plane (corresponds to ORIVECT), large-radius circular interpolation				m	51
ORIPATH	Tool orientation in relation to path				m	51
ORIPATHS	Tool orientation in relation to path, blips in the orientation characteristic are smoothed				m	51
ORIROTA	Angle of rotation to an absolute direction of rotation.				m	54
ORIROTC	Tangential rotational vector in relation to path tangent				m	54
ORIROTR	Angle of rotation relative to the plane between the start and end orientation				m	54
ORIROTT	Angle of rotation relative to the change in the orientation vector				m	54
ORIRPY	Orientation angle via RPY angle (XYZ)				m	50
ORIRPY2	Orientation angle via RPY angle (ZYX)				m	50
ORIS	Orientation modification (orientation smoothing factor)	Real			m	
ORIVECT	Large-radius circular interpolation (identical to ORIPLANE)				m	51
ORIVIRT1	Orientation angle via virtual orientation axes (definition 1)				m	50

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
ORIVIRT2	Orientation angle via virtual orientation axes (definition 2)				m	50
ORIMCS	Tool orientation in the machine coordinate system				m	25
ORIRESET	Initial setting of tool orientation with up to three orientation axes		Parameter optional (REAL)	ORIRESET(A,B,C)		
ORIWKS <sup>1</sup>	Tool orientation in the workpiece coordinate system				m	25
OS	Oscillation on/off	Integer, without sign				
OSB	Oscillating: Start point				m	
OSC	Continuous tool orientation smoothing				m	34
OSCILL	Axis assignment for oscillation- activate oscillation		Axis: 1 - 3 infeed axes		m	
OSCTRL	Oscillation control options	Integer, without sign			m	
OSD	Rounding of tool orientation by specifying rounding length with SD		Block-internal		m	34
OSE	Oscillating: End point				m	
OSNSC	Oscillating: Number of spark-out cycles (oscillating: number spark-out cycles)				m	
OSOF <sup>1</sup>	Tool-orientation smoothing OFF				m	34
OSP1	Oscillating: Left reversal point (oscillating: position 1)	Real			m	
OSP2	Oscillating: Right reversal point (oscillating: position 2)	Real			m	
OSS	Tool-orientation smoothing at end of block				m	34
OSSE	Tool-orientation smoothing at start and end of block				m	34

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
OST	Rounding of tool orientation by specifying angle tolerance in degrees with SD (maximum deviation from programmed orientation characteristic)		Block-internal		m	34
OST1	Oscillating: Stopping point in left reversal point	Real			m	
OST2	Oscillating: Stopping point in right reversal point	Real			m	
OVR	Speed override	1, ..., 200%			m	
OVRA	Axial speed override	1, ..., 200%			m	
OVRRAP	Rapid traverse override	1, ..., 100%			m	
P	Number of subroutine cycles	1, ..., 9999, integers w/o signs		e.g. L781 P... ;separate block		
PCALL	Call subroutines with the absolute path and parameter transfer		No absolute path. Behavior as for CALL			
PAROT	Align workpiece coordinate system on workpiece				m	52
PAROTOF	Deactivate workpiece-related frame rotation				m	52
PDELAYOF	Punch with delay OFF				m	36
PDELAYON <sup>1</sup>	Punch with delay ON				m	36
PL	Parameter interval length	Real, without sign			n	
PM	Per minute		Feed per minute			
PO	Polynomial	Real, without sign			n	
POLF	LIFTFAST position	Real, without sign	Geometry axis in WCS, otherwise MCS	POLF[Y]=10 target position of retracting axis	m	

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
POLFA	Start retract position of single axes with \$AA_ESR_TRIGGER		For single axes	POLFA(AX1, 1, 20.0)	m	
POLFMASK	Enable axes for retraction <b>without</b> a connection between the axes		Selected axes	POLFMASK(AX1, AX2, ...)	m	
POLFMIN	Enable axes for retraction <b>with</b> a linear connection between the axes		Selected axes	POLFMIN(AX1, AX2, ...)	m	
POLY	Polynomial interpolation				m	1
POLYPATH	Polynomial interpolation can be selected for the AXIS or VECT axis groups			POLYPATH ("AXES") POLYPATH ("VECT")	m	1
PON	Punch ON				m	35
PONS	Punch ON in IPO cycle (punch ON slow)				m	35
POS	Axis positioning			POS[X]=20		
POSA	Position axis across block boundary			POSA[Y]=20		
POSP	Positioning in part sections (oscillation) (position axis in parts)	Real: end position, part length; Integer: option				
POT	Square (arithmetic function)	Real				
PR	Per revolution			Revolutional feedrate		
PRESETON	Sets the actual value for programmed axes		One axis identifier is programmed at a time, with its respective value in the next parameter Up to eight axes possible.	PRESETON(X,10,Y,4.5)		

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
PRI0	Keyword for setting the priority for interrupt processing					
PROC	First statement in a program			Block number - PROC - identifier		
PTP	Point-to-point motion (point-to-point)		synchronized axis		m	49
PTPG0	Point-to-point motion only with G0, otherwise CP		synchronized axis		m	49
PUTFTOC	Fine tool offset for parallel dressing (continuous dressing) (put fine tool correction)		Channel numbers 1 - 10 or \$MC _CHAN_NAME	PUTFTOC(1,1,2) or PUTFTOC(CH_name)		
PUTFTOCF	Fine tool offset depending on a function defined with FCtDEF for parallel dressing (continuous dressing) (put fine tool correction function dependant)		Channel numbers 1 - 10 or \$MC _CHAN_NAME	PUTFTOCF(1,1,2) or PUTFTOCF(CH_name)		
PW	Point weight	Real, without sign			n	
QECLRNOF	Quadrant error compensation learning OFF					
QECLRNON	Quadrant error compensation learning ON					
QU	Fast additional (auxiliary) function output					
R...	Arithmetic parameters also as settable address identifier and with numerical extension	± 0.0000001, ..., 9999 9999	Number of R parameters that can be set by MD	R10=3 ;R parameter assignment X=R10 ;axis valueR[R10]=6 ;indirect prog.		
RAC	Absolute, non-modal, axis-specific radius programming		Radius programming	RAC(50)	n	
RDISABLE	Read-in disable					

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
READ	Reads one or more lines in the specified file and stores the information read in the array		The information is available as STRING			
READAL	Read alarm		Alarms are searched according to ascending numbers			
REAL	Data type: floating-point variable with sign (real numbers)	Corresponds to the 64-bit floating point format of the processor				
REDEF	Setting for machine data, NC language elements and system variables, specifying the user groups they are displayed for					
RELEASE	Release machine axes		Multiple axes can be programmed			
REP	Keyword for initialization of all elements of an array with the same value		REP(value) or DO ARRAY[n,m]=REP()			
REPEAT	Repeat a program loop		Until (UNTIL) a condition is fulfilled			
REPEATB	Repeat a program line	nnn times				
REPOSA	Repositioning linear all axes				n	2
REPOSH	Repositioning semicircle				n	2

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
REPOSHA	Repositioning all axes; geometry axes semicircle (repositioning semicircle all axes)				n	2
REPOSL	Repositioning linear				n	2
REPOSQ	Repositioning quarter circle				n	2
REPOSQA	Repositioning linear all axes; geometry axes quarter circle (repositioning quarter circle all axes)				n	2
RESET	Reset technology cycle	One or several IDs can be programmed				
RET	End of subroutine	Use in place of M17 – without function output to PLC	RET			
RIC	Relative, non-modal, axis-specific radius programming	Radius programming	RIC(50)	n		
RINDEX	Define index of character in input string	0, ..., INT	String: 1st parameter, character: 2nd parameter			
RMB	Repositioning at beginning of block (repos mode begin of block)				m	26
RME	Repositioning at end of block (repos mode end of block)				m	26
RMI <sup>1</sup>	Repositioning at interruption point (repos mode interrupt)				m	26
RMN	Reapproach to nearest path point (repos mode end of nearest orbital block)				m	26

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
RND	Round the contour corner	Real, without sign		RND=...	n	
RNDM	Modal rounding	Real, without sign		RNDM=... RNDM=0: disable modal rounding	m	
ROT	Programmable rotation	Rotation around 1st geometry axis: -180° ... +180° 2nd geometry axis: -90° ... +90° 3rd geometry axis: -180° ... +180°		ROT X... Y... Z... ROT RPL= ;separate block	n	3
ROTS	Programmable frame rotations with solid angles (rotation)			ROTS X... Y... ROTS Z... X... ROTS Y... Z... ROTS RPL= ;separate block	n	3
ROUND	Round decimal places	Real				
RP	Polar radius	Real			m/n	
RPL	Rotation in the plane	Real, without sign			n	
RT	Parameter for access to frame data: Rotation					
RTLION	G0 with linear interpolation				m	55
RTLIOF	G0 without linear interpolation (single-axis interpolation)				m	55
S	Spindle speed or (with G4, G96/G961) other meaning	REAL Display: ±999 999 999.9999 Program: ±3.4028 ex38	Spindle speed in rpm G4: Dwell time in spindle revolutions G96/G961: Cutting rate in m/min.	S...: Speed for master spindle S1...: Speed for spindle 1	m/n	

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
SAVE	Attribute for saving information at subroutine calls		The following are saved: All modal G functions and the current frame			
SBLOF	Suppress single block (single block OFF)		The following blocks are executed in single block like a block			
SBLON	Clear single block suppression (single block ON)					
SC	Parameter for access to frame data: Scaling (scale)					
SCALE	Programmable scaling (scale)			SCALE X... Y... Z... ;separate block	n	3
SCC	Selective assignment of transverse axis to G96/G961/G962. Axis identifiers may take the form of geo, channel or machine axes		Also with constant cutting rate	SCC[axis]		
SD	Spline degree	Integer, without sign			n	
SEFORM	Structuring statement in Step editor to generate the step view for HMI Advanced		Evaluated in Step editor	SEFORM (<section_name>, <level>, <icon> )		
SET	Keyword for initialization of all elements of an array with listed values			SET(value, value, ...) or DO ARRAY[n,m]=SET()		
SETAL	Set alarm					
SETDNO	Set D number of tool (T) and its cutting edge to "new"					
SETINT	Define which interrupt routine is to be activated when an NCK input is present		Edge 0 → 1 is analyzed			
SETMS	Reset to the master spindle defined in machine data					

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
SETMS(n)	Set spindle n as master spindle					
SETPIECE	Set piece number for all tools assigned to the spindle		Without spindle number: applies to master spindle			
SF	Starting point offset for thread cutting (spline offset)	0.0000,..., 359.999°			m	
SIN	Sine (trigon. function)	Real				
SOFT	Jerk-limited path acceleration				m	21
SOFTA	Switch on soft axis acceleration for the programmed axes					
SON	Nibbling ON (stroke ON)				m	35
SONS	Nibbling ON in IPO cycle (stroke ON slow)				m	35
SPATH <sup>1</sup>	Path reference for FGROUP axes is arc length				m	45
SPCOF	Switch master spindle or spindle(s) from position control to speed control		SPCOF SPCOF(n)		m	
SPCON	Switch master spindle or spindle(s) from speed control to position control		SPCON SPCON (n)		m	
SPIF1 <sup>1</sup>	Fast NCK inputs/outputs for punching/nibbling byte 1 (stroke/punch interface 1)				m	38
SPIF2	Fast NCK inputs/outputs for punching/nibbling byte 2 (stroke/punch interface 2)				m	38
SPLINE-PATH	Define spline grouping		Max. eight axes			
SPOF <sup>1</sup>	Stroke OFF, punching, nibbling OFF				m	35
JN	Number of path sections per block (stroke/punch number)	Integer			n	
JP	Length of path section (stroke/punch path)	Integer			m	

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
SPOS	Spindle position			SPOS=10 or SPOS[n]=10	m	
SPOSA	Spindle position across block boundaries			SPOSA=5 or SPOSA[n]=5	m	
SQRT	Square root (arithmetic function)	Real				
SR	Sparking-out retraction path for synchronized action	Real, without sign			n	
SRA	Sparking-out retraction path with external input axial for synchronized action			SRA[Y]=0.2	m	
ST	Sparking-out time for synchronized action	Real, without sign			n	
STA	Sparking-out time axial for synchronized action				m	
START	Start selected programs simultaneously in several channels from current program	Ineffective for the local channel		START(1,1,2) or START(CH_X, CH_Y) \$MC_CHAN_NAME		
STARTFIFO <sup>1</sup>	Execute; simultaneously fill preprocessing memory				m	4
STAT	Position of joints	Integer			n	
STOPFIFO	Stop machining; fill preprocessing memory until STARTFIFO is detected, FIFO full or end of program				m	4
STOPRE	Stop preprocessing until all prepared blocks are executed in main run					
STOPREOF	Stop preprocessing OFF					

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
STRING	Data type: Character string	Max. 200 characters				
STRINGIS	Checks the present scope of NC language and NC cycle names, user variables, macros and label names belonging especially to this command to establish whether these exist, are valid, defined or active	INT	The return value results are 000 not known 100 programmable 2XX recognized as present	STRINGIS (STRING,name)= Digit-coded return value		
STRLEN	Define string length	INT				
SUBSTR	Define index of character in input string	Real	String: 1st parameter, character: 2nd parameter			
SUPA	Suppression of current zero offset, including programmed offsets, system frames, handwheel offsets (DRF), external zero offset and overlaid motion				n	9
SYNFCT	Evaluation of a polynomial as a function of a condition in the motion-synchronous action	VAR REAL				
SYNR	The variable is read synchronously, i.e. at execution time (synchronous read)					
SYNRW	The variable is read and written synchronously, i.e. at execution time (synchronous read-write)					
SYNW	The variable is written synchronously, i.e. at execution time (synchronous write)					

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
T	Call tool (only change if specified in machine data; otherwise M6 command necessary)	1, ..., 32 000	Call using T-no. or tool identifier	For example, T3 or T=3, e.g. T="DRILL"		
TAN	Tangent (trigon. function)	Real				
TANG	Determine tangent for the follow-up from both specified leading axes					
TANGOF	Tangent follow-up mode OFF					
TANGON	Tangent follow-up mode ON					
TCARR	Request toolholder (number "m")	Integer	m=0: deselect active toolholder	TCARR=1		
TCOABS <sup>1</sup>	Determine tool length components from the orientation of the current toolholder		Necessary after reset, e.g. through manual setting		m	42
TCOFR	Determine tool length components from the orientation of the active frame				m	42
TCOFRX	Determine tool orientation of an active frame on selection of tool, tool points in X direction		Tool perpendicular to inclined surface		m	42
TCOFRY	Determine tool orientation of an active frame on selection of tool, tool points in Y direction		Tool perpendicular to inclined surface		m	42
TCOFRZ	Determine tool orientation of an active frame on selection of tool, tool points in Z direction		Tool perpendicular to inclined surface		m	42
THETA	Angle of rotation		THETA is always perpendicular to the current tool orientation	THETA=Value THETA=AC THETA=IC Polynomial for THETA PO[THT]=(...)	n	
TILT	Tilt angle	Real		TILT=Value	m	

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
TMOF	Deselect tool monitoring		T-no. required only when the tool with this number is not active	TMOF (T no.)		
TMON	Activate tool monitoring		T No. = 0: Deactivate monitoring for all tools	TMON (T no.)		
TO	Defines the end value in a FOR counter loop					
TOFF	Tool length offset in the direction of the tool length component that is effective parallel to the geometry axis specified in the index				m	
TOFFL	Tool length offset in the direction of the tool length component L1, L2 or L3				m	
TOFFR	Tool radius offset				m	
TOFFOF	Deactivate on-line tool offset					
TOFFON	Activate online tool length offset <b>(Tool offset ON)</b>		Specify a 3D offset direction	TOFFON (Z, 25) with offset direction Z, offset value 25		
TOFRAME	Set current programmable frame to tool coordinate system		Frame rotation in tool direction		m	53
TOFRAMEX	X axis parallel to tool direction, secondary axis Y, Z				m	53
TOFRAMEY	Y axis parallel to tool direction, secondary axis Z, X				m	53
TOFRAMEZ	Z axis parallel to tool direction, secondary axis X, Y				m	53
TOLOWER	Convert letters of the string into lowercase					
TOROTOF	Frame rotations in tool direction OFF				m	53

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
TOROT	Z axis parallel to tool orientation	Frame rotations ON Rotary component of the programmable frame			m	53
TOROTX	X axis parallel to tool orientation				m	53
TOROTY	Y axis parallel to tool orientation				m	53
TOROTZ	Z axis parallel to tool orientation				m	53
TOUPPER	Convert letters of the string into uppercase					
TOWSTD	Initial setting value for offsets in tool length	Inclusion of tool wear			m	56
TOWBCS	Wear values in basic coordinate system (BCS)				m	56
TOWKCS	Wear values in the coordinate system of the tool head for kinetic transformation (differs from MCS by tool rotation)				m	56
TOWMCS	Wear data in the machine coordinate system (MCS)				m	56
TOWTCS	Wear values in the tool coordinate system (tool carrier ref. point T at the tool holder)				m	56
TOWWCS	Wear values in workpiece coordinate system (WCS)				m	56
TRAANG	Transformation inclined axis	Several transformations can be set for each channel				
TRACEOF	Circularity test: Transfer of values OFF					
TRACEON	Circularity test: Transfer of values ON					
TRACON	Transformation concatenated					
TRACYL	Cylinder: Peripheral surface transformation	See TRAANG				

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
TRAFOOF	Deactivate transformation			TRAFOOF( )		
TRAILOF	Asynchronous coupled motion of axes OFF (trailing OFF)					
TRAILON	Asynchronous coupled motion of axes ON (trailing ON)					
TRANS	Programmable offset (translation)			TRANS X... Y... Z... ;separate block	n	3
TRANSMIT	Polar transformation		See TRAANG			
TRAORI	4-axis, 5-axis transformation, generic transformation (transformation oriented)		Activates the specified orientation transformation	Generic transformation TRAORI(1,X,Y,Z)		
TRUE	Logical constant: True	BOOL	Can be replaced with integer constant 1			
TRUNC	Truncate decimal places	Real				
TU	Axis angle	Integer		TU=2	n	
TURN	Number of turns for helix	0, ..., 999			n	
UNLOCK	Enable synchronized action with ID (continue technology cycle)					
UNTIL	Condition for end of REPEAT loop					
UPATH	Path reference for FGROUP axes is curve parameter				m	45
VAR	Keyword: Type of parameter passing		With VAR: Call by reference			
VELOLIMA	Reduction or overshoot of maximum axial velocity	1, ..., 200	Valid range is 1 to 200%	VELOLIMA[X]= ...[%]	m	

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
WAITC	Wait until coupling block change criterion for axes/spindles is fulfilled (wait for couple condition)		Up to two axes/spindles can be programmed	WAITC(1,1,2)		
WAITE	Wait for end of program on another channel		Channel numbers 1 - 10 or \$MC_CHAN_NAME	WAITE(1,1,2) or WAITE(CH_X, CH_Y)		
WAITM	Wait for marker in specified channel; terminate previous block with exact stop		Channel numbers 1 - 10 or \$MC_CHAN_NAME	WAITM(1,1,2) or WAITM(CH_X, CH_Y)		
WAITMC	Wait for marker in specified channel; exact stop only if the other channels have not yet reached the marker.		Channel numbers 1 - 10 or \$MC_CHAN_NAME	WAITMC(1,1,2) or WAITMC(CH_X, CH_Y)		
WAITP	Wait for end of traversing			WAITP(X);separate block		
WAITS	Waiting to reach spindle position			WAITS (main spindle) WAITS (n,n,n)		
WALCS0	WCS working area limitation deselected				m	60
WALCS1	WCS working area limitation group 1 active				m	60
WALCS2	WCS working area limitation group 2 active				m	60
WALCS3	WCS working area limitation group 3 active				m	60
WALCS4	WCS working area limitation group 4 active				m	60
WALCS5	WCS working area limitation group 5 active				m	60
WALCS6	WCS working area limitation group 6 active				m	60
WALCS7	WCS working area limitation group 7 active				m	60
WALCS8	WCS working area limitation group 8 active				m	60
WALCS9	WCS working area limitation group 9 active				m	60
WALCS10	WCS working area limitation group 10 active				m	60

*Tables*

*16.1 Statements A - G*

Name	Meaning	Value	Description, comment	Syntax	m/n <sup>3</sup>	Group <sup>2</sup>
WALIMOF	BCS working area limitation OFF			;separate block	m	28
WALIMON <sup>1</sup>	BCS working area limitation ON			;separate block	m	28
WHILE	Start of WHILE program loop		End: ENDWHILE			
WRITE	Write block in file system Appends a block to the end of the specified file		The blocks are inserted after M30			
X	Axis	Real			m/n	
XOR	Logical exclusive OR					
Y	Axis	Real			m/n	
Z	Axis	Real			m/n	

**Legend:**

- <sup>1</sup> Default setting at beginning of program (factory settings of the control, if nothing else programmed)
- <sup>2</sup> The groups are numbered according to the table in section "List of G functions/preparatory functions".
- <sup>3</sup> Absolute end points: modal (m)  
Incremental end points: non-modal (n)  
Otherwise: modal/non-modal depending on syntax of G function
- <sup>4</sup> As circle center points, IPO parameters act incrementally. They can be programmed in absolute mode with AC. The address modification is ignored when the parameters have other meanings (e.g. thread lead).
- <sup>5</sup> The OEM can add two extra interpolation types. The names can be changed by the OEM.
- <sup>6</sup> Extended address notation cannot be used for these functions.

# A

## Appendix

### A.1 List of abbreviations

A	Output
AS	Automation system
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit: User switching circuit
ASUB	Asynchronous subroutine
AV	Job planning
STL	Statement list
BA	Operating mode
Mode group	Mode group
BB	Ready to run
HMI	Human Machine Interface
BCD	Binary Coded Decimals: Decimal numbers encoded In binary code
HHU	Handheld unit
BIN	Binary files ( <b>Binary Files</b> )
BIOS	Basic Input Output System
BCS	Basic Coordinate System
UI	User interface
BOT	Boot files: Boot files for SIMODRIVE 611 digital
OP	Operator Panel
OPI	Operator Panel Interface
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CNC	Computerized Numerical Control: Computerized numerical control
COM	Communication
CP	Communications Processor
CPU	Central Processing Unit: Central processing unit
CR	Carriage Return

## Appendix

### A.1 List of abbreviations

CRT	Cathode Ray Tube picture tube
CSB	Central Service Board: PLC module
CTS	Clear To Send: Signal from serial data interfaces
CUTOM	Cutter radius compensation: Tool radius compensation
DAC	Digital-to-Analog Converter
DB	Data block in the PLC
DBB	Data block byte in the PLC
DBW	Data block word in the PLC
DBX	Data block bit in the PLC
DC	Direct Control: Movement of the rotary axis via the shortest path to the absolute position within one revolution
DCD	Data Carrier Detect
DDE	Dynamic Data Exchange
DTE	Data Terminal Equipment
DIN	Deutsche Industrie Norm (German Industry Standard)
DIO	Data Input/Output: Data transfer display
DIR	Directory: Directory
DLL	Dynamic Link Library
DOE	Data transmission equipment
DOS	Disk Operating System
DPM	Dual-Port Memory
DPR	Dual-Port RAM
DRAM	Dynamic Random Access Memory
DRF	Differential Resolver Function: Differential resolver function (DRF)
DRY	Dry Run: Dry run feedrate
DSB	Decoding Single Block: Decoding single block
DW	Data word
E	Input
I/O	Input/Output
I/R	Infeed/regenerative-feedback unit (power supply) of the SIMODRIVE 611digital
EIA code	Special punched tape code, number of holes per character always odd
ENC	Encoder: Actual value encoder
EPROM	Erasable Programmable Read Only Memory
Error	Error from printer
FB	Function block

FBS	Slimline screen
FC	Function Call: Function block in the PLC
FDB	Product database
FDD	Floppy Disk Drive
FEPROM	Flash-EPROM: Read and write memory
FIFO	First In First Out: Memory that works without address specification and whose data are read in the same order in which they were stored.
FIPO	Fine Interpolator
FM	Function Module
FPU	Floating Point Unit Floating Point Unit
FRA	Frame block
FRAME	Data record (frame)
CRC	Cutter radius compensation
FST	Feed Stop: Feed stop
CSF	Function plan (PLC programming method)
BP	Basic program
GUD	Global User Data: Global user data
HD	Hard Disk Hard disk
HEX	Abbreviation for hexadecimal number
AuxF	Auxiliary function
HMI	Human Machine Interface: Operator functionality of SINUMERIK for operation, programming and simulation.
HMS	High-resolution Measuring System
MSD	Main Spindle Drive
HW	Hardware
IBN	Startup
IF	Drive module pulse enable
IK (GD)	Implicit communication (global data)
IKA	Interpolative Compensation: Interpolatory compensation
IM	Interface Module Interconnection module
IMR	Interface Module Receive: Interconnection module for receiving data
IMS	Interface Module Send: Interconnection module for sending data
INC	Increment: Increment
INI	Initializing Data: Initializing data
IPO	Interpolator
ISA	Industry Standard Architecture
ISO	International Standardization Organization
ISO code	Special punched tape code, number of holes per character always even
JOG	Jogging: Setup mode

## Appendix

### A.1 List of abbreviations

K1 .. K4	Channel 1 to channel 4
C Bus	Communication bus
COR	Coordinate rotation
LAD	Ladder diagram (PLC programming method)
K <sub>v</sub>	Servo gain factor
K <sub>UE</sub>	Speed ratio
LCD	Liquid Crystal Display: Liquid crystal display
LED	Light-Emitting Diode: Light emitting diode
LF	Line Feed
PMS	Position measuring system
LR	Position controller
LUD	Local User Data
MB	Megabyte
MD	Machine data
MDI	Manual Data Automatic: Manual input
MC	Measuring circuit
MCS	Machine coordinate system
MLFB	Machine-readable product designation
MPF	Main Program File: NC part program (main program)
MPI	Multiport Interface Multiport Interface
MS	Microsoft (software manufacturer)
MCP	Machine control panel
NC	Numerical Control: Numerical Control
NCK	Numerical Control Kernel: NC kernel with block preparation, traversing range, etc.
NCU	Numerical Control Unit: Hardware unit of the NCK
NRK	Name for the operating system of the NCK
IS	Interface signal
NURBS	Non-Uniform Rational B-Spline
ZO	Zero offset
OB	Organization block in the PLC
OEM	Original Equipment Manufacturer
OP	Operator Panel: Operating setup
OPI	Operator Panel Interface: Interface for connection to the operator panel
OPT	Options: Options
OSI	Open Systems Interconnection: Standard for computer communications
P bus	Peripheral Bus
PC	Personal Computer

PCIN	Name of the SW for data exchange with the control
PCMCIA	Personal Computer Memory Card International Association: Standard for plug-in memory cards
PCU	PC Unit: PC box (computer unit)
PG	Programming device
PLC	Programmable Logic Control: Interface control
POS	Positioning
RAM	Random Access Memory: Program memory that can be read and written to
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
RPA	R-Parameter Active: Memory area on the NCK for R parameter numbers
RPY	Roll Pitch Yaw: Rotation type of a coordinate system
RTS	Request To Send: RTS, control signal of serial data interfaces
SBL	Single Block: Single block
SD	Setting Data
SDB	System Data Block
SEA	Setting Data Active: Identifier (file type) for setting data
SFB	System Function Block
SFC	System Function Call
SK	Softkey
SKP	SKiP: Skip block
SM	Stepper Motor
SPF	Sub Routine File: Subroutine
PLC	Programmable Logic Controller
SRAM	Static RAM (non-volatile)
TNRC	Tool Nose Radius Compensation
LEC	Leadscrew error compensation
SSI	Serial Synchronous Interface: Synchronous serial interface
SW	Software
SYF	System Files System files
TEA	Testing Data Active: Identifier for machine data
TO	Tool Offset: Tool offset
TOA	Tool Offset Active: Identifier (file type) for tool offsets

## *Appendix*

### *A.1 List of abbreviations*

TRANSMIT	TRANSform Milling Into Turning: Coordinate conversion on turning machine for milling operations
UFR	User Frame: Zero offset
SR	Subroutine
FDD	Feed Drive
RS-232-C	Serial interface (definition of the exchange lines between DTE and DCE)
WCS	Workpiece coordinate system
T	Tool
TLC	Tool length compensation
WOP	Workshop-oriented Programming
WPD	Workpiece Directory: Workpiece directory
TRC	Tool Radius Compensation
TO	Tool offset
TC	Tool change
ZOA	Zero Offset Active: Identifier (file type) for zero offset data
μC	Micro Controller

## **A.2 Feedback on the documentation**

This document will be continuously improved with regard to its quality and ease of use.  
Please help us with this task by sending your comments and suggestions for improvement  
via e-mail or fax to:

E-mail: <mailto:docu.motioncontrol@siemens.com>

Fax: +49 (0) 9131/98 - 63315

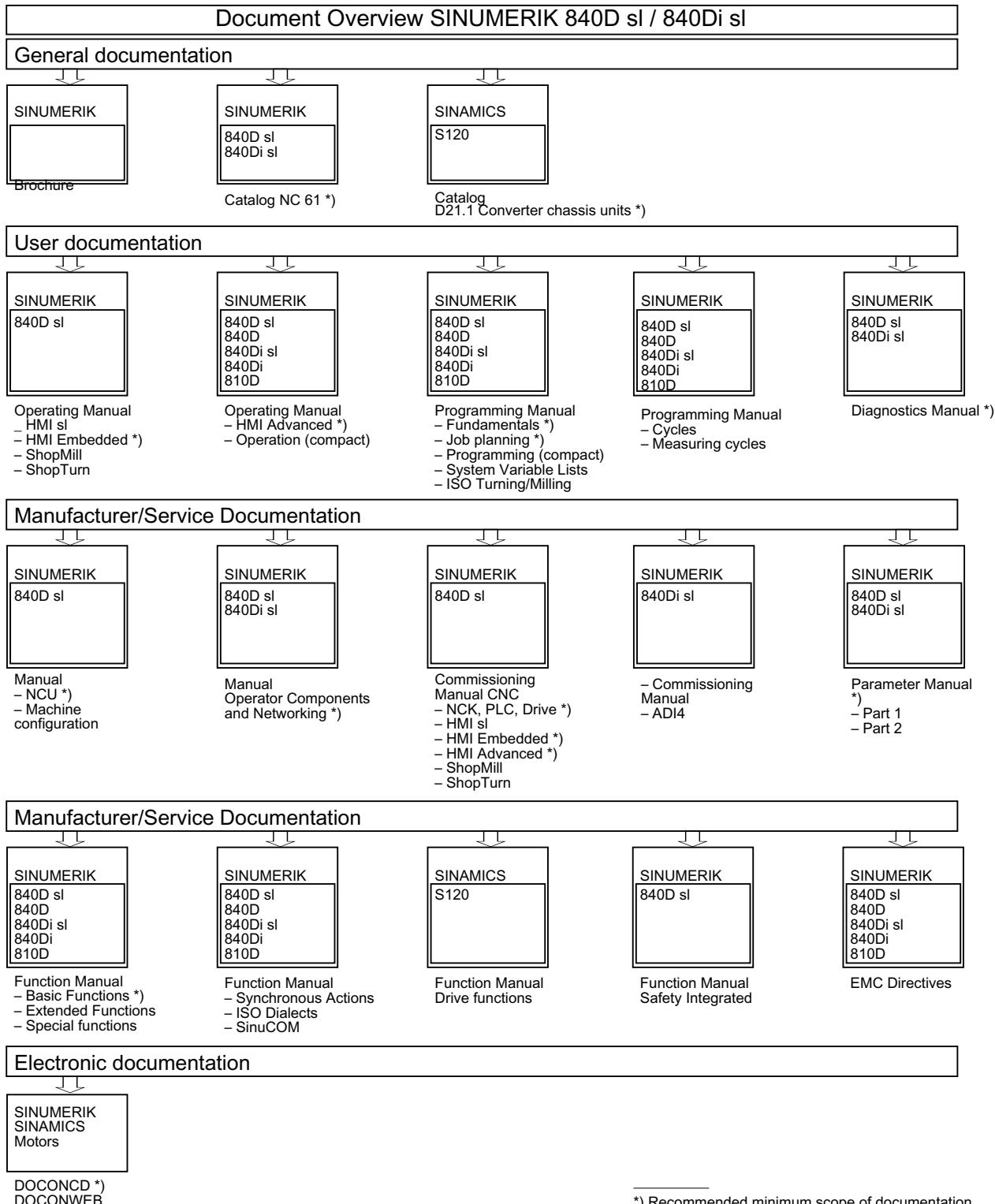
Please use the fax form on the back of this page.

*Appendix*

*A.2 Feedback on the documentation*

To: <b>SIEMENS AG</b> A&D MC MS1 Postfach 3180  <b>91050 ERLANGEN, GERMANY</b>  Fax: +49 (0) 9131/98 - 63315 (documentation)	From  Name:  Address of your Company/Dept.  Street:  Zip code:      City:  Phone:      /  Fax:      /
---	---

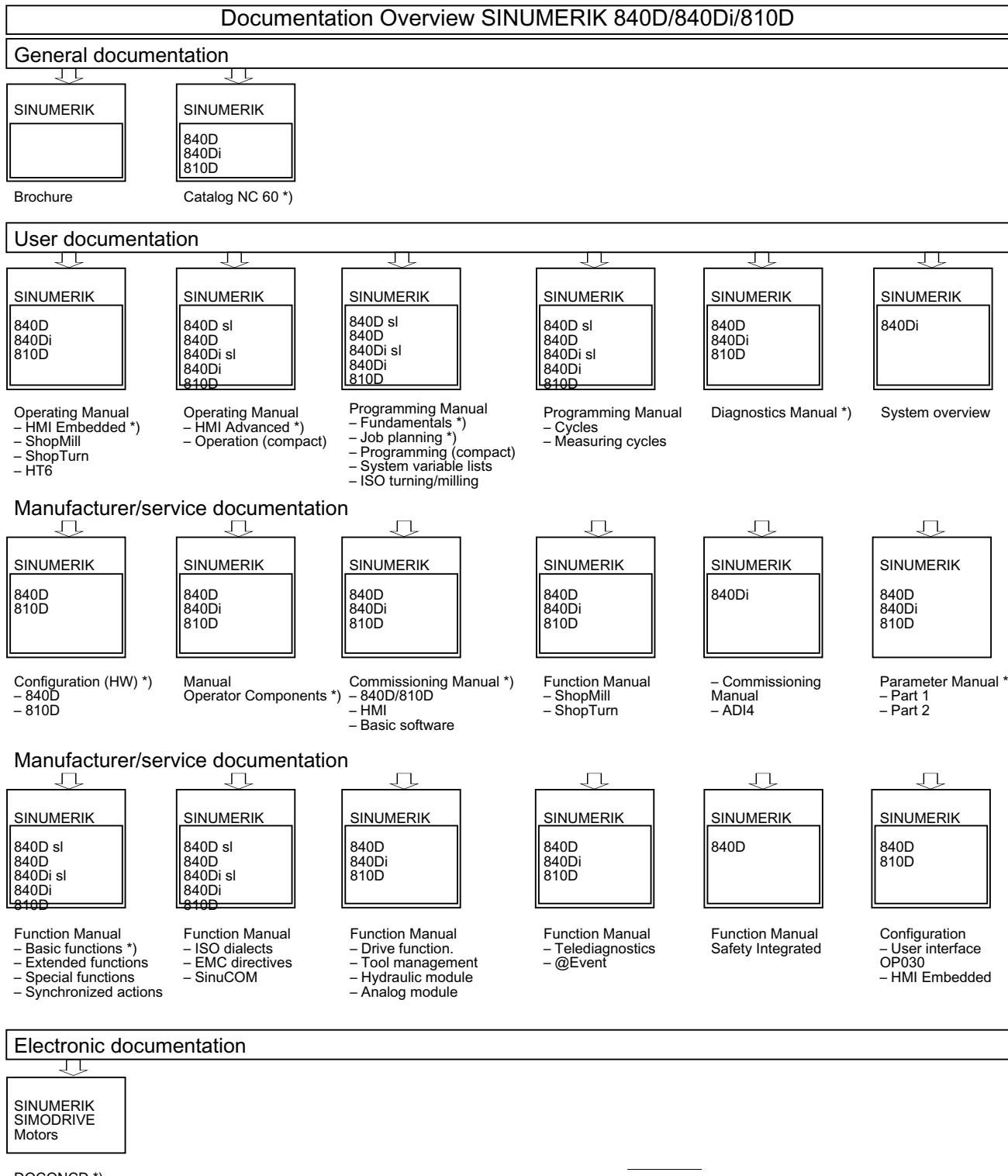
## A.3 Overview



\*) Recommended minimum scope of documentation

## Appendix

### A.3 Overview



\*) Recommended minimum scope of documentation

# Glossary

## Absolute dimensions

A destination for an axis movement is defined by a dimension that refers to the origin of the currently active coordinate system. See → Chain measure

## 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 cancel 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 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.

## Archive

Reading out of files and/or directories on an **external** memory device.

### **Asynchronous subroutine**

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 control (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 feed rate.  
Auxiliary axes are not involved in actual machining, e.g. tool feeder, tool magazine.

### **Axis address**

See → Axis identifier

### **Axis identifier**

Axes are identified using X, Y, and Z as defined in DIN 66217 for a right-handed, right-angled → coordinate system.

Rotary axes rotating around X, Y, and Z are identified using A, B, and C. Additional axes situated parallel to the specified axes can be designated using other letters.

### **Axis name**

See → Axis identifier

**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.

**Base 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 between the two coordinate systems lies in the → axis identifiers.

**Baud rate**

Rate of data transfer (Bit/s).

**Blank**

Workpiece as it is before it is machined.

**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 movement.

**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.

**Chip**

"Block" is the term given to any files required for creating and processing programs.

**Circular interpolation**

The → tool moves on a circle between specified points on the contour at a given feed rate, and the workpiece is thereby machined.

**CNC**

See → NC

**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 memory**

Data range in the control, in which the tool offset data are stored.

**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.

**Connecting cables**

Connecting cables are pre-assembled or user-assembled 2-wire cables with a connector at each end. This connecting cable connects the → CPU to a → programming device or to other CPUs by means of a → multi-point interface (MPI).

**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 → Memory-programmable control

**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. 3rd order polynomials are used.

**Cycles**

Protected subroutines 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 modules contain data definitions for global user data. These data can be initialized directly when they are defined.

**Data word**

Two-byte data unit within a → data block.

**Diagnose**

1. Operating area of the control.
2. The control has both a self-diagnostics program as well as test functions for servicing purposes: status, alarm, and service displays

**Dimensions specification, metric and inches**

Position and lead values can be programmed in inches in the machining program. Irrespective of the programmable dimensions (G70/G71), the controller is set to a basic system.

**DRF**

Differential Resolver Function: NC function which generates an incremental zero 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 zero offset**

Zero-point offset specified by the → PLC.

**Fast retraction from 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. After fast retraction, an interrupt routine can also be executed (SINUMERIK 840D).

**Feed override**

The programmed velocity is overridden 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 (1-200%) 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: → zero offset, → rotation, → scaling, → mirroring.

**Geometry**

Description of a → workpiece in the → workpiece coordinate system.

**Geometry axis**

Geometry axes are used to describe a 2- or 3-dimensional area in the workpiece coordinate system.

**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 movements:

- Circular movement in one plane
- A linear movement perpendicular to this plane

**High-level CNC language**

The high-level language offers: → user-defined variables, → system variables, → macro techniques.

**High-speed digital inputs/outputs**

The digital inputs can be used for example to start fast CNC program routines (interrupt routines). The digital CNC outputs can be used to trigger fast, program-controlled switching functions (SINUMERIK 840D).

## **HIGHSTEP**

Summary of programming options for → PLCs of the AS300/AS400 system.

### **Identifier**

In accordance with DIN 66025, words are supplemented using identifiers (names) for variables (arithmetic variables, system variables, user variables), subroutines, 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**

Also 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 control 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**

Interpolatory compensation is a tool that enables manufacturing-related leadscrew error and measuring system error compensations.

**Interrupt routine**

Interrupt routines are special → subroutines 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**

With SINUMERIK 840D, the time required for the path of a block to be traversed can be programmed for the axis motion instead of the feed velocity (G93).

**JOG**

Control operating mode (setup mode): In JOG mode, the machine can be set up. 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 control. 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 control 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 → RAM 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 prefixed by ":" introductory block, containing all the parameters required to start execution of a -> part program.

## **Main program**

The → part program designated by a number or an identifier in which additional main programs, subroutines, or → cycles can be called.

## **MDA**

Control operating mode: Manual Data Automatic. In the MDA mode, individual program blocks or block sequences with no reference to a main program or subroutine 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 cancel 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 group**

Axes and spindles that are technologically related can be combined into one mode group. Axes/spindles of a BAG can be controlled by one or more → channels. The same → mode type is always assigned to the channels of the mode group.

## **Mode of operation**

An operating concept on a SINUMERIK control. The following modes are defined: → Jog, → MDA, → Automatic.

**NC**

Numerical Control: Numerical control (NC) includes all components of machine tool control:  
→ NCK, → PLC, HMI, → COM.

---

**Note**

A more correct term for SINUMERIK 840D controls would be: Computerized Numerical Control

---

**NCK**

Numerical Control Kernel: Component of NC that executes the → part programs and basically coordinates the motion operations for the machine tool.

**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 (Non Uniform Rational B-Splines). As a result, a uniform process is available within the control for all interpolations for SINUMERIK 840D.

**OEM**

The scope for implementing individual solutions (OEM applications) for the SINUMERIK 840D has been provided for machine manufacturers, who wish to create their own operator interface or integrate process-oriented functions in the control.

**Operator Interface**

The user interface (UI) is the display medium for a CNC in the form of a screen. It features horizontal and vertical softkeys.

**Oriented spindle stop**

Stops the workpiece spindle in a specified angular position, e.g. in order to perform additional machining at a particular location.

**Oriented tool retraction**

**RETOOL:** If machining is interrupted (e.g. when a tool breaks), a program command can be used to retract the tool in a user-specified orientation by a defined distance.

**Overall reset**

In the event of an overall reset, the following memories of the → CPU are deleted:

- → Work memory
- Read/write area of → load memory
- → System memory
- → Backup memory

**Override**

Manual or programmable control feature, which enables the user to override programmed feedrates or speeds in order to adapt them to a specific workpiece or material.

**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 feed affects → path axes. It represents the geometric sum of the feed rates 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 an auxiliary program for sending and receiving CNC user data (e.g. part programs, tool offsets, etc.) via a serial interface. The PCIN program can run in 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 Logic Control:** → Programmable logic controller. Component of → NC: Programmable controller for processing the control logic of the machine tool.

### **PLC program memory**

SINUMERIK 840D: 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).

### **Positioning axis**

Axis that performs an auxiliary movement 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 subroutines of → part programs.

**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 Control**

Programmable logic controllers (PLC) are electronic controls, 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 control. The programmable logic controller 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**

Character 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 rate of an axis. For example, rapid traverse is used 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 element.

**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".

**Safety Functions**

The control 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.

**Selecting**

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.

**Serial RS-232-C interface**

For data input/output, the PCU 20 has one serial V.24 interface (RS232) while the PCU 50/70 has two V.24 interfaces. Machining programs and manufacturer and user data can be loaded and saved via these interfaces.

**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 soft keys displayed is dynamically adapted to the operating situation. The freely assignable function keys (soft keys) 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.

**SRT**

Transformation ratio

## **Standard cycles**

Standard cycles are provided for machining operations, which are frequently repeated:

- Cycles for drilling/milling applications
- 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.

## **Subroutine**

Sequence of statements of a → part program that can be called repeatedly with different defining parameters. The subroutine is called from a main program. Every subroutine can be protected against unauthorized read-out and display. → Cycles are a form of subroutines.

## **Supply System**

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.

## **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 times, counters, markers

**System variables**

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 reach 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 control 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.

**Traversing range**

The maximum permissible travel range for linear axes is  $\pm 9$  decades. The absolute value depends on the selected input and position control resolution and the unit of measurement (inch or metric).

**User memory**

All programs and data, such as part programs, subroutines, comments, tool offsets, and zero 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.

**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**

RAM is a work memory 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.

## **Zero offset**

Specifies a new reference point for a coordinate system through reference to an existing zero point and a → frame.

### 1. Settable

SINUMERIK 840D: A configurable number of settable zero offsets are available for each CNC axis. The offsets - which are selected by means of G functions - take effect alternately.

### 2. External

In addition to all the offsets which define the position of the workpiece zero, an external zero offset can be overridden by means of the handwheel (DRF offset) or from the PLC.

### 3. Programmable

Zero offsets can be programmed for all path and positioning axes using the TRANS statement.



# Index

\$

\$AA\_COUP\_ACT, 518, 546, 567  
\$AA\_COUP\_OFFSETS, 568  
\$AA\_LEAD\_SP, 545  
\$AA\_LEAD\_SV, 545  
\$AA\_MOTEND, 283  
\$AA\_TOFF[], 620  
\$AC\_ACT\_PROG\_NET\_TIME, 741  
\$AC\_ACTUAL\_PARTS, 744  
\$AC\_BLOCKTYPE, 599  
\$AC\_BLOCKTYPEINFO, 599  
\$AC\_CUT\_INV, 475  
\$AC\_CUTMOD, 475  
\$AC\_CUTMOD\_ANG, 475  
\$AC\_CUTTING\_TIME, 741  
\$AC\_CYCLE\_TIME, 741  
\$AC\_FIFO1, 596  
\$AC\_MARKER, 591  
\$AC\_OLD\_PROG\_NET\_TIME, 741  
\$AC\_OLD\_PROG\_NET\_TIME\_COUNT, 741  
\$AC\_OPERATING\_TIME, 741  
\$AC\_PARAM, 592  
\$AC\_PROG\_NET\_TIME\_TRIGGER, 742  
\$AC\_REQUIRED\_PARTS, 744  
\$AC\_SPECIAL\_PARTS, 744  
\$AC\_SPLITBLOCK, 599  
\$AC\_TIMER, 595  
\$AC\_TOTAL\_PARTS, 744  
\$AN\_POWERON\_TIME, 741  
\$AN\_SETUP\_TIME, 741  
\$MC\_COMPRESS\_VELO\_TOL, 491  
\$P\_AD, 476  
\$P\_CUT\_INV, 475  
\$P\_CUTMOD, 475  
\$P\_CUTMOD\_ANG, 475  
\$P\_TECCYCLE, 655  
\$R, 593  
\$Rn, 593  
\$SA\_LEAD\_TYPE, 545  
\$TC\_CARR1...14, 458  
\$TC\_CARR18[m], 459, 463  
\$TC\_DP1, 408  
\$TC\_DP10, 408  
\$TC\_DP11, 408  
\$TC\_DP12, 408  
\$TC\_DP13, 408  
\$TC\_DP14, 408  
\$TC\_DP15, 408  
\$TC\_DP16, 408  
\$TC\_DP17, 408  
\$TC\_DP18, 408  
\$TC\_DP19, 408  
\$TC\_DP2, 408  
\$TC\_DP20, 408  
\$TC\_DP21, 408  
\$TC\_DP22, 408  
\$TC\_DP23, 408  
\$TC\_DP24, 408  
\$TC\_DP25, 408  
\$TC\_DP3, 408  
\$TC\_DP4, 408  
\$TC\_DP5, 408  
\$TC\_DP6, 408  
\$TC\_DP7, 408  
\$TC\_DP8, 408  
\$TC\_DP9, 408  
\$TC\_ECPxy, 412  
\$TC SCPxy, 412  
\$TC\_TPG1 ... 9, 697, 698

\*

\* (arithmetic function), 46

/

/ (arithmetic function), 46

+

+ (arithmetic function), 46

<

< (comparison operator), 48

<< (concatenation operator), 60

- <= (relational operator), 48  
<> (comparison operator), 48
- =  
== (comparison operator), 48
- >  
> (comparison operator), 48  
>= (relational operator), 48
- 0**  
0 character, 57  
3D circumferential milling with limitation surfaces, 442  
3D face milling, 342  
Path curve using surface normal vectors, 342  
3D tool offset  
Circumferential milling with limitation surfaces, 443  
Compensation along the path, 438  
Insertion depth, 439  
Intersection procedure, 441  
Milling tool shapes, 436  
Path curvature, 438  
Tool data, 436  
Tool orientation, 447  
3D tool radius compensation  
3D intersection point of the equidistance, 441  
Behavior at outer corners, 452  
Circumferential milling, 434  
Face milling, 435  
Outside/inside corners, 440  
Transition circle, 441  
3D tool radius compensation, 432
- A**  
A spline, 246  
A1, A2, 459  
A2, 336  
A3, 336  
A4, 336, 342  
A5, 336, 342  
ABS, 46  
ACC, 565  
ACOS, 46  
Acquiring and finding untraceable sections, 500  
ACTBLOCNO, 180  
ACTFRAME, 290
- Actual value coupling, 557  
Adaptive control, additive, 613  
Adaptive control, multiplicative, 614  
Addresses  
Indirect programming, 36  
ADISPOSA, 280  
Alarm, 746  
-number, 746  
ALF, 107, 109  
AND, 48  
and after motion, 687  
ANG, 771  
Angle of rotation, 359  
Angle of rotation 1, 2, 459  
Angle offset/angle increment of the rotary axes, 461  
Angle reference, 563  
applim, 520  
Approach from the nearest path point, 509  
APR, 204, 208  
APW, 204, 208, 209  
APX, 209  
Arithmetic parameters  
-number n, 18  
Arithmetic parameters (R), 18  
Array, 28  
-element, 28  
Array definition, 28  
Array index, 31  
AS, 188  
ASIN, 46  
ASPLINE, 238  
Assignments, 45  
Asynchronous oscillation, 669  
ATAN2, 46  
Automatic "GET", 117  
Automatic interrupt pointer, 501  
Automatic path segmentation, 689  
Auxiliary functions, 605, 689  
AV, 563  
AX, 699  
AXCTSWE, 710  
AXCTSWED, 710  
Axial feed, 632  
Axial master value coupling, 540  
Axis  
Clamping, 710  
Directly accept, 113  
Following, 532  
Leading, 532  
Local, 712  
-replacement, 113  
AXIS, 22

- Axis container, 710  
 Axis coordination, 633  
 Axis positioning  
     Specified reference position, 624  
 Axis replacement, 118  
     Accept axis, 117  
     Get and release using synchronized actions, 627  
     Preconditions, 116  
     Release axis, 116  
     Set up variable response, 117  
     without preprocessing stop, 117  
     Without synchronization, 115  
 AXNAME, 59, 699  
 AXSTRING, 699  
 AXTCHAN, 118  
 AXTOSPI, 699  
 AXVAL, 46
- B**
- B spline, 247  
 B\_AND, 48  
 B\_NOT, 48  
 B\_OR, 48  
 B\_XOR, 48  
 B2, 336  
 B3, 336  
 B4, 336, 342  
 B5, 336, 342  
 Backlash, 737  
 BAUTO, 238  
 BFRAME, 287  
 BLOCK, 164  
 Block display, 166  
     suppress, 180  
 BNAT, 238  
 BOOL, 22  
 BOUND, 46, 53  
 BP, 41  
 BSPLINE, 238  
 BTAN, 238
- C**
- C spline, 248  
 C2, 336  
 C3, 336  
 C4, 336, 342  
 C5, 336, 342  
 CAC, 237  
 CACN, 237
- CACP, 237  
 CALCDAT, 765  
 CALL, 163, 164  
 Call by value parameters for technology cycles, 656  
 CALLPATH, 168, 196  
 CANCEL, 663  
 Cartesian PTP travel, 323  
 CASE, 73  
 CDC, 237  
 CFINE, 300  
 CHANDATA, 198  
 Channel-specific frames, 310  
 CHAR, 22  
 Check structures, 83  
 CHECKSUM, 132  
 CHKDNO, 455  
 CIC, 237  
 Circle data  
     calculating, 765  
 Circumferential milling, 434  
 Circumferential milling (3D)  
     with limitation surfaces, 443  
 Clearance control, 615  
 CLEARM, 93, 643  
 CLRINT, 105  
 CMIRROR, 46, 293  
 Coarse offset, 300  
 COARSE50, 563, 566  
 COARSEA, 280  
 COMCAD, 252  
 Command axes, 622  
 COMPCAD, 369  
 COMPCURV, 252, 369  
 COMPLETE, 199  
 Complete basic frame, 312, 313  
 COMPOF, 252, 369  
 COMPON, 252, 369, 490  
 Compressor, 252, 264  
 Computing capacity, 707  
 Concatenation  
     of strings, 60  
 Constraints for transformations, 402  
 CONTDCON, 757  
 Contour  
     -coding, 757  
     -preparation, 750  
     Repositioning, 502  
     -table, 750, 757  
 Contour element  
     travel, 764  
 Contour preparation  
     Fault feedback signal, 767

CONPRON, 750  
Conversion routines, 586  
Corner deceleration at all corners, 279  
Corner deceleration at inside corners, 279  
COS, 46  
Count loop, 88  
COUPDEF, 555, 561, 562  
COUPDEL, 555, 561  
Coupled motion, 515, 636  
    Coupled-motion axes, 518  
    Coupling factor, 518  
Coupled-axis combinations, 515  
Coupling, 477, 515  
COUPOF, 555, 566  
COUPOFS, 555, 566  
COUPON, 555, 563  
COUPONC, 555  
COUPRES, 555, 567  
CP, 391  
CPROT, 224  
CPROTDEF, 221, 223  
CROT, 46, 293  
CS, 477  
CSCALE, 46, 293  
CSPLINE, 238  
CTAB, 536, 539  
CTABDEF, 520, 525  
CTABDEL, 520, 527  
CTABEND, 520, 525  
CTABEXISTS, 527  
CTABFNO, 527  
CTABFPOL, 527  
CTABFSEG, 527  
CTABID, 527  
CTABINV, 536, 539  
CTABISLOCK, 527  
CTABLOCK, 527  
CTABMEMTYP, 527  
CTABMPOL, 527  
CTABMSEG, 527  
CTABNOMEM, 527  
CTABPERIOD, 527  
CTABPOLID, 527  
CTABSEG, 527  
CTABSEGID, 527  
CTABSEV, 536  
CTABSSV, 536  
CTABTEP, 532  
CTABTEV, 532  
CTABTMAX, 532  
CTABTMIN, 532  
CTABTSP, 532  
CTABTSV, 532  
CTABUNLOCK, 527  
CTRANS, 46, 293, 300  
Current  
    Angular offset, 568  
    Coupling status following spindle, 567  
Current channel basic frames, 312  
Current first basic frame in the channel, 312  
Current NCU-global basic frames, 311  
Current programmable frame, 313  
Current settable frame, 313  
Current system frames, 311  
Current total frame, 314  
Curve table  
    Number, 532  
Curve tables, 520, 527  
    Non-periodic curve table, 535  
    Periodic curve table, 535  
    Read in synchronized actions, 536  
CUT3DC, 432, 438  
CUT3DCC, 443  
CUT3DCCD, 443  
CUT3DF, 432  
CUT3DFF, 432  
CUT3DFS, 432  
CUTMOD, 471  
Cutting edge number, 454  
Cycle alarms, 746  
Cycles  
    Setting parameters for user cycles, 183  
    User cycles and manufacturer cycles with NC  
        programs of the same name, 183  
Cylinder surface curve transformation, 376, 377  
    Offset contour normal OFFN, 384  
Cylinder surface transformation, 322

## D

D number  
    Freely assigned, 454  
D numbers  
    Check, 455  
    Renaming, 455  
DC link backup, 726  
Deactivation position, 566  
DEF, 28, 200, 651  
DEFAULT, 73  
Default axis identifier, 590  
DEFINE, 651  
DEFINE ... AS, 188  
degrees, 536  
DELAYFSTOF, 493

DELAYFSTON, 493  
 DELDL, 413  
 DELDTG, 607  
 DELETE, 124  
 Delete couplings, 566, 567  
 Delete distance-to-go with preparation, 607  
 Deletion of distance-to-go, 607, 671  
 DELT, 423  
 Denominator polynomial, 260  
 Deselecting a transformation  
     TRAFOOF, 403  
 DISABLE, 104  
 DISPLOF, 180  
 DISPR, 502  
 DIV, 46  
 DL, 410  
 DO, 581, 676  
 Drive-independent retraction, 727  
 Drive-independent stopping, 726  
 DUPLO\_NO, 423  
 DV, 563

**E**  
 EAUTO, 238  
 EG  
     Electronic gear, 546  
 EGDEF, 547  
 EGDEL, 553  
 EGOFC, 552  
 EGOFS, 552  
 EGON, 548  
 EGONSYN, 548  
 EGONSYNE, 548  
 Electronic gear, 546  
 ELSE, 85  
 ENABLE, 104  
 ENAT, 238  
 End angle, 359  
 ENDFOR, 88  
 ENDIF, 85  
 Endless loop, 87  
 ENDLOOP, 87  
 End-of-motion criterion  
     Programmable, 280  
 Endpos, 676  
 ENDPROC, 616  
 ENDWHILE, 90  
 ETAN, 238  
 EVERY, 578  
 EXECSTRING, 44  
 EXECTAB, 764

EXECUTE, 221, 223, 767  
 EXP, 46  
 EXTCALL, 170  
 Extended measuring function, 268, 391  
 Extended stop and retract, 715  
 EXTERN, 145  
 External zero offset, 302

**F**  
 F word polynomial, 262  
 F0.1, 93  
 F10, 221  
 F3, 737  
 FA, 565, 632, 673  
 Face milling, 432, 435  
 FALSE, 16  
 Fast retraction from contour, 106  
 Faxis, 477, 515, 520, 536, 540  
 FCTDEF, 426, 609  
 FCUB, 485  
 Feed  
     Axial, 632  
     Axis, 677  
     Movement, 683  
 FENDNORM, 279  
 FGROUP axes, 262  
 FIFO variables, 596  
 File  
     -information, 129  
 FILEDATE, 129  
 FILEINFO, 129  
 FILESIZE, 129  
 FILESTAT, 129  
 FILETIME, 129  
 FINE, 563  
 Fine offset, 300  
 FINEA, 280  
 First basic frame in the channel, 310  
 Fixed stop, 645  
 FLIN, 485  
 FMA, 787  
 FNORM, 485  
 FOCOF, 645  
 FOCON, 645  
 Following axis, 540  
 FOR, 88  
 FPO, 485  
 FPR, 553  
 Frame  
     Call, 297  
     Frame chaining, 315

- FRAME, 22  
Frame calculation  
  MEAFRAME, 304  
Frame component  
  FI, 296  
  MI, 296  
  SC, 296  
  TR, 296  
Frame component RT, 296  
Frame variable  
  Assignments to G commands G54 to G599, 292  
  Predefined frame variable, 287, 297  
  Zero offsets G54 to G599, 292  
Frame variables, 285  
  Assigning values, 293  
  Calling coordinate transformations, 285  
  Defining new frames, 299  
Frames  
  Assign, 298  
  Frame chains, 299  
FRC, 786, 788  
FRCM, 788  
Friction, 737  
FROM, 578  
FTOCOF, 426  
FTOCON, 426  
FXS, 645  
FXST, 645  
FXSW, 645
- G**
- G code, 262  
G codes  
  Indirect programming, 39  
G01, 93  
G05, 390  
G07, 390  
G1, 672  
G4, 670  
G40, 432  
G450, 440  
G451, 440  
G62, 279  
G621, 279  
G64, 137  
G643, 263  
Generator operation, 726  
GEOAX, 702  
Geometry axis  
  Switching over, 702  
GET, 113  
GETACTTD, 457  
GETD, 113  
GETDNO, 455  
GETSELT, 423  
GETT50, 423  
GOTO, 70  
GOTOB, 70  
GOTOC, 70  
GOTOF, 70  
GOTOS, 69  
GUD, 22, 200, 204  
  Automatic activation, 207  
  definition, 203  
GUD variable for synchronous actions  
  User-defined GUD variables, 587
- H**
- Hold block, 500
- I**
- I1,I2, 458  
ID, 576  
Identification number, 576  
IDS, 576  
IF, 70, 85  
IFRAME, 288  
II1,II2, 679  
Inclined axis transformation, 386  
Inclined axis, TRAANG, 322, 386  
Independent drive reactions, 717  
INDEX, 64  
Indirect programming, 41  
  of addresses, 36  
  of G codes, 39  
INICF, 212  
INIPO, 212  
INIRE, 212  
INIT, 93  
INITIAL, 199  
Initial tool orientation setting ORIRESET, 333  
INITIAL\_INI, 199  
Initialization  
  of array variables, 642  
  of arrays, 28  
Initialization program, 197  
Insertion depth, 439  
Insertion depth (ISD), 432  
INT, 22  
Interpolation cycle, 709

Interpolation of the rotation vector, 358, 365  
 Interrupt routine, 100  
     Assign and start, 102  
     Deactivating/activating, 104  
     Delete, 105  
     Fast retraction from contour, 106  
     Newly assign, 103  
     Programmable traverse direction, 109  
     Programmable traverse direction, 107  
     Retraction movement, 108  
     Save modal G functions, 101  
**INTERSEC**, 762  
**IPOBRKA**, 280  
**IPOENDA**, 280  
**IPOSTOP**, 557, 563, 566  
**IPTRLOCK**, 499  
**IPTRUNLOCK**, 499  
**ISAXIS**, 699  
**ISD**, 432, 438  
**ISFILE**, 128  
**ISNUMBER**, 59  
**ISOCALL**, 166  
**ISVAR**, 734

**J**

**Jerk**  
     offset, 512  
**JERKLIM**, 512  
**Jump**  
     -condition, 71  
     -destination, 70  
     -instructions, 71  
     -marker, 71  
**Jump statement**  
     CASE, 73

**K**

**Kinematic transformation TRANSMIT, TRACYL and TRAANG**, 322  
**Kinematic type**, 463  
**Kinematics**  
     Resolved, 463  
**Kinematics type M**, 463  
**Kinematics type P**, 463  
**Kinematics type T**, 463

**L**

**L..., 145**

**Label**, 71  
**Laser power control**, 611  
**Laxis**, 477, 515, 520, 536, 540  
**LEAD**, 336  
**Lead angle**, 336  
**Leading axis**, 540  
**Leading value coupling**, 638  
**LEADOF**, 540  
**Learn compensation characteristics**, 737  
**LFPOS**, 108  
**LFTXT**, 108  
**LFWP**, 108  
**LIFTFAST**, 106, 107  
**Linear interpolation**, 264  
**Link axis**, 712  
**Link communication**, 707  
**Link module**, 709  
**Link variables**  
     Global, 709  
**LLIMIT**, 609  
**LN**, 46  
**LOCK**, 661  
**Logic operators**, 48  
**LOOP**, 87  
**LUD**, 22

**M**

**M**, 461  
**M commands**, 686  
**M17**, 140  
**MAC**  
     Automatic activation, 207  
**Machine**  
     Status, global workpiece clamping, 708  
**Macro**, 188  
**Macro techniques**, 686  
**Marker variables**, 591  
**MASLDEF**, 568  
**MASLDEL**, 568  
**MASLOF**, 568  
**MASLOFS**, 568  
**MASLON**, 568  
**Master value coupling**  
     Actual value and setpoint coupling, 540, 544  
     from static synchronized actions, 542  
     Synchronization of leading and following axis, 543  
**Master value simulation**, 545  
**MATCH**, 64  
**MATCH241Ip**, 64  
**MAXVAL**, 53  
**MCALL**, 161

- MEAC, 268  
MEAFRAME, 304  
MEAFRAME, 304  
MEAFRAME, 308  
MEAS, 265  
MEASA, 268  
Measurement  
    Continuous measurement MEAC, 277  
    DDTG MEASA, MEAWA, 274  
    Operating mode, 273  
    Recognized programming errors, 277  
    Trigger events, 273  
Measurement job  
    Status for MEASA, MEAWA, 276  
    with 2 measuring systems, 276  
Measurement results for MEASA, MEAWA, 275  
Measuring, 641  
MEAW, 265  
MEAWA, 268  
Memory  
    Program memory, 191  
    Working, 197  
Milling tool  
    -reference point (FH), 439  
    -tip (FS), 439  
MINDEX, 64  
Minimum position/maximum position of the rotary axis, 461  
MINVAL, 46, 53  
MIRROR, 289  
MMC, 739  
MOD, 46  
Modal subprogram call, 161  
MODAXVAL, 699  
Mode, 676  
Mode, 676  
MOV, 626  
MPF, 737  
MU, 389  
MZ, 389
- N**
- n, 536  
NC block compressor, 252  
NC-controlled reactions, 723  
NCU  
    Link, 707  
NCU-global basic frames, 309  
NCU-global settable frames, 309  
NCU-NCU communication, 707  
Nesting  
    of subprograms, 137  
    Nesting depth  
        of check structures, 83  
    Networked NCUs, 707  
    NEWCONF, 120  
    NEWT, 423  
    Nibbling, 685, 689  
    NOC, 563  
    NOT, 48  
    NPROT, 224  
    NPROTDEF, 221, 223  
    NUMBER, 59  
    Number of curve table, 520
- O**
- OEM addresses, 278  
OEM functions, 278  
OEMIPO1/2, 278  
OFFN, 373, 376  
Offset contour normal OFFN, 384  
Offset memory, 407  
Offset of the rotary axes, 461  
Online tool length compensation, 468  
Online tool length offset, 620  
Online tool offset, 618  
OR, 48  
ORIAxes, 347, 364  
ORIC, 447  
ORICONCCW, 351, 364  
ORICONCW, 351, 364  
ORICONIO, 351, 364  
ORICONTO, 351, 364  
ORICURVE, 355, 364  
ORID, 447  
Orientation axes, 334, 343, 346, 349  
Orientation interpolation, 351, 366  
Orientation programming, 347, 366  
Orientation relative to the path  
    Inserting intermediate blocks, 368  
    Rotation of the orientation vector, 364  
    Rotation of the tool orientation, 364  
    Rotations of the tool, 362  
orientation transformation TRAORI  
    Generic 5/6-axis transformation, 322  
    Machine kinematics, 321  
    Orientation programming, 332  
    Travel movements and orientation movements, 320  
    Variants of orientation programming, 332  
ORIEULER, 347, 364  
ORIMKS, 344  
ORIPATH, 362, 803

ORIPATHS, 362, 367, 803  
 ORIPLANE, 351, 364  
 ORIRESET(A, B, C), 333  
 ORIROTA, 358  
 ORIROTC, 358, 364  
 ORIROTR, 358  
 ORIROTT, 358  
 ORIRPY, 347, 364  
 ORIRPY2, 347  
 ORIS, 447  
 ORIVECT, 347, 364  
 ORIVIRT1, 347, 364  
 ORIVIRT2, 347, 364  
 ORIWKS, 344  
 OS, 669, 670  
 OSC, 447  
 OSCILL, 676, 679  
 Oscillating axis, 672  
 Oscillating motion  
     Infeed at reversal point, 682  
     Reversal point, 679  
     Reversal range, 679  
     Suppress infeed, 679  
 Oscillation  
     Activate, deactivate oscillation, 669  
     Asynchronous oscillation, 669  
     Control via synchronized action, 675  
     Defining the sequence of motions, 674  
     Partial infeed, 679  
 Oscillation reversal points, 673  
 OSCTRL, 670, 674  
 OSD, 447  
 OSE, 670, 674  
 OSNSC, 670, 676  
 OSP, 673  
 OSP1, 670, 676  
 OSP2, 670, 676  
 OSS, 447  
 OSSE, 447  
 OST, 447, 670  
 OST1, 670, 676  
 OST2, 670, 676  
 Override  
     Current, 648  
     Resulting, 648  
 Overview  
     Frames active in the channel, 311  
 Overwriting curve tables, 531  
 OVRA, 565

**P**

P, 159  
 Parameter  
     -transfer for subprogram call, 145  
 Parameter area  
     Configurable, 587  
 Parameterizable subprogram return, 152  
 Parameters  
     Machine, 407  
 Part program, 709  
 Partial length, 676  
 Partial length, 676  
 Path  
     Absolute, 93  
     Relative, 94  
 Path reference  
     Can be set, 262  
     Circular interpolation and linear interpolation, 264  
     Curve parameter, 262  
     G code group, 262  
     Path axes, 264  
     Path feed, 264  
     Restrictions, 264  
     Thread blocks, 264  
 Path segment, 689  
 Path segmentation, 694  
 Path segmentation for path axes, 692  
 Path segments, 689  
 Path tangent angle, 647  
 PCALL, 167  
 PDELAYOF, 685  
 PDELAYON, 685  
 Peripheral milling, 433  
 PFRAME, 289  
 PHI, 357  
 PL, 238, 255  
 PO, 255  
 PO[PHI], 357, 362  
 PO[PSI], 357, 362  
 PO[THT], 357, 362  
 PO[XH], 357  
 PO[YH], 357  
 PO[ZH], 357  
 Polar transformation, 322  
 POLF, 720  
 POLFA, 720  
 POLFMASK, 720  
 POLFMLIN, 720  
 POLY, 255  
 Polynomial  
     -Interpolation, 264  
 Polynomial coefficient, 256

- Polynomial definition, 609  
Polynomial interpolation, 255  
    Denominator polynomial, 260  
Polynomials up to the 5th order, 525  
POLYPATH, 255  
PON, 693  
PONS, 685  
POS, 622  
POSFS, 563  
Position attributes  
    Indirect programming, 41  
Position synchronism, 555  
Positioning movements, 622  
POSP, 676  
POS RANGE, 624  
POT, 46  
PREPRO, 182  
Preprocessing memory, 492  
Preprocessing stop, 606  
Preset actual value memory, 634  
Preset offset, 303  
PRESETON, 303, 634  
PRIO, 102, 107  
PRLOC, 212  
Probe status, 276  
PROC, 140  
Program  
    Branch, 73  
    Initialization, 197  
    -memory, 193  
    repetition, 159  
Program coordination  
    Channel names, 96  
    Channel numbers, 95  
Program jumps, 70  
Program loop  
    Count loop, 88  
    End of loop, 87  
    IF loop, 85  
    REPEAT loop, 91  
    WHILE loop, 90  
Program memory, 191  
    File Types, 193  
    Standard directories, 192  
Program runtimes, 740  
    Actual machining time, 741  
    Tool operation time, 741  
    Total runtime, 741  
Program section repetition, 76  
    With indirect programming CALL, 164  
Programming an inclined axis  
    G05, G07, 390
- Programming commands  
    List, 769, 820  
Programming rotation of orientation vector with THETA, 358  
Protection level  
    for NC language element, 209  
    for system variable, 209  
Protection levels  
    For machine and setting data, 208  
    For user data, 204  
Protection levels, 212  
Protection zones  
    Activate, deactivate, 224  
    Activation status, 227  
    Channel-specific protection zones, 222  
    Contour definition of protection zones, 223  
    definition, 222  
    Definitions on the machine, 221  
    Machine-specific protection zones, 222  
    Multiple activation, 228  
    Offset, 227  
    Permissible contour elements, 223  
    Selected working plane, 223  
    Status after booting, 228  
PSI, 357  
PTP, 391, 398  
PTP for TRANSMIT, 398  
PTPG0, 398  
PUD, 22  
PUNCHACC, 685  
Punching, 685, 689  
PUTFTOC, 426  
PUTFTOFC, 426  
PW, 238
- Q**
- QEC, 737  
QECDAT, 737  
QECLRN, 737  
QECLRNOF, 737  
QECLRNON, 737  
QECTEST, 737  
Quadrant error compensation  
    Activating learning operation, 737  
    Deactivating learning, 737  
    Relearning, 738
- R**
- R parameters, 593

- R..., 18  
 RDISABLE, 606  
 READ, 125  
 Readin disable, 606  
 Reading measured values, 267  
 REAL, 22  
 Reciprocation
  - Synchronous oscillation, 675
 REDEF, 208, 209, 212  
 Refpos, 625  
 Relational operators, 48  
 RELEASE, 113  
 REP, 28, 642  
 REPEAT, 76, 91  
 REPEATB, 76  
 Repeated use of curve tables, 531  
 Replaceable geometry axes, 702  
 REPOS, 100  
 REPOSA, 502  
 REPOSH, 502  
 REPOSHA, 502  
 Repositioning
  - Approaching with a new tool, 510
  - Reapproach point, 507
 REPOSL, 502  
 REPOSQ, 502  
 REPOSQA, 502  
 RESET, 661  
 Resolved kinematics, 459  
 RET, 136, 140, 152  
 Retract, 723  
 Reversal
  - Point, 675
 RINDEX, 64  
 RMB, 502  
 RME, 502  
 RMI, 502  
 RMN, 502  
 Rotary axes
  - Direction vectors V1, V2, 459
  - Distance vectors I1, I2, 458
 ROUND, 46  
 Round up, 134  
 ROUNDUP, 134  
 Runtime response
  - of check structures, 84
- S**
- S1, S2, 561, 567  
 SAVE, 101, 138  
 SBLOF, 174  
 SBLON, 174  
 SCPARA, 284  
 SD, 238  
 SD42475, 370  
 SD42476, 370  
 SD42477, 370  
 SD42674, 448  
 SD42676, 448  
 SD42900, 416  
 SD42910, 416  
 SD42920, 417  
 SD42930, 418  
 SD42935, 420  
 SD42940, 421, 474  
 SD42984, 472  
 Search for character, 64  
 Search path
  - For subprogram call, 195
  - Programmable search path, 168
 SEFORM, 220  
 Selecting a substring, 66  
 Selection of a single character, 67  
 Servo parameter set
  - Programmable, 284
 SET, 28, 642  
 SETAL, 644, 746  
 SETDNO, 456  
 SETINT, 102, 107  
 SETM, 93, 643  
 SETPIECE, 423  
 Setpoint value coupling, 557  
 Setting data, 672  
 Setup value, 412  
 SIEMENS cycles, 746  
 SIN, 46  
 Single axis motion, 694  
 Single block
  - suppression, 174
 Singular positions, 345  
 Smoothing of orientation characteristic, 364, 368  
 Software limit switch, 632  
 SON, 685, 692, 693  
 SONS, 685  
 Sparking-out stroke, 671  
 SPATH, 262  
 Speed coupling, 557  
 Speed ratio, 562  
 SPF, 737  
 SPI, 565, 699  
 SPIF1, 812  
 SPIF2, 812  
 Spindle

-replacement, 113  
Spindle motions, 635  
Spline  
  -Interpolation, 238  
  -types, 246  
Spline grouping, 250  
Spline interpolation, 264  
SPLINEPATH, 250  
SPN, 689  
SPOF, 685  
SPOS, 564  
SPP, 689  
SQRT, 46  
SR, 813  
SRA, 813  
ST, 813  
STA, 813  
START, 93  
Start/stop axis, 626  
STARTFIFO, 491  
STAT, 391, 398  
Statements  
  List, 769, 820  
Station/position change, 710  
Status of coupling, 546  
Stock removal, 749  
Stop and retract  
  Extended, 715  
Stop time, 670  
STOPFIFO, 491  
Stopping, 725  
STOPRE, 491, 672  
String  
  -concatenation, 60  
  -length, 63  
  -operations, 57  
STRING, 22  
STRINGIS, 728  
  NC addresses, 731  
STRLEN, 63  
Subprogram  
  -call with parameter transfer, 145  
  -call without parameter transfer, 145  
  -call, indirect, 163  
  -call, modal, 161  
  -name, 137  
Programmable search path, 168  
  -repetition, 159  
Subprogram with path specification and parameters, 167  
Subprograms, 135  
Subprograms with parameter transfer

Array definition, 144  
Parameter transfer between the main program and subprogram, 144  
SUBSTR, 66  
Synchronism  
  coarse, 557  
  Fine, 557  
Synchronized action, 709  
  Action, 581  
  Area of validity, 576  
  Axis positioning, 622  
  Command elements, 574  
  Condition, 578  
  Delete, 663  
  Syntax, 574  
Synchronized action parameters, 592  
Synchronized actions  
  Action overview, 602  
  ASUB, 667  
  Block search, 666  
  CANCEL, 667  
  End of program, 666  
  Main run variable, 584  
  Mode change, 664  
  NC Stop, 665  
  Power on, 664  
  Preprocessing variables, 584  
  Repositioning, 667  
  Reset, 665  
Synchronous oscillation  
  Assignment of oscillating and infeed axes, 679  
  Define infeeds, 679  
  Evaluation, interpolation cycle, 683  
  Infeed in reversal point range, 681  
  Infeed movement, 681  
  Next partial infeed, 684  
  Stop at the reversal point, 683  
  Synchronized actions, 680  
Synchronous spindle, 554  
  Block change behavior, 563  
  Define pair, 561  
  Delete coupling, 567  
  Pair, 554  
  Speed ratio SRT, 562  
SYNFCT, 612  
SYNFCT() evaluation function, 612  
System variables, 17, 583, 709  
  Global, 709

**T**

TAN, 46

- TANG, 477  
 TANGDEL, 477  
 Tangential control, 477  
 TANGOF, 477  
 TANGON, 477  
 TCARR, 464  
 TCOABS, 464  
 TCOFR, 464  
 TCOFRX, 464  
 TCOFRY, 464  
 TCOFRZ, 464  
 TE, 268  
 Technology cycles  
     Cascading, 658  
 Technology cycles, 651  
     Control cyclic processing ICYCOF, 657  
     Default parameters with initial values, 656  
 Technology cycles  
     in non-modal synchronized actions, 659  
 Technology cycles  
     IF check structures, 659  
 Technology cycles  
     Jump instructions (GOTOP, GOTOF, GOTOB), 660  
 Technology cycles  
     Unconditional jumps, 660  
 THETA, 357, 358  
 TILT, 336  
 Tilt angle, 336  
 Time requirement  
     Synchronized actions, 649  
 Time use evaluation, 649  
 Timer variable, 595  
 TLIFT, 477  
 TMOF, 697  
 TMON, 697  
 TOFFOF, 620  
 TOFFOF, 468  
 TOFFON, 620  
 TOFFON, 468  
 TOLOWER, 62  
 Tool  
     -compensation memory, 407  
     -length compensation, 464  
     -monitoring, grinding-specific, 697  
     -offsets, additive, 410  
     -orientation for frame change, 467  
     -parameters, 407  
     -radius compensation, 414  
 Tool management, 423  
 Tool monitoring, grinding-specific, 430  
 Tool offset  
     Compensation memory, 407  
     Coordinate system for wear values, 418  
     Online, 426  
 Tool orientation, 447  
 Tool radius compensation  
     3D circumferential milling without limitation surfaces, 442  
     Corner deceleration, 279  
 Toolholder, 464  
     Can be orientated, 464  
     Deleting/changing/reading data, 463  
     Kinematics, 458  
 Toolholder with orientation capability  
     Number of the toolholder, 461  
     System variables, 459  
 Toolholder with orientation capability, 458  
 Torsion, 737  
 TOUPPER, 62  
 TOWBCS, 418  
 TOWKCS, 418  
 TOWMCS, 418  
 TOWSTD, 418  
 TOWTCS, 418  
 TOWWCS, 418  
 TRAANG, 322, 386  
 TRACYL, 322, 376, 384  
 TRACYL transformation, 378  
 TRAFOOF, 373, 376, 386, 404  
 TRAILOF, 515  
 TRAILON, 515  
 Transformation TRAORI, 330  
 Transformation types  
     General function, 317  
 Transformation with a swiveling linear axis, 328  
 Transformation, five-axis  
     Programming in Euler angles, 338  
     Programming in RPY angles, 338  
     Programming of path curve in surface normal vectors, 342  
     Programming using LEAD/TILT, 334  
 Transformation, three-, four-axis transformations, 330  
 Transformations  
     Chained, 404  
     Chained transformations, 320  
     Initial tool orientation setting regardless of kinematics, 319  
     Kinematic transformations, 319  
     Orientation transformation, 318  
     Three, four and five axis transformation (TRAORI), 318  
 TRANSMIT, 322, 373, 376, 398  
 TRANSMIT transformation, 374  
 TRAORI, 326, 330

Travel-dependent acceleration PUNCHACC, 687  
TRUE, 16  
TRUNC, 46  
TU, 391, 398  
Type of coupling, 557

## U

U1,U2, 679  
uc.com, user cycles, 185  
ULIMIT, 609  
UNLOCK, 661  
UNTIL, 91  
UPATH, 262  
User data  
  definition, 203  
Users  
  -data, 200

## V

V1,V2, 459  
Value Range  
  Variables, 16  
VAR, 140  
Variable  
  Assignments, 45  
  Type conversion, 56  
Variables, 15  
  -definition, 22, 587  
  -name, 24  
  Range of values, 16  
  -type, 22  
  Type conversion, 58  
  -types, 15, 16  
  user-defined, 22  
VELOLIM, 513

## W

WAIT, 95  
Wait markers, 643  
WAITC, 555, 566  
WAITE, 93  
WAITM, 93  
WAITMC, 93  
Wear value, 412  
WHEN, 578  
WHEN-DO, 676, 680  
WHENEVER, 578  
WHENEVER-DO, 676, 680

WHILE, 90  
Winlimit, 625  
Work offset  
  External zero offset, 302  
  PRESETON, 303  
Working memory, 197  
  Data areas, 197  
  Reserved block names, 202  
Workpiece  
  -directories, 193  
  -main directory, 193  
Workpiece counter, 744  
WRITE, 121

## X

x, 423  
XOR, 48