SIEMENS

SINUMERIK 840D sl/ 840Di sl/840D/840Di/810D

Job planning

Programming Manual

Valid for

Control

SINUMERIK 840D sl/840DE sl
SINUMERIK 840Di sl/840DiE sl
SINUMERIK 840D powerline/840DE powerline
SINUMERIK 840Di powerline/840DiE powerline
SINUMERIK 810D powerline/810DE powerline

Software	*Version*
NCU system software for 840D sl/840DE sl	1.3
NCU system software for 840Di sl/DiE sl	1.0
NCU system software for 840D/840DE	7.4
NCU system software for 840Di/840DiE	3.3
NCU system software for 810D/810DE	7.4

03/2006 Edition
6FC5398-2BP10-1BA0

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

Disclaimer of Liability

Siemens AG
Automation and Drives
Postfach 48 48
90437 NÜRNBERG
GERMANY

Order No.: 6FC5398-2BP10-1BA0
Edition 05/2006

Copyright © Siemens AG 2006.
Technical data subject to change

Preface

Foreword

SINUMERIK® Documentation

The SINUMERIK documentation is organized in 3 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 menu option "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 questions, please get in touch with our hotline:

Europe and Africa time zone

A&D Technical Support

Tel.:	+49 (0) 180 / 5050 - 222
Fax:	+49 (0) 180 / 5050 - 223
Internet:	http://www.siemens.com/automation/support-request
E-mail:	mailto:adsupport@siemens.com

Asia and Australia time zone

A&D Technical Support

Tel.:	+86 1064 719 990
Fax:	+86 1064 747 474
Internet:	http://www.siemens.com/automation/support-request
E-mail:	mailto:adsupport@siemens.com

America time zone

A&D Technical Support

Tel.:	+1 423 262 2522
Fax:	+1 423 262 2289
Internet:	http://www.siemens.com/automation/support-request
E-mail:	mailto:adsupport@siemens.com

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 (0) 9131 / 98 - 63315
E-mail:	mailto:motioncontrol.docu@siemens.com

Fax form: See the reply form at the end of this publication

SINUMERIK Internet address

http://www.siemens.com/sinumerik

EC declaration of conformity

The EC Declaration of Conformity for the EMC Directive can be found/obtained from:

- the internet:

  http://www.ad.siemens.de/csinfo

  under product/order no. 15257461
- the relevant branch office of the A&D MC group of Siemens AG.

Export version

The following functions are not available in the export version:

Function	810DE	840DE sl	840DE	840DiE sl	840DiE
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 1)	–	–	–	–	–
Synchronized actions 1) (Basic version, no options)	#	#	#	#	#
Master-value coupling and curve-table interpolation	#	#	#	#	#
Sag compensation, multi-dimensional	#	#	#	#	#
Synchronized actions, stage 2 1)	–	–	#	–	#
Electronic gear 1)	–	–	#	–	#
Electronic transfer	–	–	#	–	#
	# Restricted functionality - Function not available				

1) The restricted functions for the SINUMERIK 810DE powerline/SINUMERIK 840DE sl/ SINUMERIK 840DE powerline/SINUMERIK 840DiE sl/SINUMERIK 840DiE powerline export versions impose a limit of "max. 4 interpolating axes".

Description

Fundamentals

This Programming Guide "Fundamentals" 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.

Job planning

The Programming Guide "Job Planning" is intended for use by technicians with in-depth, comprehensive programming knowledge. By virtue of a special programming language, the SINUMERIK 840D sl/840Di sl/840D/840Di/810D control enables the user to program complex workpiece programs (e.g., for free-form surfaces, channel coordination, etc.) and greatly facilitates the programming of complicated operations.

The commands and statements described in this Guide are not specific to one particular technology.

They can be used for a variety of tasks, such as

- Turning, milling and grinding
- Cyclical machines (packaging, woodworking)
- Laser power controls.

Table of contents

1 Flexible NC programming

1.1 Variables and arithmetic parameters (user-defined variables, arithmetic parameters, system 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:

User-defined variables	Name and type of variable defined by the user, e.g., arithmetic parameter.
Arithmetic variables	Special, predefined arithmetic variable whose address is R plus a number. The predefined arithmetic variables are of the REAL type.
System variables	Variable provided by the control that can be processed in the program (write, read). System variables provide access to zero offsets, tool offsets, actual values, measured values on the axes, control states, etc. (See Appendix for the meaning of the system variables).

Variable types

Type	Meaning	Value range
INT	Integers with leading sign	$\pm(2^{31} - 1)$
REAL	Real numbers (fractions with decimal point, LONG REAL in acc. with IEEE)	$\pm(10^{-300} \dots 10^{+300})$

BOOL	Boolean values: TRUE (1) and FALSE (0)	1.0
CHAR	ASCII character specified by the code	0 … 255
STRING	Character string, number of characters in [...], maximum of 200 characters	Sequence of values with 0 ... 255
AXIS	Axis identifiers only (axis addresses)	Any axis identifiers in the channel
FRAME	Geometric data for translation, rotation, scaling, mirroring, see the "Frames" Chapter	

Arithmetic variables

Address R provides 100 arithmetic variables of type REAL by default.

The exact number of arithmetic variables (up to 32535) is defined in machine data.

Example: R10=5

System variables

The control provides system variables that can be contained and processed in all running programs.

System variables provide machine and control states. Some system variables cannot be assigned values.

Summary of system variables

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

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

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

Example: $AA_IM

Meaning: Current axis-specific value in the machine coordinate system.

1.2 Variable definition (DEF user-defined variables LUD, GUD, PUD)

Function

In addition to the predefined variables, programmers can define and initialize their own variables.

Local variables (LUD) are only valid in the program where they are defined.

Global variables (GUD) are valid in all programs.

Machine data are used to redefine the local user variables (LUD) defined in the main program as program-global user variables (PUD).

Machine manufacturer

See machine manufacturer's specifications.

Programming

Variable type INT

`DEF INT name`

or

`DEF INT name=value`

Variable type REAL

`DEF REAL name`

or

`DEF REAL name1,name2=3,name4`

or

`DEF REAL name[array_index1,array_index2]`

Variable type BOOL

`DEF BOOL name`

Variable type CHAR

`DEF CHAR name`

or

`DEF CHAR name[array_index2]=("A","B",…)`

Variable type STRING

`DEF STRING[string_length] name`

Variable type AXIS

`DEF AXIS name`

or

`DEF AXIS name[array_index]`

Variable type FRAME

`DEF FRAME name`

Note

If a variable is not assigned a value on definition, the system sets zero as the default.

Variables must be defined at the beginning of the program before they are used. The definition must be made in a separate block; only one variable type can be defined per block.

Parameters

INT	Variable type integer, i.e. whole number
REAL	Variable type real, i.e. factional number with decimal point
BOOL	Variable type Boolean, i.e. 1 or 0 (TRUE or FALSE)
CHAR	Variable type char, i.e. ASCII-coded character (0 to 255)
STRING	Variable type string, i.e. character string
AXIS	Variable type axis, i.e. axis addresses and spindles
FRAME	Variable type frame, i.e. geometric data

Example

Variable type	Meaning
INT	
DEF INT NUMBER	This creates a variable of type integer with the name NUMBER. System initializes with zero.
DEF INT NUMBER=7	This creates a variable of type integer with the name NUMBER. The system initializes the variable with 7.
REAL	
DEF REAL DEPTH	This creates a variable of type real with the name DEPTH. System initializes with zero (0.0).
DEF REAL DEPTH=6.25	This creates a variable of type real with the name DEPTH. The variable is initialized with 6.25.
DEF REAL DEPTH=3.1,LENGTH=2,NUMBER	More than one variable can be defined in a line.

BOOL	
DEF BOOL IF_TOO_MUCH	This creates a variable of type BOOL with the name IF_TOO_MUCH. System initializes with zero (FALSE).
DEF BOOL IF_TOO_MUCH=1 or DEF BOOL IF_TOO_MUCH=TRUE or DEF BOOL IF_TOO_MUCH=FALSE	This creates a variable of type BOOL with the name IF_TOO_MUCH.
CHAR	
DEF CHAR GUSTAV_1=65	A code value for the corresponding ASCII character or the ASCII character itself
DEF CHAR GUSTAV_1="A"	can be assigned to a variable of type CHAR (code value 65 corresponds to letter "A").
STRING	
DEF STRING[6] MUSTER_1="BEGIN"	Variables of type string can contain a string (sequence of characters). The maximum number of characters is enclosed in square brackets after the variable type.
AXIS	
DEF AXIS AXIS_NAME=(X1)	The variables of type AXIS have the name AXIS_NAME and contain the axis designation of a channel - here X1. (Axis names with extended address are specified within parentheses.)
FRAME	
DEF FRAME BEVEL_1	Variables of type FRAME have names like BEVEL_1.

Note

A variable of type AXIS can contain an axis identifier and a spindle identifier of a channel.

Note

Axis names with an extended address must be enclosed in parentheses.

Example: Redefine local (LUD) and program-global user variables (PUD)

If they are defined in the main program, they will also be valid at all levels of the subroutines called. They are created with parts program start and deleted with parts program end or reset.

If machine data $MN_LUD_EXTENDED_SCOPE is set, it is not possible to define a variable with the same name in the main and subroutines.

```
$MN_LUD_EXTENDED_SCOPE=1
PROC MAIN                                  ;Main program
DEF INT VAR1                               ;PUD definition
...                                        ;Subroutine call
SUB2
...
M30

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

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

Variable names

A variable name consists of up to 31 characters. The first two characters must be a letter or an underscore.

The "$" sign can not be used for user-defined variables because it is used for system variables.

Example: Program-local variables

```
DEF INT COUNTER
LOOP: G0 X…                                 ;Loop
COUNT=COUNT+1
IF COUNT<50 GOTOB LOOP
M30
```

Example: Querying existing geometry axes

```
DEF AXIS ABSCISSA;                          ;1. geometry axis
IF ISAXIS(1) == FALSE GOTOF CONTINUE
ABSCISSA = $P_AXN1
CONTINUE:
```

Example: Indirect spindle programming

```
DEF AXIS SPINDLE
SPINDLE=(S1)
OVRA[SPINDLE]=80                            ;Spindle override = 80%
SPINDLE=(S3)
…
```

1.3 Array definitions (DEF, SET, REP)

Function

An array is a memory area defined using the variable type with name and size. Arrays with up to two dimensions can be defined.

Note

Maximum array size

When defining arrays, the maximum array size of the 1st and 2nd dimension is 32767 for the array index [n, m].

Initialization of arrays

Initialization values can be assigned to the array elements:

- during the program execution

or

- already with the array definition.

In 2-dimensional arrays, the right array index is incremented first.

Programming

```
DEF CHAR NAME[n,m]
```
or
```
DEF INT NAME[n,m]
```
or
```
DEF REAL NAME[n,m]
```
or
```
DEF AXIS NAME[n,m]
```
or
```
DEF FRAME NAME[n,m]
```
or
```
DEF STRING[string_length] NAME[m]
```
or
```
DEF BOOL[n,m]
```

- Initialization with value lists; SET

Array definition options

```
DEF Type VARIABLE = SET(VALUE)
DEF Type ARRAY[n,m] = SET(VALUE, value, …)
```
or
```
DEF Type VARIABLE = Value
DEF Type ARRAY[n,m] = (value, value, …)
```

Note

SET is optional in the array definition.

Initializing during the program run

```
ARRAY[n,m]= SET(value, value, value,…)
ARRAY[n,m]= SET(expression, expression, expression,…)
```

- Initialization with the same values, REP

Array definition options

```
DEF Type ARRAY[n,m] = REP(value)
```

Note

Variables of type FRAME cannot be initialized.

Initializing during the program run

```
ARRAY[n,m] = REP(value)
ARRAY[n,m] = REP(expression)
```

Note

Variables of type FRAME are permissible and can be initialized very simply in this way.

Parameters

`DEF Variable type`	Array definition
`SET VALUE or expression`	Initialization with value lists for the array definition or in the program execution
`REP VALUE or expression`	Initialization with the same values for the array definition or in the program execution

`CHAR NAME[n,m]`
`INT NAME[n,m]`
`REAL NAME[n,m]`
`AXIS NAME[n,m]`
`FRAME NAME[n,m]`
`BOOL[n,m]`
	Variable type (CHAR, INTEGER, REAL, AXIS, FRAME, BOOL)

`STRING[string_length]`
`NAME[m]`
	Data type STRING can only be defined for 1-dimensional arrays.
	The string length is specified after the data type String.

`NAME`	Variable name
`Type VARIABLE`	Variable type (CHAR, INTEGER, REAL, AXIS, FRAME, BOOL)
`ARRAY[n,m]= SET(value, value,…)`	Initialization of all elements of an array with the listed values for the array definition
`TYPE ARRAY[n,m] = REP(value)`	Initialization of all elements of an array with the same value for the array definition

`ARRAY[n,m]= SET(value, value,…)`
`ARRAY[n,m]= SET(expression,…)`
	Initialization of all elements of an array with the listed values in the program execution

`ARRAY[n,m]= REP(value)`
`ARRAY[n,m]= REP(expression)`
	Initialization of all elements of an array with the same value in the program execution

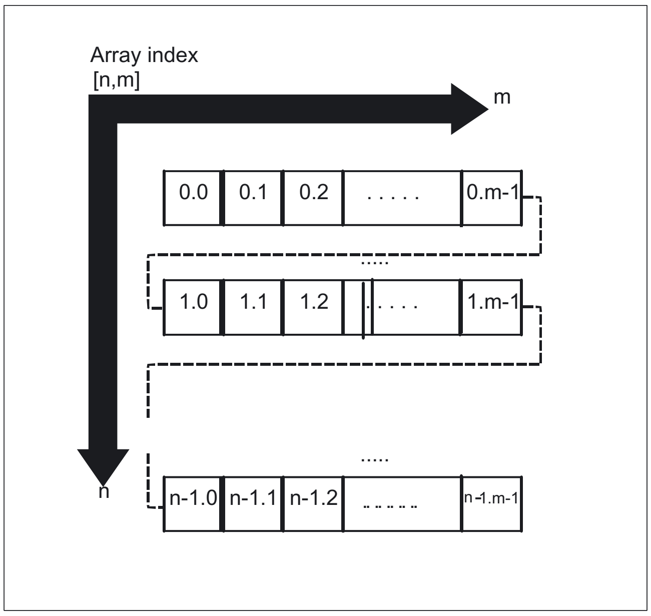
`ARRAY[n, m]`	Array index
`n`	Array size for 1st dimension
`m`	Array size for 2nd dimension
`Maximum array size`	e.g. DEF INT NAME[32767]

Arrays with variables of type STRING can only be 1-dimensional.

Array_index [n,m]

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 with index [0,0]; for example, for array size [3,4] the maximum possible array index is [2,3].



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. It should not be set higher than actually required.

Default: 812 bytes

If no large arrays are defined, select: 256 bytes.

Example: Definition of BOOL arrays

Global user data must contain PLC machine data for switching the control on/off.

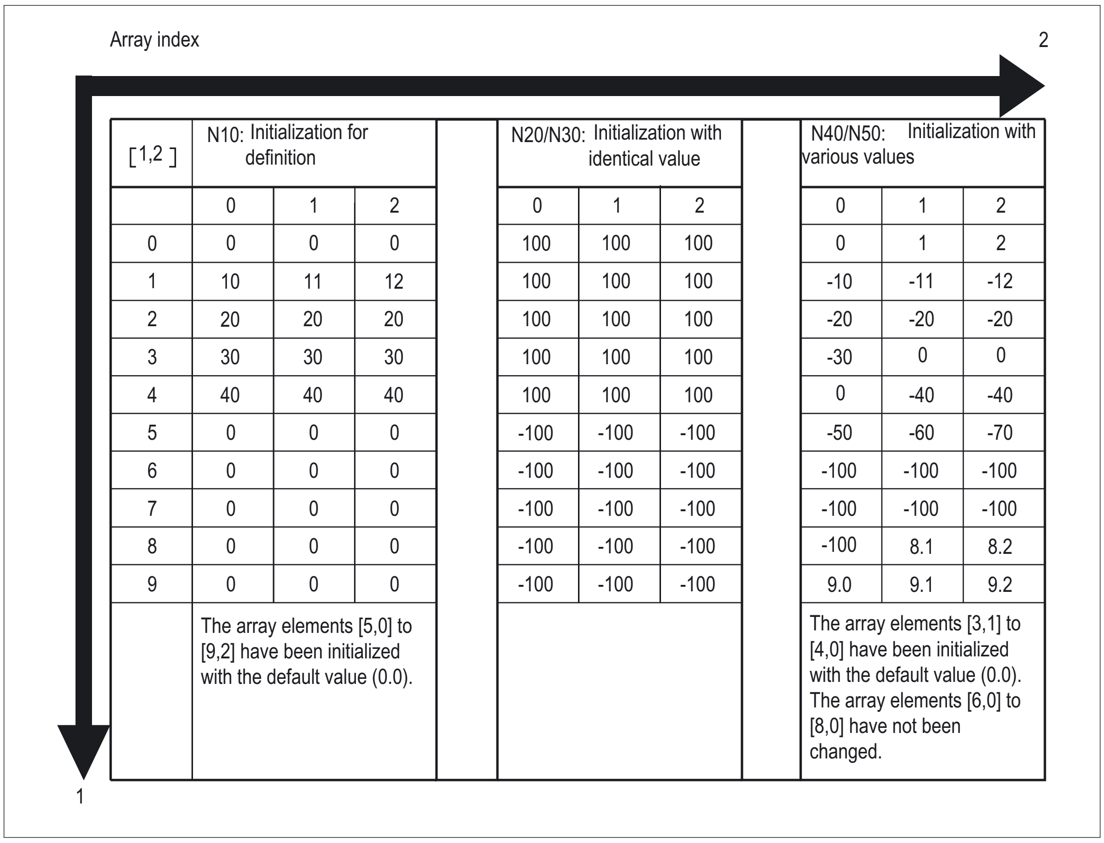
Example Definition of arrays with maximum array size for the 1st and 2nd dimension

```
DEF INT NAME[32767,32767]
```

Example: Initialization of complete variable arrays

The current assignment is shown in the drawing.

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


Array index
2
1

[1,2]	N10: Initialization for definition			N20/N30: Initialization with identical value			N40/N50: Initialization with various values		
	0	1	2	0	1	2	0	1	2
0	0	0	0	100	100	100	0	1	2
1	10	11	12	100	100	100	-10	-11	-12
2	20	20	20	100	100	100	-20	-20	-20
3	30	30	30	100	100	100	-30	0	0
4	40	40	40	100	100	100	0	-40	-40
5	0	0	0	-100	-100	-100	-50	-60	-70
6	0	0	0	-100	-100	-100	-100	-100	-100
7	0	0	0	-100	-100	-100	-100	-100	-100
8	0	0	0	-100	-100	-100	-100	8.1	8.2
9	0	0	0	-100	-100	-100	9.0	9.1	9.2
	The array elements [5,0] to [9,2] have been initialized with the default value (0.0).						The array elements [3,1] to [4,0] have been initialized with the default value (0.0). The array elements [6,0] to [8,0] have not been changed.		

Initialization with value lists for the array definition, SET

- 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.

Example:

```
DEF REAL ARRAY[2,3]=(10, 20, 30, 40)
```

Initialization with value lists in the program execution, SET

- Initialization is the same as in array definition.
- Expressions are possible values in this case too.
- Initialization starts at the programmed array indexes. Values can also be assigned selectively to subarrays.

Example: Assignment of expressions

```
DEF INT ARRAY[5, 5]
ARRAY[0,0] = SET(1, 2, 3, 4, 5)
ARRAY[2,3] = SET(VARIABLE, 4*5.6)
```

The axis index of axis variables is not traversed:

Example: Initialization in one line

```
$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
```

Initialization with the same values for the array definition, REP

All array elements are assigned the same value (constant).

Variables of type FRAME cannot be initialized.

Example:

```
DEF REAL ARRAY5[10,3] = REP(9.9)
```

Initialization with the same values in the program execution

- 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 also be assigned selectively to subarrays.

Example: Initialization of all elements with one value

```
DEF FRAME FRM[10]
FRM[5] = REP(CTRANS (X,5))
```

1.4 Indirect programming

Function

Indirect programming permits general-purpose use of programs. The extended address (index) is substituted by a variable of suitable type.

Indirect G code programming

Indirect programming of G codes using variables facilitates effective cycle programming. Two parameters

G code groups with integer constants

G code numbers with integer/real type variables

are available for this purpose.

Programming

```
ADDRESS[INDEX]
```

or

```
G[<group_index>] = <integer/real_variable>
```

Indirect programming of G codes using variables for effective cycle programming

Parameters

All addresses are parameterizable except:

- N - block number
- L - subroutine

Indirect programming is not possible for settable addresses.

Example: X[1] in place of X1 is not permissible.

`ADDRESS`	`Address with parameter details as index`
`[INDEX]`	`Index variable, e.g., spindle no., axis ....`
`G<group_index`	`G code groups: Integer constants with which the G code group is selected.`
`<Integer/real_variable>`	`G code numbers: Variable of the integer or real type with which the G code number is selected`

Valid G code groups

Only modal G code groups can be programmed indirectly.

Non-modal G code groups are rejected with alarm 12470.

Valid G code numbers

Arithmetic functions are not permissible in indirect G code programming.

The G code number must be stored in a variable of type integer or real. Invalid G code numbers are rejected with alarm 12475.

If it is necessary to calculate the G code number, this must be done in a separate parts program line before the indirect G code programming.

Note

All the valid G codes are shown in the PG, in the "List of G functions/preparatory functions" section in various groups. See /PG/ Programming Guide Fundamentals, "Tables"

Example

`Spindle`	
`S1=300`	`;Direct programming`
`DEF INT SPINU=1`	`;Indirect programming:`
`S[SPINU]=300`	`;Speed 300 rpm for the spindle whose number is stored in the SPINU variable (in this example 1).`
`Feed`	
`FA[U]=300`	`;Direct programming`
`DEF AXIS AXVAR2=U`	`;Indirect programming:`
`FA[AXVAR2]=300`	`;Feedrate for positioning axis whose address name is stored in the variable of type AXIS with the variable name AXVAR2.`
`Measured value`	
`$AA_MM[X]`	`;Direct programming`
`DEF AXIS AXVAR3=X`	`;Indirect programming:`
`$AA_MM[AXVAR3]`	`;Measured value in machine coordinates for the axis whose name is stored in variable AXVAR3.`
`Array element`	
`DEF INT ARRAY1[4,5]`	`;Direct programming`
`DEFINE DIM1 AS 4`	`;Indirect programming:`
`DEFINE DIM2 AS 5`	`Array dimensions must be stated as constant values.`
`DEF INT ARRAY[DIM1,DIM2]`	
`ARRAY[DIM1-1,DIM2-1]=5`	
`Axis assignment with axis variables`	
`X1=100 X2=200`	`;Direct programming`
`DEF AXIS AXVAR1 AXVAR2`	`;Indirect programming:`
`AXVAR1=(X1) AXVAR2=(X2)`	`;Definition of variables`
`AX[AXVAR1]=100 AX[AXVAR2]=200`	`;Assignment of the axis names, traversal of axes that are stored in the variables to 100 or 200.`
`Interpolation parameters with axis variables`	
`G2 X100 I20`	`;Direct programming`
`DEF AXIS AXVAR1=X`	`;Indirect programming:`
`G2 X100 IP[AXVAR1]=20`	`;Definition and assignment of the axis name`
	`;Indirect programming of the center`
`Indirect subroutine call`	
`CALL "L" << R10`	`;Call of the program whose number is in R10`

Note

R parameters can also be considered 1-dimensional arrays with abbreviated notation (R10 is equivalent to R[10]).

Example: Indirect G code programming

Settable zero offset G code group 8

```
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
```

Plane selection G code group 6

```
N2010 R10 = $P_GG[6]                   ; Read G code for current plane
...
N2090 G[6] = R10                       ; G17
```

1.4.1 Run string as parts program line (EXECSTRING)

Function

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

Programming

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

Parameters

`EXECSTRING`	Transfer of a string variable with the parts program line to run
`(<string_variable>)`	Parameters with the parts program line actually to be executed

Note

All parts program constructions that can be programmed in a parts program can be output. That excludes PROC and DEF instructions and all use of INI and DEF files.

Example: Indirect parts program line

```
N100 DEF STRING[100] BLOCK             ;String variable to be included in ;parts
                                       program line
N110 DEF STRING[10] MFCT1 = "M7"

N200 EXECSTRING(MFCT1 << " M4711")     ;Run parts program line "M7 M4711"

N300 R10 = 1
N310 BLOCK = "M3"
N320 IF(R10)
N330 BLOCK = BLOCK << MFCT1
N340 ENDIF
N350 EXECSTRING(BLOCK)                 ;Run parts program line "M3 M4711"
```

1.5 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.

Parameters

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

```
R1=10.518 R2=4 VARI1=45             ;Assignment of a numeric value
X=47.11 Y=R2
R1=R3 VARI1=R4                      ;Assignment of a suitable type variable
R4=-R5 R7=-VARI8                    ;Assignment with opposite sign
                                    ;(only permitted for INT and REAL types)
```

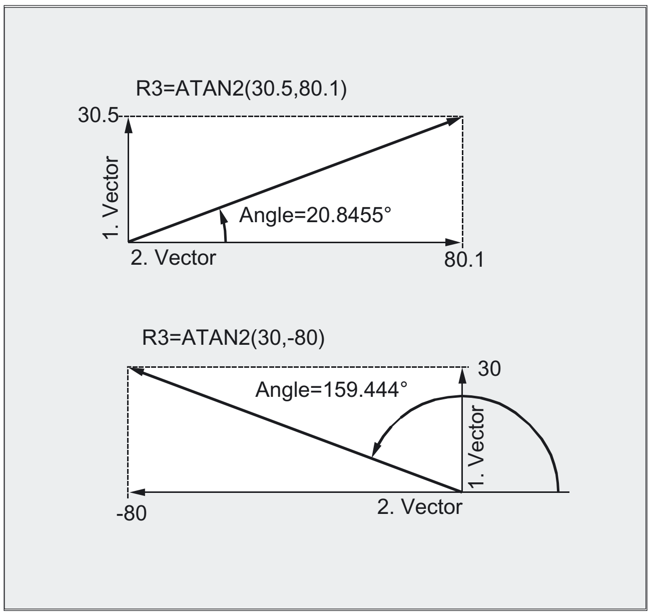
1.6 Arithmetic operations/functions

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 (–° < 0 < +180°). 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 correction for comparison commands"

Programming

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

Parameters

Operators/Mathematical functions

+	Addition
-	Subtraction
*	Multiplication
/	Division **Caution**: (type INT)/(type INT)=(type REAL) ;example: 3/4 = 0.75
DIV	Division, for variable type INT and REAL **Caution**: (type INT)DIV(type INT)=(type INT) ;example: 3 DIV 4 = 0
MOD	Modulo division (INT or REAL) produces remainder of INT division, e.g., 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
CTRANS()	Translation
CROT ()	Rotation
CSCALE()	Change of scale
CMIRROR()	Mirroring

Example: Initialization of complete variable arrays

```
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 (R12+R22)
RESFRAME= FRAME1:FRAME2              ;The concatenation operator links frames
FRAME3=CTRANS(…):CROT(…)             to form a resulting frame or assigns values
                                     to frame components
```

1.7 Comparison and logical 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 used for the `BOOL` type. 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

Relational operators

==

or

<>

or

>

or

<

or

>=

or

<=

Logic operators

`AND`

or

`OR`

or

`NOT`

or

`XOR`

Spaces must be left between BOOLEAN operands and operators.

Bit-by-bit logic operators

`B_AND`

or

`B_OR`

or

`B_NOT`

or

`B_XOR`

Parameters

Meaning of relational operators

`==`	equal to
`<>`	not equal to
`>`	greater than
`<`	less than
`>=`	greater than or equal to
`<=`	less than or equal to

Meaning of logic operators

AND	AND
OR	OR
NOT	Negation
XOR	Exclusive OR

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

Meaning of bit logic operators

B_AND	Bit-serial AND
B_OR	Bit-serial OR
B_NOT	Bit-serial negation
B_XOR	Bit-serial exclusive OR

Note

The operator B_NOT refers to one operand only,
it comes after the operator.

Example: 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: logic operators

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

or

```
IF NOT R10 GOTOB START
```

NOT is only applied to one operand.

Example: bit logic operators

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

1.7.1 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.

Programming

Precision correction on comparison errors

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

Parameters

`TRUNC()`	Truncate decimal places

Relative equality considered 10^{-12} 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 equality with (>) and (<) can be deactivated by setting machine data MD 10280: 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

```
N40 R1=61.01 R2=61.02 R3=0.01              ;Assignment of initial values
N41 IF ABS(R2-R1) > R3 GOTOF ERROR         ;Jump was performed previously
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)      ;Precision correction
 R13=TRUNC(R3*1000)
IF ABS(R12-R11) > R13 GOTOF ERROR          ;Jump is no longer executed
M30                                        ;End of program
ERROR: SETAL(66000)
```

Example: calculate and evaluate the quotient of both operands

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

1.8 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
```

1.9 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 this 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*	Yes1)	yes*	–	–	–
INT	yes	yes	Yes1)	Yes 2)	–	–	–
BOOL	yes	yes	yes	yes	yes	–	–
CHAR	yes	yes	Yes 1)	yes	yes	–	–
STRING	–	–	Yes 4)	Yes 3)	yes	–	–
AXIS	–	–	–	–	–	yes	–
FRAME	–	–	–	–	–	–	yes

Explanation

*	At type conversion from REAL to INT, fractional values that are >=0.5 are rounded up, others are rounded down (cf. ROUND function).
1)	Value <> 0 is equivalent to TRUE; value == 0 is equivalent to FALSE
2)	If the value is in the permissible range
3)	If only 1 character
4)	String length 0 = >FALSE, otherwise TRUE

Note

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

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

1.10 String operations

Overview

Further string manipulations are provided in addition to the conventional operations "Assignment" and "Comparison" described in this section:

Parameters

Type conversion to STRING:	
`STRING_ERG = <<any type`[1)]	Result type: STRING
`STRING_ERG = AXSTRING (AXIS)`	Result type: STRING
Type conversion from STRING:	
`BOOL_ERG = ISNUMBER (STRING)`	Result type: BOOL
`REAL_ERG = NUMBER (STRING)`	Result type: REAL
`AXIS_ERG = AXNAME (STRING)`	Result type: AXIS
Concatenation of strings:	
`any type`[1)] `<< any Type` [1)]	Result type: STRING
Conversion to lower/upper case:	
`STRING_ERG = TOUPPER (STRING)`	Result type: STRING
`STRING_ERG = TOLOWER (STRING)`	Result type: STRING
Length of the string:	
`INT_ERG = STRLEN (STRING)`	Result type: INT
Look for character/string in the string:	
`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
Selection of a substring:	
`STRING_ERG = SUBSTR (STRING, INT)`	Result type: INT
`STRING_ERG = SUBSTR (STRING, INT, INT)`	Result type: INT
Selection of a single character:	
`CHAR_ERG = STRINGVAR [IDX]`	Result type: CHAR
`CHAR_ERG = STRINGARRAY [IDX_FELD, IDX_CHAR]`	Result type: CHAR

[1)] "any type" stands for the variable types INT, REAL, CHAR, STRING, and BOOL.

Special meaning of the 0 char

The 0 char is interpreted internally as end-of-string. Replacing a character by the 0 character truncates the string.

Example

```
DEF STRING[20] STRG = "Axis .
stopped"
STRG[6] = "X"                         ;Returns the message "Axis X stopped"
MSG(STRG)
STRG[6] = 0
MSG(STRG)                             ;Returns the message "Axis"
```

1.10.1 Type conversion to STRING

Function

This enables use of variables of different types in a message (MSG).

Performed implicitly with use of the operator << 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.

Programming

Syntax

STRING_ERG = AXSTRING (AXIS)	**Result type: STRING**

Semantics:

AXSTRING (AXIS) returns the specified axis identifier as a string.

Parameters

Variables of type AXIS can be converted to STRING by the AXSTRING function.

FRAME variables cannot be converted.

Example:

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

Example

```
DEF STRING[32] STRING_ERG
STRING_ERG = AXSTRING(X)                ;Now: STRING_ERG == "X"
```

1.10.2 Type conversion of STRING

Function

The NUMBER function converts from STRING to REAL.

If ISNUMBER returns the value FALSE, the CALL of NUMBER with the same parameter will issue an alarm.

The AXNAME function converts a string to data type AXIS. An alarm is output if the string cannot be assigned to any configured axis identifier.

Programming

Syntax

REAL_ERG = NUMBER (STRING)	Result type: REAL
BOOL_ERG = ISNUMBER (STRING)	Result type: BOOL
AXIS_ERG = AXNAME (STRING)	Result type: AXIS

Semantics:

NUMBER (STRING) returns the number represented by the string as a REAL.

ISNUMBER (STRING) returns TRUE, if the string is a valid REAL by the rules of the language. It is thus possible to check whether the string can be converted to a valid number.

AXNAME (STRING) converts the specified string to an axis identifier.

Example

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

1.10.3 Concatenation of strings

Function

This functionality puts a string together out of separate components.

The chaining function is implemented via 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.

Programming

Syntax

any type << any type	Result type: STRING

Semantics

The strings specified (possibly implicitly converted non-string types) are concatenated.

This operator can also be used as a "unary" operator with a single operand. This can be used for explicit type conversion to STRING (not for FRAME and AXIS).

Types FRAME and AXIS cannot be used with this operator.

Syntax

<< any type	Result type: STRING

Semantics

The specified type is implicitly converted to STRING type.

This can be used to put together a message or a command out of text lists and insert parameters into it (e.g. a module name):

MSG(STRG_TAB[LOAD_IDX]<<MODULE_NAME)

Caution

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

Example: concatenation of strings

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

1.10.4 Conversion to lower/upper case

Function

This functionality permits conversion of all letters of a string to standard capitalization.

Syntax

STRING_ERG = TOUPPER (STRING)	Result type: STRING
STRING_ERG = TOLOWER (STRING)	Result type: STRING

Semantics

All lower case letters are converted to either upper or lower case letters.

Example

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

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

1.10.5 Length of the string

Function

This functionality sets the length of a string.

Syntax

INT_ERG = STRLEN (STRING)	Result type: INT

Semantics

It returns a number of characters that are not the 0 character, counting from the beginning of the string.

Example

This can be used to ascertain the end of the string, for example, in conjunction with the single character access described below:

```
IF(STRLEN (MODULE_NAME) > 10) GOTOF ERROR
```

1.10.6 Look for character/string in the string

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.

Programming

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: They return the position in the string (first parameter) where the search has been successful. If the character/string cannot be found, the value "–1" is returned. In this case, the first character is in position 0.

Parameters

`INDEX`	`searches for the character specified as the second parameter in the string specified as the second parameter (from the beginning).`
`RINDEX`	`searches for the character specified as the second parameter in the string specified as the second parameter (from the end).`
`MINDEX`	`same as the INDEX function except that a list of characters is specified (as a string) and the index of the first character found is returned.`
`MATCH`	`searches for a string in a string.`

This can be used to break up a string by certain criteria, for example, at blanks or path separators ("/").

Example: separating an input string into path and module names:

```
DEF INT PATHIDX, PROGIDX
DEF STRING[26] INPUT
DEF INT LISTIDX
INPUT = "/_N_MPF_DIR/_N_EXECUTE_MPF"
LISTIDX = MINDEX (EINGABE, "M,N,O,P") + 1  The value returned in LISTIDX is 3
                                           because "N" is the first char from the
                                           selection list in parameter INPUT,
                                           searching from the beginning.
PATHIDX = INDEX (INPUT, "/") +1            ;Therefore: PATHIDX = 1
PROGIDX = RINDEX (INPUT, "/") +1           ;Therefore: PATHIDX = 1
                                           ;The SUBSTR function introduced in the
                                           next section can be used to break up
                                           variable INPUT into the components "Path"
                                           and "Module":
VARIABLE = SUBSTR (INPUT, PATHIDX,         ;returning "_N_MPF_DIR"
PROGIDX-PATHIDX-1)
VARIABLE = SUBSTR (INPUT, PROGIDX)         ;returning "_N_EXECUTE_MPF"
```

1.10.7 Selection of a substring

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.

Programming

Syntax

STRING_ERG = SUBSTR (STRING,INT)	**Result type: INT**
STRING_ERG = SUBSTR(STRING,INT, INT)	**Result type: INT**

Semantics

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

In the second case, the result string goes up to the maximum length specified in the third parameter.

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

A negative initial position or length triggers an alarm.

Example

```
DEF STRING [29] ERG
ERG = SUBSTR ("ACK: 10 to 99", 10, 2)       ;Therefore: ERG == "10"
```

1.10.8 Selection of a single character

Function

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

Programming

Syntax

CHAR_ERG = STRINGVAR [IDX]	Result type: CHAR
CHAR_ERG = STRINGARRAY [IDX_FELD, IDX_CHAR]	Result type: 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.

```
DEF STRING [50] MESSAGE = "Axis n has
reached position"
MESSAGE [6] = "X"
MSG (MESSAGE)                              ;returns message "Axis X has
                                           reached position"
```

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 subroutine calls.

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

```
DEF STRING [50] STRG
DEF CHAR ACK
…
STRG = $P_MMCA
ACK = STRG [0]                             ;Evaluation of acknowledgment component
```

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

```
DEF STRING [50] STRG
DEF CHAR CHR1
EXTERN UP_CALL (VAR CHAR1)                ;Call-by-reference parameter!
…
CHR1 = STRG [5]
UP_CALL (CHR1)                            ;Call-by-reference
STRG [5] = CHR1
```

1.11 CASE statement

Function

The CASE statement enables various branches to be executed according to a value of type INT.

The program jumps to the point specified by the jump destination, depending on the value of the constant evaluated in the CASE statement.

Programming

```
CASE (expression) OF constant1 GOTOF LABEL1 … DEFAULT GOTOF LABELn
CASE (expression) OF constant1 GOTOB LABEL1 … DEFAULT GOTOB LABELn
```

Parameters

CASE	Keyword for jump statement
GOTOB	Jump statement with jump destination backward (toward the beginning of program)
GOTOF	Jump statement with forward jump destination (toward the end of program)
GOTO	Jump statement with the jump destination first forward and then backward (the direction first to the end of the program and then to the start of the program)
GOTOC	Suppress Alarm 14080 "Destination not found".
	Jump statement with the jump destination first forward and then backward (the direction first to the end of the program and then to the start of the program)
LABEL	Destination (label within the program)
LABEL:	The name of the jump destination is followed by a colon
Expression	Arithmetic expression
Constant	Constant of type INT
DEFAULT	Program path if none of the previously named constants applies

Note

For more information on the GOTO commands, see Chapter 10, Arithmetic parameters and program jumps

In cases where the constant matches none of the predefined values, the DEFAULT statement can be used to determine the branch destination.

If the DEFAULT statement is not programmed, the jump destination is the block following the CASE statement.

Example 1

```
CASE(expression) OF 1 GOTOF LABEL1 2 GOTOF LABEL2 … DEFAULT GOTOF LABELn
```

"1" and "2" are possible constants.

If the value of the expression = 1 (INT constant), jump to block with LABEL1

If the value of the expression = 2 (INT constant), jump to block with LABEL2

…

otherwise jump to the block with LABELn

Example 2

```
DEF INT VAR1 VAR2 VAR3
CASE(VAR1+VAR2-VAR3) OF 7 GOTOF LABEL1 9 GOTOF LABEL2 DEFAULT GOTOF LABEL3
LABEL1: G0 X1 Y1
LABEL2: G0 X2 Y2
LABEL3: G0 X3 Y3
```

1.12 Control structures

Function

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

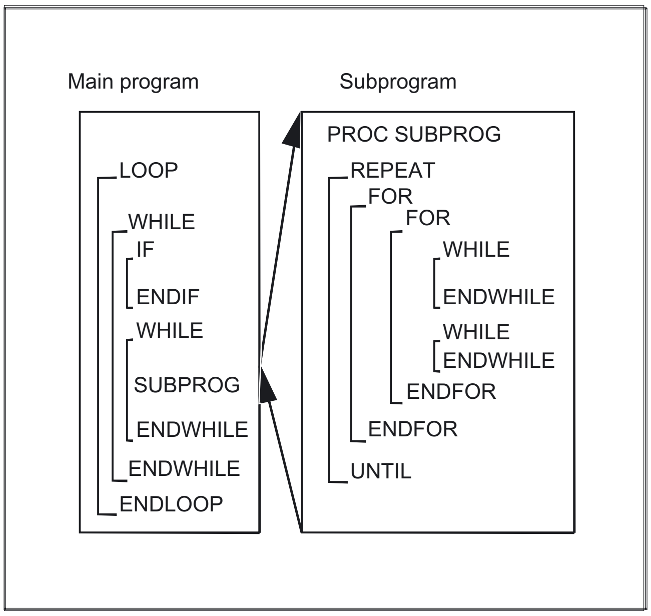
In addition to the program branches described in this chapter, these commands can be used to define additional alternatives and program loops.

These commands enable the user to produce well-structured and easily legible programs.

Programming

Nesting depth

Control structures apply locally within programs. A nesting depth of up to 8 control structures can be set up on each subroutine level.



Caution

Control 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 control structures or on branch destinations. No such check is made when the macro is defined.

Parameters

IF	Selection between 2 alternatives
LOOP	Endless loop
FOR	Count loop
WHILE	Loop with condition at beginning of loop
REPEAT	Loop with condition at end of loop

Example: endless program

```
%_N_LOOP_MPF
LOOP
 IF NOT $P_SEARCH                              ;No block search
 G01 G90 X0 Z10 F1000
 WHILE $AA_IM[X] <= 100
 G1 G91 X10 F500                               ;Drilling pattern
 Z-F100
 Z5
 ENDWHILE
 Z10
 ELSE                                          ;Block search
 MSG("No drilling during block search")
 ENDIF
 $A_OUT[1] = 1                                 ;Next drilling plate
 G4 F2
ENDLOOP
M30
```

Example: production of a fixed quantity of parts

```
%_N_WKPCCOUNT_MPF

DEF INT WKPCCOUNT
FOR WKPCCOUNT = 0 TO 100
 G01 …
ENDFOR
M30
```

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 control structures.

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

Restrictions

Blocks with control structure elements cannot be suppressed. Labels may not be used in blocks of this type.

Control structures are processed interpretively. When a loop end is detected, a search is made for the loop beginning, allowing for the control 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 control structures and program branches.

A check can be made to ensure that control structures are nested correctly when cycles are preprocessed.

Sequence

1. IF-ELSE-ENDIF

An IF-ELSE-ENDIF block is used to select one of two alternatives:

IF (expression)

NC blocks

ELSE

NC blocks

ENDIF

If the value of the expression is TRUE, i.e., the condition is fulfilled, then the next program block is executed. If the condition is not fulfilled, then the ELSE program branch is executed.

The ELSE branch can be omitted.

2. Endless loop control LOOP

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

LOOP

NC blocks

ENDLOOP

3. Counter loop FOR

The FOR loop is used if it is necessary to repeat an operation by a fixed number of runs. In this case, the count variable is incremented from the start value to the end value. The start value must be lower than the end value. The variable must be of type INT.

FOR Variable = start value TO end value

NC blocks

ENDFOR

4. Program loop with condition at start of loop WHILE

The WHILE program loop is executed for as long as the condition is fulfilled.

WHILE expression

NC blocks

ENDWHILE

5. Program loop with condition at end of loop REPEAT

The REPEAT loop is executed once and repeated continuously until the condition is fulfilled.

REPEAT

NC blocks

UNTIL (expression)

1.13 Program coordination

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	- Current directory/_N_name_MPF "current directory" stands for the selected workpiece directory or the standard directory /_N_MPF_DIR.
INIT (n,"/_N_MPF_DIR/_N_name_MPF")	- 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:	**Up to SW 3:**
INIT(2,"/_N_WKS_DIR/_DRESS_MPF") G01 F0.1 START	At least one executable block must be programmed between an **init** command (without synchronization) and an **NC start**. With subroutine calls "_SPF" must be added to the path
INIT (2,"/_N_WKS_DIR/_N_UNDER_1_SPF")	

- Relative path specification

Example:	The same rules apply to relative path definition as for program calls.
INIT(2,"DRESS")	
INIT(3,"UNDER_1_SPF")	With subroutine 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 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 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_MPF100_MPF

`N10 INIT(2,"MPF200")`	
`N11 START(2)`	`;Processing in channel 2`
`.`	

`N80 WAITM(1,1,2)`
`.`
	`;Wait for WAIT mark 2 in channel 1 and`
	`;in channel 2 and execution continued in channel 1`

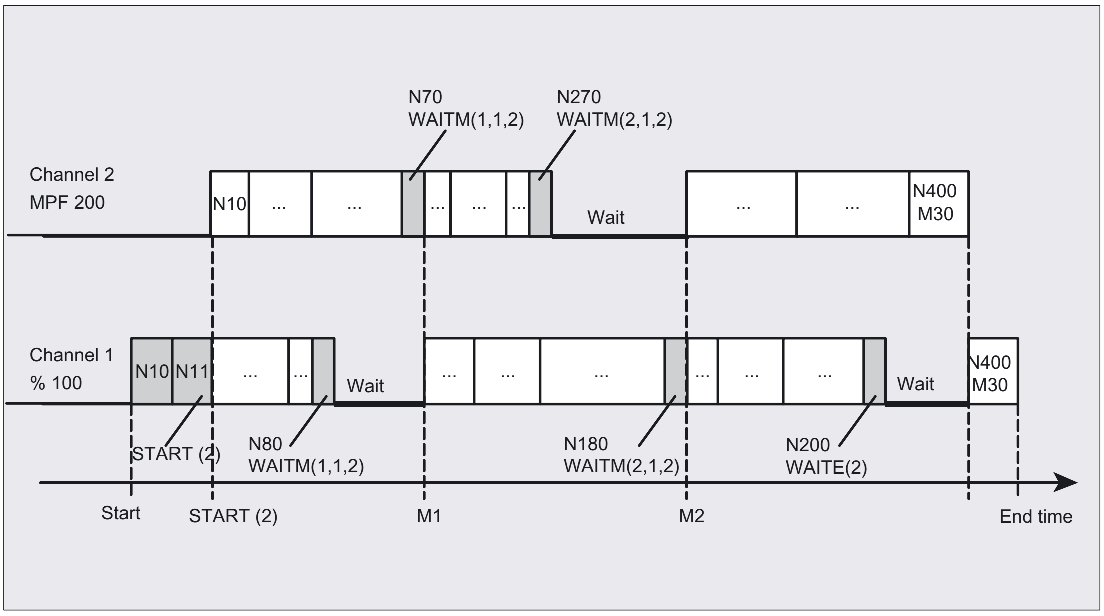
`N180 WAITM(2,1,2)`
`.`
	`;Wait for WAIT mark 2 in channel 2 and`
	`;in channel 2 and execution continued in channel 1`

`N200 WAITE(2)`	`;Wait for end of program in channel 2`
`N201 M30`	`;Program end channel 1, total end`
`…`	

Channel 2:

%_N_MPF200_MPF

```
;$PATH=/_N_MPF_DIR
                                        ;Processing in channel 2
N70 WAITM(1,1,2)                        ;Wait for WAIT mark 2 in channel 1 and
.                                       ;in channel 2 and execution continued in
                                        channel 1
N270 WAITM(2,1,2)                       ;Wait for WAIT mark 2 in channel 2 and
.                                       ;in channel 2 and execution continued in
                                        channel 1
N400 M30                                ;End of program in channel 2
```



Example: program from workpiece

```
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

```
N10 INIT(2,"MYPROG")                    ;Select program /_N_MPF_DIR/_N_MYPROG_MPF
                                        ;in channel 2.
```

Example of channel name and channel number with integer variable

`$MC_CHAN_NAME[0]=` "CHAN_X"
`$MC_CHAN_NAME[1]=` "CHAN_Y"

```
START(1, 2)                          ;Run start in 1st and 2nd channel
```

Similar to this, programming with the channel identifiers:

```
START(CHAN_X, CHAN_Y)                ;Run start in 1st and 2nd channel
                                     ;The channel_X and channel_Y identifiers
                                     represent $MC_CHAN_NAME internally due to the
                                     machine data, channel numbers 1 and 2 also run
                                     a start in the 1st and 2nd channel accordingly.
```

Programming with an integer variable:

```
DEF INT chanNo1, chanNo2)            ;Define channel number
chanNo1=CHAN_X chanNo2=CHAN_Y
START(chanNo1, chanNo2)
```

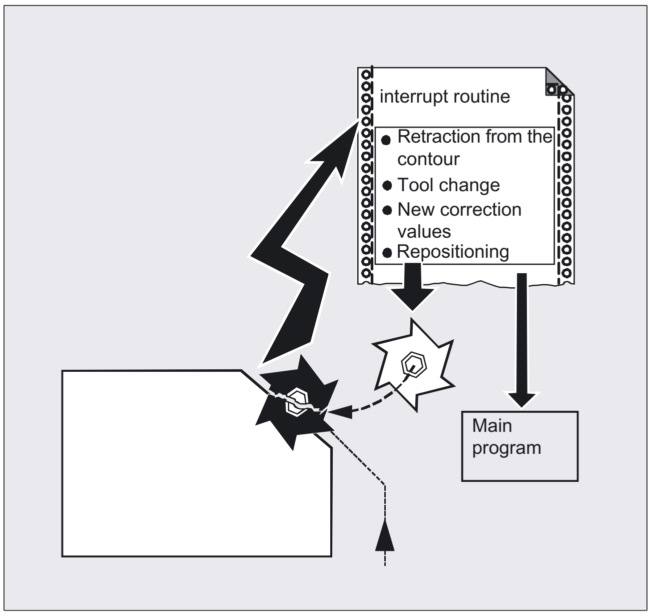
1.14 Interrupt routine (SETINT, DISABLE, ENABLE, CLRINT)

Function

The relationships concerned with programming an interrupt routine will be illustrated using a typical example:

The tool breaks during machining. This triggers a signal that stops the current machining process and simultaneously starts a subroutine – this subroutine is called an 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.



For further information on REPOS, see "Repositioning".

Programming

```
SETINT(3) PRIO=1 NAME
SETINT(3) PRIO=1 LIFTFAST
SETINT(3) PRIO=1 NAME LIFTFAST
G… X… Y… ALF=…
DISABLE (3)
ENABLE (3)
CLRINT (3)
```

Parameters

`SETINT(n)`	`Start interrupt routine if input n is enabled, n (1...8) stands for the number of the input`
`PRIO=1`	`Define priority 1 to 128 (1 has top priority)`
`LIFTFAST`	`Fast retraction from contour`
`NAME`	`Name of the subroutine to be executed`
`ALF=…`	`Programmable traverse direction (in motion block)`
`DISABLE(n)`	`Deactivate interrupt routine number n`
`ENABLE(n)`	`Reactivate interrupt routine number n`
`CLRINT(n)`	`Clear interrupt assignments of interrupt routine number n`

Retraction movement

The direction of the retraction movement is programmed by means of the G code **LFTXT** or **LFWP** with the variable **ALF**.

- LFTXT
  The plane of the retraction movement is determined by the path tangent and the tool direction. This G code (default setting) is used to program the response on a fast lift.
- LFWP
  The plane of the retraction movement is the active working plane selected with G codes G17, G18 or G19. The direction of the retraction movement is not dependent on the path tangent. This allows a fast lift to be programmed parallel to the axis.
- LFPOS
  Retraction of the axis declared with POLFMASK to the absolute axis position programmed with POLF. See also NC-controlled retraction in Function Manual M3
  ALF has no affect on the lift direction for several axes and for several axes in a linear system.

In the plane of the retraction movement, **ALF** is used, as before, to program the direction in discrete steps of 45 degrees. With **LFTXT**, the retraction is defined in the tool direction for ALF=1.

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

- **G17**:X/Y-levelALF=1retraction in X-direction
  ALF=3retraction in Y-direction
- **G18**:Z/X-levelALF=1retraction in Z-direction
  ALF=3retraction in X-direction
- **G19**:Y/Z-levelALF=1retraction in Y-direction
  ALF=3retraction in Z-direction

Example

In this example, a broken tool is to be replaced automatically by an alternate tool. Machining is continued with the new tool. Machining is then continued with the new tool.

Main program

```
N10 SETINT(1) PRIO=1 W_CHANGE ->        When input 1 is enabled, the tool is
 -> LIFTFAST                            automatically retracted from the contour
                                        with liftfast (code no. 7 for tool radius
                                        compensation G41). Interrupt routine
                                        W_CHANGE is subsequently 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
```

Subroutine

```
PROC W_CHANGE SAVE                      Subroutine with storage of current
                                        operating state
N10 G0 Z100 M5                          ;Tool changing position, spindle stop
N20 T11 M6 D1 G41                       ;Change tool
N30 REPOSL RMB M3                       ;Repositioning and return
                                        ;to main program
-> programmed in a single block.
```

Caution

If you do not program any of the REPOS commands in the subroutine, the axis is moved to the end of the block that follows the interrupted block.

Create interrupt routine as subroutine

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

Example:

```
PROC LIFT_Z
N10
N50 M17
```

Program name LIFT_Z, followed by the NC blocks, finally end-of-program `M17` and return to main program.

Note

SETINT statements can be programmed within the interrupt routine and used to activate additional interrupt routines. They are triggered via the input.

You will find more information on how to create subroutines in Chapter "Subroutines, Macros".

Save interrupt position, SAVE

The interrupt routine can be identified with `SAVE` in the definition.

Example:

```
PROC LIFT_Z SAVE
N10
N50 M17
```

At the end of the interrupt routine the modal G functions are set to the value they had at the start of the interrupt routine by means of the SAVE attribute. The programmable zero offset and the basic offset are reestablished in addition to the settable zero offset (modal G function group 8). If the G function group 15 (feed type) is changed, e.g. from `G94` to `G95`, the appropriate F value is also reestablished.

Machining can thus be resumed later at the point of interruption.

Assign and start interrupt routine, SETINT

The control has signals (inputs 1 to 8)

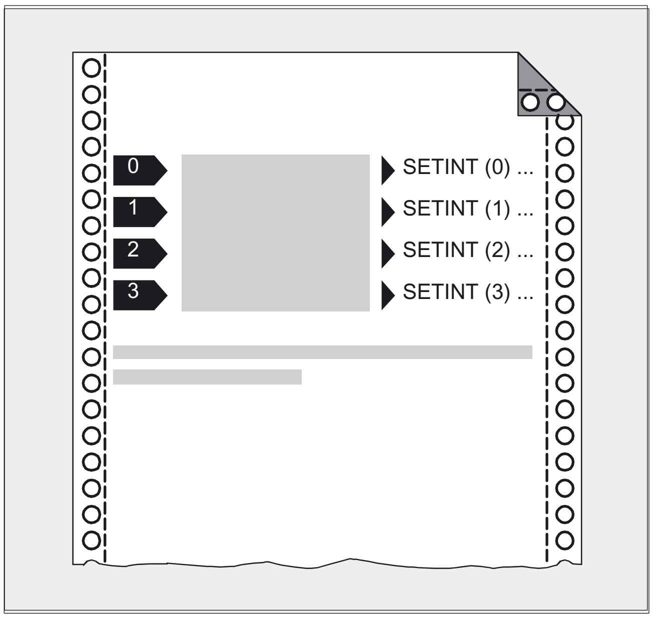
to interrupt the program run and start the corresponding interrupt routine.

The assignment of input to program is made in the main program.

Example:

```
N10 SETINT(3) PRIO=1 LIFT_Z
```

When input 3 is enabled, routine LIFT_Z is started immediately.



Start several interrupt routines, define the priority, PRIO=

If several SETINT instructions are programmed in your NC program and several signals can therefore occur at the same time, you must assign the priority of the interrupt routines to determine the order in which they are executed: PRIO 1 to 128, 1 has highest priority.

Example:

```
N10 SETINT(3) PRIO=1 LIFT_Z
N20 SETINT(2) PRIO=2 LIFT_X
```

The routines are executed successively in the order of their priority if the inputs are enabled at the same time. First `SETINT(3)`, then `SETINT(2)`.

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

Deactivate/reactivate interrupt routine, DISABLE, ENABLE

You can deactivate interrupt routines in the NC program with `DISABLE(n)` and reactive them with `ENABLE(n)` (n stands for the input number).

The input/routine assignment is retained with `DISABLE` and reactivated with `ENABLE`.

Reassign interrupt routines

If a new routine is assigned to an assigned input, the old assignment is automatically canceled.

Example:

```
N20 SETINT(3) PRIO=2 LIFT_Z
...
...
N120 SETINT(3) PRIO=1 LIFT_X
```

Clear assignment, CLRINT

Assignments can be cleared with `CLRINT(n)`.

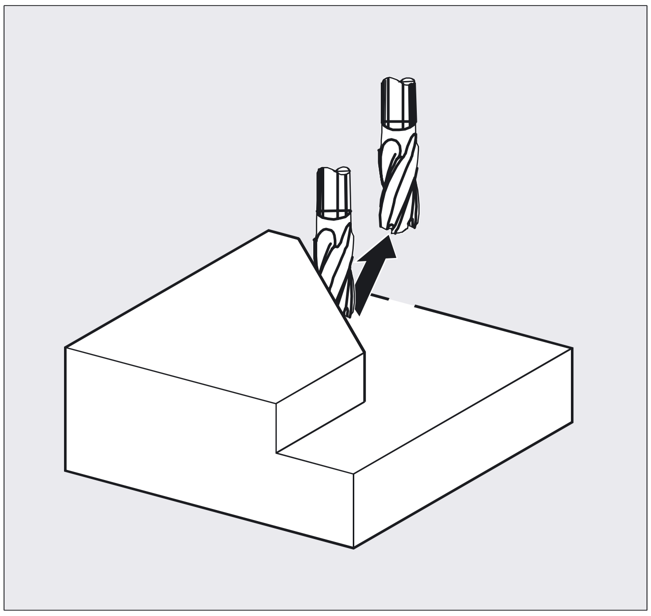
Example:

```
N20 SETINT(3) PRIO=2 LIFT_Z
N50 CLRINT(3)
```

The assignment between input 3 and the routine LIFT_Z is cleared.

Rapid lift from contour, LIFTFAST

When the input is switched, `LIFTFAST` retracts the tool rapidly from the workpiece contour.



If the SETINT instruction includes an interrupt routine as well as `LIFTFAST` , the liftfast is executed **before** the interrupt routine.

Example:

```
N10 SETINT(2) PRIO=1 LIFTFAST
```

or

```
N30 SETINT(2) PRIO=1 LIFT_Z LIFTFAST
```

In both cases, the liftfast is executed when input 2 with top priority is enabled.

- With N10, execution is stopped with alarm 16010 (as no asynchronized subroutine, `ASUB`, was specified).
- The `ASUB` "LIFT-Z" is executed with N30.

When determining the lift direction, a check is performed to see whether a frame with mirror is active. If one is active, right and left are inverted for the lift direction with regard to the tangent direction. The direction components in tool direction are not mirrored. This behavior is activated via MD $MC_LIFTFAST_WITH_MIRROR=TRUE

Sequence of motions with lift fast

The distance through which the geometry axes are retracted from the contour on liftfast can be defined in machine data.

Interrupt routine without LIFTFAST

Decelerates on the path and starts the interrupt routine as soon as motion on the path stops.

This position is stored as the interrupt position and is approached with `REPOS` with `RMI` at the end of the interrupt routine.

Interrupt routine with LIFTFAST

Decelerates on the path and simultaneously performs the FIFTFAST motion as an overlaid motion. If the path motion and LIFTFAST motion stop, the interrupt routine starts.

The position on the contour is stored as the interrupt position at which the LIFTFAST motion was started, thus leaving the path.

The interrupt routine behaves with `LIFTFAST` and `ALF=0` identical as the interrupt routine without `LIFTFAST`.

Programmable traversing direction, ALF=...

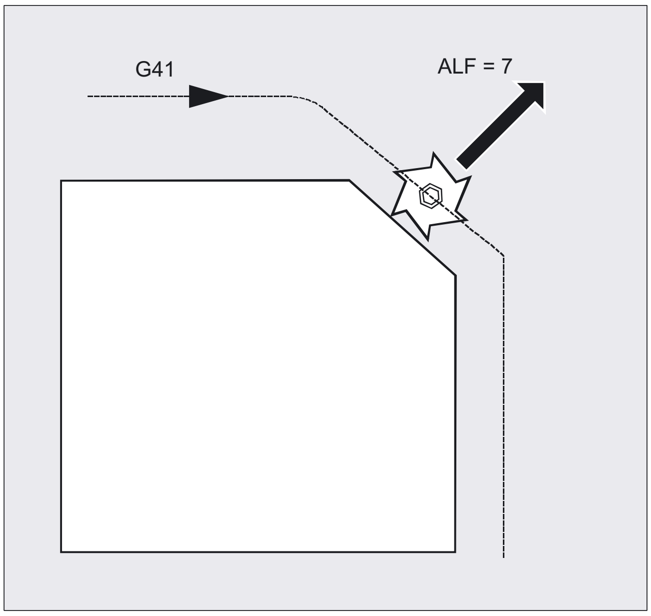
You enter the direction in which the tool is to travel on liftfast in the NC program.

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

Example:

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

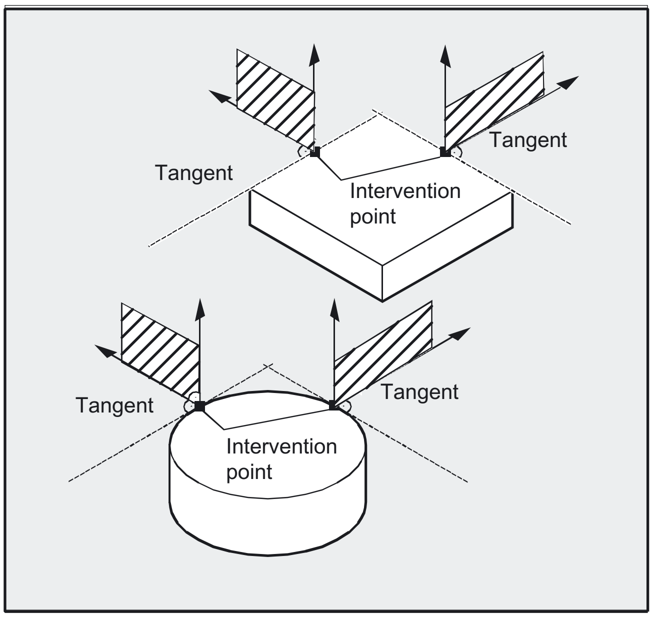
The tool moves – with `G41` activated (direction of machining to the left of the contour) – away from the contour perpendicularly as seen from above.



Reference plane for describing the traversing directions

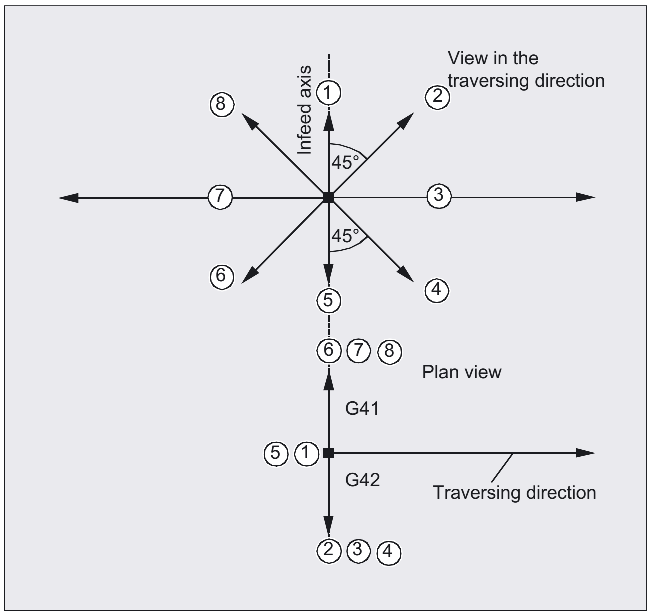
At the point of application of the tool to the programmed contour, the tool is clamped at a plane which is used as a reference for specifying the liftoff movement with the corresponding code number.

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 number with traversing directions summarized

The code numbers and the traversing directions in relation to the reference plane are shown in the diagram on the right.



`ALF=0` deactivates the liftfast function.

Caution

If tool radius compensation is activated, the codings 2, 3, 4 and the codings 6, 7, 8 should **not** be used
for `G41` and
for `G42`, respectively.

In these cases, the tool would approach the contour and collide with the workpiece.

1.15 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.

Programming

`RELEASE (axis name, axis name, ...)` or `RELEASE (S1)`

`GET (axis name, axis name, ...)` or `GET (S2)`

or

`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).

Parameters

`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 of an axis replacement 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 is to be transferred between the 2 channels and then assigned to channel 1 after power ON.

Program "MAIN" in channel 1

```
%_N_MAIN_MPF
INIT (2,"TRANSFER2")                  ;Select program TRANSFER2 in channel 2
N… START (2)                          ;Start program in channel 2
N… GET (AX2)                          ;Accept axis AX2

…
N… RELEASE (AX2)                      ;Enable axis AX2
N… WAITM (1,1,2)                      ;Wait for WAIT marker in channel 1 and 2
                                      ;for synchronizing in both channels
N…                                    ;Rest of program after axis replacement
N… M30
```

Program "Replace2" in channel 2

```
%_N_TRANSFER2_MPF
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
N…                                        ;Rest of program after axis replacement
N… M30
```

Example of axis replacement without synchronization

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

```
N01 G0 X0
N02 RELEASE(AX5)
N03 G64 X10
N04 X20
N05 GET(AX5)                              ;If synchronization is not necessary,
                                          ;this is not an executable block.
N06 G01 F5000                             ;Not an executable block.
N07 X20                                   ;Not an executable block because X position
                                          ;as for N04.
N08 X30                                   ;First executable block after N05.
N09 …
```

Example: activating an axis replacement without a preprocessing stop

Prerequisite

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

```
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).

1.16 Transfer axis to another channel (AXTOCHAN)

Function

The AXTOCHAN NC 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 parts program and from a synchronized action.

Programming

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

Parameters

AXTOCHAN	Request axis for a specific channel
Axis name	Axis assignment in 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.

```
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".

1.17 NEWCONF: Setting machine data effective

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.

Programming

```
NEWCONF
```

Parameter

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

Cross-channel execution of NEWCONF from the parts program

If axial machine data from the parts program are changed and then activated with `NEWCONF`, `NEWCONF` will only activate the machine data containing changes affecting the parts 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

```
N10 $MA_CONTOUR_TOL[AX]=1.0           ; Change machine data
N20 NEWCONF                           ; Set machine data active
```

1.18 WRITE: Write file

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 parts 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)

Programming

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

Parameters

Machine manufacturer

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

The MD 11420: LEN_PROTOCOL_FILE sets the maximum length of the log files in KB. 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: File is in use`
	`12: No resources available`
	`13: No access rights`
	`20: Other error`

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. **Example:** PROTFILE _N_PROTFILE _N_PROTFILE_MPF /_N_MPF_DIR_/_N_PROTFILE_MPF/
STRING	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

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

1.19 DELETE: Delete file

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.

Programming

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

Parameters

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: 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_MPF_DIR/_N_PROTFILE_MPF/

Example

```
N10 DEF INT ERROR
N15 STOPRE                                    ;Preprocessing stop
N20 DELETE (ERROR,                            ;deletes file TEST1 in the
 "/_N_SPF_DIR/_N_TEST1_SPF")                  ;subroutine branch
N30 IF ERROR
N40 MSG ("Error with DELETE command:"
<<ERROR)
N50 M0
N60 ENDIF
```

1.20 READ: Read lines in the file

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).

Programming

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

Parameter

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, 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).
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.

Examples

```
N10 DEF INT ERROR                           ;error variable
N20 STRING[255] RESULT[5]                   ;result variable
...
N30 READ(ERROR, "TESTFILE", 1, 5,           ;file name without domain and file
 RESULT)                                    ;identifier
...
N30 READ(ERROR, "TESTFILE_MPF", 1, 5,       ;file name without domain and with
 RESULT)                                    ;file identifier
...
N30 READ(ERROR,"_N_TESTFILE_MPF",1,5,       ;file name with domain and file
 RESULT)                                    ;identifier
...
N30 READ(ERROR,"/_N_CST_DIR/_N_TESTFILE     ;file name with domain and file
 _MPF", 1, 5 RESULT)                        ;identifier and path specification
^...
N40 IF ERROR <>0                            ;error evaluation
N50 MSG("ERROR "<<ERROR<<"
 WITH READ COMMAND")
N60 M0
N70 ENDIF
...
```

1.21 ISFILE: File present in the NCK user memory

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.

Programming

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

Parameters

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

```
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
...
```

1.22 FILEDATE/TIME/SIZE/STAT/INFO: File information

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.

Programming

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

Parameter

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/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 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

```
N10 DEF INT ERROR                           ;error variable
N20 STRING[32] RESULT                       ;result variable
...
N30 FILEINFO(ERROR, "TESTFILE", RESULT)     ;file name without domain and file
                                            ;identifier
...
N30 FILEINFO(ERROR, "TESTFILE_MPF",         ;file name without domain and with
 RESULT)                                    ;file identifier
...
N30 FILEINFO(ERROR,"_N_TESTFILE_MPF",       ;file name with domain and file
 RESULT)                                    ;identifier
...
N30 FILEINFO                                ;file name with domain and file
(ERROR,"/_N_MPF_DIR/_N_TESTFILE_MPF",       ;identifier and path specification
 RESULT)
...
N40 IF ERROR <>0                            ;error evaluation
N50 MSG("ERROR "<<ERROR<<"
 WITH FILE INFO COMMAND")
N60 M0
N70 ENDIF
...
Returns in the RESULT event variable: "77777 12345678 26.05.00 13:51:30"
```

1.23 CHECKSUM: Form the checksum over an array

Function

With CHECKSUM you form a checksum over an array.

Application:

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

Programming

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

Parameter

CHECKSUM	Form the checksum over an array
error	Error variable for return 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

```
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"
```

1.24 ROUNDUP: Round up

Function

The ROUNDUP function returns for

- positive input values
  the next larger integer
- negative input values
  the next smaller integer

If the input value is an integer type value (a whole number), the value is returned unmodified.

Programming

`ROUNDUP(Variable Real)`

Parameters

ROUNDUP	Rounds up to the next larger integer (observing the sign).
Variable	Input value of the type real
Real	Variables type for fractions containing decimal points

ROUNDUP in the NC parts program

```
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])
```

Examples

`ROUNDUP(3.1)` produces 4.0
`ROUNDUP(3.6)` produces 4.0
`ROUNDUP(-3.1)` produces -3.0
`ROUNDUP(-3.6)` produces -3.0
`ROUNDUP(3.0)` produces 3.0
`ROUNDUP(3)` produces 3.0

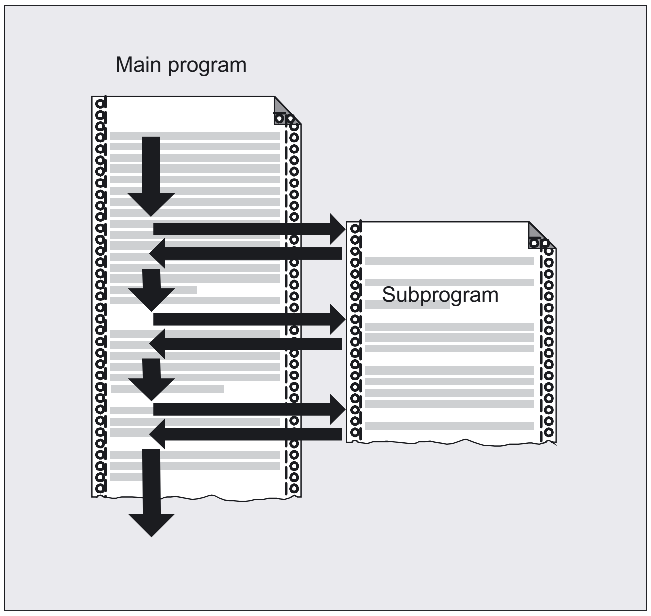
Subroutines, Macros 2

2.1 Using subroutines

Function

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

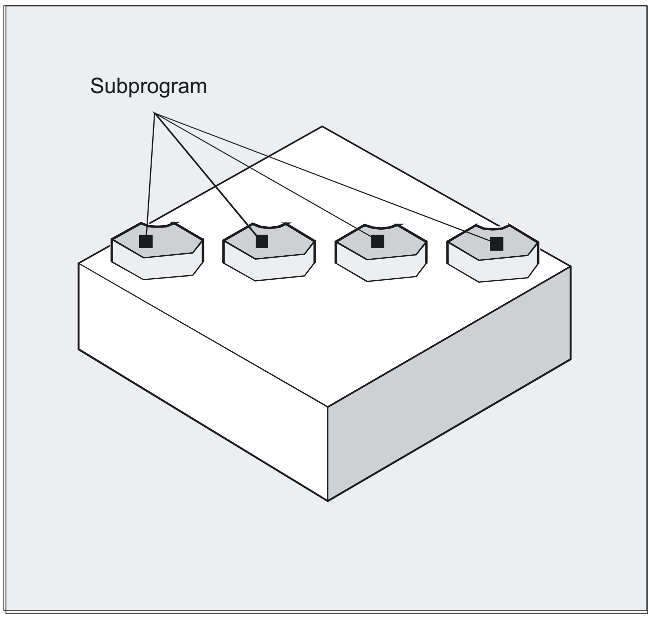
Basically, there is no difference between a main program and a subroutine. The subroutine contains either machining operations or sequences of operations that are to be performed several times.



Application

Machining sequences that recur are only programmed once in a subroutine. Examples include certain contour shapes, which occur repeatedly, and machining cycles.

The subroutine can be called and executed in any main program.



Structure of a subroutine

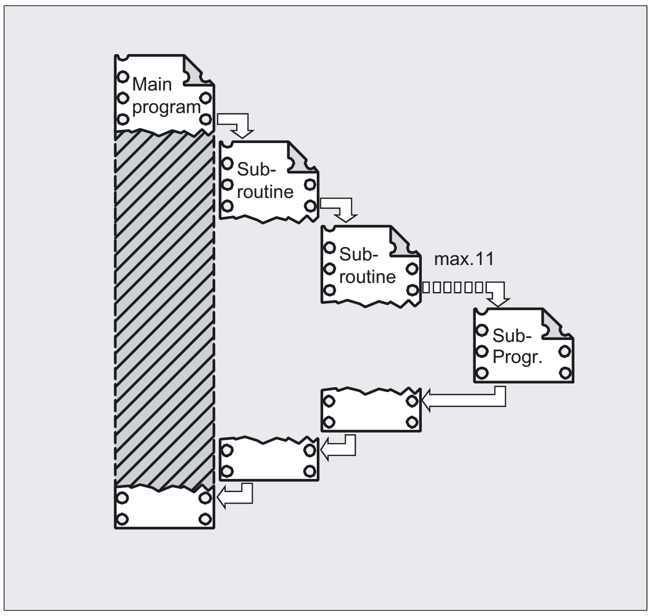
The structure of a subroutine is identical to that of the main program.

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

Nesting of subroutines

A subroutine can itself contain subroutine calls. This subroutine also contains another subroutine call, etc.
The maximum number of program levels or the nesting depth is 12.

This means: Up to 11 nested subroutine calls can be issued from the main program.



Restrictions on subroutines in interrupt routines and the cycle processing

It is also possible to call subroutines in interrupt routines. For work with subroutines you must keep four levels free or only nest seven subroutine calls.

For SIEMENS machining and measuring cycles you require three levels. If you call a cycle from a subroutine you must do this no deeper than level 5 (if four levels are reserved for interrupt routines).

2.2 Subroutines with SAVE mechanism

Function

For this, specify the additional command SAVE with the definition statement with PROC.

Programming

In the subroutine

```
PROC subroutine name SAVE
```

The SAVE attribute sets modal G functions to the same value at the end of subroutines that they had at the beginning. If this action results in a change to the

G function group 8 (settable zero offset)

or

G function group 52 (frame rotations of a rotational workpiece)

or

G function group 53 (frame rotation in direction of tool),

then the relevant frames are restored.

- The active basic frame is not changed when the subroutine returns
- The programmable zero offset is restored

Parameters

The behavior of the settable zero shift and the basic frame can be changed using the machine data MD 10617: FRAME_SAVE_MASK.

For more information on this, see
/FB1/ Function Manual Basic Functions; Axes, Coordinate Systems, Frames (K2), "Subroutine return with SAVE".

Example

Subroutine definition

```
PROC CONTOUR (REAL VALUE1) SAVE
N10 G91 …
N100 M17
```

Main program

```
%123
N10 G0 X… Y… G90
N20…
N50 CONTOUR (12.4)
N60 X… Y…
```

In the CONTOUR subroutine G91 incremental dimension applies. After returning to the main program, absolute dimension applies again because the modal functions of the main program were stored with SAVE.

2.3 Subroutines with parameter transfer (PROC, VAR)

Function

Program start, PROC

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

Subroutine end M17, RET

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

Programming

The parameters relevant for parameter transfer must be listed at the beginning of the subroutine with their type and name.

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

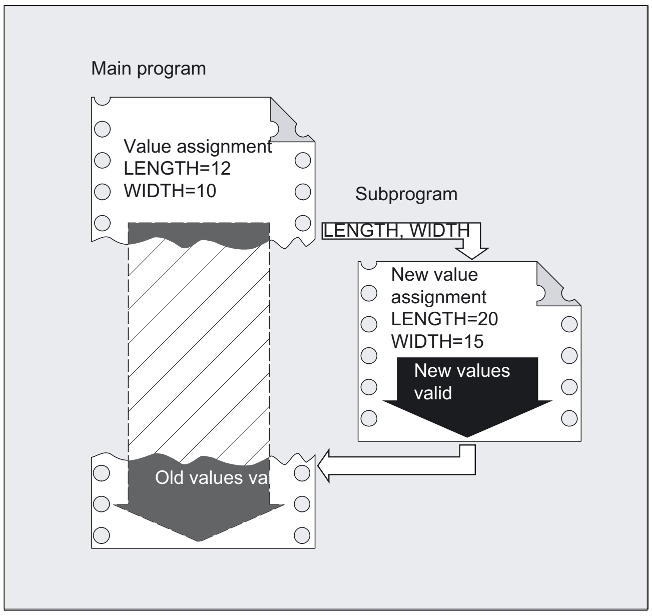
PROC	First instruction in a program
PROGRAM NAME	Subroutine name that should accept the relevant values of the parameters
VARIABLE_TYPE VARIABLE	Variable types with specification of the variable values. Several values can be specified.
VAR	Keyword for the type of the parameter transfer
FIELDNAME	Elements 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 subroutine

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



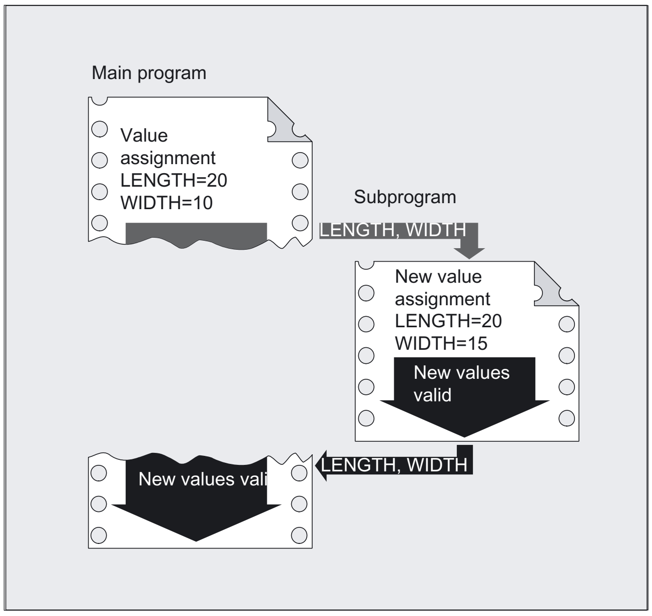
The values assigned in N20 in the main program are transferred in N30 when the subroutine is called. Parameters are transferred in the sequence stated.

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

Second method of parameter transfer:

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

If the parameters transferred are changed as the subroutine 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 subroutine also causes the parameter to change in the main program (see Fig.).

Example: variable array lengths

```
%_N_DRILLING_PLATE_MPF                           Main program
DEF REAL TABLE[100,2]                            ;Define position table
EXTERN DRILLING_PATTERN (VAR REAL[,2],INT)
TABLE[0,0]=-17.5                                 ;Define positions
…
TABLE[99.1]=45
DRILLING_PATTERN(TABLE,100)                      ;Subroutine call
M30
```

Example: creating a drilling pattern using a transferred variable-length position table

```
%_N_DRILLING_PATTERN_SPF                                Subroutine
PROC DRILLING_PATTERN(VAR REAL ARRAY[,2],->             ;Parameter delivery
-> INT NUMBER)
DEF INT COUNTER
STEP: G1 X=ARRAY[COUNTER,0]->                           ;Machining sequence
-> Y=ARRAY[COUNTER,1] F100
Z=IC(-5)
Z=IC(5)
COUNT=COUNT+1
IF COUNT<NUMBER GOTOB STEP
RET                                                     ;Subroutine end
```

Interruption of continuous-path mode

To prevent continuous-path mode from being interrupted:

Make sure the subroutine does **not** have the SAVE attribute. For further information about the SAVE mechanism, refer to the section, Subroutine with SAVE Mechanism.

RET must be programmed in a separate NC block.

```
PROC CONTOUR
N10…
…
N100 M17
```

Parameter transfer between main program and subroutine

If you are working with parameters in the main program, you can use the values calculated or assigned in the subroutine as well. For this purpose the values of the **current parameters** of the main program are passed to the **formal parameters** of the subroutine when the subroutine is called and then processed in subroutine execution.

Array definition

The following applies to the definition of the formal parameters: With two-dimensional arrays the number of arrays in the first dimension does not need to be specified, but the comma must be written.

Example:

```
VAR REAL ARRAY[,5]
```

With certain array dimensions it is possible to process subroutines with arrays of variable length. However, when defining the variables you must define how many elements it is to contain. The explanations of the array definition are contained in "Flexible NC Programming" in the array definition section with the same name.

2.4 Call subroutines (L or EXTERN)

Function

Calling subroutines without parameter transfer

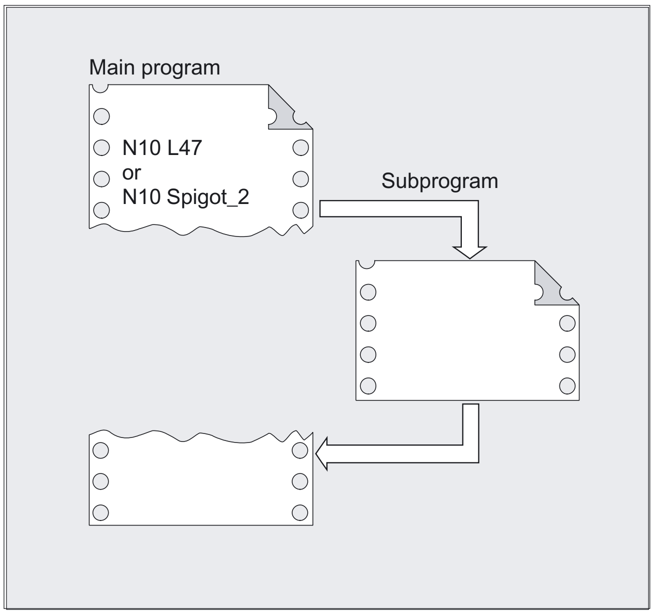
In the main program, you call the subroutine either

- with the L address and the subroutine number or
- with the program name.

Example:

`N10 L47` or

`N10 SPIGOT_2`



Programming

Subroutine with parameter transfer, explanation of EXTERN

`EXTERN`

Subroutines with parameter transfer must be listed with `EXTERN` in the main program before they are called, e.g., at the beginning of the program.

The name of the subroutine and the variable types are declared in the sequence in which they are transferred, see example.

Subroutines with parameter transfer

In the main program you call the subroutine by specifying the program name and parameter transfer. When transferring parameters you can transfer variables or values directly (not for VAR parameters), see example.

Parameters

`L address`	`Subroutine number`
`EXTERN`	`Broadcast a subroutine with specified parameters. You only have to specify EXTERN if the subroutine is in the workpiece or in the global subroutine directory. You do not have to declare cycles as EXTERN .`

Incomplete parameter transfer

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

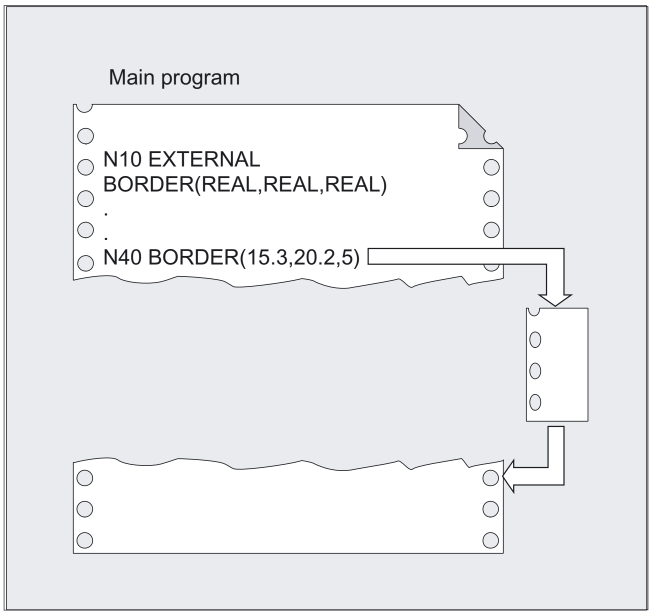
The comma must always be written to indicate the sequence. If the parameters are at the end of the sequence you can omit the comma as well.

Caution

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

Example: Subroutine with parameter transfer, declaration with EXTERN

```
N10 EXTERN BORDER(REAL, REAL, REAL)
…
N40 BORDER(15.3,20.2,5)
N10 Declaration of the subroutine, N40 Subroutine call with
parameter transfer.
```

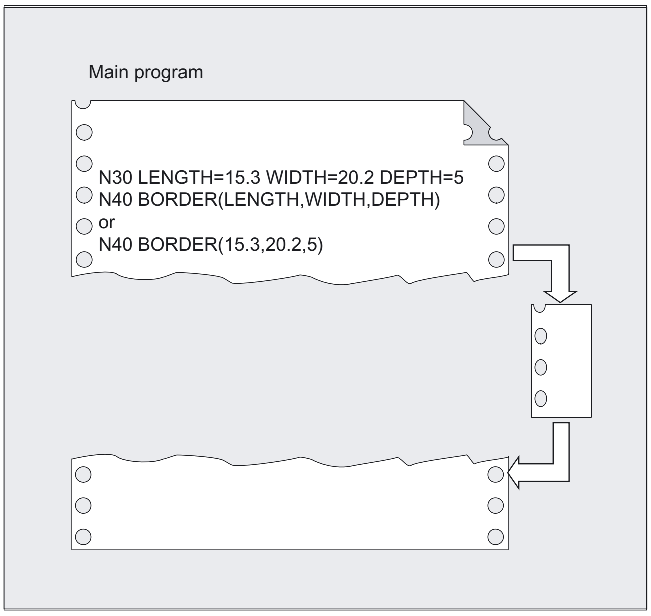


Example: Subroutine call with parameter transfer

```
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)
```


Main program
N30 LENGTH=15.3 WIDTH=20.2 DEPTH=5
N40 BORDER(LENGTH,WIDTH,DEPTH)
or
N40 BORDER(15.3,20.2,5)

Example: subroutine

```
PROC SUB1 (INT VAR1, DOUBLE VAR2)
IF $P_SUBPAR[1]==TRUE
;Parameter VAR1 was programmed in the subroutine call
ELSE
;Parameter VAR1 was not programmed in the subroutine call
;and initialized by the system with the default value 0
ENDIF
IF $P_SUBPAR[2]==TRUE
;Parameter VAR2 was programmed in the subroutine call
ELSE
;Parameter VAR2 was not programmed in the subroutine call
;programmed and initialized by the system with the default value 0.0
ENDIF
;Parameter 3 is not defined
IF $P_SUBPAR[3]==TRUE -> Alarm 17020
M17
```

Description

Caution

Subroutine definition corresponds to subroutine call

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

Definition in the subroutine:

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

Call in the main program:

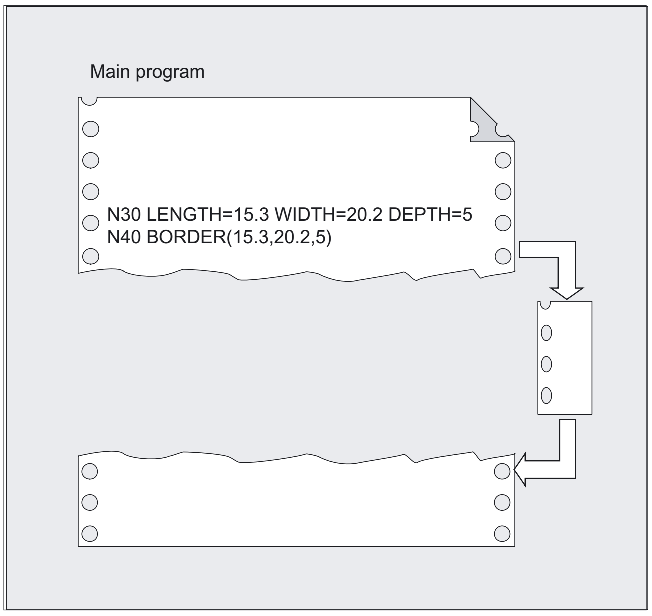
```
N30 BORDER(LENGTH, WIDTH, DEPTH)
```

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 subroutines 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, parts program processing is aborted with alarm output.

Call main program as subroutine

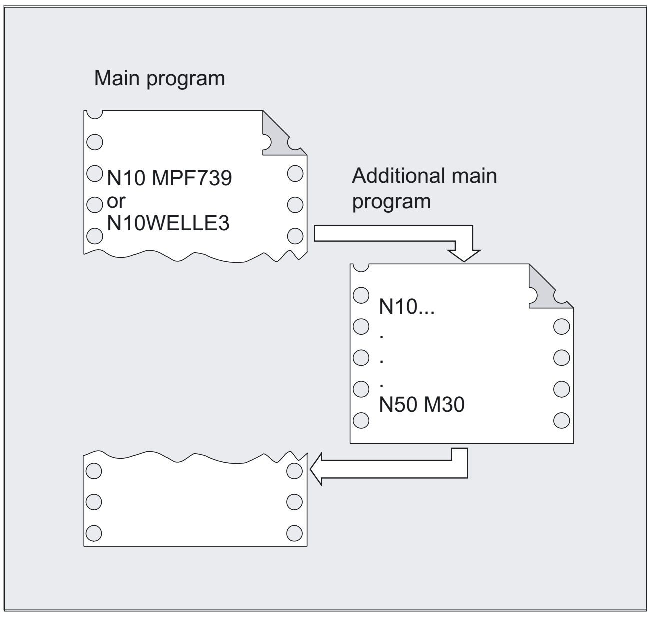
A main program can also be called as a subroutine. The end of program M2 or M30 set in the main program is evaluated as M17 in this case (end of program with return to the calling program).

You program the call specifying the program name.

Example:

`N10 MPF739` or

`N20 Shaft3`



Note

A subroutine can also be started as a main program.

2.5 Parameterized subroutine return (RET)

Function

Usually, a RET or M17 end of subroutine returns to the calling program and execution of the parts program continues with the lines following the subroutine call. However, some applications may require program resumption at another position:

- Continuation of execution after call-up of the cutting cycles in ISO dialect mode, after the contour definition.
- Return to main program from any subroutine level (even after ASUB) for error handling.
- Return over two or more program levels for special applications in compile cycles and in ISO dialect mode.

Programming

```
RET (<blocknumber/label>, <block after block with
blocknumber/label>,
<number of return levels>), <return to program start>)
```

or

```
RET (<block_number/label>, < >, < >)
```

or subroutine return over several levels

(return to the specified number of subroutine levels).

```
RET (, , <number of return levels>, <return to program start>)
```

Parameters

The parameterizable command RET can fulfill these requirements of the continuation or the return with 4 parameters:

1. <block_number/label>
2. <block after block with block number/label>
3. <number of return levels>
4. <return to beg. of program>

`RET`	Subroutine end (use instead of M17)
`<block_number/label>`	Parameter: Block number or label as STRING (constant or variable) of the block at which to resume execution. Execution is resumed in the calling program at the block with the "Block number/label".
`<block after block with block number/label>,`	Parameter of type INTEGER If the **value is greater than 0**, execution is resumed at "Block number/label". If the **value is equal to 0**, the subroutine return goes to the block with <block number/label>.
`<no_of_return_levels>,`	Parameter of type INTEGER with the permissible **values 1 to 11**. Value = 1: The program is resumed in the current program level -1 (like RET without parameters). Value = 2: The program is resumed in the current program level -2, skipping one level, etc.
`<return to beg. of program>,`	Parameter of type BOOL **Value 1 or 0**. Value = 1 If the return goes to the main program and ISO dialect mode is active there, execution will be resumed at the beginning of the program.

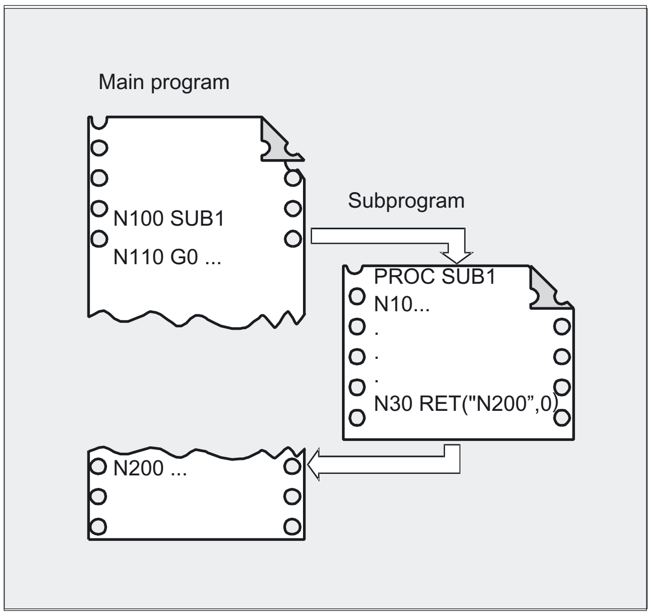
Example of error handling: Resumption in the main program after ASUP processing

```
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 2 (ASUB execution)
      ... RET("N10900", , ...               ; Program level 3
      N19100 RET(N10900,,$P_STACK)          ; Subroutine return
N10900                                      ; Resumption in main program
N10910 MCALL                                ; Deactivate modal subroutine
N10920 G0 G60 G40 M5                        ; Correct further modal settings
```

Description

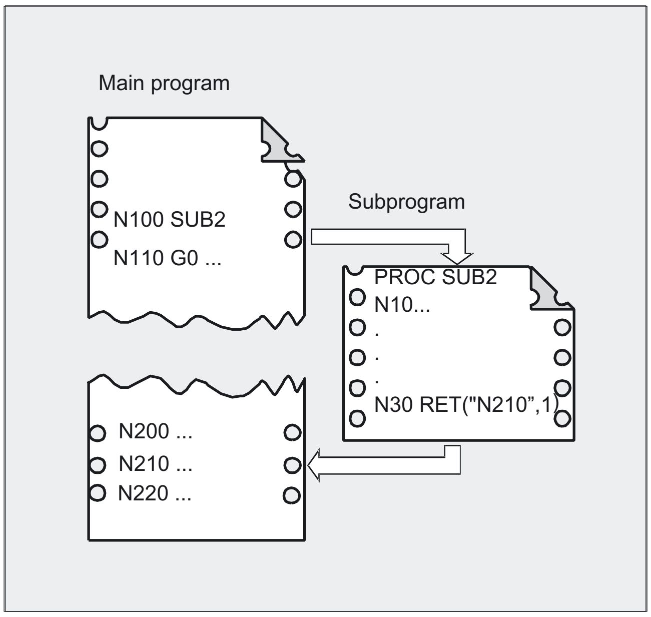
1. <block_number/label>

Execution is resumed in the calling program (main program) at the block with the <block number/label>.



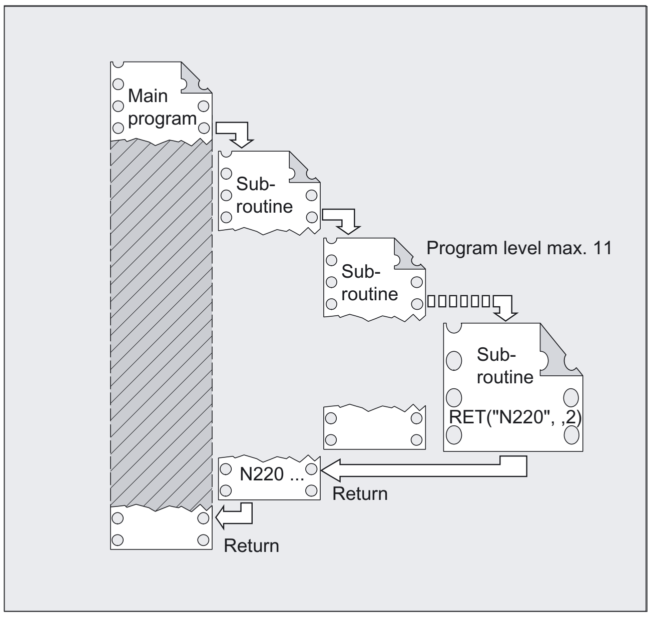
2. <block after block with block number/label>

The subroutine return goes back to the block with <block number/label>.



3. <number of return levels>

The program is resumed in the current program level minus <number of return levels>.



Impermissible return levels

If, for the number of return levels,

- a negative value or
- a value larger than the currently active program levels –(maximum 11)

is programmed, alarm 14091 is output with parameter 5.

Return with SAVE statements

On return over two or more program levels, the SAVE statements of each program level are evaluated.

Modal subroutine active on return

If a modal subroutine is active on a return over two or more program levels and if the deselection command MCALL is programmed for the modal subroutine in one of the skipped subroutines, the modal subroutine will remain active.

Caution

The user must **always ensure** that execution continues with the correct modal settings on return over two or more program levels. This is done, for example, by programming an appropriate main block.

2.6 Subroutine with program repetition (P)

Function

If a subroutine 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 subroutine call.

Parameters

Caution

Subroutine 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 subroutine.

`P`	`Number of subroutine passes`
`Value range:`	`1, ..., 9999 (unsigned integers)`

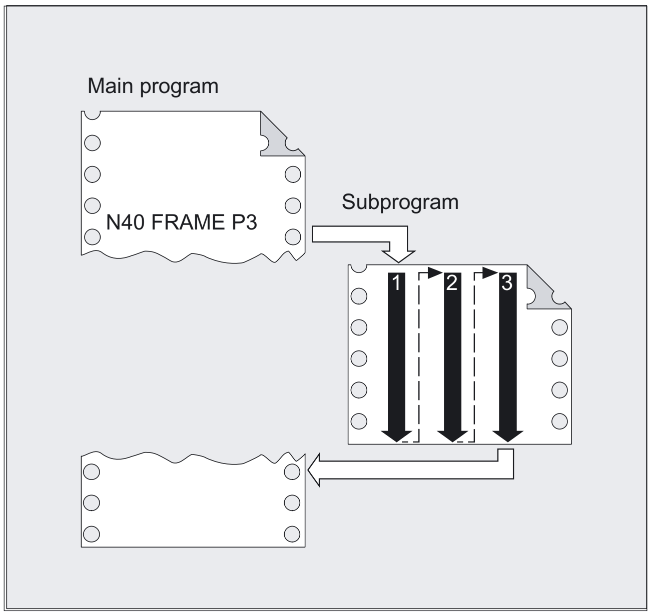
Caution

The following applies to every subroutine call:

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

Example

```
N40 FRAME P3
```



The subroutine FRAME must be executed 3 times in succession.

2.7 Modal subroutine (MCALL)

Function

This function causes the subroutine to be called and executed automatically after each block that contains traversing movement. In this way you can automate the calling of subroutines that are to be executed at different positions on the workpiece; for example, for the production of drilling patterns.

Deactivating the modal subroutine call

With MCALL without a subroutine call or by programming a new modal subroutine call for a new subroutine.

Parameters

MCALL	Modal subroutine call
L address	Subroutine number

Caution

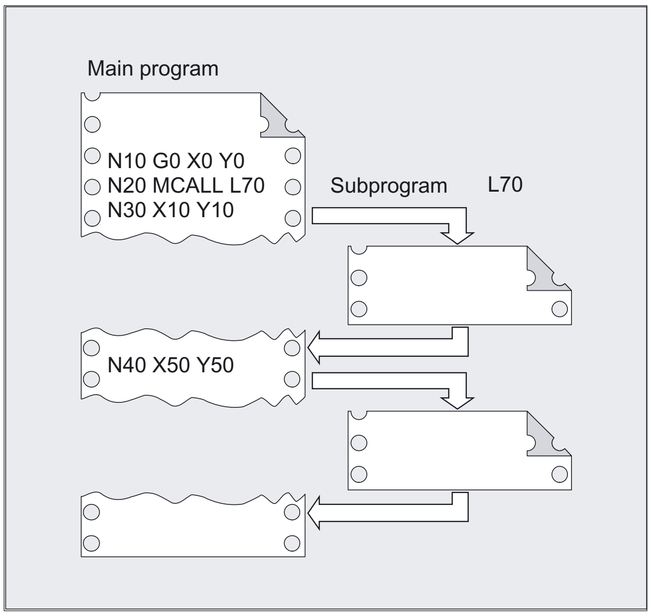
In a program run, only one MCALL call can apply at any one time. Parameters are only transferred once with an MCALL. In the following situations the modal subroutine is also called without motion programming: When programming the addresses S and F if G0 or G1 is active. G0/G1 is on its own in the block or was programmed with other G codes.

Example

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

In blocks N30 to N40, the program position is approached and subroutine L70 is executed.

```
N10 G0 X0 Y0
N20 MCALL L70
N30 L80
```



In this example, the following NC blocks with programmed path axes are in subroutine L80. L70 is called by L80.

2.8 Indirect subroutine call (CALL)

Function

Depending on the prevailing conditions at a particular point in the program, different subroutines can be called. The name of the subroutine is stored in a variable of type STRING. The subroutine call is issued with CALL and the variable name.

Programming

```
CALL <program name>
```

Parameters

CALL	Keyword for indirect subroutine call
<program_name>	Variable or constant of type string
	Name of the program containing the program section to run

Caution

The indirect subroutine call is only possible for subroutines without parameter transfer. For direct calling of the subroutine, store the name in a string constant

Example

Direct call with string constant

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

Indirect call via variable

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

The subroutine Part1 is assigned the variable PROGNAME. With CALL and the path name you can call the subroutine indirectly.

2.9 Repeating program sections with indirect programming (CALL)

Function

CALL is used to call up subroutines indirectly in which the program section repetitions defined with BLOCK are run according to the start label and end label.

Programming

```
CALL <program_name> BLOCK <start_label> TO <end_label>
CALL BLOCK <start_label> TO <end_label>
```

Parameters

`CALL`	Keyword for indirect subroutine call
`<program_name>` (option)	Variable or constant of type string, name of the program containing the program section to run. If no `<program_name>` is programmed, the program section with `<start_label>` `<end_label>` in the current program is searched for and run.
`BLOCK ... TO ...`	Keyword for indirect program section repetition
`<start_label>` `<end_label>`	Variable or constant of type string Refers to the beginning or end of the program section to run

Example

```
DEF STRING[20] STARTLABEL, ENDLABEL
STARTLABEL = "LABEL_1"
ENDLABEL = "LABEL_2"
...
CALL "CONTOUR_1" BLOCK STARTLABEL TO ENDLABEL ...
M17
PROC CONTOUR_1 ...
LABEL_1                                         ; Beginning of program section
                                                ; repetition
N1000 G1 ...
LABEL_2                                         ; End of program section repetition
```

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

Function

The indirect program call ISOCALL is used to call up a program in ISO language. The ISO mode set in the machine data is activated. At the end of the program, the original mode is reactivated. If no ISO mode is set in the machine data, the subroutine is called in Siemens mode.

For further information about the ISO mode, see
/FBFA/ ISO Dialects functional description.

Programming

```
ISOCALL <program_name>
```

Parameters

ISOCALL	Subroutine call with which the ISO mode set in the machine data is activated.
<program_name>	Variable or constant of type string Name of the program in ISO language

Example: Calling a contour with cycle programming from ISO mode

```
%_N_0122_SPF                               Contour description in ISO mode
N1010 G1 X10 Z20
N1020 X30 R5
N1030 Z50 C10
N1040 X50
N1050 M99
N0010 DEF STRING[5] PROGNAME = "0122"      ;Siemens parts program (cycle)
...
N2000 R11 = $AA_IW[X]
N2010 ISOCALL PROGNAME
N2020 R10 = R10+1                          ;Run program 0122.spf in ISO mode
N2300 ...
N2400 M30
```

2.11 Calling subroutine with path specification and parameters (PCALL)

Function

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

Programming

```
PCALL <path/program_name>(parameter 1, …, parameter n)
```

Parameters

PCALL	Keyword for subroutine call with absolute path name
<path_name>	Absolute path name beginning with "/", including subroutine names If no absolute path name is specified, PCALL behaves like a standard subroutine call with a program identifier. The program identifier is written without the leading _N_ and without an extension. If you want the program name to be programmed with the leading _N_ and the extension, you must declare it explicitly with the leading _N_ and the extension as Extern.
Parameters 1 to n	Current parameters in accordance with the PROC statement of the subroutine.

Example

```
PCALL/_N_WKS_DIR/_N_SHAFT_WPD/SHAFT(parameter1, parameter2, ...)
```

2.12 Extend search path for subroutine calls with CALLPATH

Function

The CALLPATH command is used to extend the search path for subroutine calls. That allows you to call subroutines from a non-selected workpiece directory without specifying the complete absolute path name of the subroutine.
Search path extension precedes the user cycle entry (_N_CUS-DIR).

Deselection of the search path extension

The search path extension is deselected with the following events:

- CALLPATH with empty string
- CALLPATH without parameters
- End of parts program
- Reset

Programming

Adding subroutines stored outside the existing NCK file system to the existing NCK file system.

CALLPATH <path_name>

Parameters

CALLPATH	Keyword for programmable search path extension. The CALLPATH command is programmed in a separate parts program line.
<path_name>	Variable or constant of type string. The field contains the absolute path of a directory beginning with "/" to extend the search path. The path must be specified complete with prefixes and suffixes (e.g. /_N_WKS_DIR/_N_WST_WPD). If <path_name> contains the empty string or if CALLPATH is called without parameters, the search path statement will be reset. The maximum path length is 128 bytes.

Note

CALLPATH checks whether the programmed path name really exists. An error aborts program execution with correction block alarm 14009.

Example

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

That sets this search path (position 5 is new):

1. current directory/subroutine identifier
2. current directory/subroutine identifier_SPF
3. current directory/subroutine identifier_MPF
4. /_N_SPF_DIR/subroutine identifier_SPF
5. **/_N_WKS_DIR/_N_MYWPD/subroutine identifier_SPF**
6. N_CUS_DIR/_N_MYWPD/subroutine identifier_SPF
7. /_N_CMA_DIR/subroutine identifier_SPF
8. /_N_CST_DIR/subroutine identifier_SPF

Note

CALLPATH can also be programmed in INI files. Then it applies for the duration of execution of 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 initialization program is then reset again.

2.13 Search path adaptation of the subroutines prepared during startup

Function

Machine data can be used to set a situation where the PROC instructions of the NC programs saved in the cycle directories are read prepared with parameters for the subroutine call during startup. The sequence for this is identical to that which the cycle directories searches when calling up the subroutine. The user cycles are thereby first addressed, followed by the manufacturer cycles and finally the standard cycles.

Application

The user or machine manufacturer can therefore copy NC programs from the standard cycles to then adapt these to their own requirements. The NC programs modified in this way can then be saved in a directory of the same name for user or manufacture cycles.

Machine manufacturer

All files which are prepared during startup can be identified with the PREPRO key word in the PROC instruction line with appropriately set machine data. Please see the machine manufacturer's specifications for further details.

Read subroutine with preparation and subroutine call

The cycle directories are addressed in the same order both for subroutines prepared with parameters during startup and during subroutine 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.

2.14 Execute external subroutine (EXTCALL)

Function

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

Programming

```
EXTCALL (<path/program_name>)
```

Parameters

`EXTCALL\`	Keyword for subroutine call
`<path/program_name>`	Constant/variable of type STRING An absolute path name or program name can be specified. The program name is written with/without the leading _N_ and without an extension. An extension can be appended to the program name using the <_> character.
Example:	
`EXTCALL ("/_N_WKS_DIR/_N_SHAFT_WPD/_N_SHAFT_SPF") or EXTCALL ("SHAFT")`	

Note

External subprograms are not permitted to include jump commands such as GOTOF, GOTOB, CASE, FOR, LOOP, WHILE or REPEAT.

IF-ELSE-ENDIF constructions are possible.

Subroutine calls and nested EXTCALL calls, may be used.

POWER ON, RESET

RESET and POWER ON cause external subroutine calls to be interrupted and the associated load memory to be erased. The reloading programs does however remain selected and is also immediately ready for SINUMERIK solution line systems for implementation as before with no limitations.

Example: HMI Advanced

The program to be reloaded is stored on the local hard disk of HMI Advanced.

In the setting data SD 42700: EXT_PROG_PATH is stored in the following path: "/_N_WKS_DIR/_N_WST1". The _N_MAIN_MPF main program is present in user memory and has been selected.

```
N10 PROC MAIN
N20 ...
N30 EXTCALL "ROUGHING"                           ; Call of external subroutine
                                                 ; ROUGHING
N40 ...
N50 M30
Subroutine "ROUGHING" (located in the HMI Advanced directory structure under
workpieces->WST1):
N10 PROC ROUGHING
N20 G1 F1000
N30 X=... Y=... Z=...
N40 ...
N90 M17
```

Example: HMI embedded powerline

The program to be reloaded is located on the network drive or ATA card

EXTCALL Windows path name

Call for ATA card (HMI embedded only for SINUMERIK powerline systems) e.g.
EXTCALL C:\Werkstücke\Kontur2.spf

Example: HMI Advanced or HMI Embedded with network drive option

Call for network drive (HMI Embedded or HMI Advanced)
EXTCALL \\R4711\Workpieces\Contour.1.spf

External program path

The call path can be set flexibly in SD 42700: EXT_PROG_PATH. SD 42700 contains a path definition that builds the absolute path name of the program to be called in conjunction with the programmed subroutine identifier.

Call of external subroutine for SINUMERIK with HMI Advanced

An external subroutine is called by means of parts program command **EXTCALL**.
From the

- subroutine names programmed with EXTCALL and
- setting data SD 42700: EXT_PROG_PATH provides the program path for the external subroutine call by the concatenation of
- the content of SD 42700: EXT_PROG_PATH (e.g. /_N_WKS_DIR/_N_WKST1_WPD)
- the character "/" as a separator
  (if a path has been specified via SD 42700: EXT_PROG_PATH)
- the subroutine path or subroutine identifier specified with EXTCALL.

SD 42700: EXT_PROG_PATH is a blank. If the external subroutine is called without an absolute path name, the same search path is executed on the HMI Advanced as for calling a subroutine from NCK memory.

1. current directory / subroutine identifier
2. current directory / subroutine identifier_SPF
3. current directory / subroutine identifier_MPF
4. /_N_SPF_DIR / subroutine identifier_SPF
5. /_N_CUS_DIR / subroutine identifier_SPF
6. /_N_CMA_DIR / subroutine identifier_SPF
7. /_N_CST_DIR / subroutine identifier_SPF

"current directory" represents the directory in which the main program has been selected
"subroutine designation" represents the subroutine identifier programmed in
EXTCALL.

Note

The following applies to HMI embedded powerline:

An absolute path must always be specified in HMI Embedded.

The following applies to HMI embedded solution line:

Specified programs with EXTCALL and an absolute path are only used if the specified target also exists. Program execution is canceled if a target is not found. This is the case, for example, if the path is specified for a network drive with a mandatory option which is not present.

Execution from external program saving

There are differences between how an external program is reloaded from an available memory. These depend on the HMI operating panel used. The following memory variants are available for SINUMERIK powerline and solution line systems:

- Process network drive as additional memory for HMI Embedded and Advanced.
- Process hard disk as option only for HMI Advanced powerline and solution.
- Process ATA card or via V.24 interface especially for HMI Embedded powerline.
- Process CompactFlash Card or USB drive for HMI solution line systems.

Programs to be reloaded can be saved with the existing option both on a network drive and a CompactFlash card. An external user memory connected to the TCU is only fully supported via USB interface X203. We cannot recommend using a USB FlashDrive as a persistent memory.

Please note the relevant operating processes in the corresponding instructions.

References
/BEM/ HMI Embedded powerline or solution line
/BAD/ HMI Advanced powerline or solution line

Adjustable load memory (FIFO buffer)

A load memory is required in the NCK in order to process a program in "Execution from external" mode (main program or subroutine). 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 buffer must be set for each program (main program or subroutine) to run concurrently in "Processing from external source" mode.

Machine manufacturer

Please contact the machine manufacturer if the size and number of reloading buffers is to be extended. For further information about "Processing from external source", see /FB1/ Function Manual, Basic Functions; BAG, Channel, Program Operation Mode (K1).

2.15 Subroutine call with M, T and D functions

Function

T, M and D functions can be replaced with a subroutine call by making the appropriate machine data settings. This method can be used, for example, to call the tool change routine. During block search with calculation and SERUPRO, subroutine calls with M, T and D functions are processed in the same way as standard subroutine calls.

Example: tool change with M6

M function M6 is replaced by tool change routine TC_UP_M6.

```
N10 PROC ROUGHING3
N20 G1 F1000
N30 X=... Y=... Z=...
N40 T1234 M6 ;                           ; Call TC_UP_M6
M30
Associated subroutine TC_UP_M6:
N110 PROC TC_UP_M6
...
N130 G53 D0 G0 X=... Y=... Z=... ;       ; Approach tool change point
N140 M6 ;                                ; Execute tool change
...
N190 M17
```

For further information about "Subroutine calls using M, T and D functions" see
/FB1/ Function Manual, Basic Functions; BAG, Channel, Program Operation Mode (K1).

2.16 Suppress individual block (SBLOF, SBLON)

Function

Program-specific single block suppression

For all single block types, the programs marked with SBLOF are executed in their entirety like one block. SBLOF is written in the PROC line and is valid until the end of the subroutine or until it is aborted. At the return command, the decision is made whether to stop at the end of the subroutine.

Return jump with M17: Stop at the end of the subroutine

Return jump with RET: No stop at the end of the subroutine

SBLOF is also valid in subroutines, which are called.

Example for subroutine without stop in single block:

```
PROC EXAMPLE SBLOF
G1 X10
RET
```

Programming

PROC ... SBLOF	; Command can be programmed in a PROC or a separate block
SBLON	; The command must be programmed in a separate block

Single block suppression in the program

SBLOF must be alone in a block. Single block is deactivated after this block until

- the next SBLON or
- the end of the active subroutine level.

Parameters

SBLOF	Deactivate single block
SBLON	Reactivate single block

Example: single block suppression in the program

```
N10 G1 X100 F1000
N20 SBLOF
N30 Y20                                     ;Deactivate single block
N40 M100
N50 R10=90
N60 SBLON
N70 M110                                    ;Reactivate single block
N80 ...
```

The area between N20 and N60 is executed as one step in single block mode.

Example: cycle is to act like a command for a user

Main program

```
N10 G1 X10 G90 F200
N20 X-4 Y6
N30 CYCLE1
N40 G1 X0
N50 M30
Program cycle:1
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 an active single block, i.e., the Start key must be pressed once for machining with CYCLE1.

Example: an ASUP, which is started by the PLC in order to activate a modified zero offset and tool offsets, is to be executed invisibly.

```
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: use MD 10702 IGNORE_SINGLEBLOCK_MASK, bit 12 = 1 to prevent stopping

In single block type SBL2 (stop at each parts program line) in the SBLON statement.

```
;SBL2 is active
;$MN_IGNORE_SINGLEBLOCK_MASK = 'H1000'        ;In the MD 10702: set bit 12 = 1
N10 G0 X0                                     ;Stop at this parts program line
N20 X10                                       ;Stop at this parts program line
N30 CYCLE                                     ;Traversing block generated by the
                                              ;cycle
 PROC CYCLE SBLOF                             ;Suppress single block stop
 N100 R0 = 1
 N110 SBLON                                   ;Because of MD 10702: bit 12 = 1
                                              ;prevents stopping
 N120 X1                                      ;Stop at this parts program line
 N140 SBLOF
 N150 R0 = 2
 RET
N50 G90 X20                                   ;Stop at this parts program line
M30
```

Example: single block suppression for program nesting

```
                                              ;Single block is active
N10 X0 F1000                                  ;Stop at this block
N20 UP1(0)
     PROC UP1(INT _NR) SBLOF                  ;Single block OFF
     N100 X10
     N110 UP2(0)
           PROC UP2(INT _NR)
           N200 X20
           N210 SBLON                         ;Single block ON
           N220 X22                           ;Stop at this block
           N230 UP3(0)
                 PROC UP3(INT _NR)
                 N302 SBLOF                   ;Single block OFF
                 N300 X30
                 N310 SBLON                   ;Single block ON
                 N320 X32                     ;Stop at this block
                 N330 SBLOF                   ;Single block OFF
                 N340 X34
                 N350 M17                     ;SBLOF active
           N240 X24                           ;Stop at this block
                                              ;SBLON active
           N250 M17                           ;Stop at this block
                                              ;SBLON active
     N120 X12
     N130 M17                                 ;Stop at this return block
                                              ;SBLOF of the PROC statement active
N30 X0                                        ;Stop at this block
N40 M30                                       ;Stop at this block
```

Restrictions

- The current block display can be suppressed in cycles using DISPLOF.
- If DISPLOF is programmed together with SBLOF, the cycle call continues to be displayed on single block stops within the cycle.
- If the single block stop in the system ASUB or the user ASUB is suppressed with Bit0 = 1 or Bit1 = 1 for MD 10702: IGNORE_SINGLEBLOCK_MASK, the SBLON in the ASUB can be programmed to reactivate the single block stop.
- The single block stop in the user ASUB is suppressed with MD 20117: IGNORE_SINGLEBLOCK_ASUP and can no longer be activated by programming the SBLON.
- By selecting SBL3 you can suppress the SBLOF command.
- Ignore single block stop in the single block type 2. Single block type 2 (SBL2) does **not** stop in the SBLON block, if Bit12 = 1 is set in MD 10702: IGNORE_SINGLEBLOCK_MASK.

Note

Further information about the block display with/without single block suppression, see /FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1), "Single Block" chapter.

Single block disable for unsynchronized subroutines

To run an ASUB in single block mode in one step, the ASUB must contain a PROC statement with SBLOF. This also applies to the function "editable system ASUB" in MD 11610: ASUP_EDITABLE.

Example of "editable system ASUP":

```
N10 PROC ASUB1 SBLOF DISPLOF
N20 IF $AC_ASUP=='H200'
N30 RET                                       ;No REPOS on mode change
N40 ELSE
N50 REPOSA                                    ;REPOS in all other cases
N60 ENDIF
```

Program control in single block mode

With the single block function, the user can process a parts program block by block. The single block function has the following settings:

- SBL1: IPO single block with stop after each machine function block.
- SBL2: Single block with stop after each block.
- SBL3: Stop in the cycle (by selecting SBL3 you can suppress the SBLOF command).

Single block suppression for program nesting

If SBLOF is programmed in the PROC statement in a subroutine, stopping is performed on the subroutine return jump with M17. That prevents the next block in the calling program from already running. If single block suppression is activated with SBLOF (without SBLOF in the PROC statement), execution stops after the next machine function block of the calling program. If that is not wanted, SBLON must be programmed in the subroutine before the return (M17). Execution does not stop on a return to a higher-level program with RET.

2.17 Suppress current block display (DISPLOF)

Function

DISPLOF suppresses the current block display for a subroutine. DISPLOF is placed at the end of the PROC statement. Instead of the current block, the call of the cycle or the subroutine is displayed.

By default the block display is activated. Deactivation of block display with DISPLOF applies until the return from the subroutine or end of program.

Programming

```
PROC … DISPLOF
```

If further subroutines are called from the subroutine with the DISPLOF attribute, the current block display is suppressed in these as well. If a subroutine with suppressed block display is interrupted by an unsynchronized subroutine, the blocks of the current subroutine are displayed.

Parameters

DISPLOF	Suppress current block display

Example: suppress current block display in the cycle

```
%_N_CYCLE_SPF                                   ;$PATH=/_N_CUS_DIR
PROC CYCLE (AXIS TOMOV, REAL POSITION) SAVE DISPLOF
                                                ;Suppress current block display
                                                ;Now the cycle call is displayed as
                                                ;the current block
                                                ;e.g.: CYCLE(X, 100.0)
DEF REAL DIFF                                   ;Cycle contents

G01 …
…
RET                                             ;Subroutine return, the following
                                                ;block of the calling program is
                                                ;displayed again
```

2.18 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

Programming

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:

PROC MY_CYCLE_1 (REAL PAR1, INT PAR2, CHAR PAR3, STRING[10] PAR4)	
;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 subroutine name
...	
M17	

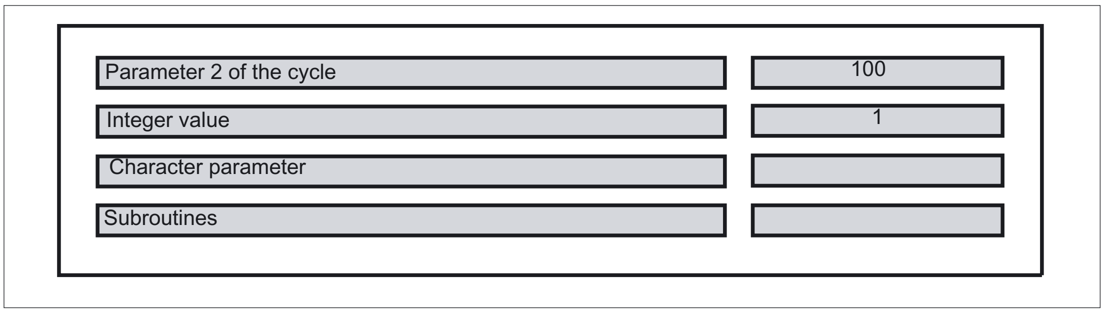
PROC SPECIAL CYCLE (REAL VALUE1, INT VALUE2)	
;The cycle has the following transfer parameters:	
;	
;VALUE1:	Real value without value range limitation and default
;VALUE2:	Integer value without value range limitation and default
...	
M17	

Associated file uc.com:

```
%_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///Subroutine name)

//C26(SPECIALCYCLE)
(R///Entire length)
(I/*123456/3/Machining type)
M17
```

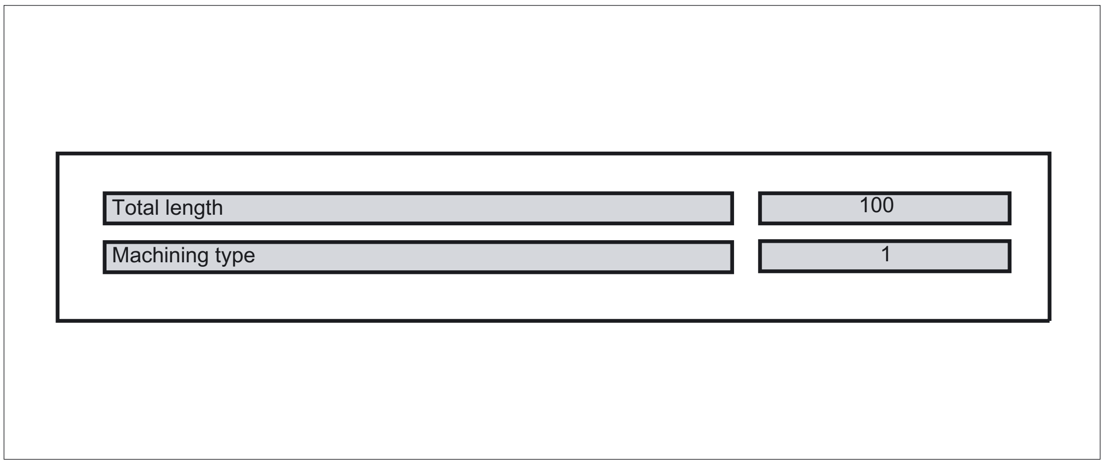
Example: both cycles

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.

2.19 Macro technique (DEFINE...AS)

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 subroutine 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.

Use of macros

Sequences of statements that recur are only programmed once as a macro in a separate macro module and once at the beginning of the program. The macro can then be called in any main program or subroutine and executed.

Programming

Macros are identified with the keyword DEFINE...AS.

The macro definition is as follows:

```
DEFINE NAME AS <statement>
```

Example:

Macro definition:

```
DEFINE LINE AS G1 G94 F300
```

Call in the NC program:

```
N20 LINE X10 Y20
```

Activate macro

The macro is active when it is loaded into the NC ("Load" soft key).

Parameters

Caution

Keywords and reserved names must not be redefined with macros.

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

DEFINE	Define macro
NAME	This is the name of the macro
AS	STRING macro definition
Statement	Programming statements, e.g., G, M, H and L functions

With macros you can define any identifiers, G, M, H functions and L program names. Two-digit H and L functions can be programmed.

Three-digit M/G function

Supports programming of three-digit M and G functions.

Example:

```
N20 DEFINE M100 AS M6
N80 DEFINE M999 AS M6
```

Note

Macros can also be declared in the NC program. Only identifiers are permissible as macro names. G function macros can only be defined in the macro module globally for the entire control.

Nesting of macros is not possible.

Example: macro definitions

`DEFINE M6 AS L6`	A subroutine is called at tool change to handle the necessary data transfer. The actual M function is output in the subroutine (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: macro file

After reading the macro file into the control, activate the macros (see above). The macros can now be used in the parts program.

```
%_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 right, coolant on
DEFINE M14 AS M4 M7             ;Spindle left, coolant on
DEFINE M15 AS M5 M9             ;Spindle stop, coolant off
DEFINE M6 AS L6                 ;Call tool change program
DEFINE G80 AS MCALL             ;Deselect drilling cycle
M30
```

File and Program Management 3

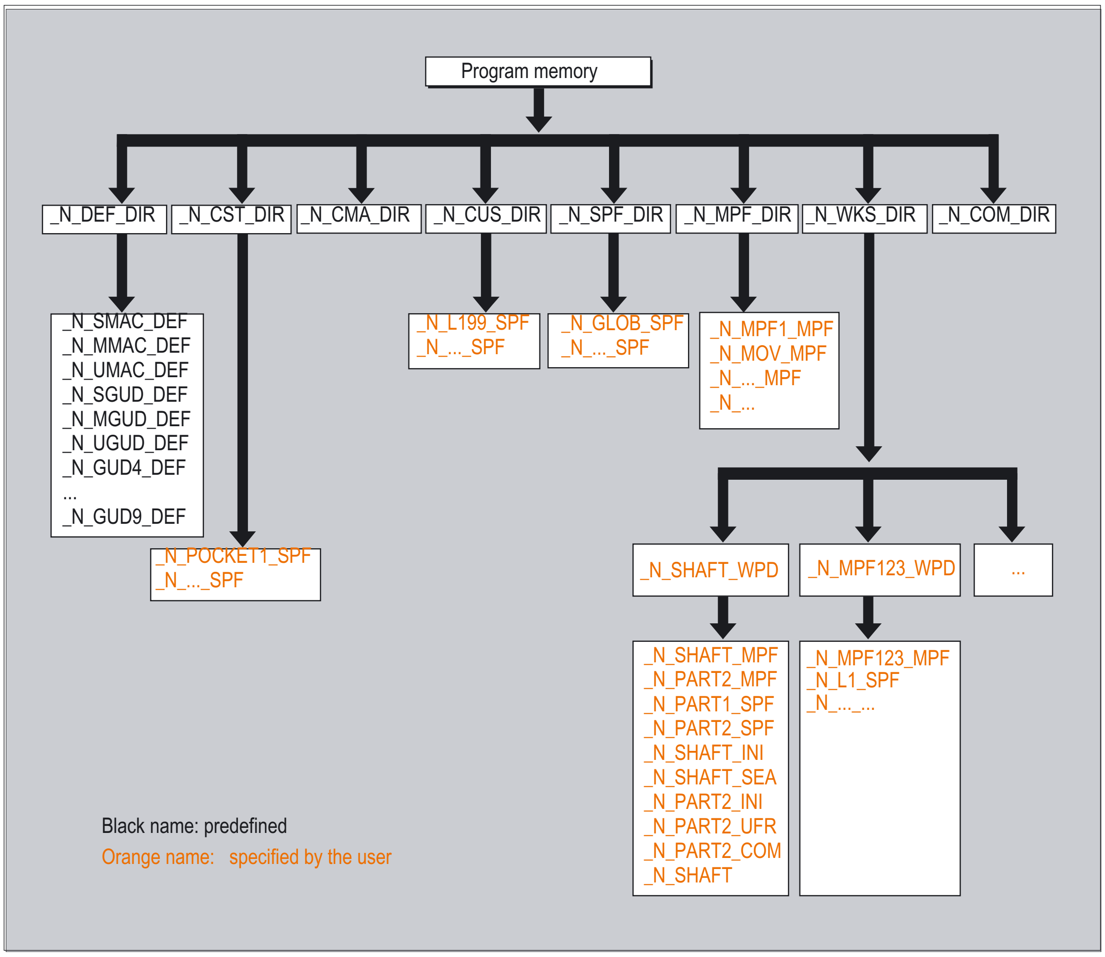
3.1 Program memory

Function

The files and programs are stored in the program memory and are thus permanently stored (passive file system).

Example: Main programs and subroutines, macro definitions.

Main programs and subroutines are stored in the main memory. 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).



Directories

Its standard complement of directories is as follows:

1.	_N_DEF_DIR	Data modules and macro modules
2.	_N_CST_DIR	Standard cycles
3.	_N_CMA_DIR	Manufacturer cycles
4.	_N_CUS_DIR	User cycles
5.	_N_WKS_DIR	Workpieces
6.	_N_SPF_DIR	Global subroutines
7.	_N_MPF_DIR	Standard directory for main programs
8.	_N_COM_DIR	Standard directory for comments
9.	_N_EXT_DIR	External parts program memory

File types

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

*name*_MPF	Main program
*name*_SPF	Subroutine
*name*_TEA	Machine data
*name*_SEA	Setting data
*name*_TOA	Tool offsets
*name*_UFR	Zero offsets/frames
*name*_INI	Initialization files
*name*_GUD	Global user data
*name*_RPA	R parameters
*name*_COM	Comment
*name*_DEF	Definitions for global user data and macros

Description

Workpiece directory, _N_WKS_DIR

The workpiece directory exists in the standard setup of the program directory under the name `_N_WKS_DIR`. The workpiece directory contains all the workpiece directories for the workpieces that you have programmed.

Workpiece directories, identification 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, subroutines, any initialization programs and comment files.

Initialization programs are executed once as specified in the machine data
MD 11280: WPD_INI_MODE after the program selection with the first parts program start.

Example: The workpiece directory `_N_SHAFT_WPD`, created for workpiece `SHAFT` contains the following files:

_N_SHAFT_MPF	Main program
_N_PART2_MPF	Main program
_N_PART1_SPF	Subroutine
_N_PART2_SPF	Subroutine
_N_SHAFT_INI	General initialization program for the data of the workpiece
_N_SHAFT_SEA	Setting data initialization program
_N_PART2_INI	General initialization program for the data for the Part 2 program
_N_PART2_UFR	Initialization program for the frame data for the Part 2 program
_N_SHAFT_COM	Comment file

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.

;$PATH statement

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

Example:

```
;$PATH=/_N_WKS_DIR/_N_SHAFT_WPD
```

The file is stored at the specified path.

Note

If the path is missing, files of file type SPF are stored in `/_N_SPF_DIR`, files with extension `_INI` in the working memory and all other files in `/_N_MPF_DIR`.

Example with the path for the previous example:

`SHAFT: _/N_WELLE_MPF` is stored in `/_N_WKS_DIR/_N_WELLE_WPD`

```
%_N_SHAFT_MPF
;$PATH=/_N_WKS_DIR/_N_SHAFT_WPD
N40 G0 X… Z…
●
M2
```

`SHAFT: _/N_WELLE_SPF` is stored in `/_N_SPF_DIR`

•

%_N_SHAFT_SPF

```
●
M17
```

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.

Example:

The workpiece directory `/_N_WKS_DIR/_N_SHAFT_WPD` contains the files `_N_SHAFT_SPF` and `_N_SHAFT_MPF`.

HMI Advanced only: See "Operator's Guide" /BAD/ "Job list" and "Selecting a program for execution".

Search paths for subroutine call

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

Example of subroutine call with absolute path specification:

```
CALL"/_N_CST_DIR/_N_CYCLE1_SPF"
```

Programs are usually called without specifying a path:

Example:

```
CYCLE1
```

Search path order

1.	Current directory / *name*	Workpiece directory or standard directory _N_MPF_DIR
2.	Current directory / *name_SPF*	
3.	Current directory / *name_MPF*	
4.	/_N_SPF_DIR / *name_SPF*	Global subroutines
5.	/_N_CUS_DIR / *name_SPF*	User cycles
6.	/_N_CMA_DIR / *name_SPF*	Manufacturer cycles
7.	/_N_CST_DIR / *name_SPF*	Standard cycles

Programming search paths for subroutine call

CALLPATH command

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

Example:

```
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 subroutine calls with CALLPATH, see section "Extending the search path for subroutine calls with CALLPATH".

3.2 Working memory

Function

The working memory contains the current system and user data with which the control operates (active file system).

Example: Active machine data, tool offset data, zero offsets.

Parameters

Initialization programs

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

*name*_TEA	Machine data
*name*_SEA	Setting data
*name*_TOA	Tool offsets
*name*_UFR	Zero offsets/frames
*name*_INI	Initialization files
*name*_GUD	Global user data
*name*_RPA	R parameters

Data areas

The data can be organized in different areas in which they are to apply. For example, a control can use several channels
(not 810D CCU1, 840D NCU 571) and can usually use several axes. The following exist:

Identifier	Data areas
NCK	NCK-specific data
CHn	Channel-specific data (n specifies the channel number)
AXn	Axis-specific data (n specifies the number of the machine axis)
TO	Tool data
COMPLETE	All data

Example: create initialization program on the 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.

Example: procedure for multi-channel controls

CHANDATA (channel number) for multiple channels is permitted only in the `N_INITIAL_INI` file.

`N_INITIAL_INI` is the installation file with which all data of the control is initialized.

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

Caution

CHANDATA statement

In the parts program, the CHANDATA statement may only be used for the channel on which the NC program is running, i.e. the statement can be used to protect NC programs from being executed accidentally on a different channel.

Program processing is aborted if an error occurs.

Note

INI files in job lists do not contain any CHANDATA statements.

Saving the initialization programs

The files in the working memory can be saved on an external PC and read in again from there.

- The files are saved with `COMPLETE`.
- `INITIAL` is used to create an INI file over all areas: `_N_INITIAL_INI`.

Loading initialization programs

INI programs can also be selected and called as parts programs if they only use the data of a single channel. It is thus also possible to initialize program-controlled data.

Information on file types is given in the Operator's Guide.

3.3 Defining user data

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.

The user data definition (GUD) can be prepared in the Services operating area of the HMI user interface. 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.

Programming

The GUD variables are programmed with the DEF command:

```
DEF range preprocessing stop type name[.., ...]=value
```

Parameters

range	Range identifies the variable as a GUD variable and defines its validity scope: NCK NCK-wide CHAN channel-wide
preprocessing_stop	Optional attribute preprocessing stop: SYNR Preprocess stop while reading SYNW Preprocess stop while writing SYNRW Preprocess stop while reading/writing
Type	Data type BOOL REAL INT AXIS FRAME STRING CHAR
name	Variable name
[.., ...]	Optional run limits for array variables
Value	Optional initialization value, several values for arrays, each separated with a comma or REP (w1), SET(w1, W2, ...), (w1, w2, ...) Initialization values are not possible for type Frame.

Example: definition file, global data (Siemens)

```
%_N_SGUD_DEF
;$PATH=/_N_DEF_DIR
DEF NCK REAL RTP                            ;Retraction plane
DEF CHAN INT SDIS                           ;Safety clearance
M30
```

Example: definition file, global data (machine manufacturer)

```
%_N_MGUD_DEF
;$PATH=/_N_DEF_DIR
;Global data definitions of the machine manufacturer
DEF NCK SYNRW INT QUANTITY                  ;Implicit preprocess stop while
                                            ;reading/writing
                                            ;Specific data present in the control
                                            ;Access from all channels
DEF CHAN INT TOOLTABLE[100]                 ;Tool table for channel-spec. View of
                                            ;the tool number at magazine locations
M30                                         ;Separate table created for each channel
```

Reserved block names

The following modules 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 no measuring cycles / standard cycles are present, the modules reserved for them can be freely defined.

1. **Defining user data (GUD)**

1. Save module `_N_INITIAL_INI`.
2. Create a definition file for user data in the Services HMI operating area
3. Load definition file into the program memory of the control
4. Activating definition files
5. Data backup

6. **Creating a definition file for user data**

   Definition files can be prepared on the external PC or in the Services operating area. Predefined file names also exist (also see "Reserved module 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.

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

   The control always creates a default 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.

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

   When the GUD definition file is loaded into the NC ("Load" soft key), it becomes active. See "Automatic activation ..." 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 dialogue HMI 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.

9. **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 archived separately.
   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.

3.4 Protection levels for user data, MD, SD and NC commands

3.4.1 Defining protection levels for user data (GUD)

Function

Access criteria can be defined for GUD modules to protect them against manipulation. In cycles GUD variables can be queried that are protected in this way from change via the HMI user 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.

Programming

Protection levels for the whole module are specified in the headers.

```
%_N_MGUD_DEF                                ;module type
;$PATH=/_N_DEF_DIR                          ;path
APR value APW n                             ;protection levels in separate line
```

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.

Parameters

Protection level:	Access protection (**A**ccess **P**rotection)
APW n	for writing (**W**rite)
APR n	for reading (**R**ead)
n	Protection level n
	from 0 or 10 (highest level)
	to 7 or 17 (lowest level)
Meaning of the protection levels n:	
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
APW 0-7, APR 0-7 The read and write protection acts on the user interface **and** in the NC program or in the MDA operation.	These values are permissible in GUD modules and in protection levels for individual variables in the REDEF instruction.
APW 10-17, APR 10-17: The read and write protection acts here on the user interface.	This values are only permissible for module-specific GUD protection level.

Note

To protect a complete file, the commands must be placed before the first definitions in the file. In other cases, they go into the **REDEF** instruction of the relevant data, see section "Protection levels for NC commands".

Example: definition file with write access protection

(Machine manufacturer), read (key switch 2 on the user interface):

```
%_N_GUD6_DEF
;$PATH=/_N_DEF_DIR
APR 15 APW 12                       ;Protection levels for all following
                                    ;variables
DEF CHAN REAL_CORRVAL
DEF NCK INT MYCOUNT
…
M30
```

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

10. The file _N_GUD7_DEF has value 3 as write protection. The value 3 is then overwritten with value 2.
11. 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.

3.4.2 Automatic activation of GUDs and MACs

Function

Definition files for GUD and macro definitions for HMI Advanced are edited in the Services operating 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: prompt on exiting the editor

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

"OK": A prompt appears that asks whether you want the save the currently active data.

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

3.4.3 Change the protection data for the machine and setting data (REDEF MD, SD)

Function

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

Programming

```
REDEF Machine data/setting data protection level
```

Parameters

`REDEF`	`Redefinition (`**`REDEF`**`inition) e.g. Set the machine and setting data`
`Machine data / setting data`	`Machine data or setting data to which a protection level is to be assigned.`
`Protection level:`	`Access protection (`**`A`**`ccess `**`P`**`rotection)`
`APW n`	`for writing (`**`W`**`rite)`
`APR n`	`for reading (`**`R`**`ead)`
`n`	`Protection level n`
	`from 0 or 10 (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.

REDEF extensions

For further information about the operation of the `REDEF` statement in the parts program, see the section "Protection levels for NC commands".

Example: changing rights in individual MDs

```
%_N_SGUD_DEF
;$PATH=/_N_DEF_DIR
REDEF $MA_CTRLOUT_SEGMENT_NR APR 2 APW 2
REDEF $MA_ENC_SEGMENT_NR APR 2 APW 2
REDEF $SN_JOG_CONT_MODE_LEVELTRIGGRD APR 2 APW 2
M30
```

Example: resetting rights in individual MDs to the original values

```
%_N_SGUD_DEF
;$PATH=/_N_DEF_DIR
REDEF $MA_CTRLOUT_SEGMENT_NR APR 7 APW 2
REDEF $MA_ENC_SEGMENT_NR APR 0 APW 0
REDEF $SN_JOG_CONT_MODE_LEVELTRIGGRD APR 7 APW 7
M30
```

3.4.4 Protection levels for NC commands (REDEF)

Function

The existing protection level concept for access to machine/setting data and GUDs has been expanded by the parts program commands listed above. For this purpose, a protection level 0 to 7 is assigned to a parts program command with the REDEF command.

Note

This command will now only be executed during parts program execution when the corresponding execution right exists.

Programming

G codes in accordance with the "List of G functions/preparatory functions"

`REDEF` (NC language element) APX value

or write access by the parts program or synchronous actions on the system variable.

`REDEF` (system variable) APW value

or change the write or read access to machine and setting data as previously

`REDEF` (machine data/setting data) APW value

`REDEF` (machine data/setting data) APR value

Parameters

The REDEF command acts globally for all channels and mode groups

REDEF	**Effect and application of the REDEF command**
NC language element	Language element to which a protection level is to be assigned for execution:
	1. Predefined subroutines/functions (see list with same name).
	2. "DO" statement keyword for synchronized actions
	3. G functions (G functions/preparatory functions).
	4. Program identifier for cycle The cycle must be stored in one of the cycle directories and contain a PROC statement.
System variables	System variable to which a protection level is to be assigned for write access. Read access is always possible. (see "List of system variables"):
Machine data/setting data	Machine data or setting data to which a protection level is to be assigned for read/write access.
APX	Vocabulary word for access protection
APW, APR	Execute
	Write, read
Value	Numerical value of the protective level (0 to 7)
	from 0 or 10 (highest level)
	to 7 (lowest level)
value 7	Keyswitch position 0 corresponds to the default setting of all available parts program commands

Example: subroutine call in definition files

```
N10 REDEF GEOAX APX 3
N20 IF(ISFILE("/_N_CST_DIR/_N_SACCESS_SUB1_SPF"))
N30 PCALL /_N_CST_DIR/_N_SACCESS_SUB1_SPF
N40 ENDIF
N40 M17
```

Description

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`

Subroutine call in definition files

It is possible to call subroutines containing `REDEF` statements from the above definition files. The `REDEF` statements must always be at the beginning of the data part just like the DEF statements. The subroutines must have the extension SPF or MPF and inherit the write-protection of the definition files set with $MN_ACCESS_WRITE_xACCESS.

Note

Extension of the REDEF command

As soon as the "Protection levels for NC commands" function is active, the redefinitions for the machine data/setting data created in GUD definition files must be stored in the new definition files for protection level assignments, i.e. the setting of protection levels for machine and setting data is permitted only in the previously mentioned protection level definition files and rejected in the GUD definition files with the alarm 15420.

Note

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 the value assignments via parts program command. On the user interface, the protection level concept of the HMI Advanced/Embedded applies.

For further information about the "protection level concept", please refer to:
/BAD/, HMI Operator's Guide, in the "Keyswitch" and "Machine Data" section
/IAD/, Installation and Start-Up Guide, "Setting Parameters for Control Unit"

3.5 REDEF Changing the attributes of the NC language elements

Function

The extension of the REDEF statement makes available the functions described in the previous subsection for defining data objects and protection levels into a general interface for setting attributes and values.

Programming

`REDEF NC language element attribute value`

or

`REDEF name` (no further parameter details)

Parameters

NC language element	This includes:	
	GUD	
	R parameters	
	Machine data/setting data	
	Synchronous variables ($AC_PARAM, $AC_MARKER, $AC_TIMER)	
	Synchronous variables that can be written from parts programs (see PGA1)	
	User frames (G500, etc.)	
	Magazine/tool configurations	
name	The settings for APX, APR, APW are set to default values and INIPO, INIRE, INICF, PRLOC are reset again.	
Attribute	**Permissible for:**	
Initializations		
INIPO	GUD, R parameters, synchronous vars	
INIRE	GUD, R parameters, synchronous vars	
INICF	GUD, R parameters, synchronous vars	
PRLOC	Setting data	
Synchronization	**Permissible for:**	**Setting a default value:**
SYNR	GUD	Preprocess stop while reading
SYNW	GUD	Preprocess stop while writing
SYNRW	GUD	Preprocess stop while reading and writing
Access authorization	**Permissible for:**	
APW	Machine/setting data	Access right during write
APR	Machine/setting data	Access right during read
		For machine and setting data you can overwrite the preset access authorization subsequently. The permissible values range from
		'0' (Siemens password) to '7' (keyswitch setting 0)

Optional parameters

`Value (optional)`	Optional parameters for attributes INIPO, INIRE, INICF, PRLOC: Subsequent start value(s) forms:

`Single value`
`value list`
	e.g. 5
	e.g. (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) for variable with 10 elements with

`REP (w1)`	w1: the value list to be repeated for variable with several elements, e.g. REP(12)

`SET(w1, w2, w3, ...)`
`(w1, w2, w3, ...)`
	or
	value list

`n:`	required protection level parameter for attributes for APR or APW For **GUD**, the definition can contain a start value (DEF NCK INT _MYGUD=5). If this start value is not stated (e.g. in DEF NCK INT _MYINT), the start value can be defined subsequently in the REDEF statement. 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.

Meaning of the attributes

`INIPO`	**INI**t for **P**ower **O**n The data are overwritten with the default(s) on battery-back restart of the NC.
`INIRE`	**INI**t for operator panel front **Re**set or TP end At the end of a main program, for example, with M2, M30 , etc. or on cancellation with the reset, the data are overwritten with the defaults. INIRE also applies for INIPO.
`INICF`	**INI**t for **N**ew**C**onf request or NEWCONF TP command On NewConf request or TP command NEWCONF, the data are overwritten with the default values. INICF also applies to INIRE and INIPO.
`PRLOC`	Only **pr**ogram-**loc**al change If the data is changed in a parts program, subroutine, cycle, or ASUB, it will be restored to its original value at the end of the main program (end with, for example, M2, M30, etc. or on cancellation by operator panel front reset). This attribute is only permissible for programmable setting data, see programmable setting data

The user is responsible for **synchronization** of the events triggering initialization. For example, if an end of parts program is executed in two **different channels**, the variables are initialized in each. That affects global and axial data!

Programmable setting data and the writable system variables from the parts program

The following SD 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_VELO_G25	G25
43220	$SA_SPIND_MAX_VELO_G26	G26
43230	$SA_SPIND_MAX_VELO_LIMS	LIMS
43300	$SA_ASSIGN_FEED_PER_REV_SOURCE	FPRAON
43420	$SA_WORKAREA_LIMIT_PLUS	G26
43430	$SA_WORKAREA_LIMIT_MINUS	G25
43510	$SA_FIXED_STOP_TORQUE	FXST
43520	$SA_FIXED_STOP_WINDOW	FXSW
43700	$SA_OSCILL_REVERSE_POS1	OSP1
43710	$SA_OSCILL_REVERSE_POS2	OSP2
43720	$SA_OSCILL_DWELL_TIME1	OST1
43730	$SA_OSCILL_DWELL_TIME2	OST2
43740	$SA_OSCILL_VELO	FA
43750	$SA_OSCILL_NUM_SPARK_CYCLES	OSNSC
43760	$SA_OSCILL_END_POS	OSE
43770	$SA_OSCILL_CTRL_MASK	OSCTRL
43780	$SA_OSCILL_IS_ACTIVE	OS
43790	$SA_OSCILL_START_POS	OSB

The PGA1 "List of the system variables" contains the listing of the system variables. All system variables that are marked W (write) or WS (write with preprocess stop) in column parts program can be initialized with the RESET instruction.

Example

```
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 parts program:
DEF _MYGUD2 INIRE                           ;Initialization
M17
```

This sets "_MYGUD2" back to "2" on operator panel front reset / end of parts program whereas "_MYGUD1" and "_MYGUD3" retain their value.

Example: modal speed limitation in the parts program (setting data)

```
/_N_DEF_DIR/_N_SGUD_DEF
REDEF $SA_SPIND_MAX_VELO_LIMS PRLOC         ;Setting data for limit speed
M17
```

```
/_N_MPF_DIR/_N_MY_MPF
N10 SETMS (3)
N20 G96 S100 LIMS=2500
...
M30
```

Let the limit speed defined in setting data ($SA_SPIND_MAX_VELO_LIMS) speed limitation be 1200 rpm. Because a higher speed can be permitted in a set-up and completely tested parts program, LIMS=2500 is programmed here. After the end of the program, the value configured in the setting data takes effect here again.

Reset settings to default values and delete initializations again

New definition	Attribute	Reset default value initializations
REDEF	NC language element	APX = 7
REDEF	Machine data/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

```
REDEF MASLON APX 2
REDEF SYG RS INIRE APW3
REDEF R[ ] INIRE
REDEF MASLON                                ;Set APX to 7
REDEF SYG RS                                ;Set APW 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**
  Note, however, that when the INI attributes for these variables are set, that an appropriately large **memory for INIT values**, can be set using MD 18150: MM_GUD_VAL_MEM, must be available. In the machine data 11270: 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 that deviates from the compiled value. 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 REDEF <name> INIRE, INIPO; INICF; PRLOC is used to change the behavior of a system variable or GUD, the machine data DEFAULT_VALUES_MEM_MASK must be set to 1 (memory for initialization values active). Otherwise, alarm 12261 "Initialization not allowed" is output.

3.6 SEFORM structuring statement in the Step editor

Function

The SEFORM statement is evaluated in the Step editor to generate the step view for HMI Advanced. The step view available in the HMI Advanced improves the readability of the NC subroutine. The SEFORM structuring statement supports the Step editor (editor-based program support) over the three specified parameters.

Programming

```
SEFORM(STRING[128] section_name, INT level, STRING[128] icon)
```

Parameters

`SEFORM`	`Function call of structuring statement with parameters: section_name, level, and icon`
`section_name`	`Identifier of the operation`
`level`	`Index for the main or sublevel.`
	`=0 corresponds to the main level`
	`=1, ... corresponds to sublevel 1 to n`
`icon`	`Name of the icon displayed for this section.`

Note

The SEFORM statements 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 statement. The information remains until overwritten by the next SEFORM statement. Reset and end of parts program clear the content.

The level and icon parameters are checked by the parts program processing of the NCK but not further processed.

For more information about editor-based programming support, see:
/BAD/ Operator's Guide HMI Advanced.

Protection zones 4

4.1 Definition of the protection zones (CPROTDEF, NPROTDEF)

Function

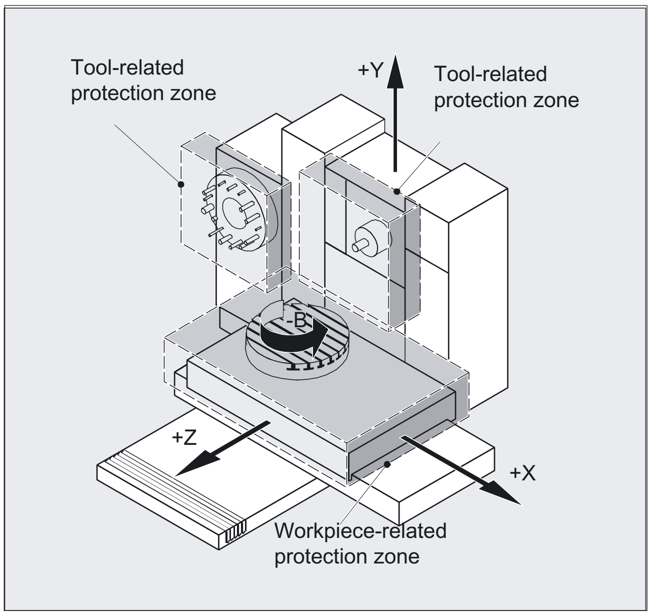
You can use protection zones to protect various elements on the machine, their components and the workpiece against incorrect movements.

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).


Tool-related protection zone
+Y
Tool-related protection zone
-B
+Z
+X
Workpiece-related protection zone

Programming

```
DEF INT NOT_USED
CPROTDEF(n,t,applim,appplus,appminus)
NPROTDEF(n,t,applim,applus,appminus)
```

```
EXECUTE(NOT_USED)
```

Parameters

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 = **Tool-related** protection zone
	FALSE = **workpiece**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
applus	Value of the limit in the positive direction in the 3rd dimension
appminus	Value of the limit in the negative direction in the 3rd dimension
NOT_USED	Error variable has no effect in protection zones with EXECUTE

Description

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 parts 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

With the 810D, a maximum of 4 contour elements are available for defining one protection zone (max. of 4 channel-specific and 4 NCK-specific protection zones).

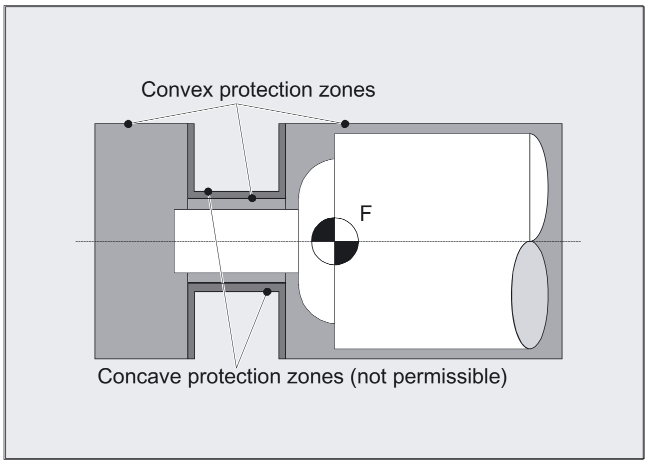
If a full circle describes the protection zone, it must be divided into two half circles. The order G2, G3 or G3, G2 is not permitted. A short G1 block must be inserted, if necessary.

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.



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.

4.2 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.

Programming

```
CPROT (n,state,xMov,yMov,zMov)
NPROT (n,state,xMov,yMov,zMov)
```

Parameters

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

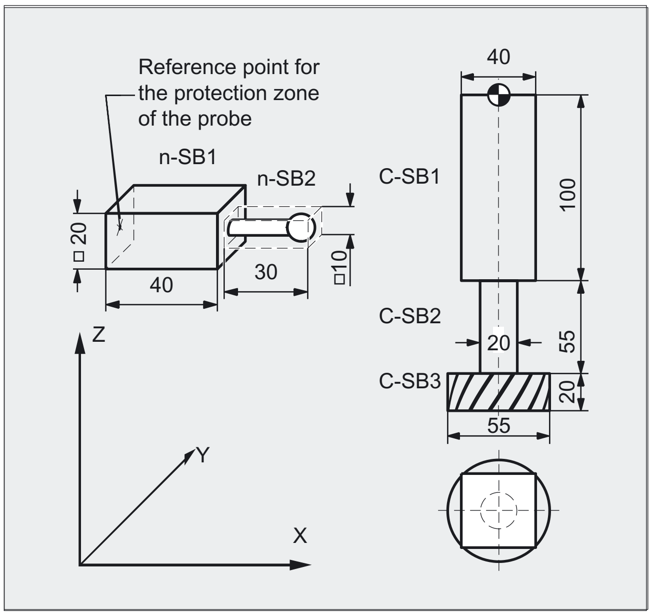
Example of milling

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.



```
DEF INT PROTECTB                       Definition of a Help variable
Definition of protection zones         Set orientation
G17
NPROTDEF(1,FALSE,3,10,-10)             Protection zone n-SB1
G01 X0 Y-10
X40
Y10
X0
Y-10
EXECUTE(PROTECTB)
```

```
NPROTDEF(2,FALSE,3,5,-5)                Protection zone n-SB2
G01 X40 Y-5
X70
Y5
X40
Y-5
EXECUTE(PROTECTB)

CPROTDEF(1,TRUE,3,0,-100)               Protection zone c-SB1
G01 X-20 Y-20
X20
Y20
X-20
Y-20
EXECUTE(PROTECTB)

CPROTDEF(2,TRUE,3,-100,-150)            Protection zone c-SB2
G01 X0 Y-10
G03 X0 Y10 J10
X0 Y-10 J-10
EXECUTE(PROTECTB)

CPROTDEF(3,TRUE,3,-150,-170)            Protection zone c-SB3
G01 X0 Y-27,5
G03 X0 Y27,5 J27,5
X0 Y27,5 J-27,5
EXECUTE(PROTECTB)
Activation of protection zones:
NPROT(1,2,-120,60,80)                   Activate protection zone n-SB1 with offset
NPROT(2.2,-120,60,80)                   Activate protection zone n-SB2 with offset
CPROT(1,2,0,0,0)                        Activate protection zone c-SB1 with offset
CPROT(2,2,0,0,0)                        Activate protection zone c-SB2 with offset
CPROT(3,2,0,0,0)                        Activate protection zone c-SB3 with offset
```

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.

4.3 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 subroutine. It must be alone in a block.

Programming

```
Status=CALCPOSI(_STARTPOS, _MOVDIST, _DLIMIT, _MAXDIST, _BASE_SYS,
_TESTLIM)
```

Parameters

Status	0: Function OK; the defined path can be traversed completely. –: In _DLIMIT at least one component is negative –: An error occurred in a transformation calculation. If the defined path cannot be traversed completely, a positive, decimally coded value is returned: **Units digit (type of violated limit):** 1: Software limits are limiting the traverse path. 2: Working area limitation is limiting the traverse path. 3: Protection zones are limiting limit the traverse path. If several limits are violated at once (e.g. software limits and protection zones), the limit leading to the greatest limitation of the traverse path is indicated in a units digit. **Tens digit** 10: The start value is violating the limit. 20: The defined straight line is violating the limit. This value is also returned if the end point does not violate any limit itself but a limit value would be violated on the path from the start to the end point (e.g. by passing through a protection zone, curved software limits in the WCS for non-linear transformations, e.g. Transmit). **Hundreds digit** 100: The positive limit value is violated (only if the units digit is 1 or 2, i.e. for software limits and working area limitation). 100: Only an NCK protection zone is violated (only if the units digit is 3). 200: The negative limit value is violated (only if the units digit is 1 or 2, i.e. for software limits and working area limitation). 200: Only a channel-specific protection zone is violated (only if the units digit is 3). **Thousands digit** 1000: Factor by which the number of the axis is multiplied that violates the limit (only if the units digit is 1 or 2, i.e. for software limits and working area limitation). The axis count starts at 1 and refers in the case of violated software limits (units digit = 1) to the machine axes and in the case of a working area limitation (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 leading to the greatest limitation of the traverse path is indicated in the hundreds and thousands digit of the protection zone.
_STARTPOS	Start value for abscissa [0], ordinate [1], and applicate [2] in the (WCS)
_MOVEDIST	Path definition incremental 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 a non-linear transformation, if no geometry axis can be uniquely assigned.`
	`[4]: Minimum clearance assigned to a rotary machine axis for a non-linear transformation, if no geometry axis can be uniquely assigned. Only for special transformations, if SW limits are to be monitored.`

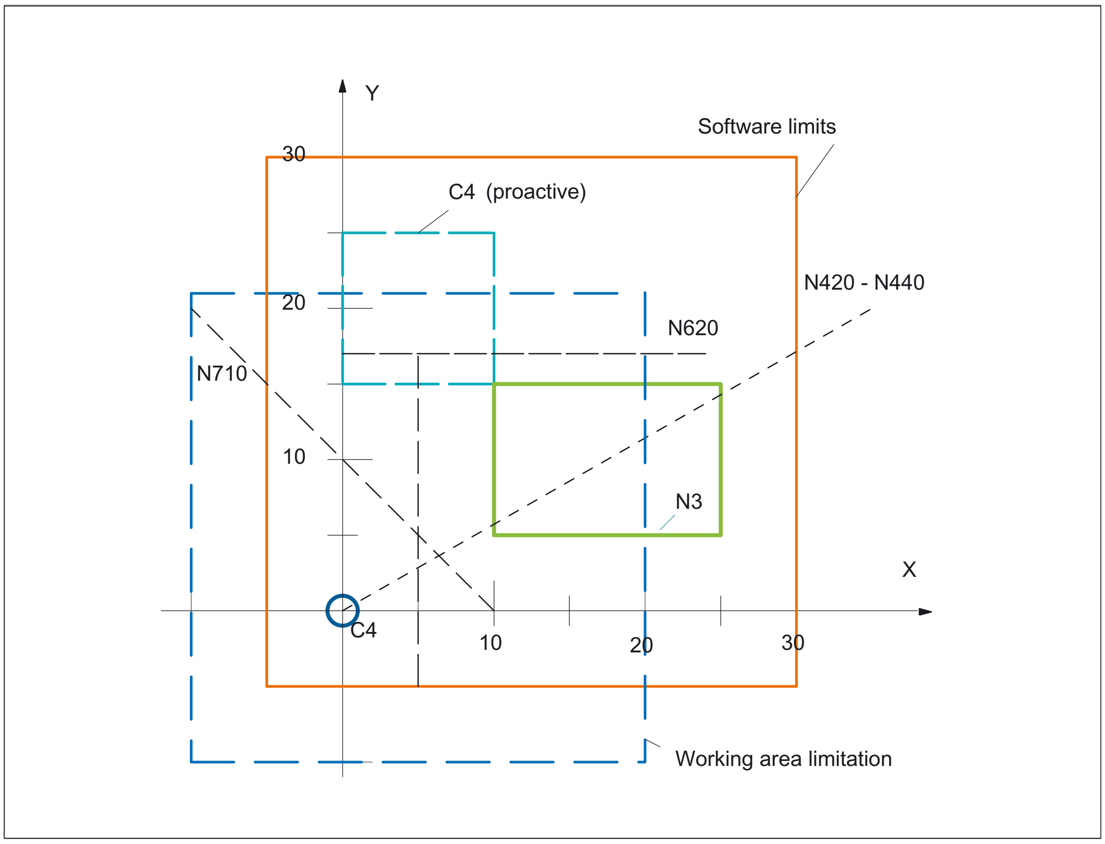
`_MAXDIST`	`Array [0] - [2] for return value. Incremental path in all three geometry axes without violating the defined minimum clearance of an axis limit in the machine axes involved.`
	`If the traverse path is not restricted, the content of this return parameter is the same as the content of _MOVDIST.`

`_BASE_SYS`	`FALSE or parameters not stated:`
	`In evaluating the position and length data, the G code from G code group 13 (G70, G71, G700, G710; inch/metric) is evaluated. If G70 is active and the basic system is metric (or G71 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 recalculated using the CALCPOSI function.`
	`TRUE:`
	`In evaluation of the position and length data, the basic system of the control is always used depending on the value of the active G code of group 13.`

`_TESTLIM`	`Limitations to be checked (binary coded):`
	`1: Monitoring software limits`
	`2: Monitoring working area limitations`
	`3: Monitoring activated protection zones`
	`4: Monitoring pre-activated protection zones`
	`Combinations by adding values. Default: 15; check all.`

Example

The example in the figure shows X software limits and working area limitations. In addition, three protection zones are defined: the two channel-specific protection zones C2 and C4, and the N3 NCK protection zone C2 is a circular, active, tool-related protection zone with 2 mm radius. C4 is a square, pre-activated, and workpiece-related protection zone with side length 10 mm and N3 is a rectangular, active protection zone with side lengths 10 mm and 15 mm. In the following NC, initially the protection zones and the operating range limits are defined as indicated, and the CALCPOSI function is then called with various parameters. The events of each CALCPOSI call are summarized in the table at the end of the example.



```
N10 def real _STARTPOS[3]
N20 def real _MOVDIST[3]
N30 def real _DLIMIT[5]
N40 def real _MAXDIST[3]
N50 def int _SB
N60 def int _STATUS
N70 cprotdef(2, true, 0)                  ;Tool-related 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)
N190 nprotdef(3, false, 0)                ;workpiece-elated 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/deactivate
N270 cprot(4,1,0,0,0)                     ;protection zones
N280 nprot(3,2,0,0,0)
N290 g25 XX=-YY=-                         ;define working area limitations
N300 g26 xx= 20 yy= 21
N310 _STARTPOS[0] = 0.
N320 _STARTPOS[1] = 0.
N330 _STARTPOS[2] = 0.
N340 _MOVDIST[0] = 35.
N350 _MOVDIST[1] = 20.
N360 _MOVDIST[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
N420 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST)
N430 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST,,3)
N440 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST,,1)
N450 _STARTPOS[0] = 5.                    ;other starting point
N460 _STARTPOS[1] = 17.
N470 _STARTPOS[2] = 0.
N480 _MOVDIST[0] = 0.                     ;other destination
N490 _MOVDIST[1] =-.
N500 _MOVDIST[2] = 0.
; various function calls
N510 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST,,14)
N520 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST,,6)
N530 _DLIMIT[1] = 2.
N540 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST,,6)
N550 _STARTPOS[0] = 27.
N560 _STARTPOS[1] = 17.1
N570 _STARTPOS[2] = 0.
N580 _MOVDIST[0] =-.
N590 _MOVDIST[1] = 0.
N600 _MOVDIST[2] = 0.
N610 _DLIMIT[3] = 2.
N620 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST,,12)
N630 _STARTPOS[0] = 0.
N640 _STARTPOS[1] = 0.
N650 _STARTPOS[2] = 0.
N660 _MOVDIST[0] = 0.
N670 _MOVDIST[1] = 30.
N680 _MOVDIST[2] = 0.
N690 trans x10
N700 arot z45
N710 _STATUS = calcposi(_STARTPOS,_MOVDIST,_DLIMIT,_MAXDIST)
N720 M30
```

Results of the tests in the example:

Block no. N...	_STATUS	_MAXDIST [0] (= X)	_MAXDIST [1] (= Y)	Comments
420	3123	8.040	4.594	Protection zone N3 violated.
430	1122	20.000	11.429	No protection zone monitoring, working area limitation violated.
440	1121	30.000	17.143	Now only monitoring of the software limits active.
510	4213	0.000	0.000	Start point violates protection zone C4.
520	0000	0.000	–.000	Pre-activated protection zone C4 not monitored. Defined path can be traversed completely.
540	2222	0.000	–.000	Because _DLIMIT[1]=2, the traverse path is restricted by the working area limitation.
620	4223	–.000	0.000	Distance from C4 in total 4 mm due to C2 and _DLIMIT[3]. Distance C2 – N3 of 0.1 mm does not lead to limitation of the traverse path.
710	1221	0.000	21.213	Frame with translation and rotation active. The permissible traversal path in _MOVDIST applies in the translated 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.

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 _MOVDIST 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).

Special Motion Commands 5

5.1 Approaching coded positions (CAC, CIC, CDC, CACP, CACN)

Function

The machine data can be used to enter for two axes a maximum of 60 (0 to 59) positions for each in the position tables.

Programming

```
CAC (n)
```

or

```
CIC (n)
```

or

```
CACP (n)
```

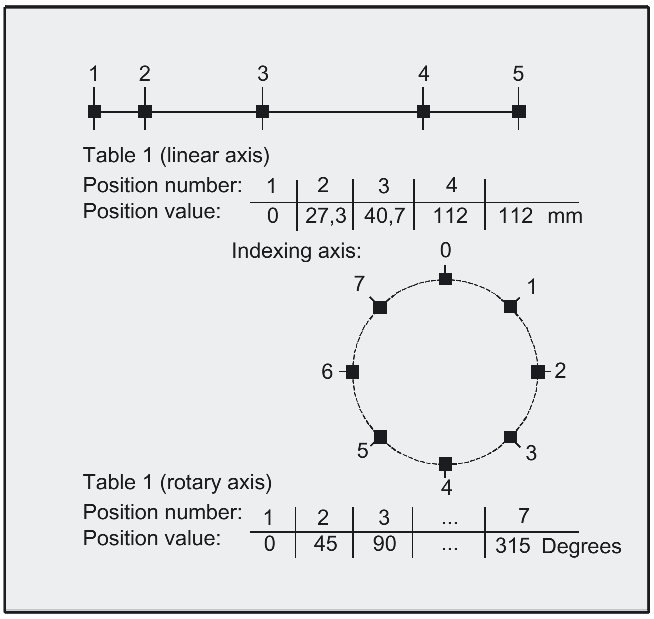
or

```
CACN (n)
```

Parameters

`CAC (n)`	Approach absolute coded position
`CIC (n)`	Approach coded position incrementally by n spaces in plus direction (+) or in minus direction (-)
`CDC (n)`	Approach coded position via shortest possible route (rotary axes only)
`CACP (n)`	Approach coded position absolutely in positive direction (rotary axes only)
`CACN (n)`	Approach coded position absolutely in negative direction (rotary axes only)
`(n)`	Position numbers 1, 2, ... max. 60 positions for each axis

Example: positioning table for linear axis and rotary axis


1
2
3
4
5
Table 1 (linear axis)
Position number: 1 2 3 4
Position value: 0 27,3 40,7 112 112 mm
Indexing axis:
0
1
2
3
4
5
6
7
Table 1 (rotary axis)
Position number: 1 2 3 ... 7
Position value: 0 45 90 ... 315 Degrees

Note

If an axis is situated between two positions, it does not traverse in response to an incremental position command with CIC (...). It is always advisable to program the first travel command with an absolute position value.

Example 2

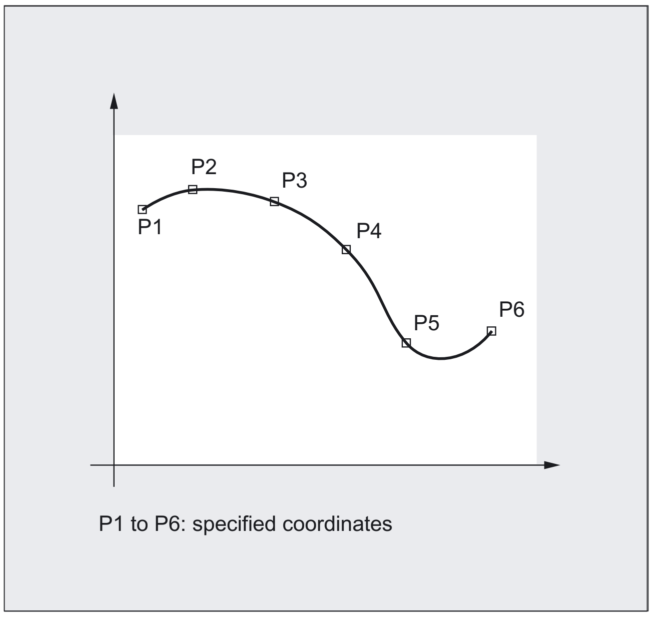
```
N10 FA[B]= 300                   ;Feed for positioning axis B
N20 POS[B]=    CAC     (10)      ;Approach coded position 10 (absolutely)
N30 POS[B]=    CIC     (-4)      ;Travel 4 spaces back from the current position
```

5.2 Spline interpolation (ASPLINE, BSPLINE, CSPLINE, BAUTO, BNAT, BTAN)

Function

The spline interpolation function can be used to link series of points along smooth curves. Splines can be applied, for example, to create curves using a sequence of digitized points.

There are several types of spline with different characteristics, each producing different interpolation effects. In addition to selecting the spline type, the user can also manipulate a range of different parameters. Several attempts are normally required to obtain the desired pattern.



In programming a spline, you link a series of points along a curve. You can select one of three spline types:

- A spline (akima spline)
- B spline (non-uniform, rational basis spline, NURBS)
- C spline (cubic spline)

Programming

```
ASPLINE X Y Z A B C
```

or

```
BSPLINE X Y Z A B C
```

or

```
CSPLINE X Y Z A B C
```

Parameters

ASPLINE	The Akima spline passes as tangent through the intermediate points.
BSPLINE	The B spline does not pass directly through the control points but only near them. The programmed positions are not interpolation points but only control points.
CSPLINE	Cubic spline with transitions to the interpolation points both tangentially and in terms of curvature.

A, B and C splines are modally active and belong to the group of motion commands.
The tool radius offset may be used.
Collision monitoring is carried out in the projection in the plane.

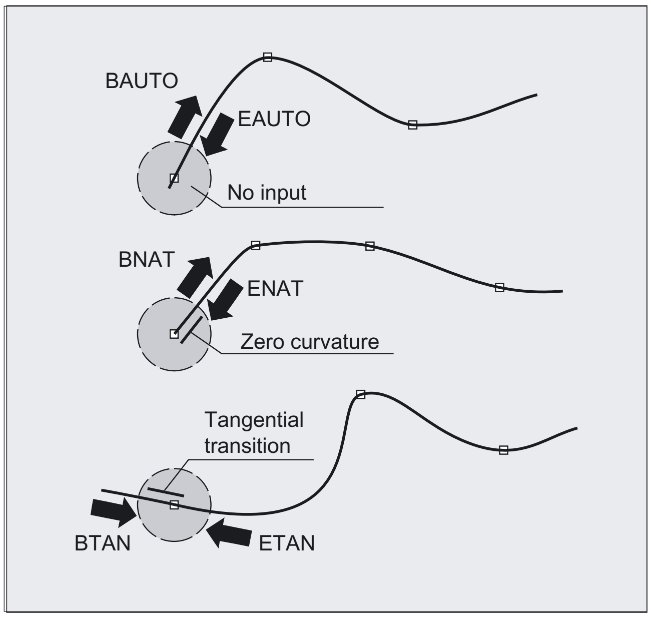
Note

Parameters for A-SPLINE and C-SPLINE

For the Akima spline (A spline) and the Cubic spline (C spline), restrictions for the transition behavior at the start and the end of the spline curve can be programmed.

These restrictions for the transition behavior are divided into two groups with statements each with three commands as follows:

Start of spline curve:	
BAUTO	No command input; start is determined by the position of the first point
BNAT	Zero curvature
BTAN	Tangential transition to preceding block (initial setting)
End of spline curve:	
EAUTO	No command input; end is determined by the position of the last point
ENAT	Zero curvature
ETAN	Tangential transition to next block (initial setting)



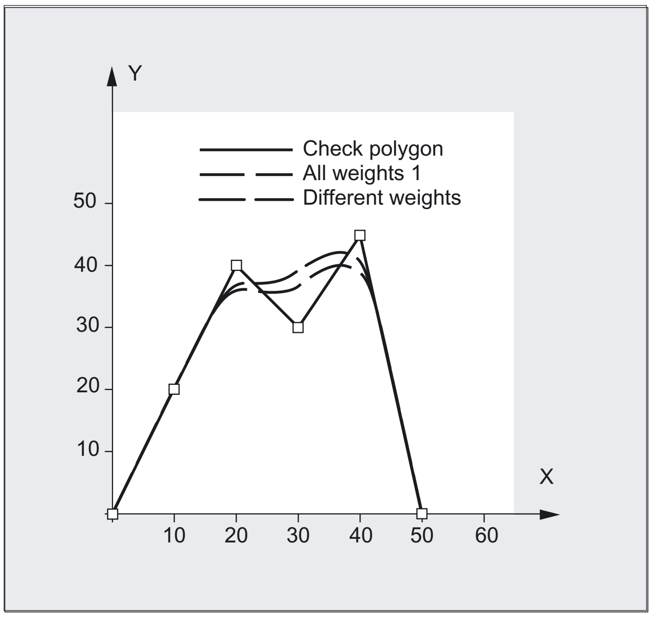
Note

Parameters for B-SPLINE

The programmed restrictions (see A- or C-spline) do not have any affect on the B-spline. The B spline is always tangential to the check polygon at its start and end points.

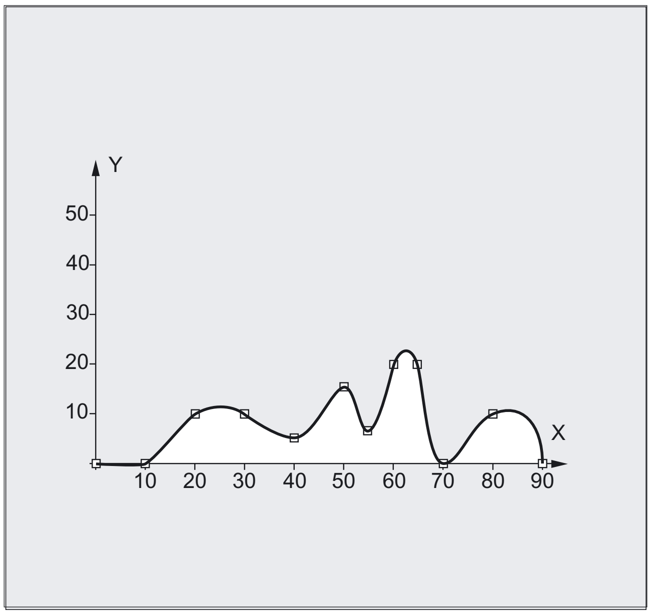
Point weight:	
`PW = n`	The weight details can be programmed as a so-called point-weight (PW) for each interpolation point.
Value range:	
`<= n <= 3`	in increments of 0.0001
Effect:	
`n > 1`	The check point exerts more "force" on the curve.
`n < 1`	The check point exerts less "force" on the curve.
Spline degree:	
`SD = 2`	A third degree polygon is used as standard, but a second degree polygon is also possible.
Distance between nodes:	
`PL = value`	The distances between nodes are suitably calculated internally. The control can also machine predefined node distances that are specified in the so-called parameter-interval-length (PL).
Value	Value range as for path dimension

Example: B spline



All weights 1	Different weights	Check polygon
N10 G1 X0 Y0 F300 G64	N10 G1 X0 Y0 F300 G64	N10 G1 X0 Y0 F300 G64
N20 BSPLINE	N20 BSPLINE	N20 ;omitted
N30 X10 Y20	N30 X10 Y20 PW=2	N30 X10 Y20
N40 X20 Y40	N40 X20 Y40	N40 X20 Y40
N50 X30 Y30	N50 X30 Y30 PW=0.5	N50 X30 Y30
N60 X40 Y45	N60 X40 Y45	N60 X40 Y45
N70 X50 Y0	N70 X50 Y0	N70 X50 Y0

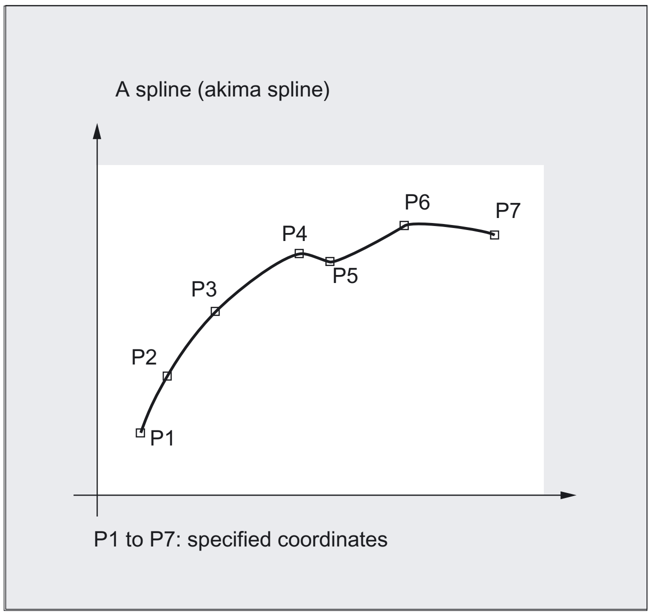
Example: C spline, zero curvature at start and end



```
N10 G1 X0 Y0 F300
N15 X10
N20 BNAT ENAT                          ;C spline, zero curvature at start and end
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
```

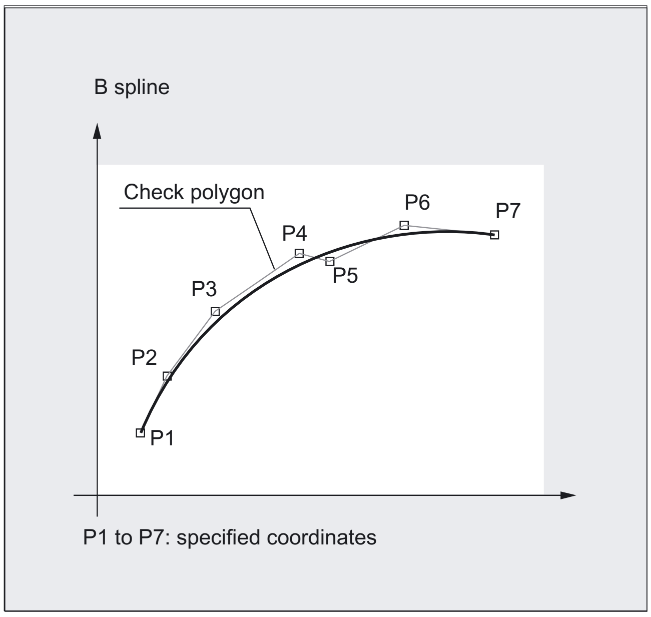
A-Spline

The A spline (Akima spline) passes exactly through the intermediate points. While it produces virtually no undesirable oscillations, it does not create a continuous curve in the interpolation points. The akima spline is local, i.e. a change to an interpolation point affects only up to six adjacent points. The primary application for this spline type is therefore the interpolation of digitized points. A polynomial of third degree is used for interpolation.



B spline

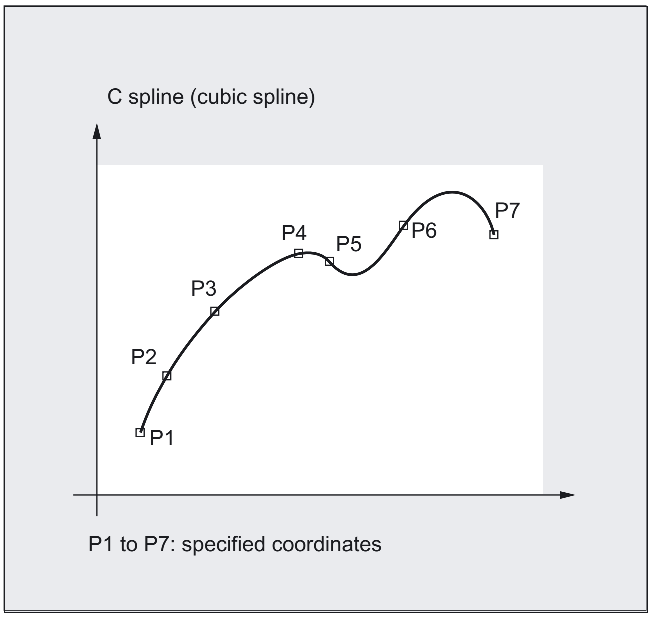
With a B spline, the programmed positions are not intermediate points, but merely check points of the spline, i.e. the curve is "drawn towards" the points, but does not pass directly through them. The lines linking the points form the check polygon of the spline. B splines are the optimum means for defining tool paths on sculptured surfaces. Their primary purpose is to act as the interface to CAD systems. A third degree B spline does not produce any oscillations in spite of its continuously curved transitions.



C spline

In contrast to the akima spine, the cubic spline is continuously curved in the intermediate points. It tends to have unexpected fluctuations however. It can be used in cases where the interpolation points lie along an analytically calculated curve. C splines use third degree polynomials.

The spline is not local, i.e. changes to an interpolation point can influence a large number of blocks (with gradually decreasing effect).

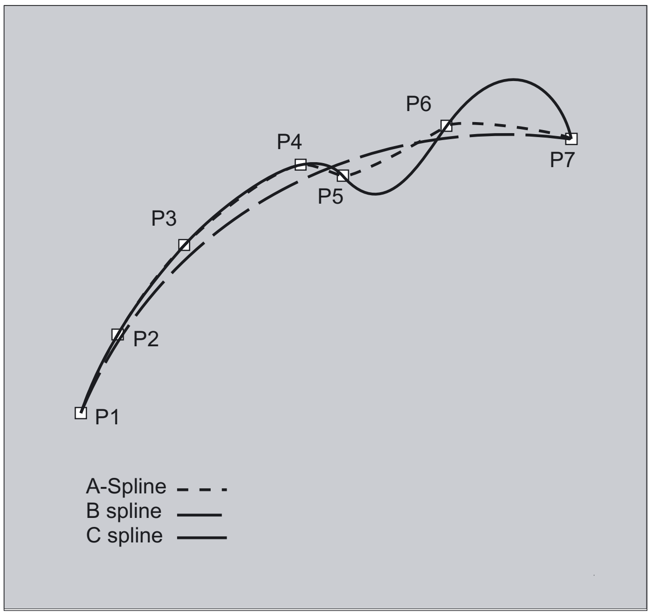


Comparison of three spline types with identical interpolation points:

A spline (akima spline)

B spline (Bezier spline)

C spline (cubic spline)



Settings for splines

The G codes ASPLINE, BSPLINE and CSPLINE link block endpoints with splines. For this purpose, a series of blocks (endpoints) must be simultaneously calculated. The buffer size for calculations is ten blocks as standard. Not all block information is a spline endpoint. However, the control requires a certain number of spline endpoint blocks from ten blocks.

These are for:

A spline:	At least 4 blocks out of every 10 must be spline blocks. These do not include comment blocks and parameter calculations.
B spline:	At least 6 blocks out of every 10 must be spline blocks. These do not include comment blocks and parameter calculations.
C spline:	From each 10 blocks at least the contents of machine data $MC_CUBIC_SPLINE_BLOCKS+1 must be spline blocks (also in standard case 9). The number of points must be entered in machine data $MC_CUBIC_SPLINE_BLOCKS (standard value 8) which are used for calculating the spline segment.

Note

An alarm is output if the tolerated value is exceeded and likewise when one of the axes involved in the spline is programmed as a positioning axis.

5.3 Spline grouping (SPLINEPATH)

Function

The SPLINEPATH command is used to select the axes to be interpolated in the spline grouping. Up to eight path axes can be involved in a spline interpolation grouping. The SPLINEPATH statement defines which axes are to be involved in the spline.

Programming

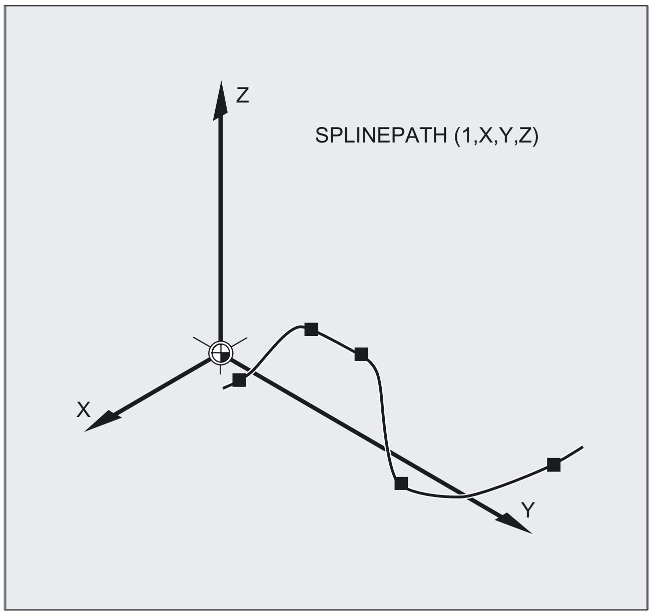
```
SPLINEPATH(n,X,Y,Z,…)
```

The instruction is programmed in a separate block. If SPLINEPATH is not explicitly programmed, then the first three axes in the channel are traversed as the spline grouping.

Parameters

`SPLINEPATH`	`Define spline grouping`
`n= 1`	`Fixed point`
`X,Y,Z,…`	`Path axes details`

Example: spline grouping with three path axes



```
N10 G1 X10 Y20 Z30 A40 B50 F350
N11 SPLINEPATH(1,X,Y,Z)                     ;Spline grouping
N13 CSPLINE BAUTO EAUTO X20 Y30 Z40 A50 B60 ;C spline
N14 X30 Y40 Z50 A60 B70                     ;Interpolation points
…
N100 G1 X… Y…                               ;Deselection of spline
                                            ;interpolation
```

5.4 Compressor (COMPOF/ON, COMPCURV, COMPCAD)

Function

With G code COMPON block transitions are only constant in **speed**, while acceleration of the participating axes can be in jumps at block transitions. This can increase oscillation on the machine.

With G code COMPCURV, the block transitions are with **constant acceleration**. This ensures both smooth velocity and acceleration of all axes at block transitions. The COMPCAD G code can be used to selected another compression that can be optimized with regard to **surface quality and velocity**.

Machine manufacturer

The compressor functions can be configured and thus dependent on machine data settings.

Programming

`COMPON`

or

`COMPOF`

or

`COMPCURV`

or

`COMPCAD`

Operating conditions for programmed NC blocks

This compression operation can only be executed on linear blocks (G1). It is interrupted by any other type of NC instruction, e.g., an auxiliary function output, but not by parameter calculations. Only those blocks containing nothing more than the block number, G1, axis addresses, feed and comments are compressed. All other blocks are executed unchanged (no compression). Variables may not be used.

Parameters

COMPON/ /	Compressor on, continuous in the velocity
COMPOF	Compressor OFF
COMPCURV	Compressor on, with constant curvature polynomial (continuous acceleration)
COMPCAD	Compressor on, optimized surface quality (velocity-optimized)

Example COMPON

```
N10 COMPON                             ;or COMPCURV, compressor ON
N11 G1 X0.37 Y2.9 F600                 ;G1 must be programmed before the
                                       ;end point and feed
N12 X16.87 Y-.698
N13 X16.865 Y-.72
N14 X16.91 Y-.799…
N1037 COMPOF                           ;Compressor OFF
…
```

Note

All blocks are compressed for which a simple syntax is sufficient, e.g.,

N19 X0.103 Y0. Z0.

N20 X0.102 Y-0.018

N21 X0.097 Y-0.036

N22 X0.089 Y-0.052

N23 X0.078 Y-0.067

Traverse blocks with extended addresses such as C=100 or A=AC(100) are also compressed.

Example COMPCAD

```
G00 X30 Y6 Z40
G1 F10000 G642
SOFT
COMPCAD                                   ;Compressor interface optimization ON
STOPFIFO
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 OFF
G00 Z50
M30
```

Requirements

Machine manufacturer

Three sets of machine data are provided for the compressor function:

- $MC_COMPRESS_BLOCK_PATH_LIMIT
  A maximum path length is set. All the blocks along this path are suitable for compression. Longer blocks are not compressed.
- $MA_COMPRESS_POS_TOL
  A tolerance can be set for each axis. This value specifies the maximum deviation of the generated spline curve from the programmed end points. The higher the values, the more blocks can be compressed.
- $MC_COMPRESS_VELO_TOL
  The maximum permissible path feed deviation with active compressor can be preset in conjunction with FLIN and FCUB.

COMPCAD

- $MN_MM_EXT_PROG_BUFFER_SIZE should be large, e.g., 100 (KB).
- $MC_COMPRESS_BLOCK_PATH_LIMIT must be significantly increased in value, e.g., 50 (mm).
- $MC_MM_NUM_BLOCKS_IN_PREP must be >= 60, to allow machining of much more than 10 points.
- FLIN and FCUB cannot be used.

Recommended for large block lengths and optimum velocity:

- $MC_MM_MAX_AXISPOLY_PER_BLOCK = 5
  $MC_MM_PATH_VELO_SEGMENTS = 5
  $MC_MM_ARCLENGTH_SEGMENTS = 10

Description

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.

The compressor allows a certain number (max. 10) of short path sections to be combined in a single path section.

The modal G code COMPON or COMPCURV can be used to activate an "NC block compressor". This function collects a series of linear blocks during linear interpolation (the number is limited to 10) and approximates them within a tolerance specified in machine data via a 3rd-degree (COMPON) or 5th-degree (COMPCURV) polynomial. One traversing block is processed by the NC instead of a large number of small blocks.

COMPCAD

COMPCAD is processor and memory-intensive. It should only be used if surface quality enhancement measures cannot be incorporated in the CAD/CAM program. Features:

- COMPCAD produces polynomial blocks with a continuous acceleration.
- With adjacent paths, deviations head in the same direction.
- A limit angle can be defined with setting data $SC_CRIT_SPLINE_ANGLE; COMPCAD will leave the corners from this angle.
- The number of blocks to be compressed is not limited to 10.
- COMPCAD eliminates poor surface transitions. In doing so, however, the tolerances are largely adhered to but the corner limit angle is ignored.
- The rounding function G642 can also be used.

COMPON, COMPCURV and COMPCAD extensions

The compressors COMPON, COMPCURV and COMPCAD are extended in a way that even NC programs for which orientation was programmed via directional vectors, can be compressed respecting a specifiable tolerance.

Orientation transformation TRAORI

The "Compressor for orientation" function requires the availability of the Orientation transformation option. The restrictions mentioned above under "Conditions of usage" have been relieved to allow position values via parameter settings now also.

NC block structure in general:

```
N10 G1 X=<...> Y=<...> Z=<...>        ;Axis positions as parameter expressions with
A=<...>                               < ... > parameter expression, e.g., X=R1*(R2+R3)
      B=<...> F=<...> ; Comment
```

Activation

You can activate "Compressor for orientations" via one of the following commands: COMPON, COMPCURV (COMPCAD not possible).

The compressors can be used with active orientation transformation (TRAORI) as well as on

- 5-axis machines and on
- 6-axis machines on which, in addition to tool orientation, the rotation of the tool can also be programmed

See Transformations, "Orientation compression" for more detailed information about the use of compressors on 5- and 6-axis machines.

References: /FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformation (F2).

5.5 Polynomial interpolation (POLY, POLYPATH)

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.

Programming

`POLY PO[X]=(`x_e, a_2, a_3`) PO[Y]=(`y_e, b_2, b_3`) PO[Z]=(`z_e, c_2, c_3`) PL=n` polynomial of the 3rd degree

or expansion to polynomials of the 5th degree and new polynomial syntax

`POLY X=PO(`x_e, a_2, a_3, a_4, a_5`) Y=PO(`y_e, b_2, b_3, b_4, b_5`) Z=PO(`z_e, c_2, c_3, c_4, c_5`) PL=n`

```
POLYPATH ("AXES", VECT")
```

Parameters

`POLY`	Activation of polynomial interpolation with a block containing POLY
`POLYPATH`	Polynomial interpolation can be selected for both the AXIS or VECT axis groups
`PO [axis identifier/variable]=(…,…,…)`	End points and polynomial coefficients
`X, Y, Z`	Axis identifier
x_e, y_e, z_e	Specification of end position for relevant axis; value range as for path dimension
a_2, a_3, a_4, a_5	The coefficients a_2, a_3, a_4, and a_5 are written with their value; Value range as for path dimension. The last coefficient in each case can be omitted if it equals zero.
`PL`	Length of parameter interval over which the polynomials are defined (definition range of function f(p)). The interval always starts at 0. p can be set to values between 0 and PL. Theoretical value range for PL: 0,0001 … 99 999,9999. The PL value applies to the block that contains it. PL=1 is applied if no PL value is programmed.

Activate/deactivate POLY

Polynomial interpolation belongs to the first G group along with G0, G1, G2, G3, A spline, B spline and C spline. If it is active, there is no need to program the polynomial syntax: Axes that are programmed with their name and end point only are traversed linearly to their end point. If all axes are programmed in this manner, the control system responds as if G1 were programmed.

Polynomial interpolation is deactivated by another command in the G group (e.g. G0, G1).

Polynomial coefficient

The PO value `(PO[]=)` or `...=PO(...)` specifies all polynomial coefficients for an axis. Several values, separated by commas, are specified according to the degree of the polynomial. Different polynomial degrees can be programmed for different axes within one block.

New polynomial syntax with PO: The previous syntax remains valid.

Subroutine call POLYPATH

With POLYPATH the polynomial interpolation can be specified selectively for the following axis groups:

- `POLYPATH ("AXES")`
  All path axes and special axes.
- `POLYPATH ("VECT")` orientation axes
  (with orientation transformation).

As standard, the programmed polynomials are interpreted as polynomial for both axis groups.

Examples:

```
POLYPATH ("VECT")
```

Only the orientation axes are selected for the polynomial interpolation; all other axes are traversed linearly.

`POLPATH ( )`
Deactivates the polynomial interpolation for all axes

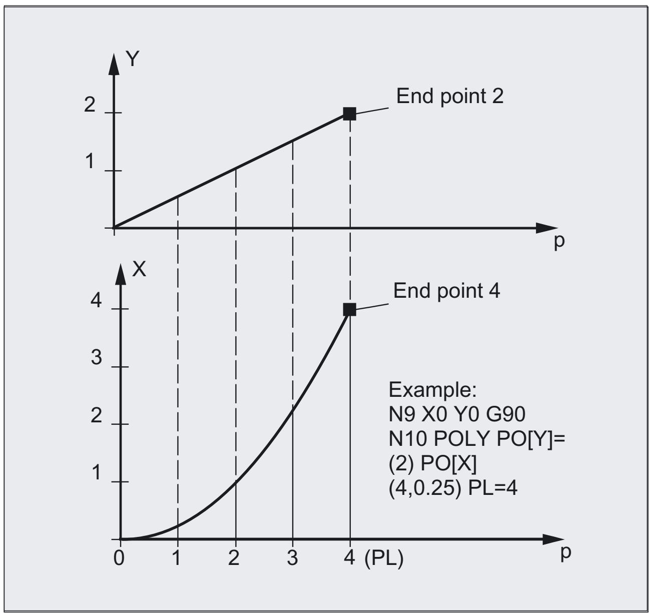
Example

```
N10 G1 X… Y… Z… F600
N11 POLY PO[X]=(1,2.5,0.7) ->              ;Polynomial interpolation ON
-> PO[Y]=(0.3,1,3.2) PL=1.5
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 settings for axes
N27 PO[X]=(10.2.5) PO[Y]=(2.3)             ;No PL value programmed; PL=1 applies
N30 G1 X… Y… Z.                            ;Polynomial interpolation OFF
…
```

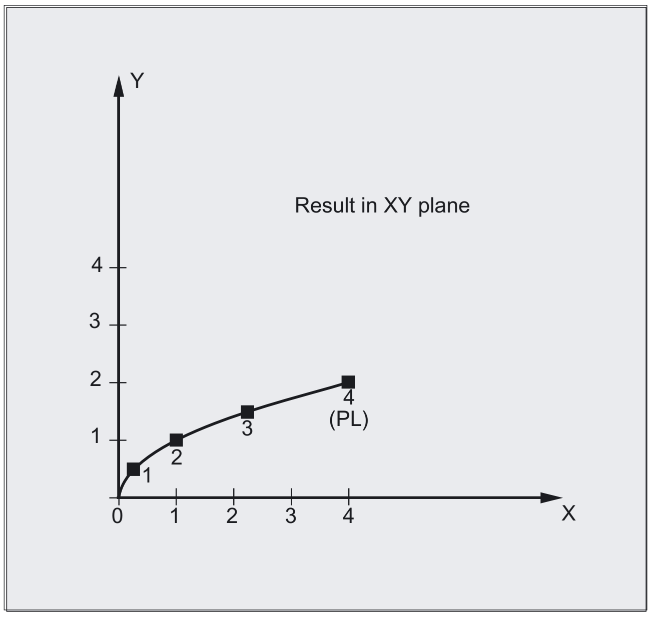
Example of applicable polynomial syntax with PO

```
Polynomial syntax used previously remains   ;New polynomial syntax (SW 6 and higher)
valid
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 of a curve in the X/Y plane



```
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 each selected path axis follows a function (polynomial, max. 3rd degree) or (polynomial, max. 5th degree)

The equation used to express the polynomial function is generally as follows:

$f(p)= a_0 + a_1p + a_2p^2 + a_3p^3$

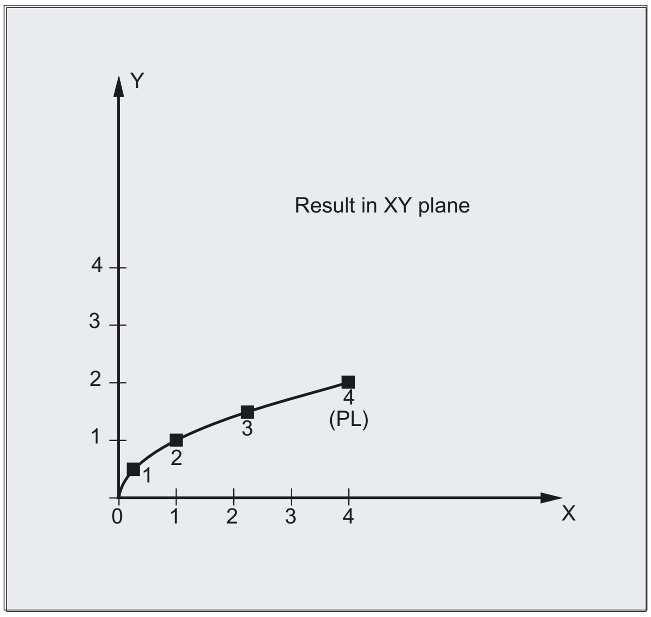
or

$f(p)= a_0 + a_1p + a_2p^2 + a_3p^3 + a_4p^4 + a_5p^5$

Key:

a_n: Constant coefficients

p: Parameters



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:

$f(p) = a_0 + a_1p$

The following settings apply:

a_0 = axis position at the end of the preceding block

a_1 = difference between axis position at the end of the definition range (PL) and the start position

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 endpoint 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 subroutine 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 specification of axes names), 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:

```
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
                                            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:

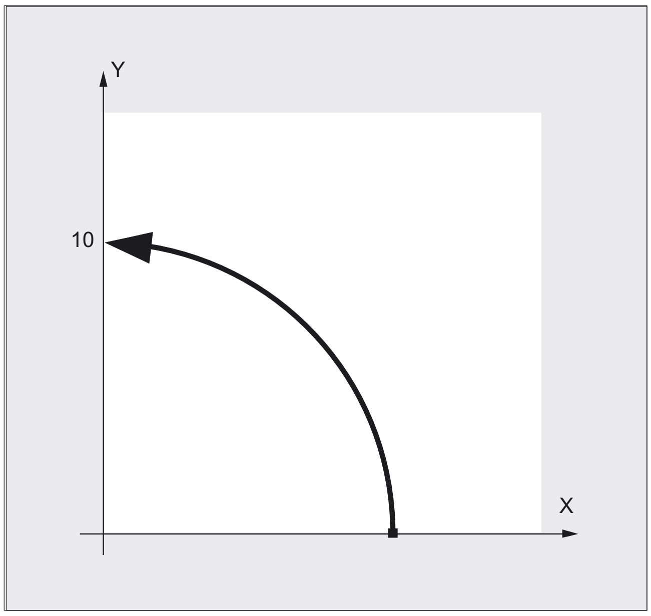
X(p)=10(1)/(1+p2) and Y(p)=20p/(1+p2) where 0<=p<=1

As a result of the programmed start points, end points, coefficient a_2 and PL=1, the intermediate values are as follows:

Numerator (X)=10+0*p–p2

Numerator (Y)=0+20*p+0*p2

Denominator = 1+2*p+1*p2



An alarm is output if a denominator polynomial with zeros is programmed within the interval [0,PL] when polynomial interpolation is active. Denominator polynomials have no effect on the motion of special axes.

Note

Tool radius compensation can be activated with G41, G42 in conjunction with polynomial interpolation and can be applied in the same way as in linear or circular interpolation modes.

5.6 Settable path reference (SPATH, UPATH)

Function

During polynomial interpolation the user may require two different relationships between the velocity-determining FGROUP axes and the other path axes: The latter are to be controlled

- either synchronized with the path of the FGROUP axes
- or synchronized with the curve parameter.

Therefore, for the axes not contained in FGROUP there are two ways to follow the path:

1. Either they travel synchronized with path S (SPATH)
2. or synchronized with the curve parameter U of FGROUP axes (UPATH).

Both types of path interpolation are used in different applications and can be switched via G codes SPATH and UPATH.

Programming

```
SPATH
```
or
```
UPATH
```

Parameters

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 determine the relationship of the F word polynomial (FPOLY, FCUB, FLIN) with the path movement.

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.

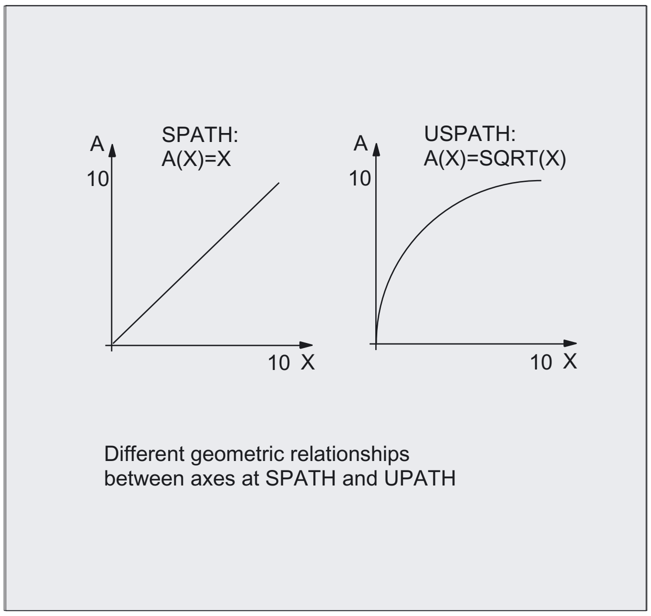
Example 1

The example below shows a square with 20 mm side lengths and corners rounded with G643. The maximum deviations from the exact contour are specified by the machine data MD 33100: COMPRESS_POS_TOL[...] when a contour is smoothed with G643.

```
N10 G1 X… Y… Z… F500
N20 G643                                                ;Block-internal corner rounding with
                                                        ;G643
N30 XO Y0
N40 X20 Y0                                              ;mm edge length for 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.



```
N10 G1 X0 A0 F1000 SPATH
N20 POLY PO[X]=(10, 10) A10
or
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 positions arise for synchronized axis A along the path of 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(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.

5.7 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.

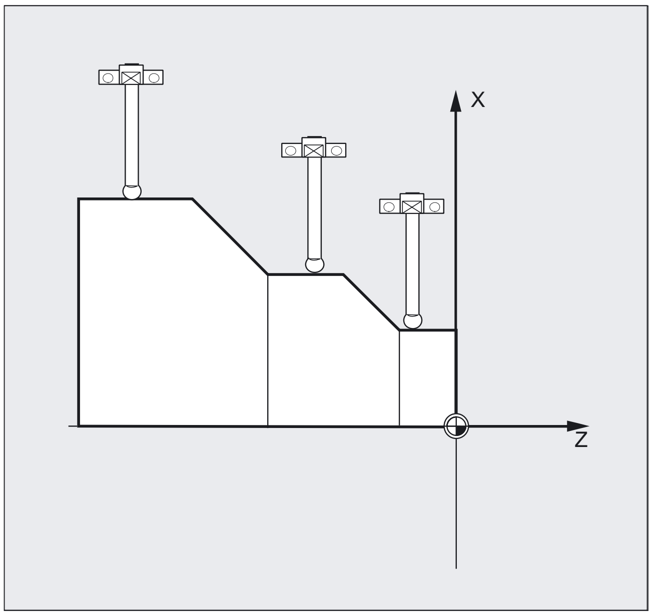
Read measurement result

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.



Programming

Programming measuring 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`
`MEAS=±2`
	`G... X... Y... Z...`
	`G... X... Y... Z...`
		(+1/+2 measurement with deletion of distance-to-go and rising edge)
		(–/– measurement with deletion of distance-to-go and falling edge)

`MEAW=±1`
`MEAW=±2`
	`G... X... Y... Z...`
	`G... X... Y... Z...`
		(+1/+2 measurement without deletion of distance-to-go and rising edge)
		(–/– measurement without deletion of distance-to-go and falling edge)

Parameters

`MEAS=±1`	Measurement with probe 1 at measuring input 1
`MEAS=±2*`	Measurement with probe 2 at measuring input 2
`MEAW=±1`	Measurement with probe 1 at measuring input 1
`MEAW=±2*`	Measurement 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 measuring 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.

Description

Measuring job status

Status variable `$AC_MEA[n]` (n= number of probe) can be scanned if the switching state of the touch-trigger probe needs to be evaluated in the program:

0 Measuring job not satisfied

1 Measuring job completed successfully (probe has switched)

Note

If the probe is deflected during program execution, this variable is set to 1. At the beginning of a measurement block, the variable is automatically set to correspond to the starting state of the probe.

Sensor

The positions of all path and positioning axes (maximum number of axes depends on control configuration) in the block that have moved are recorded. In the case of MEAS, the motion is braked in a defined manner after the probe has switched.

Note

If a GEO axis is programmed in a measuring block, then the measured values are stored for all current GEO axes.

If an axis that participates in a transformation is programmed in a measurement block, the measured values for all axes that participate in this transformation are recorded.

5.8 Extended measuring function (MEASA, MEAWA, MEAC) (option)

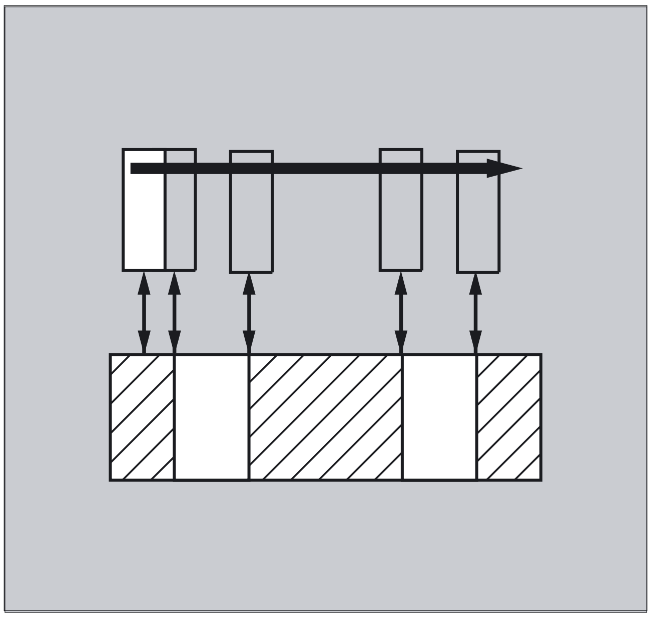
Function

Several probes and several measuring systems can be used for the axial measuring.

When MEASA, MEAWA is programmed, up to four measured values are acquired for the programmed axis in each measuring run and stored in system variables in accordance with the trigger event.

Continuous 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 measuring run is also 4 with MEAC:

- Under `$AA_MM1 to 4[axes]` in the machine coordinate system
- Under `$AA_MM1 to 4[axes]` in the workpiece coordinate system



Programming

`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,…, TE 4)
```

or

```
MEAWA[axis]=(mode, TE1,…, TE 4)
```

or

```
MEAC[axis]=(mode, measurement memory,TE1,…, TE 4)
```

Parameters

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	Two-digit setting for operating mode consisting of **Measuring mode** (ones decade) and 0: Mode **0**: Cancel measuring job 1: Mode **1**: Up to 4 different trigger events can be activated **concurrently**. Trigger events 2: Mode **2**: Up to 4 trigger events can be activated **consecutively** 3: Mode **3**: Up to 4 trigger events can be activated **consecutively** but no monitoring of trigger event 1 on start (alarms 21700/21703 are suppressed) Note: Mode 3 not possible with MEAC **Measuring system** (tens' decade) 0 or no setting: active measuring system 1: Measuring system 1 2: Measuring system 2 3: Both measuring systems
TE 1…4	**Trigger event** 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

```
...
N100 MEASA[X] = (1,1,-1) G01 X100 F100       ;Measurement in mode 1 with active
                                             ;measuring system. Wait for measuring
                                             ;signal with rising/falling edge from
                                             ;probe 1 on travel path to X = 100.
N110 STOPRE                                  ;Preprocessing stop
N120 IF $AC_MEA[1] == FALSE gotof END        ;Check success of measurement.
N130 R10 = $AA_MM1[X]                        ;Store measured value acquired on
                                             ;first programmed trigger event
                                             ;(rising edge)
N140 R11 = $AA_MM2[X]                        ;Store measured value acquired on
                                             ;second programmed trigger event
                                             ;(falling edge)
N150 END:
```

Example of measuring with delete distance-to-go in mode 1

b) with 2 measuring systems

```
...
N200 MEASA[X] = (31,1-1) G01 X100 F100        ;Measurement in mode 1 with both
                                              ;measuring systems. Wait for measuring
                                              ;signal with rising/falling edge from
                                              ;probe 1 on travel path to X = 100.
N210 STOPRE                                   ;Preprocessing stop
N220 IF $AC_MEA[1] == FALSE gotof END         ;Check success of measurement.
N230 R10 = $AA_MM1[X]                         ;Save measured value of the
                                              ;measuring system 1 for rising edge.
N240 R11 = $AA_MM2[X]                         ;Save measured value of the
                                              ;measuring system 2 for rising edge.
N250 R12 = $AA_MM3[X]                         ;Save measured value of the
                                              ;measuring system 1 for falling edge.
N260 R13 = $AA_MM4[X]                         ;Save measured value of the
                                              ;measuring system 2 for falling edge.
N270 END:
```

Example of measuring with delete distance-to-go in mode 2

(evaluation in programmed sequence)

```
...
N100 MEASA[X] = (2,1,-1,2,-2) G01 X100 F100   ;Measurement in mode 2 with active
                                              ;measuring system. Wait for measuring
                                              ;signal in the following order:
                                              ;Rising edge of probe 1,
                                              ;falling edge of probe 1,
                                              ;rising edge of probe 2,
                                              ;falling edge of probe 2,
                                              ;on travel path to X = 100.
N110 STOPRE                                   ;Preprocessing stop
N120 IF $AC_MEA[1] == FALSE gotof             ;Check success of measurement
                                              ;with probe 1
PROBE2
N130 R10 = $AA_MM1[X]                         ;Store measured value acquired on
                                              ;first programmed trigger event
                                              ;(rising edge probe 1).
N140 R11 = $AA_MM2[X]                         ;Store measured value acquired on
                                              ;second programmed trigger event
                                              ;(rising edge probe 1).
N150 PROBE2:
N160 IF $AC_MEA[2] == FALSE gotof END         ;Check success of measurement
                                              ;with probe 2
N170 R12 = $AA_MM3[X]                         ;Store measured value acquired on
                                              ;third programmed trigger event
                                              ;(rising edge probe 2).
N180 R13 = $AA_MM4[X]                         ;Store measured value acquired on
                                              ;fourth programmed trigger event
                                              ;(rising edge probe 2).
N190 END:
```

Example of continuous measuring in mode 1

(evaluation in chronological sequence)

a) Measurement of up to 100 measured values

```
...
N110 DEF REAL MEASVALUE[100]
N120 DEF INT loop = 0
N130 MEAC [X] = (1,1,-1) G01 X1000 F100      ;Measure in mode 1 with active
                                             ;measuring system, store measured
                                             ;values under $AC_FIFO1, wait for
                                             ;measuring signal with falling edge
                                             ;from probe 1 on travel path to
                                             ;X = 1000.
N135 STOPRE
N140 MEAC[X] = (0)                           ;Terminate measurement when
                                             ;axis position is reached.
N150 R1 = $AC_FIFO1[4]                       ;Store number of accumulated measured
                                             ;values in parameter R1.
N160 FOR loop = 0 TO R1-1
N170 MEASVALUE[loop] = $AC_FIFO1[0]          ;Read measured values from $AC_FIFO1
                                             ;and store.
N180 ENDFOR
```

Example of continuous measuring in mode 1

(evaluation in chronological sequence)

b) Measuring with deletion of distance-to-go after 10 measured values

```
...
N10 WHEN $AC_FIFO1[4]>=10 DO                 ;Delete distance to go
 MEAC[x]=(0) DELDTG (x)
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.

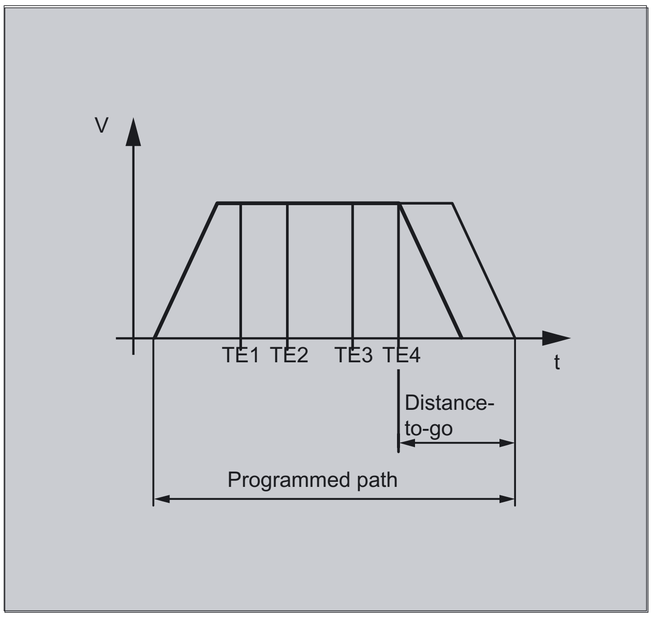
Note

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_MEA[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_MEA` 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
  ```

5.9 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.

5.10 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.

Note

This function is not part of the standard scope of SINUMERIK and must be activated for the relevant software versions.

References:

/FBA/ Functional description ISO Dialects.

Programming

```
FENDNORM
G62 G41
```

or

```
G621
```

Parameters

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 control mode G64, G641

The corner is approached at a reduced feed rate resulting from:

F * (override for feed rate reduction) * feed rate override

The maximum possible feed rate reduction is attained at the precise point where the tool is to change directions at the corner, with reference to the center path.

G621 applies analogously with G62 at each corner, of the axes defined by FGROUP.

5.11 Programmed end-of-motion criterion (FINEA, COARSEA, IPOENDA, IPOBRKA, ADISPOSA)

Function

Similar to the block change criterion for continuous-path interpolation (G601, G602 and G603), the end-of-motion criterion can be programmed in a parts program for single axis interpolation or in synchronized action for the command/PLC.

The end-of-motion criterion set will affect how quickly or slowly parts program blocks and technology cycle blocks with single-axis movements are completed. The same applies for PLC via FC15/16/18.

Programming

`FINEA[Axis]`

or

`COARSEA[Axis]`

or

`IPOENDA[Axis]`

or

`IPOBRKA(axis,[, [value as percentage]])` More than one value can be specified

or

`ADISPOSA(axis, [Int][, [Real]])` More than one value can be specified

Parameters

FINEA	Motion end when "Exact stop FINE" reached
COARSEA	Motion end when "Exact stop COARSE" reached
IPOENDA	Motion end when "Interpolator stop" reached
IPOBRKA	Block change in braking ramp possible (SW 6.2 and higher)
ADISPOSA	Size of tolerance window for end-of-motion criterion (SW 6.4 and higher)
Axis	Channel axis name (X, Y, ...)
Value as percentage	When relative to the braking ramp the block change should be as %
Int	Mode 0: Tolerance window not active
	Mode 1: Tolerance window with respect to set position
	Mode 2: Tolerance window with respect to actual position
Real	Size of tolerance window. This value is entered synchronized with the main run in the setting data 43610: ADISPOSA_VALUE

Example of end-of-motion on reaching the interpolator stop

```
...
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
...
```

Example for block change condition "Braking ramp" in the parts program:

```
         ;Default effective
N40 POS[X]=100
         ;Block change occurs when X-axis reaches position 100 and fine exact stop
N20 IPOBRKA(X,100)     ;Activate block change criterion braking ramp
N30 POS[X]=200         ;Block change occurs as soon as X-axis starts to brake
N40 POS[X]=250
         ;The x-axis does not brake at position 200 but continues to
         ;position 250, the block change occurs as soon as the
         ;X-axis starts to brake
N50 POS[X]=0           ;The X-axis brakes and moves back to position 0
                       ;The block change occurs at position 0 and fine exact stop
N60 X10 F100
N70 M30
...
```

Example for the braking ramp in synchronous actions block change condition

```
In the technology
cycle:
FINEA                  ;End of motion criterion fine exact stop
POS[X]=100             ;Technology cycle block change occurs when X-axis
                       ;has reached position 100 and fine exact stop
IPOBRKA(X,100)         ;Activate block change criterion braking ramp
POS[X]=100             ;POS[X]=100; technology cycle block change occurs,
                       ;as soon as the X-axis starts to brake
POS[X]=250             ;The X-axis does not brake at position 200 but continues
                       ;to position 250, as soon as the X-axis starts to brake
                       ;the block change in the technology cycle occurs
POS[X]=250             ;The X-axis brakes and moves back to position 0
                       ;The block change occurs at position 0 and fine exact stop
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".

5.12 Programmable servo parameter set (SCPARA)

Function

Using SCPARA, it is possible to program the parameter block (consisting of MDs) in the parts program and in synchronized actions (previously only via PLC).

DB3n DBB9 bit3

To ensure no conflicts occur between PLC and NC, an additional bit is defined on the PLC–>NCK interface:

DB3n DBB9 bit3 "Parameter set selection by SCPARA disabled".

If parameter set selection via SCPARA is disabled, there is no error message if the latter is programmed nevertheless.

Programming

```
SCPARA[Axis]= value
```

Parameters

SCPARA	Define parameter block
Axis	Channel axis name (X, Y, ...)
Value	Desired parameter block (1<= value <=6)

Note

The current parameter set can be scanned by system variable `$AA_SCPAR[axis]`.

For G33, G331 and G332, the most suitable parameter block is selected by the control.

If the **servo parameter set** has to be **changed** in both a parts program or a synchronized action and on the PLC, the PLC user program must be extended.

References:
/FB1/ Function Manual Basic Functions; Feedrates (V1), "Feedrate Impact" section.

Example

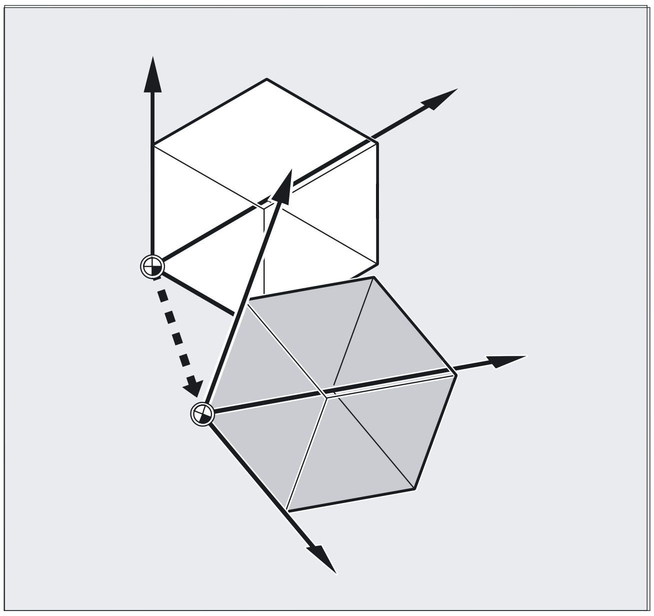
```
...
N110 SCPARA[X]= 3          ;The 3rd parameter block is selected for axis X.
...
```

Frames 6

6.1 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

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.

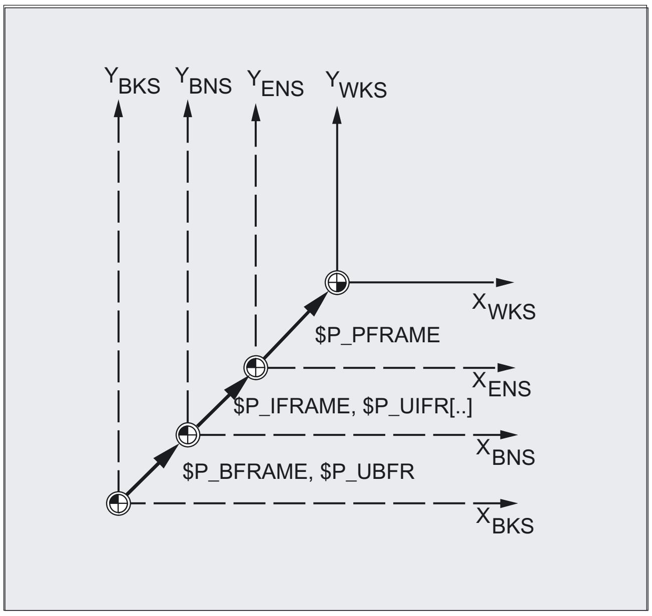
Example: `$P_PFRAME=CTRANS(X,10)`

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

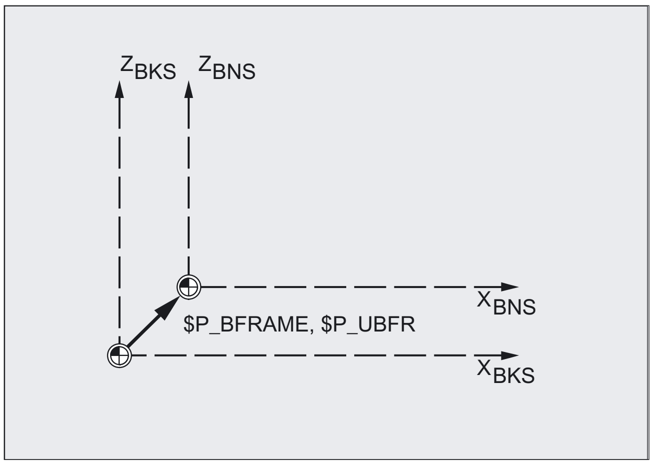
6.1.1 Predefined frame variable ($P_BFRAME, $P_IFRAME, $P_PFRAME, $P_ACTFRAME)

$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

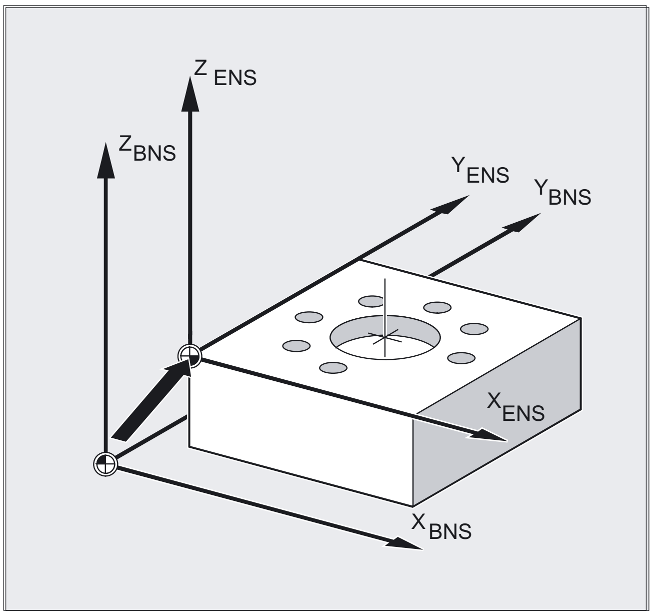
- you have to program a `G500, G54...G599`, or
- you have to describe `$P_BFRAME` with `$P_UBFR`



$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.

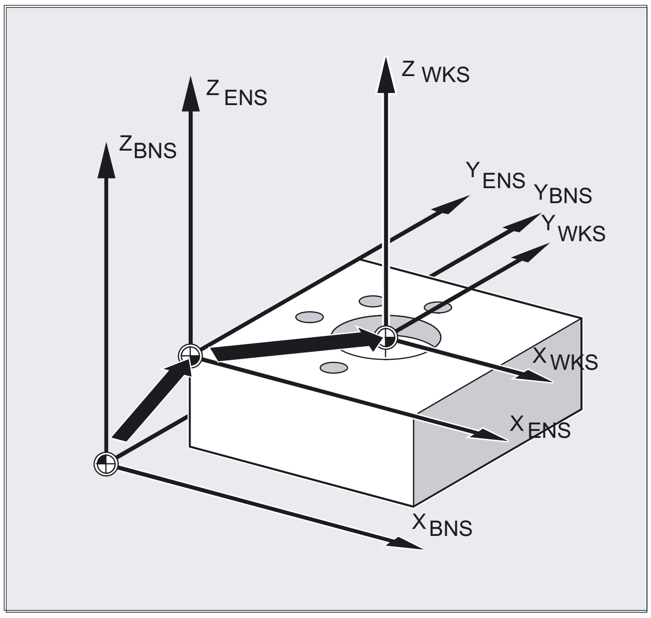


$P_PFRAME

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



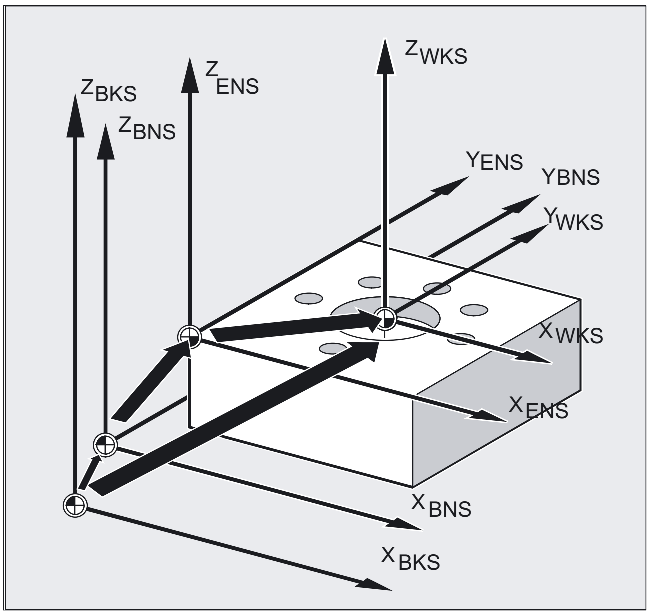
$P_ACTFRAME

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.

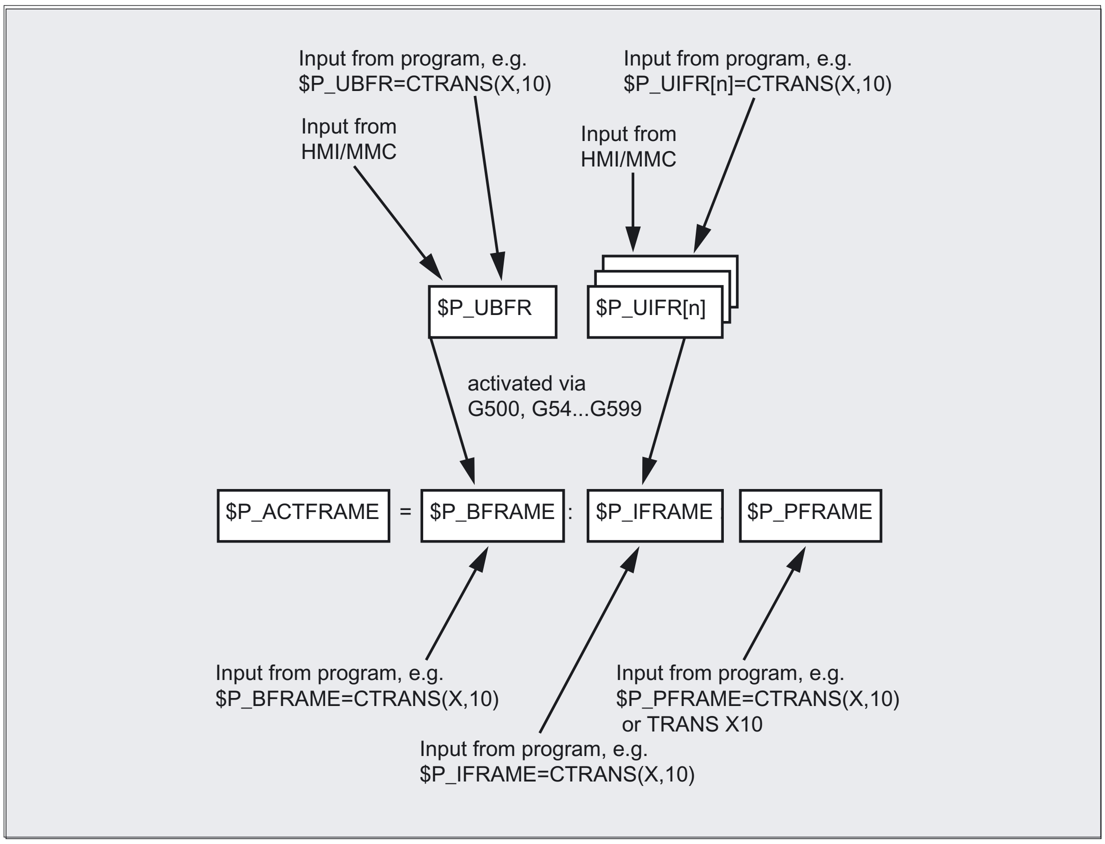
System frames, see Section "Frames that Act in the Channel"

`$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.

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, ...`

6.2 Frame variables / assigning values to frames

6.2.1 Assigning direct values (axis value, angle, scale)

Function

You can directly assign values to frames or frame variables in the NC program.

Programming

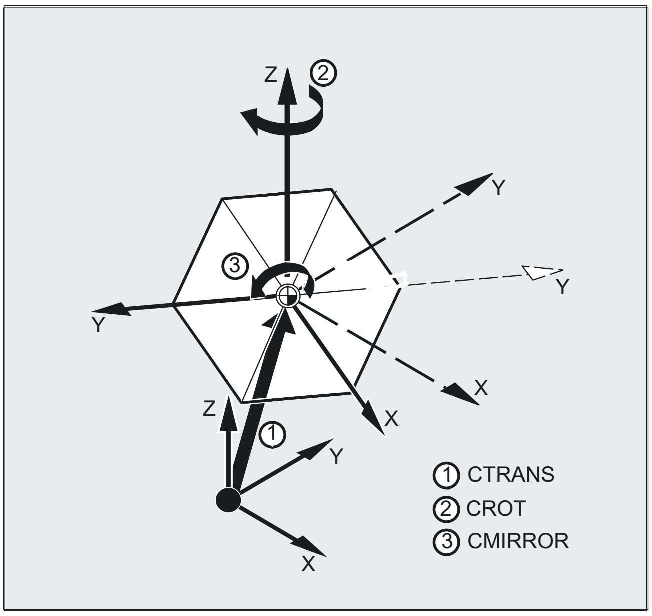
```
$P_PFRAME=CTRANS (X, axis value, Y, axis value, Z, axis value, …)
```
or
```
$P_PFRAME=CROT (X, angle, Y, angle, Z, angle, …)
```
or
```
$P_UIFR[..]=CROT (X, angle, Y, angle, Z, angle, …)
```
or
```
$P_PFRAME=CSCALE (X, scale, Y, scale, Z, scale, …)
```
or
```
$P_PFRAME=CMIRROR (X, Y, Z)
```
Programming $P_BFRAME is carried out analog to $P_PFRAME.

Parameters

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

```
$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
```

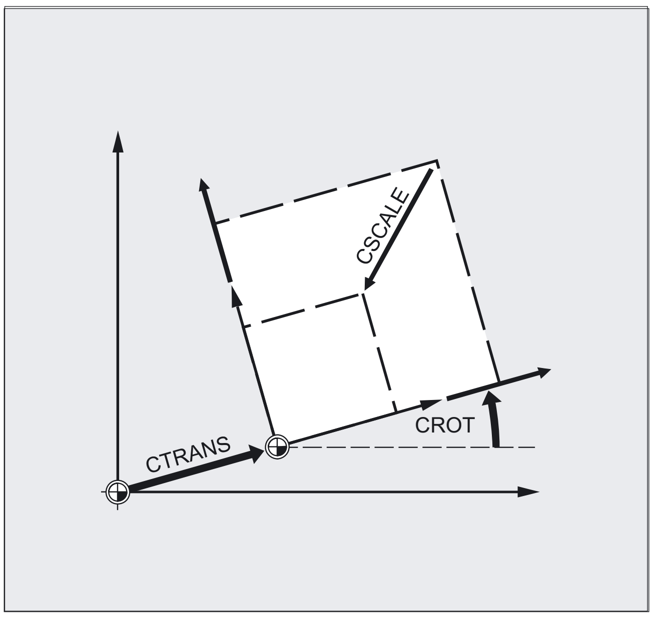
Description

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`.

6.2.2 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.

Programming

`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.

Parameters

$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 change,
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:	-89.999° to +90°
Rotation around 3rd geometry axis:	-180° to +180°

Description

Calling frame

By specifying the system variable $P_UIFRNUM you can access the current zero offset set with $P_UIFR or G54, G55, ...
($P_UIFRNUM contains the number of the currently set frame).

All other stored settable $P_UIFR frames are called up by specifying the appropriate number $P_UIFR[n].

For predefined frame variables and user-defined frames, specify the name, e.g., $P_IFRAME.

Calling data

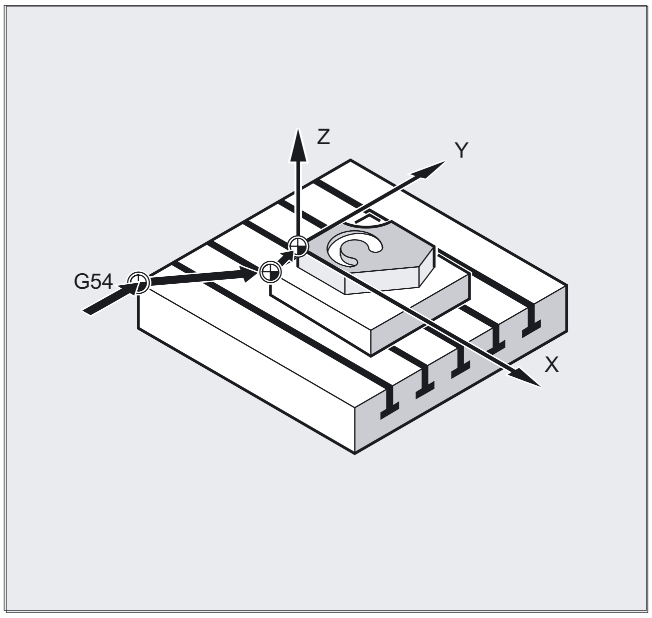
The axis name and the frame component of the value you want to access or modify are written in square brackets, e.g., [X, RT] or [Z, MI].

6.2.3 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.

Programming

Assigning frames

`DEF FRAME SETTING1`
`SETTING1=CTRANS(X,10)`
`$P_PFRAME=SETTING1`
	`Assign the values of the user frame SETTING1 to the current programmable frame.`

`DEF FRAME SETTING4`
`SETTING4=$P_PFRAME`
`$P_PFRAME=SETTING4`
	`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 : .

6.2.4 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.

Programming

```
DEF FRAME PALETTE1
```

or

```
PALETTE1=CTRANS(…):CROT(…)…
```

Parameters

DEF FRAME	Creating new frames
PALETTE1	Name of the new frame
=CTRANS(...): CROT(...)...	Assigning values to the possible functions

6.2.5 Specifying frame rotations (ROT, ROTS, TOFRAME, TOROT, PAROT)

Function

Frame rotations can be used to define application-specific orientations in the area.

Parameters

ROT	Individual rotations for all geometry axes.
ROTS, AROTS, CROTS	Rotation by specifying a solid angle (max. 2); see description in /FB1/ K2: Coordinate systems.
TOFRAME	Rotation by frame "TOFRAME", with Z axis pointing in the tool direction.
TOROT	Rotation by frame "TOROT", which only overwrites the rotation component of frames that have already been programmed.
PAROT	Workpiece-oriented frame rotation. The rotation component is determined by the rotation component of an oriented toolholder.

6.3 Coarse and fine offsets (CFINE; CTRANS)

Function

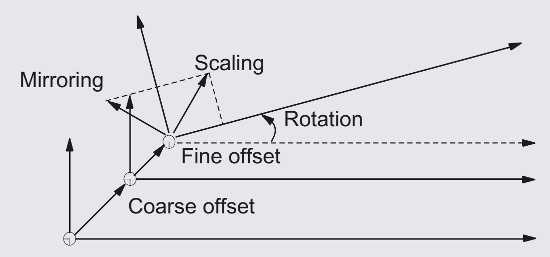
Fine offset

A fine offset of the basic frames and of all other settable frames can be programmed with command `CFINE (X, ..,Y, ...)`.

Fine offset can only take place if MD 18600: MM_FRAME_FINE_TRANS=1.

Coarse offset

The coarse offset is defined with `CTRANS(...)`.



Frame structure with fine offset

Coarse and fine offset add up to the total offset.

Programming

`$P_UBFR=CTRANS(x, 10) : CFINE(x, 0.1) : CROT(x, 45)`	;Chaining of offset, ;fine offset and rotation
`$P_UIFR[1]=CFINE(x, 0.5 y, 1.0, z, 0.1)`	;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

Parameters

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

With MD18600: MM_FRAME_FINE_TRANS is used to configure the fine offset for the following variants:

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.

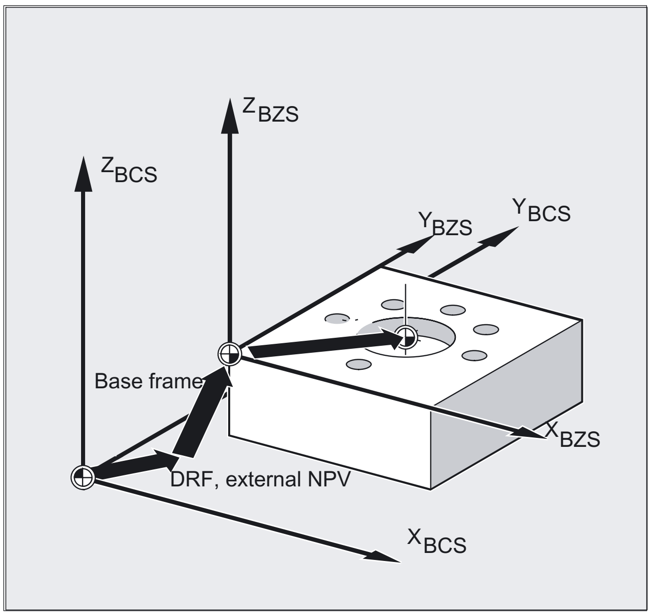
The programmable frame has no fine offset. If the programmable frame is assigned a frame with fine offset, then the total offset is established by adding the coarse and the fine offset. When reading the programmable frame the fine offset is always zero.

6.4 DRF offset

Offset using the handwheel, DRF

In addition to all the translations described in this section, you can also define zero offsets with the handwheel (DRF offset).

In the basic coordinate system, DRF offset affects geometry axes and special axes:



However, a handwheel assignment must be made for the machine axis (e.g., via "Activate handwheel" NC/PLC interface signals), to which the geometry axis and special axis can be mapped. You will find more information in the appropriate Operator's Guide.

Clear DRF offset, DRFOF

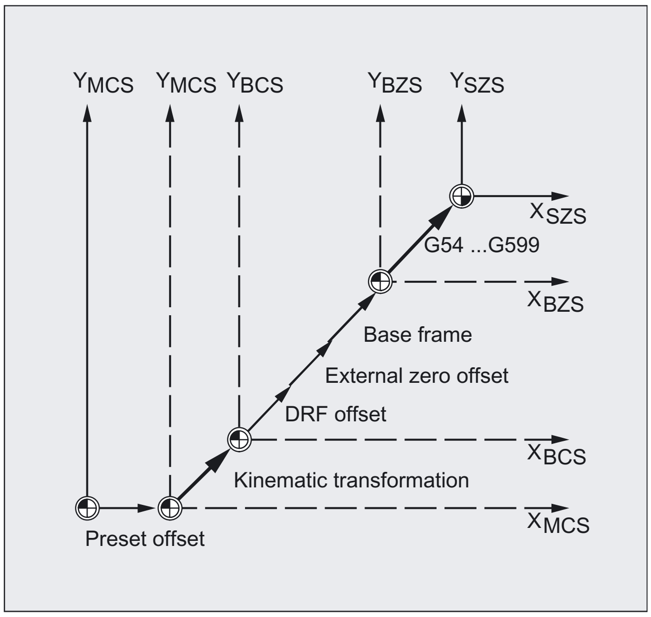
DRFOF clears the handwheel offset for all axes assigned to the channel. DRFOF is programmed in a separate NC block.

6.5 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).

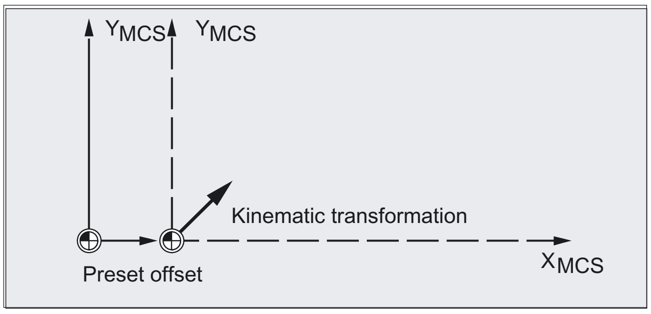
6.6 Preset offset (PRESETON)

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).

Caution

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.



Programming

```
PRESETON(axis, value, ...)
```

Parameters

`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.

6.7 Deactivating frames (DRFOF, G53, G153, and SUPA)

Function

The programmable frames are cleared by assigning a "zero frame" (without axis specification) to the programmable frame.

Programming

`DRFOF`

or

`G53`

or

`G153`

or

`SUPA`

Parameters

DRFOF	Deactivate (clear) the handwheel offsets (DRF)
G53	Non-modal deactivation of programmable and all settable frames
G153	Non-modal deactivation of programmable frames, basic frames and all settable frames
SUPA	Non-modal deactivation of all programmable frames, basic frames, all settable frames and handwheel offsets (DRF)

Example of the assignment of a zero frame

```
$P_PFRAME=TRANS( )
$P_PFRAME=ROT( )
$P_PFRAME=SCALE( )
$P_PFRAME=MIRROR( )
```

6.8 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 parts 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.

Programming

```
MEAFRAME IDEAL_POINT,MEAS_POINT,FIT_QUALITY)
```

Parameters

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.
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".

Example

```
; parts program 1
;
DEF FRAME CORR_FRAME
;
;Setting measuring points
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); for test
DEF REAL FIT_QUALITY = 0
;
DEF REAL ROT_FRAME_LIMIT = 5 ;permits max. 5 degree rotation
;of the parts 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]
;========================================================
;
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 ;limitation of the 1st RPY
;angle
SETAL(65020)
GOTOF NO_FRAME
ENDIF
;
N260 IF CORR_FRAME[Y,RT] > ROT_FRAME_LIMIT ;limitation of the 2nd
;RPY
;angle
SETAL(65021)
GOTOF NO_FRAME
ENDIF
;
N270 IF CORR_FRAME[Z,RT] > ROT_FRAME_LIMIT ;limitation of the 3rd RPY
;angle
SETAL(65022)
GOTOF NO_FRAME
ENDIF
;
N300 $P_IFRAME=CORR_FRAME ;activate the probe frame via a settable frame
;
;check the frame by positioning the geometry axes at the ideal points
;
N400 X=IDEAL_POINT[0,0] Y=IDEAL_POINT[0,1] Z=IDEAL_POINT[0,2]
N410 SHOW_MCS_POS1[0]=$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, because zero frame preset (no value set)
;
NO_FRAME:
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.

6.9 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.

6.9.1 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".

6.9.2 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

Current system frames	For:
$P_PARTFRAME	TCARR and PAROT
$P_SETFRAME	PRESET and scratching
$P_EXTFRAME	External zero offset
$P_NCBFRAME[n]	Current NCU-global basic frames
$P_CHBFRAME[n]	Current channel basic frames
$P_BFRAME	Current first basic frame in the channel
$P_ACTBFRAME	Complete basic frame
$P_CHBFRMASK and $P_NCBFRMASK	Complete basic frame
$P_IFRAME	Current settable frame
Current system frames	For:
$P_TOOLFRAME	TOROT and TOFRAME
$P_WPFRAME	Workpiece reference points
$P_TRAFRAME	Transformations
$P_PFRAME	Current programmable frame
Current system frame	For:
$P_CYCFRAME	cycles
P_ACTFRAME	Current total frame
FRAME chaining	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.

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.

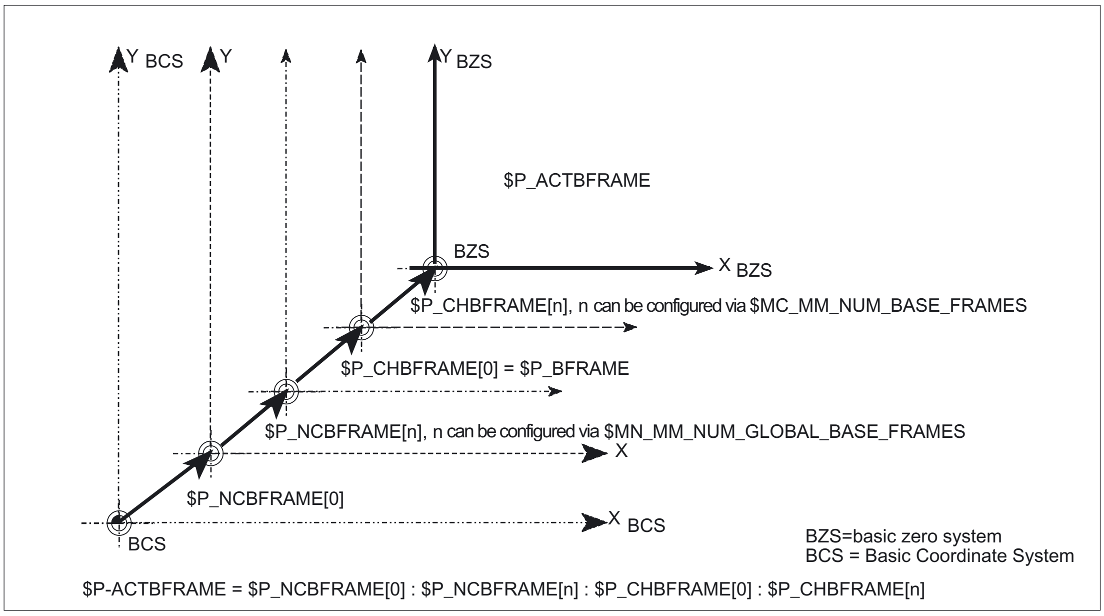
`$P_UBFR` is identical to `$P_CHBFR[0]`. The system variable always has a valid default value. If there is no channel-specific basic frame, an alarm is issued at read/write: "Frame: instruction not permissible".

$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].`


Y BCS
Y
Y BZS
$P_ACTBFRAME
BZS
X BZS
$P_CHBFRAME[n], n can be configured via $MC_MM_NUM_BASE_FRAMES
$P_CHBFRAME[0] = $P_BFRAME
$P_NCBFRAME[n], n can be configured via $MN_MM_NUM_GLOBAL_BASE_FRAMES
X
$P_NCBFRAME[0]
X BCS
BCS
BZS=basic zero system
BCS = Basic Coordinate System
$P-ACTBFRAME = $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)

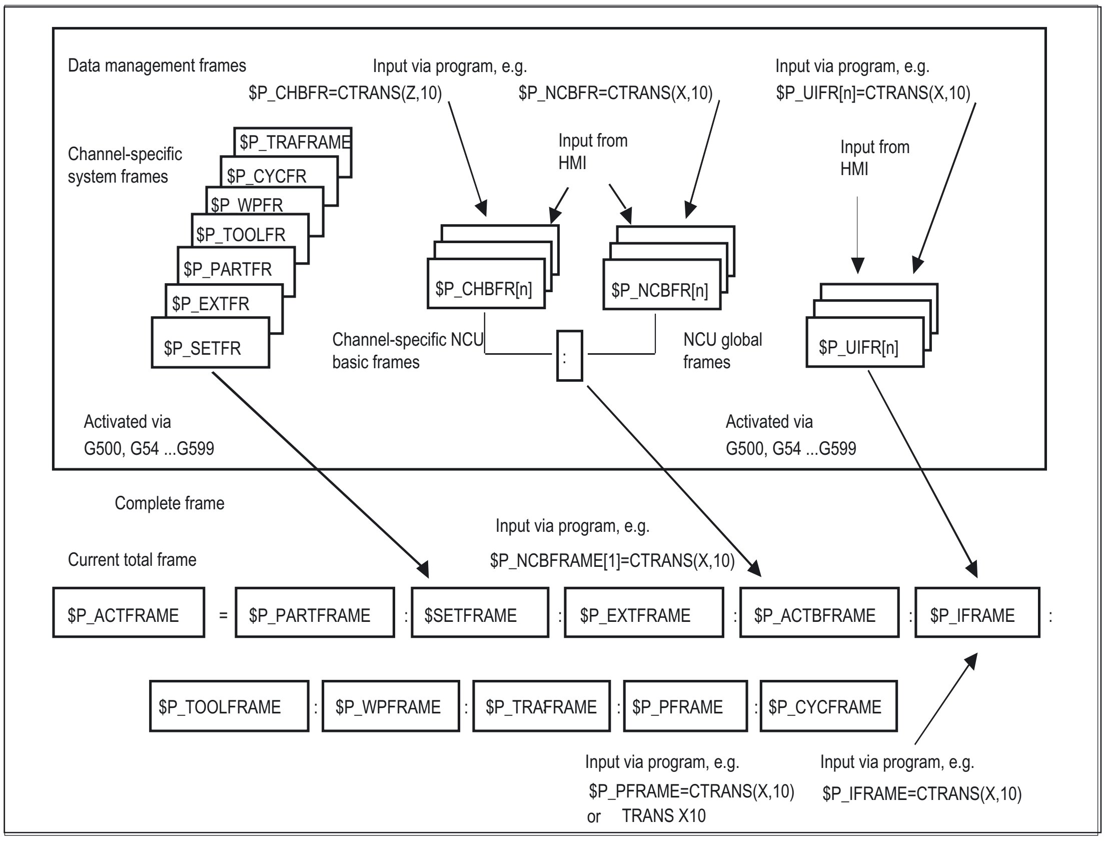
.

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.

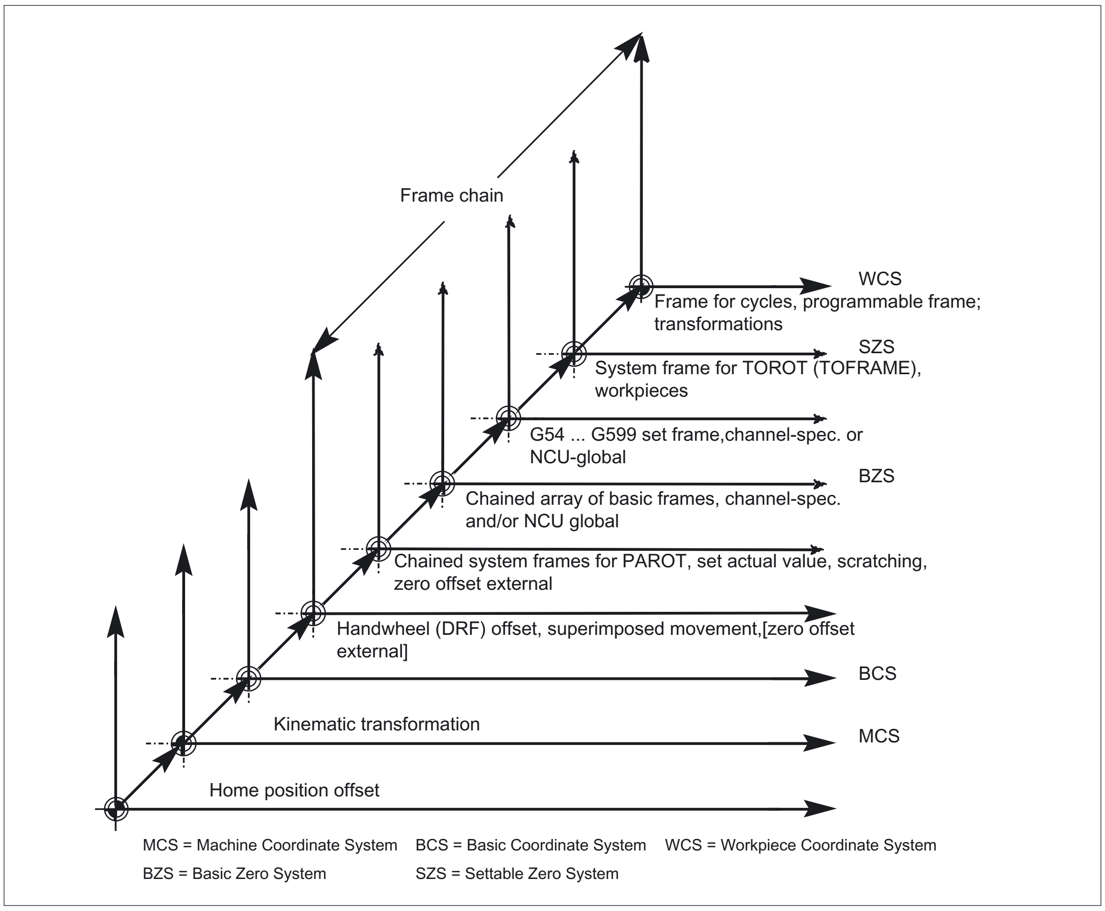
`$P_ACTFRAME` corresponds to

```
$P_PARTFRAME : $P_SETFRAME : $P_EXTFRAME : $P_ACTBFRAME : $P_IFRAME :
$P_TOOLFRAME : $P_WPFRAME : $P_TRAFRAME : $P_PFRAME : $P_CYCFRAME
```


Data management frames
Input via program, e.g.
$P_CHBFR=CTRANS(Z,10)
$P_NCBFR=CTRANS(X,10)
Input via program, e.g.
$P_UIFR[n]=CTRANS(X,10)
Channel-specific system frames
$P_TRAFRAME
$P_CYCFR
$P_WPFR
$P_TOOLFR
$P_PARTFR
$P_EXTFR
$P_SETFR
Input from HMI
Input from HMI
$P_CHBFR[n]
$P_NCBFR[n]
$P_UIFR[n]
Channel-specific NCU basic frames
:
NCU global frames
Activated via G500, G54 ...G599
Activated via G500, G54 ...G599
Complete frame
Input via program, e.g.
$P_NCBFRAME[1]=CTRANS(X,10)
Current total frame
$P_ACTFRAME
=
$P_PARTFRAME
:
$SETFRAME
:
$P_EXTFRAME
:
$P_ACTBFRAME
:
$P_IFRAME
:
$P_TOOLFRAME
:
$P_WPFRAME
:
$P_TRAFRAME
:
$P_PFRAME
:
$P_CYCFRAME
Input via program, e.g.
$P_PFRAME=CTRANS(X,10)
or TRANS X10
Input via program, e.g.
$P_IFRAME=CTRANS(X,10)

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.


Frame chain
WCS
Frame for cycles, programmable frame; transformations
SZS
System frame for TOROT (TOFRAME), workpieces
G54 ... G599 set frame,channel-spec. or NCU-global
BZS
Chained array of basic frames, channel-spec. and/or NCU global
Chained system frames for PAROT, set actual value, scratching, zero offset external
Handwheel (DRF) offset, superimposed movement,[zero offset external]
BCS
Kinematic transformation
MCS
Home position offset
MCS = Machine Coordinate System
BCS = Basic Coordinate System
WCS = Workpiece Coordinate System
BZS = Basic Zero System
SZS = Settable Zero System

7 Transformations

7.1 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)
```

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)

Chained transformations

Two transformations can be switched one after the other. For the second transformation chained here, the motion parts for the axes are taken from the first transformation.

The first transformation can be:

- orientation transformation TRAORI
- polar transformation TRANSMIT
- cylinder transformation TRACYL
- inclined axis transformation TRAANG

The second transformation must be a TRAANG type transformation for an inclined axis.

7.1.1 Orientation movements for transformations

Travel movements and orientation movements

The traversing movements of the programmed orientations are determined primarily by the type of machine. For three-, four-, and five-axis type transformations with TRAORI, the rotary axes or pivoting linear axes describe the orientation movements of the tool.

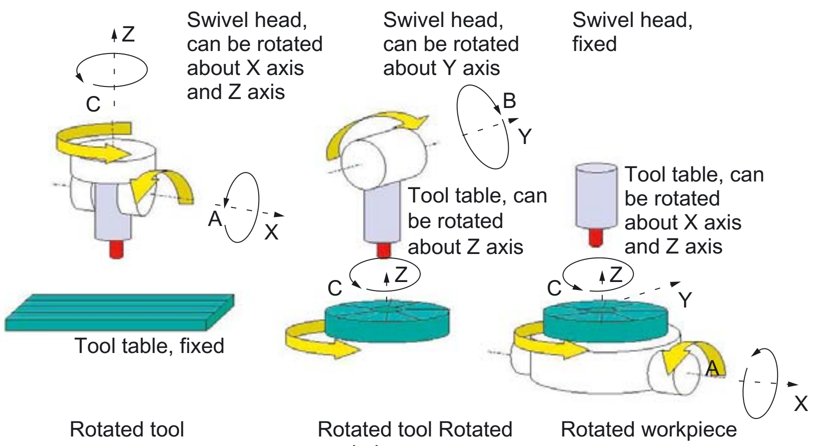
Changes in the position of the rotary axes involved in the orientation transformation will induce compensating movements on the remaining machine axes. The position of the tool tip remains unchanged.

Orientation movements of the tool can be programmed using the rotary axis identifiers A…, B…, C… of the virtual axes as appropriate for the application either by entering Euler or RPY angles or directional or surface normal vectors, normalized vectors for the rotary axis of a taper or for intermediate orientation on the peripheral surface of a taper.

In the case of kinematic transformation with TRANSMIT, TRACYL and TRAANG, the 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.


Z
C
Swivel head, can be rotated about X axis and Z axis
A
X
Tool table, fixed
Swivel head, can be rotated about Y axis
B
Y
Tool table, can be rotated about Z axis
C
Z
Swivel head, fixed
Tool table, can be rotated about X axis and Z axis
C
Z
Y
A
X
Rotated tool
Rotated tool Rotated workpiece
Rotated workpiece
Machine type 1 with axis sequence CA
Machine type 3 with axis sequence BC
Machine type 2 with axis sequence AC

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 **two** translatory axes (linear axes) and **one** axis of rotation (rotary axis).
Four-axis transformation machine types 1 and 2	Programming of tool orientation only in the plane, which is vertical to the rotary axis. There are **three** translatory axes (linear axes) and **one** axis of rotation (rotary axis).
Five-axis transformation machine types 3 Single-axis swivel head and single-axis rotary table	Programming of orientation transformation. Kinematics with **three** linear axes and **two** orthogonal rotary axes. The rotary axes are parallel to two of the three linear axes. The first rotary axis is moved by two Cartesian linear axes. It rotates the third linear axis with the tool. The second rotary axis rotates the workpiece.

Generic 5/6-axis transformations

Machine type	Programming of orientation transformation
Generic five/six-axis transformation machine types 4 Two-axis swivel head with tool which rotates around itself and single-axis rotary table	Programming of orientation transformation. Kinematics with **three** linear axes and **three** orthogonal rotary axes. The rotary axes are parallel to two of the three linear axes. The first rotary axis is moved by two Cartesian linear axes. It rotates the third linear axis with the tool. The second rotary axis rotates the workpiece. The basic tool orientation can also be programmed with additional rotation of the tool around itself with the THETA rotary angle.

When calling "generic three-, four-, and five/six-axis transformation", the basic orientation of the tool can also be transferred. The restrictions in respect of the directions of the rotary axes no longer apply. If the rotary axes are not exactly vertical to one another or existing rotary axes are not exactly parallel with the linear axes, "generic five-/six-axis transformation" can provide better results in respect of tool orientation.

Kinematic transformations TRANSMIT, TRACYL and TRAANG

For milling on turning machines or an axis that can be set for inclined infeed during grinding, the following axis arrangements apply by default in accordance with the transformation declared:

TRANMIT	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.

7.1.2 Overview of orientation transformation TRAORI

Programming types available in conjunction with TRAORI

Machine type	Programming with active transformation TRAORI
Machine types 1, 2, or 3 two-axis swivel head or two-axis rotary table or a combination of single-axis swivel head and single-axis rotary table.	The axis sequence of the orientation axes and the orientation direction of the tool can either be configured on a **machine-specific** basis using machine data depending on the machine kinematics or on a **workpiece-specific** basis with programmable orientation independently of the machine kinematics. The directions of rotation of the orientation axes in the reference system are programmed with: - ORIMKS reference system = machine coordinate system - ORIWKS reference system = workpiece coordinate system The default setting is ORIWKS.
	Programming of orientation axes with: A, B, C of machine axis position directly A2, B2, C2 angular programming of virtual axes with - ORIEULER using Euler angle (default) - ORIRPY using RPY angle - ORIVIRT1 using virtual orientation axes 1st definition - ORIVIRT2 using virtual orientation axes 2nd definition with differentiation of interpolation type: **Linear interpolation** - ORIAXES of orientation axes or machine axes **Large-radius circular interpolation** (interpolation of orientation vector) - ORIVECT of orientation axes Programming of orientation axes by entering A3, B3, C3 of vector components (direction/surface normal) Programming of resulting tool orientation A4, B4, C4 of surface normal vector at start of block A5, B5, C5 of surface normal vector at end of block LEAD angle for tool orientation TILT angle for tool orientation **Interpolation of orientation vector** on the peripheral surface of a taper Changes in orientation on the peripheral surface of a taper located **anywhere in space** by means of interpolation: - ORIPLANE in the plane (large-radius circular interpolation) - ORICONCW on the peripheral surface of a taper clockwise - ORICONCCW on the peripheral surface of a taper counter-clockwise A6, B6, C6 direction vectors (rotary axis of taper) -OICONIO interpolation on the peripheral surface of a taper with: A7, B7, C7 intermediate vectors (initial and ultimate orientation)

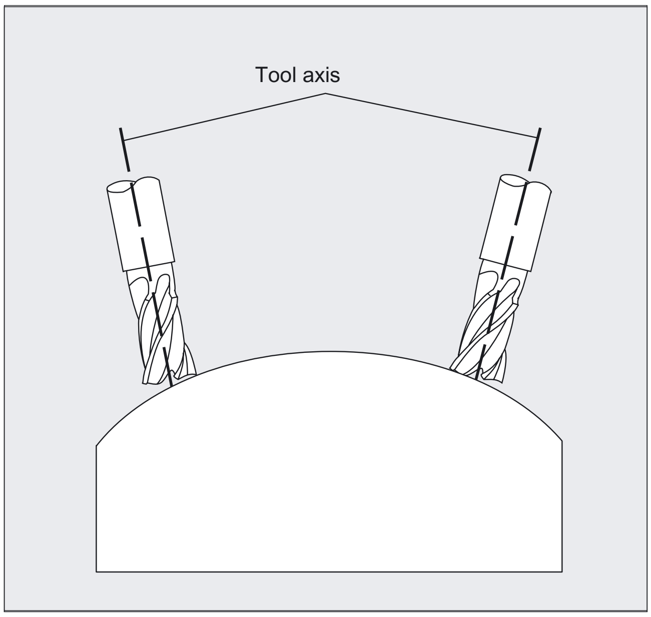
Machine type	Programming with active transformation TRAORI
	or - ORICONTO on the peripheral surface of a taper, tangential transition Changes in orientation in relation **to a path** with - ORICURVE specification of the movement of two contact points using PO[XH]=(xe, x2, x3, x4, x5) orientation polynomials up to the fifth degree PO[YH]=(ye, y2, y3, y4, y5) orientation polynomials up to the fifth degree PO[ZH]=(ze, z2, z3, z4, z5) orientation polynomials up to the fifth degree - ORIPATHS smoothing of orientation characteristic with A8, B8, C8 reorientation phase of tool corresponding to: direction and path length of tool during retraction movement
Machine types 1 and 3	Programming **of rotations** for tool orientation with LEAD angle, angle relative to surface normal vector PO[PHI] programming of a polynomial up to the fifth degree TILT angle rotation about path tangent (Z direction) PO[PSI] programming of a polynomial up to the fifth degree
Other machine types with additional tool rotation around itself require a 3rd round axis	THETA angle of rotation (rotation about tool direction in Z) THETA= value reached at end of block THETA=AC(...) absolute non-modal switching to dimensions THETA=IC(...) non-modal switching to chain dimensions THETA=Θ_e interpolate programmed angle G90/G91 PO[THT]=(..) programming of a polynomial up to the fifth degree
Orientation transformation, e.g. generic 6-axis transformation. Rotations of orientation vector.	programming of the rotation vector - ORIROTA rotation, absolute - ORIROTR relative rotation vector - ORIROTT tangential rotation vector
Orientation relative to the path for orientation changes relative to the path or rotation of the rotary vector tangentially to the path	Changes in orientation **relative to the path** with - ORIPATH tool orientation relative to the path - ORIPATHS also in the event of a blip in the orientation characteristic programming of rotation vector - ORIROTC tangential rotation vector, rotation to path tangent

7.2 Three, four and five axis transformation (TRAORI)

7.2.1 General relationships of universal tool head

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.

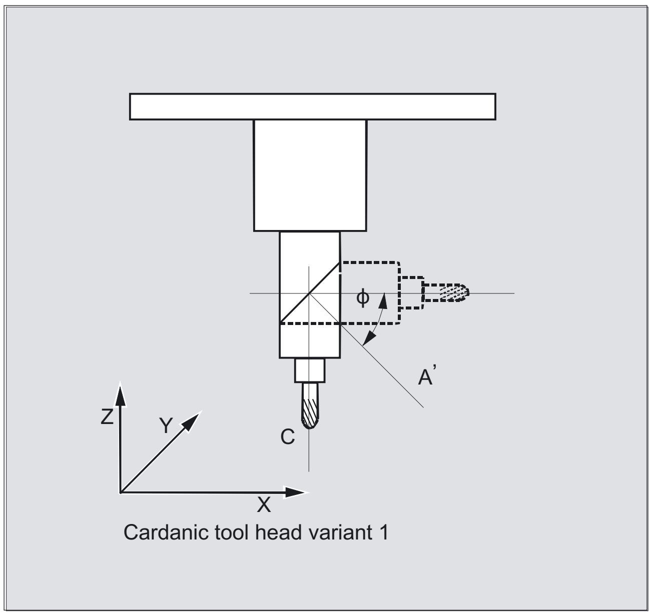


The machine design to achieve this is stored in the axis data.

5-Axis Transformation

Cardanic tool head

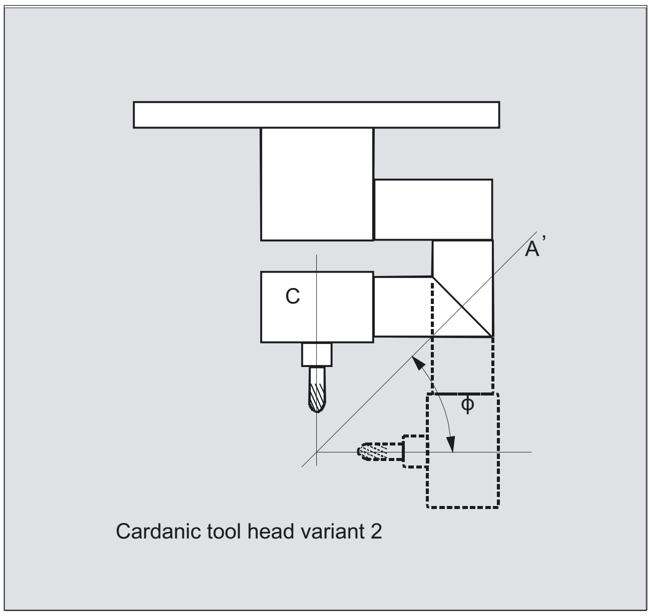
Three linear axes (X, Y, Z) and two orientation axes (C, A) define the setting angle and the operating point of the tool here. One of the two orientation axes is created as an inclined axis, in our example A' - in many cases, placed at 45°.



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 φ to the X axis.

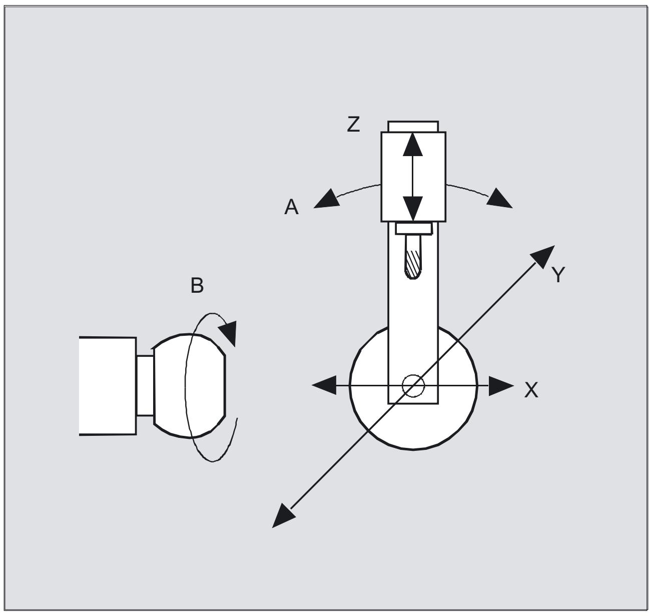
The following possible relations are generally valid:

A' lies below the angle φ to the	X axis
B' lies below the angle φ to the	Y axis
C' lies below the angle φ to the	Z axis

Angle φ can be configured in the range 0° to +89° using machine data.

With swiveling linear axis

This is an arrangement with a moving workpiece and a moving tool. The kinematics consists of three linear axes (X, Y, Z) and two orthogonally arranged rotary axes. The first rotary axis is moved, for example, over a compound slide of two linear axes, the tool standing parallel to the third linear axis. The second rotary axis turns the workpiece. The third linear axis (swivel axis) lies in the compound slide plane.



The axis sequence of the rotary axes and the orientation direction of the tool can be set up using the machine data as appropriate for the machine kinematics.

There are the following possible relationships:

Axes:	**Axis sequences:**
1. Rotary axis	A A B B C C
2. Rotary axis	B C A C A B
Swiveled linear axis	Z Y Z X Y X

For more detailed information about configurable axis sequences for the orientation direction of the tool, see

References: /FB3/ Function Manual, Special Functions; 3- to 5-Axis Transformations (F2), Universal Milling Head section, "Parameter Setting".

7.2.2 Three, four and five axis transformation (TRAORI)

Function

The user can configure two or three translatory axes and one rotary axis. The transformations assume that the rotary axis is orthogonal on the orientation plane.

Orientation of the tool is possible only in the plane perpendicular to the rotary axis. The transformation supports machine types with movable tool and movable workpiece.

Three- and four-axis transformations are configured and programmed in the same way as five-axis transformations.

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.

7.2.3 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.

7.2.4 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, ORIAXPOS` and `ORIPY2`. The values programmed with the orientation axes are also interpreted as orientation angles in accordance with the active G code of group 50.
3. If the machine data for defining the orientation angle is set to one via G code and the machine data for defining the orientation axes is set to zero via G code, the changeover is
   not dependent on the active G code of group 50:
   The angles programmed using `A2, B2, C2` are interpreted in accordance with one of the active G codes `ORIEULER, ORIRPY, ORIVIRT1, ORIVIRT2 ORIAXPOS` and `ORIPY2`. The values programmed with the orientation axes are always interpreted as round axis positions irrespective of the active G code of group 50.

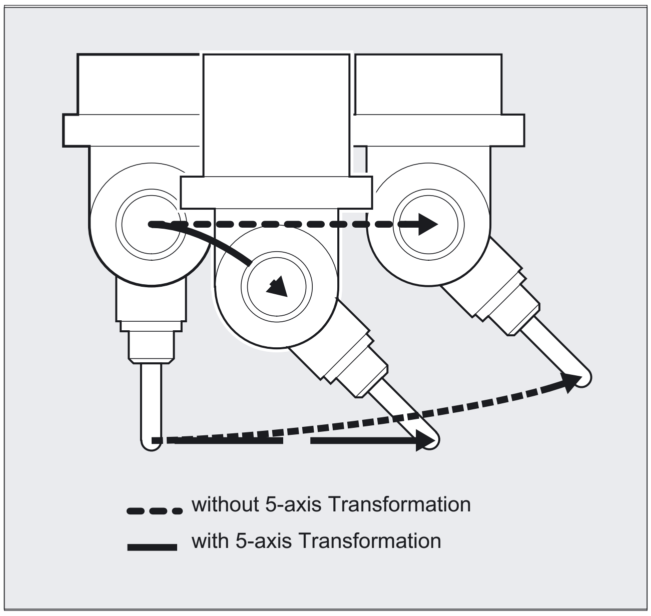
Programming

`G1 X Y Z A B C`	Programming 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)

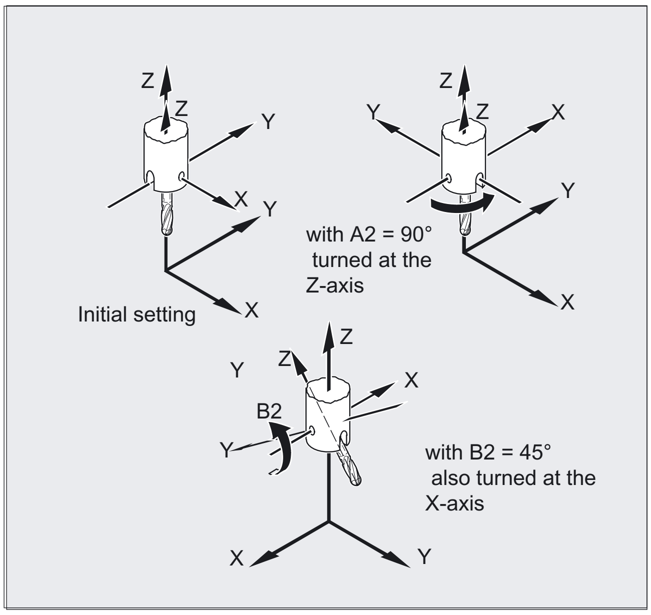
Machine manufacturer

The machine manufacturer can use machine data to define various variants. Please refer to the machine manufacturer's instructions.

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.


Z
Z
Y
X
Y
Initial setting
X
Z
Y
Z
X
Y
with A2 = 90° turned at the Z-axis
X
Z
Y
Z
X
B2
Y
with B2 = 45° also turned at the X-axis
X
Y

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).

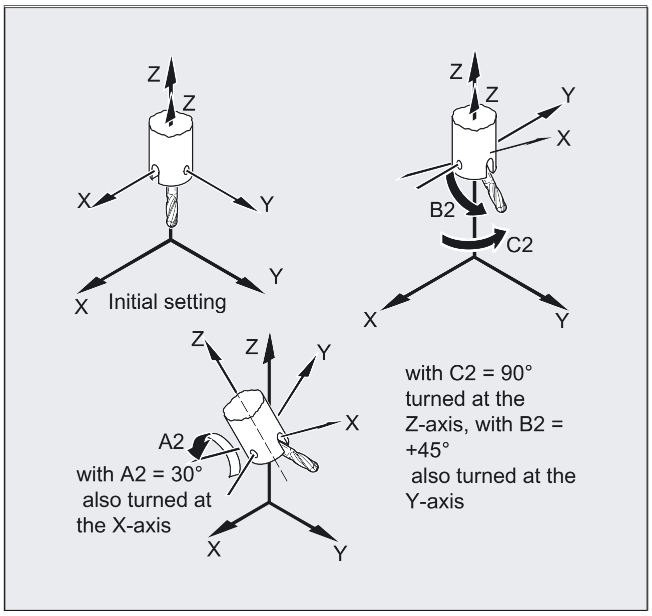
Note

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.

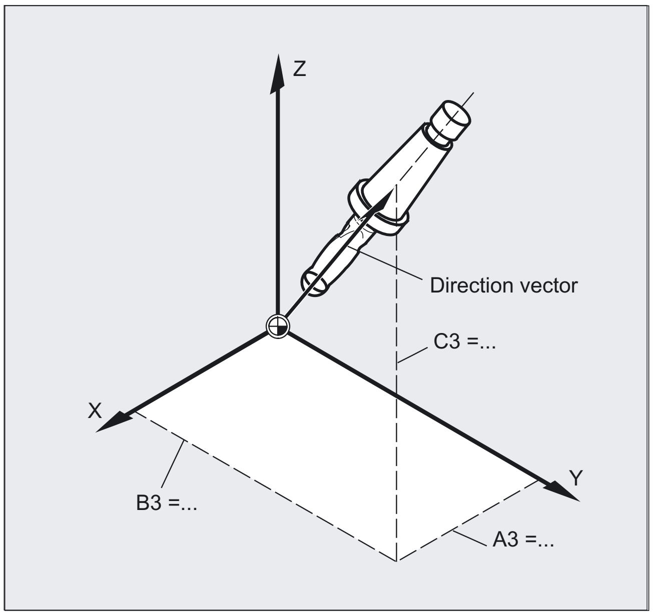

Z
Z
X
Y
Y
X
Initial setting
Z
Z
Y
X
B2
C2
X
Y
with C2 = 90° turned at the Z-axis, with B2 = +45° also turned at the Y-axis
Z
Z
Y
X
A2
with A2 = 30° also turned at the X-axis
X
Y

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 LEADʼ= and TILT=

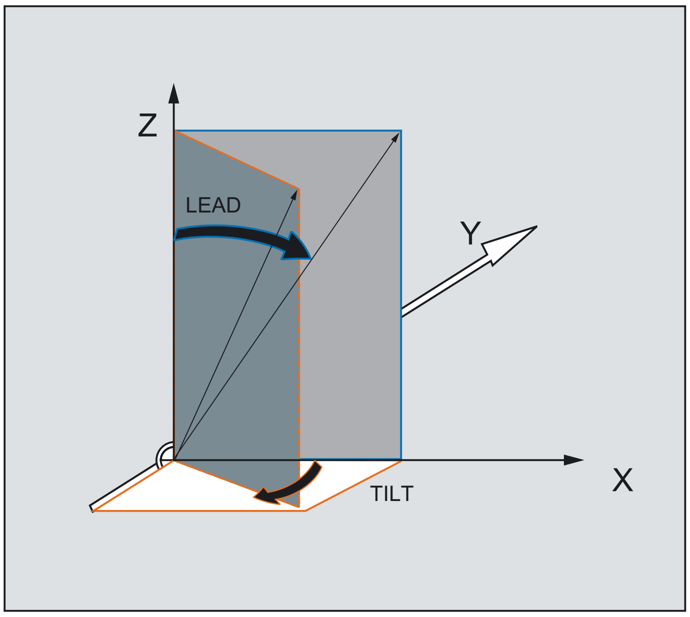
The resultant tool orientation is determined from:

- 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.

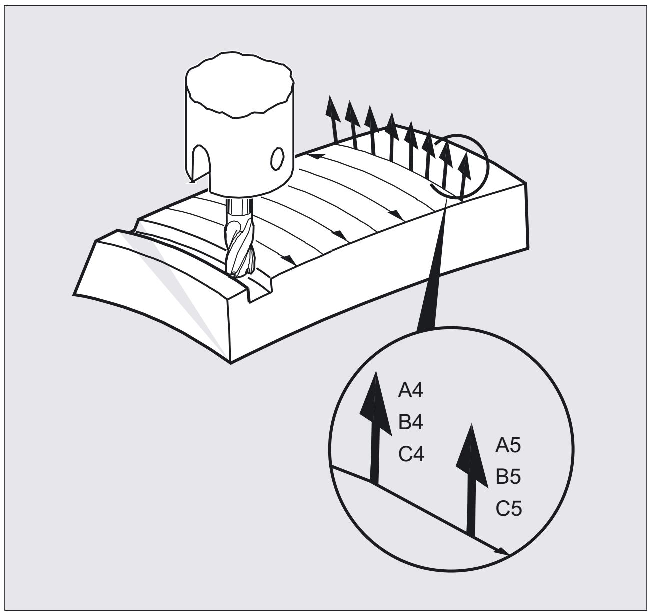
Definition of tool orientation with LEAD= and TILT=



7.2.5 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.

7.2.6 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

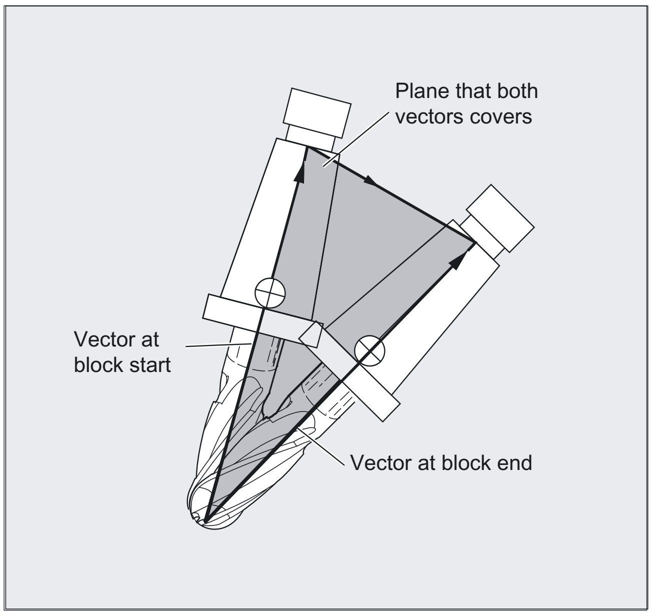
`ORIWKS` is the basic setting. In the case of a 5-axis program, if it is not immediately obvious on which machine it is to run, always choose `ORIWKS`. Which movements the machine actually executes depend on the machine kinematics.

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.

7.2.7 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`
`N... G1 X Y Z A B C`
	Linear or large-radius circular interpolation

or	or
`N... ORIPLANE`	orientation interpolation of the plane
or	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	or

`N... ORIVIRT1` or `ORIVIRT2`
`N... G1 X Y Z A3= B3= C3=`
	virtual orientation axes definition 1 or 2
	direction vector programming

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_ORIAX_TURN_TAB_1
	(definition 2), definition according to MD $MC_ORIAX_TURN_TAB_2
A3= B3= C3=	Direction vector programming of direction axis

Description

Machine manufacturer

MD `$MC_ORI_DEF_WITH_G_CODE` specifies how the programmed angles `A2, B2, C2` are defined:

The definition is made using MD `$MC_ORIENTATION_IS_EULER` (standard) or the definition is made using G group 50 (`ORIEULER`, `ORIRPY`, `ORIVIRT1`, `ORIVIRT2`).

MD `$MC_ORI_IPO_WITH_G_CODE` specifies which interpolation mode is active: `ORIWKS/ORIMKS` or `ORIAXES/ORIVECT`.

JOG mode

Interpolation for orientation angles in this mode of operation is always linear. During continuous and incremental traversal via the traversing keys, only one orientation axis can be traversed. Orientation axes can be traversed simultaneously using the handwheels.

For manual travel of the orientation axes, the channel-specific feed override switch or the rapid traverse override switch work at rapid traverse override.

A separate velocity setting is possible with the following machine data:

```
$MC_JOG_VELO_RAPID_GEO
$MC_JOG_VELO_GEO
$MC_JOG_VELO_RAPID_ORI
$MC_JOG_VELO_ORI
```

The Cartesian manual travel function can be used in JOG operation for
SINUMERIK 840D power line and 840D solution line with "Handling transformation package" and
Sinumerik 810D power line to set the translation of geometric axes in the MCS, WCS and TCS reference systems independently of each other.

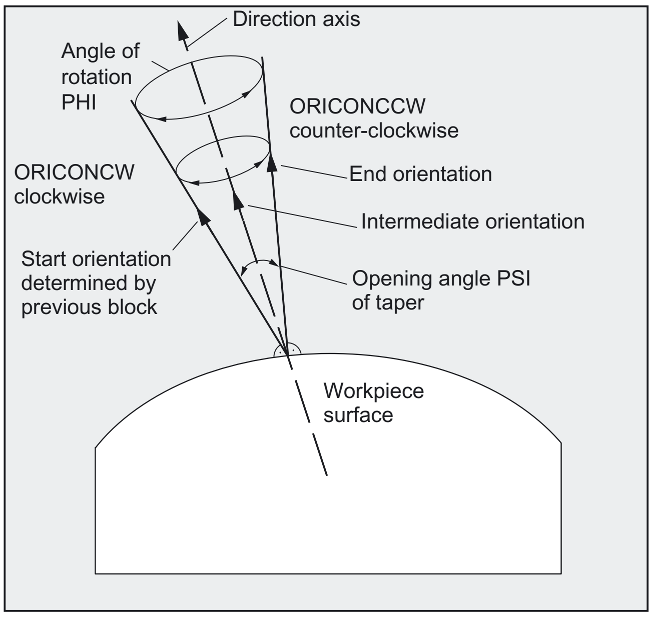
References

/FB2/ Description of Functions Extended Functions; Kinematic Transformation (M1)

7.2.8 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`	**Interpolation** on the outside of a taper with
`N... A6= B6= C6= A3= B3= C3=`	direction vector in the clockwise/counterclockwise direction of the taper and end orientation or
`or`	
`N... ORICONTO`	tangential transition and
`N... G1 X Y Z A6= B6= C6=`	specification of end orientation
or	or
`N... ORICONIO`	specification of end orientation and
`N... G1 X Y Z A7= B7= C7=`	intermediate orientation on the outside of the taper with
`N... PO[PHI]=(a2, a3, a4, a5)`	polynomials for angle of rotation and
`N... PO[PSI]=(b2, b3, b4, b5)`	polynomials for opening angle

Parameters

`ORIPLANE`	Interpolation in the plane (large-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
…                                 ;Tool orientation in the plane
N40 ORIPLANE                      ;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.
…                                 ;Orientation programming on outside of the
                                  ;taper
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`.

7.2.9 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. It is also possible to activate a spline interpolation with BSPLINE. Other types of splines (ASPLINE and CSPLINE) and the activation of a compressor using COMPON, COMPCURV or COMPCAD are not permissible.

The motion of the two contact points of the tool can be predefined up to the 5th degree when programming the orientation polynomials for coordinates.

Extended orientation interpolation with additional curve in space and polynomials for coordinates

```
N... ORICURVE
N... PO[XH]=(xe, x2, x3, x4, x5)
N... PO[YH]=(ye, y2, y3, y4, y5)
N... PO[ZH]=(ze, z2, z3, z4, z5)
```

Specification of the motion of the second contact point of the tool and additional polynomials of the coordinates in question

Parameters

ORICURVE	Interpolation of the orientation specifying a movement between two contact points of the tool
XH YH ZH	Identifiers of the coordinates of the second contact point of the tool of the additional contour as a curve in space
Possible polynomials PO[XH]=(xe, x2, x3, x4, x5) PO[YH]=(ye, y2, y3, y4, y5) PO[ZH]=(ze, z2, z3, z4, z5)	Apart from using the appropriate end points, the curves in space can also be programmed using polynomials.
xe, ye, ze	End points of the curve in space
xi, yi, zi	Coefficients of the polynomials up to the 5th degree

Note

Identifiers XH YH ZH for programming a second orientation path

The identifiers must be selected such that no conflict arises with the other identifiers or linear axes

X Y Z axes

and rotary axes such as

A2 B2 C2 Euler angle or RPY angle

A3 B3 C3 direction vectors

A4 B4 C4 or A5 B5 C5 surface normal vectors

A6 B6 C6 rotation vectors or A7 B7 C7 intermediate point coordinates

or other interpolation parameters.

7.3 Orientation polynomials (PO[angle], PO[coordinate])

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.

Programming

Type 1 orientation polynomials for **angles**

`N… PO[PHI]=(a2, a3, a4, a5)`	3-axis to 5-axis transformation
or	
`N… PO[PSI]=(b2, b3, b4, b5)`	3-axis to 5-axis transformation

Type 2 orientation polynomials for **coordinates**

`N… PO[XH]=(xe, x2, x3, x4, x5)`
`N… PO[YH]=(ye, y2, y3, y4, y5)`
`N… PO[ZH]=(ze, z2, z3, z4, z5)`
	Identifiers for the coordinates of the second orientation path for tool orientation

In both cases, with 6-axis transformations, a polynomial can also be programmed for the **rotation** using

`N… PO[THT]=(c2, c3, c4, c5)`	Interpolation of the rotation relative to the path
or	
`N… PO[THT]=(d2, d3, d4, d5)`	Interpolation absolute, relative and tangential to the change of orientation

of the orientation vector. This is possible if the transformation supports a rotation vector with an offset that can be programmed and interpolated using the THETA angle of rotation.

Parameters

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.

7.4 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.

Programming

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`
or
`N... ORIROTR`
or
`N... ORIROTT`
or
`N... ORIROTC`
	Define the interpolation of the rotation vector

`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

Parameters

ORIROTA	Angle of rotation to an absolute direction of rotation.
ORIROTR	Angle of rotation relative to the plane between the start and end orientation.
ORIROTT	Angle of rotation as a tangential rotation vector to the change of orientation
ORIROTC	Angle of rotation as a tangential rotation vector to the path tangent
THETA	Rotation of the orientation vector
THETA=value	Angle of rotation in degrees reached by the end of the block.
THETA=Θe	Angle of rotation with end angle Θ_e of rotation vector
THETA=AC(...)	Non-modal switchover to absolute dimensions
THETA=AC(...)	Non-modal switchover to incremental dimensions
Θe	End angle of rotational vector both absolute with G90 and relative with G91 (incremental dimensioning) is active
PO[THT]=(....)	Polynomial for angle of rotation

Example of rotations of orientations

```
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
                                     ;Orientation
N50 A3=0 B3=1 C3=0 PO[THT]=(180,90)  ;In Y direction and rotation about
                                     ;180 degrees
N60 A3=0 B3=1 C3=0 THETA=IC(-90)     ;Remains constant and rotation to 90 degrees
N70 ORIROTT                          ;Angle of rotation relative to change of
                                     ;orientation
N80 A3=1 B3=0 C3=0 THETA=30          ;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.

7.5 Orientations relative to the path

7.5.1 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.

Programming

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), "Orientation" section

---

Parameters

Various settings can be made for the interpolation of angles `LEAD` and `TILT` via machine data:

- The tool-orientation reference programmed using `LEAD` and `TILT` is retained for the entire block.
- Lead angle `LEAD`: rotation about the direction vertical to the tangent and normal vector `TILT`: rotation of the orientation about the normal vector.
- Lead angle `LEAD`: rotation about the direction vertical to the tangent and normal vector Tilt angle `TILT`: rotation of the orientation in the direction of the path tangent.
- Angle of rotation `THETA`: rotation of the tool about itself with an additional third rotary axis acting as an orientation axis in 6-axis transformation.

Note

Orientation relative to the path not permitted in conjunction with OSC, OSS, OSSE, OSD and OST

Orientation interpolation relative to the path, that is `ORIPATH` or `ORIPATHS` and `ORIOTC`, cannot be programmed in conjunction with orientation characteristic smoothing with a G code from group 34. OSOF has to be active for this.

7.5.2 Rotation of the tool orientation relative to the path (ORIPATH, ORIPATHS, angle of rotation)

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.

Programming

Rotation of tool orientation and tool

The type of tool orientation relative to the path is activated using ORIPATH or ORIPATHS.

`N... ORIPATH`	Activate type of orientation relative to the path
`N... ORIPATHS`	Activate type of orientation relative to the path with smoothing of the orientation characteristic

Activating the three angles that can be rotated:

`N... LEAD=`	Angle for the programmed orientation relative to the surface normal vector
`N... TILT=`	Angle for the programmed orientation in the plane, vertical to the path tangent relative to the surface normal vector
`N... THETA=`	Angle of rotation relative to the change of orientation in the tool direction of the third rotary axis

The values of the angles at the end of block are programmed using `LEAD=value`, `TILT=value` or `THETA=value`. In addition to the constant angles, polynomials can be programmed for all three angles up to the 5th degree.

`N... PO[PHI]=(a2, a3, a4, a5)` or	Polynomial for the LEAD angle
`N... PO[PSI]=(b2, b3, b4, b5)` or	Polynomial for the TILT angle
`N... PO[THT]=(d2, d3, d4, d5)`	Polynomial for the THETA angle of rotation

The higher polynomial coefficients, which are zero, can be omitted when programming. Example: `PO[PHI]=a2` results in a parabola for the `LEAD` angle.

Parameters

Tool orientation relative to the path

ORIPATH	Tool orientation relative to the 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.

7.5.3 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].

Programming

`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.

Parameters

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=θ_e	Angle of rotation with end angle Θ_e of rotation vector
THETA=AC(...)	Non-modal switchover to absolute dimensions
THETA=IC(…)	Non-modal switchover to incremental dimensions
PO[THT]=(c_2, c_3, c_4, c_5)	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
	Description, see the Rotations of the tool orientation section
`A= B= C=`	Programming the machine axis position
`ORIEULER`	Orientation programming via Euler angle
`ORIRPY`	Orientation programming via RPY angle
`A2= B2= C2=`	Angle programming of virtual axes
`ORIVIRT1`	Orientation programming using virtual orientation axes
`ORIVIRT2`	(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`	Interpolation on the peripheral surface of a taper
`A7= B7= C7=`	Intermediate orientation (programming as normalized vector)

`ORICURVE`
`XH YH ZH, e.g., with polynomials PO[XH]=(xe, x2, x3, x4, x5)`
	Interpolation of the orientation specifying a movement between two contact points of the tool. In addition to the end points, additional curve polynomials can also be programmed.

Note

If the tool orientation with active ORIAXES is interpolated via the orientation axes, the angle of rotation is only initiated relative to the path at the end of block.

7.5.4 Smoothing of orientation characteristic (ORIPATHS A8=, B8=, C8=)

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)

Programming

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
```

Parameters

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.

7.6 Compression of the orientation COMPON (A..., B..., C..., THETA)

Function

NC programs in which the orientation is programmed by means of direction vectors can be compressed if kept within specified limits. The compressor can only be used for orientations in conjunction with an orientation transformation.

Machine manufacturer

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

NC block structure in general

The blocks to be compressed may only contain a block number, linear interpolation G1, axis addresses, feedrate, and a comment and their program syntax is, therefore, as follows:

```
N... G1 X=... Y=... Z=... A=... B=...      ;Comment
F=...
```

The position values can be entered directly, e.g., X90, or indirectly via parameter settings X=R1*(R2+R3).

With active orientation transformation TRAORI

The tool orientation can be programmed independently of the kinematics.

On a machine with **3-axis to 5-axis transformation**, the following applies:

```
N... TRAORI
A3=... B3=... C3=...                       ;Direction vector
A2=... B2=... C2=...                       ;Euler angle or RPY angle
```

On a machine **with 6-axis transformation** , the rotation of the tool can be programmed in addition to the tool orientation.

```
N... X... Y... Z... A3=... B3=... C3=... THETA=... F=... or
N... X... Y... Z... A2=... B2=... C2=... THETA=... F=...
```

If the tool orientation is specified via rotary axis positions, e.g., as:

```
N... X... Y... Z... A=... B=... THETA=... F=...
```

the compression is interpreted differently, depending on whether large-radius circular interpolation is performed or not. If large-radius circular interpolation is not performed, the compressed orientation change is represented by axial polynomials for the rotary axes.

Parameters

The parameter assignments that previously applied to the compressor can also be used for rotary axis positions.

TRAORI	Activate orientation transformation
COMPON	Compressor ON
G1	Linear interpolation
X= Y= Z=	Linear axis addresses
A= B= C=	Rotary axis positions; direct programming
A2= B2= C2=	Rotary axis addresses in Euler angles or RPY angles
A3= B3= C3=	Rotary axis addresses as direction vectors
THETA	Rotation of the orientation vector
F	Path feedrate

For more detailed information about programming the THETA=... angle of rotation, please see "Rotations of the tool orientation (ORIROTA/TR/TT, ORIROTC, THETA)".

Note

Compression only with active large-radius circular interpolation

This is the case when the tool orientation changes in the plane that is defined by the start and end orientation. The conditions that apply to large-radius circular interpolation must be set via machine data.

1st machine data: G code for orientation interpolation = FALSE
ORIWKS is active and orientation is programmed as a vector with A3, B3, C3 or A2, B2, C2.

2nd machine data: G code for orientation interpolation = TRUE
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 ORICONxx or ORICURVE G codes is active or if polynomials are programmed for the orientation angle (PO[PHI] and PO[PSI]), large-radius circular interpolation is not performed, i.e., blocks of this type are not compressed.

Example: "Compressor for orientations"

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 generates a smooth motion of the orientation.

```
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 deviations of the contour:
                                         ;0.05 mm
$SC_COMPRESS_ORI_TOL = 5                 ;Maximum deviations of the orientation:
                                         ;5 degrees
TRAORI                                   ;The movement describes a circle
COMPCURV                                 ;generated from polygons.
N100 X0 Y0 A3=0 B3==1                    ;While the orientation moves on a taper
N110 FOR COUNTER = 0 TO NUMBER           ;around the Z axis with an opening angle
N120 ANGLE= 360 * COUNTER/NUMBER         ;of 45 degrees.
N130 X=RADIUS*COS(ANGLE)Y=RADIUS*
 SIN(ANGLE) A3=SIN(ANGLE)
 B3=(ANGLE) C3=1
N140 ENDFOR
...
```

Description

Accuracy

You can only compress NC blocks if you allow the contour to deviate from the programmed contour. You can set the maximum deviation as a compressor tolerance in the setting data. The higher the tolerances, the more blocks can be compressed.

Axis accuracy

For each axis, the compressor creates a spline curve, which deviates from the programmed end points of each axis by no more than the tolerance set with the axial machine data.

Contour accuracy

The maximum geometrical contour deviations (geometry axes) and the tool orientation are monitored. This is achieved using the setting data for:

1. Maximum tolerance for the contour
2. Maximum angular displacement for the tool orientation
3. Maximum angular displacement for the angle of rotation THEATA of the tool
   (only available on 6-axis machines)

You can use the channel-specific MD 20482 COMPRESSOR_MODE to set tolerance specifications:

0: Axis accuracy: Axial tolerances for all axes (geometry axes and orientation axes)

1: Contour accuracy: Specification of the contour tolerance (1.), tolerance for the orientation using axial tolerances (a.)

2: Specification of the maximum angular displacement for tool orientation (2.), tolerance for the contour using axial tolerances (a.)

3: Specification of the contour tolerance with (1.) and specification of the maximum angular displacement for tool orientation with (2.)

It is only possible to specify a maximum angular displacement for tool orientation if an orientation transformation (TRAORI) is active.

7.7 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 offsets can be active at the same time.

The online tool length offset function can be used for:

- orientation transformation TRAORI
- orientable toolholder TCARR

Machine manufacturer

Online tool length offset is an option, which must be enabled in advance. This function is only practical in conjunction with an active orientation transformation or an active orientable toolholder.

Programming

```
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 offset in motion-synchronous actions, see "Actions in synchronized actions".

Parameters

TOFFON	**T**ool **Off**set **ON** (activate online tool length offset) When activating, an offset value can be specified for the relevant direction of compensation and this is immediately recovered.
TOFFOF	**T**ool **Off**set **ON** (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

Example of tool length offset selection

```
MD 21190: TOFF_MODE =1                   ;Absolute values are approached
MD 21194: TOFF_VELO[0] =1000
MD 21196: TOFF_VELO[1] =1000
MD 21194: TOFF_VELO[2] =1000
MD 21196: TOFF_ACCEL[0] =1
MD 21196: TOFF_ACCEL[1] =1
MD 21196: TOFF_ACCEL[2] =1
N5 DEF REAL XOFFSET
N10 TRAORI(1)                            ;Transformation ON
N20 TOFFON(Z)                            ;Activation of online tool length offset
                                         ;for the Z tool direction
N30 WHEN TRUE DO $AA_TOFF[Z] = 10        ;For the Z tool direction, a tool
G4 F5                                    ;length offset of 10 is interpolated
...
N100 XOFFSET = $AA_TOFF_VAL[X]           ;Assign current offset in X direction
N120 TOFFON(X, -XOFFSET)                 ;for the X tool direction, the tool
G4 F5                                    ;length offset will be returned to 0 again
```

Example of tool length offset deselection

```
N10 TRAORI(1)                            ;Transformation ON
N20 TOFFON(X)                            ;Activating the Z tool direction
N30 WHEN TRUE DO $AA_TOFF[X] = 10        ;For the X tool direction, a tool
G4 F5                                    ;length offset of 10 is interpolated
...
N80 TOFFOF(X)                            ;Positional offset of the X tool direction
                                         ;is deleted: …$AA_TOFF[X] = 0
                                         ;No axis is traversed;
                                         ;to the current position in WCS, the
                                         ;positional offset is added 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_VELO` 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).

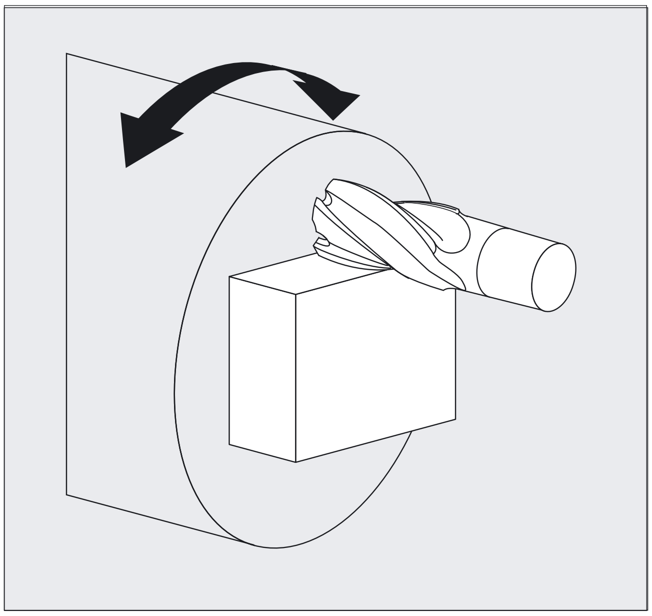
7.8 Kinematic transformation

7.8.1 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.

Programming

`TRANSMIT` or `TRANSMIT(n)`

or

`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.

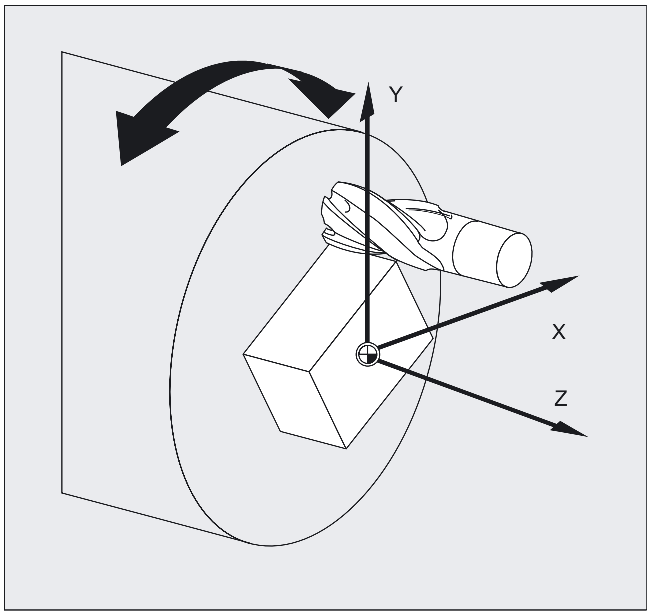
Parameters

`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



```
N10 T1 D1 G54 G17 G90 F5000 G94              ;Tool selection
N20 G0 X20 Z10 SPOS=45                       ;Approach start position
N30 TRANSMIT                                 ;Activate TRANSMIT function
N40 ROT RPL=-45                              ;Set frame
N50 ATRANS X-2 Y10
N60 G1 X10 Y-10 G41 OFFN=1                   ;Square roughing; allowance 1 mm
N70 X-10
N80 Y10
N90 X10
N100 Y-10
N110 G0 Z20 G40 OFFN=0                       ;Change tool
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 start 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)

7.8.2 Cylinder surface transformation (TRACYL)

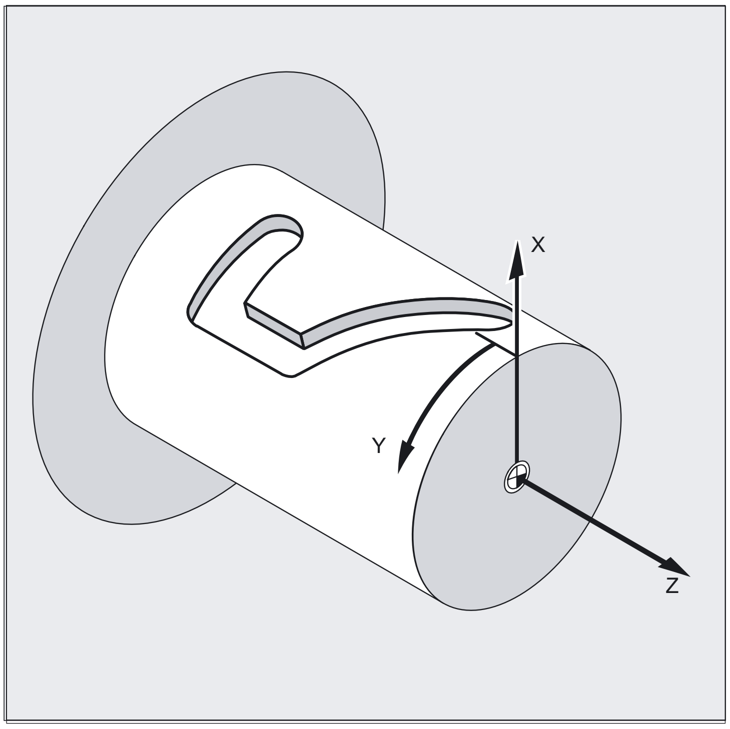
Function

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).

Programming

`TRACYL(d)` or `TRACYL(d, n)` or

for transformation type 514

`TRACYL(d, n, groove side offset)`

or

`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.

Parameters

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 of the definition of a tool

The following example is suitable for testing the parameterization of the `TRACYL` cylinder transformation:

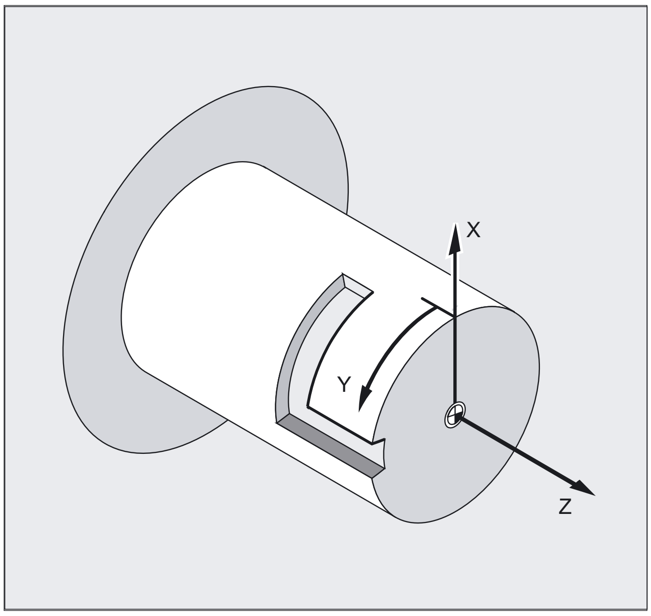
Tool parameters Number (DP)	Meaning	Remarks
$TC_DP1[1,1]=120	Tool type	Milling tool
$TC_DP2[1,1]=0	Tool point direction	only for turning tools

Geometry	Length offset	
$TC_DP3[1,1]=8.	Length offset vector	Calculation acc. to type
$TC_DP4[1,1]=9.		and plane
$TC_DP5[1,1]=7.		

Geometry	Radius	
$TC_DP6[1,1]=6.	Radius	Tool radius
$TC_DP7[1,1]=0	Slot width b for slotting saw, rounding radius for milling tools	
$TC_DP8[1,1]=0	Projection k	For slotting saw only
$TC_DP9[1,1]=0		
$TC_DP10[1,1]=0		
$TC_DP11[1,1]=0	Angle for taper milling tools	

Wear	Tool length and radius compensation	
$TC_DP12[1,1]=0	Remaining parameters to $TC_DP24=0	Base dimensions/ adapter

Example of making a hook-shaped groove:



Activate cylinder surface transformation

```
N10 T1 D1 G54 G90 F5000 G94              ;Tool selection, clamping compensation
N20 SPOS=0                               ;Approach start position
N30 G0 X25 Y0 Z105 CC=200
N40 TRACYL (40)                          ;Enable cylinder peripheral curve
                                         ;transformation
N50 G19                                  ;Plane selection
```

Making a hook-shaped groove

```
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
N150 G1 X25                              ;Move clear
N160 TRAFOOF
N170 G0 X25 Y0 Z105 CC=200               ;Approach start position
N180 M30
```

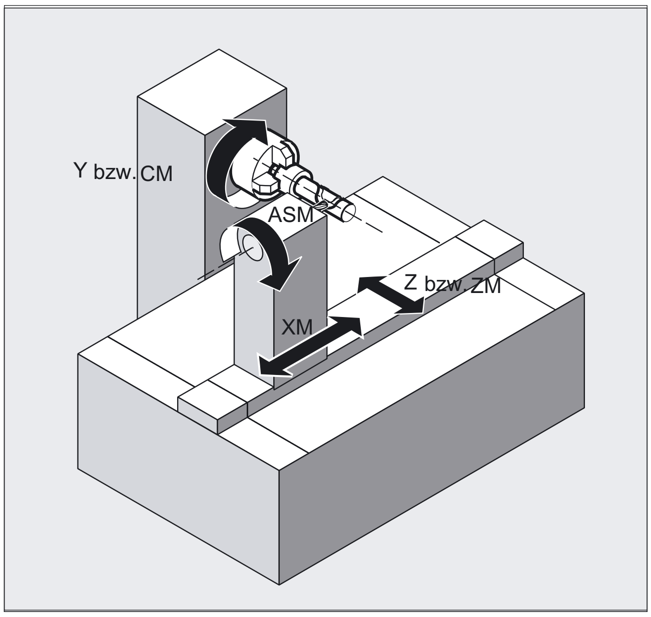
Description

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.

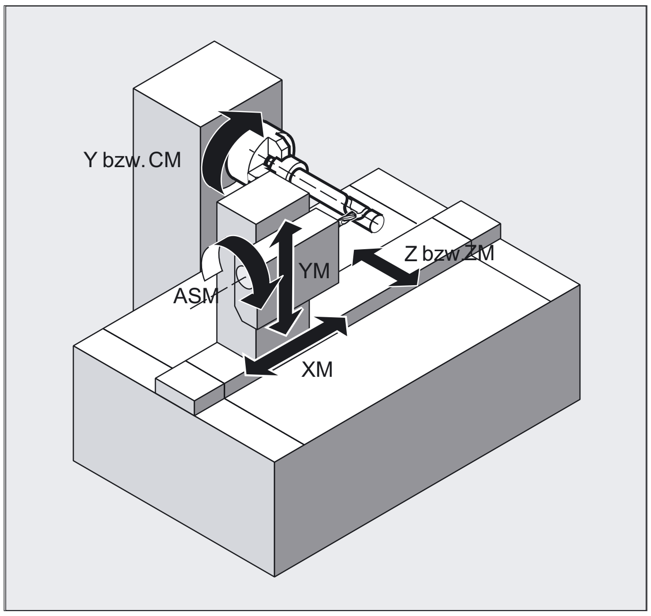


With groove wall offset (transformation type 513):

Kinematics as above, but an additional longitudinal axis parallel to the peripheral direction

The linear axes are positioned perpendicular to one another.

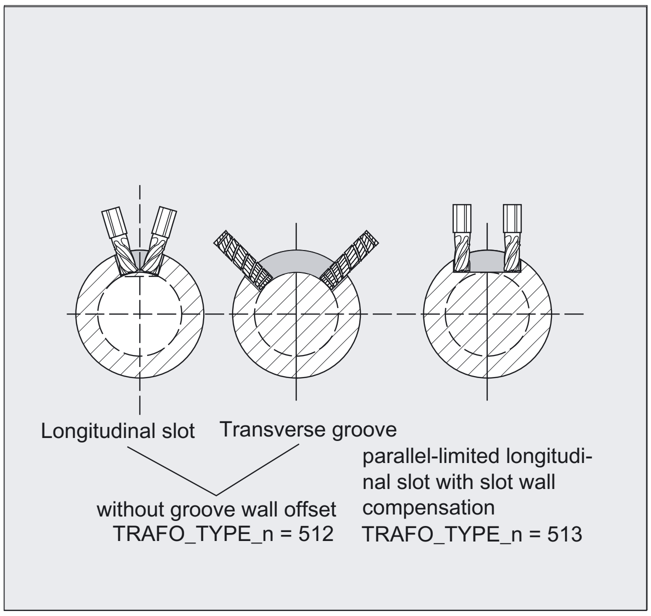
The velocity control makes allowance for the limits defined for the rotations.



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.


Longitudinal slot
Transverse groove
without groove wall offset
TRAFO_TYPE_n = 512
parallel-limited longitudinal slot with slot wall compensation
TRAFO_TYPE_n = 513

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

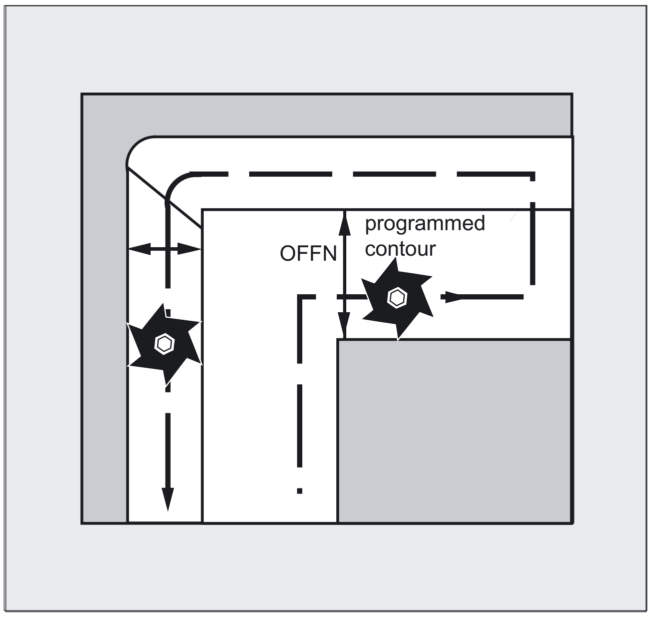
/FB2/ Function Manual Extended Functions; Kinematic Transformations (M1)

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 >= 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.

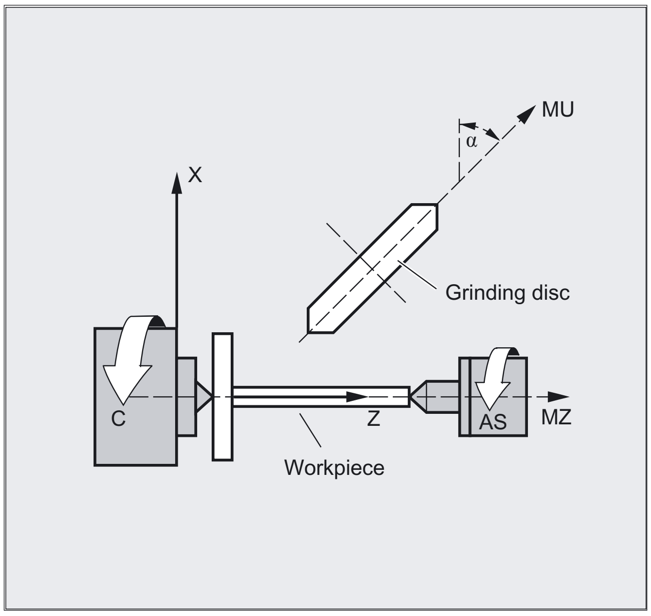
With TRAFO_TYPE_n = 513, half the groove width is programmed in `OFFN`. The contour is retracted with OFFN-TRC.

7.8.3 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.



Programming

`TRAANG(α)` or `TRAANG(α, n)`

or

`TRAFOOF`

Parameters

`TRAANG( ) or`
`TRAANG( ,n)`
	Activate transformation with the parameterization of the previous selection.

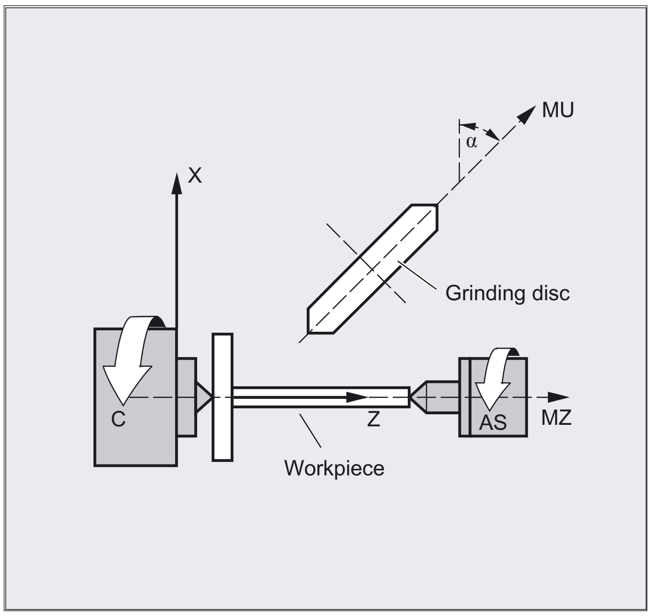
`TRAANG(α)`	Activates the first specified inclined axis transformation
`TRAANG(α,n)`	Activates the nth agreed inclined axis transformation. The maximum value of n is 2. TRAANG(α,1) corresponds to TRAANG(α).
α	Angle of the inclined axis Permissible values for α are: -90 degrees < α < + 90 degrees
`TRAFOOF`	Transformation off
n	Number of agreed transformations

Angle α omitted or zero

If α (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 α = 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



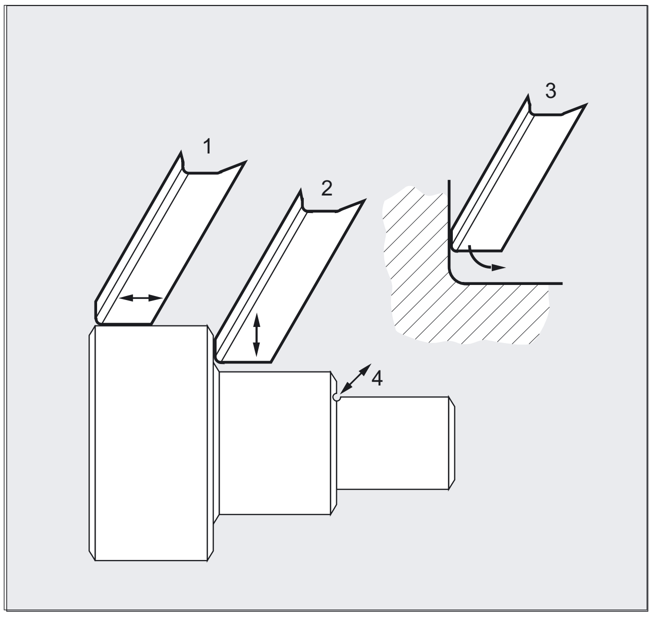
```
N10 G0 G90 Z0 MU=10 G54 F5000 ->              ;Tool selection,
                                              ;clamping compensation,
-> G18 G64 T1 D1                              ;Plane selection
N20 TRAANG(45)                                ;Enable inclined axis transformation
N30 G0 Z10 X5                                 ;Approach start position
N40 WAITP(Z)                                  ;Enable axis for reciprocation
N50 OSP[Z]=10 OSP2[Z]=5 OST1[Z]=-2 ->         ;Reciprocation, until dimension
-> OST2[Z]=-2 FA[Z]=5000                      ;reached
N60 OS[Z]=1                                   ;(for reciprocation, see
                                              "Reciprocation" chapter)
N70 POS[X]=4.5 FA[X]=50
N80 OS[Z]=0
N90 WAITP(Z)                                  ;Enable reciprocating axes as
                                              ;positioning axes
N100 TRAFOOF                                  ;Deactivate transformation
N110 G0 Z10 MU=10                             ;Move clear
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.

7.8.4 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)

Programming

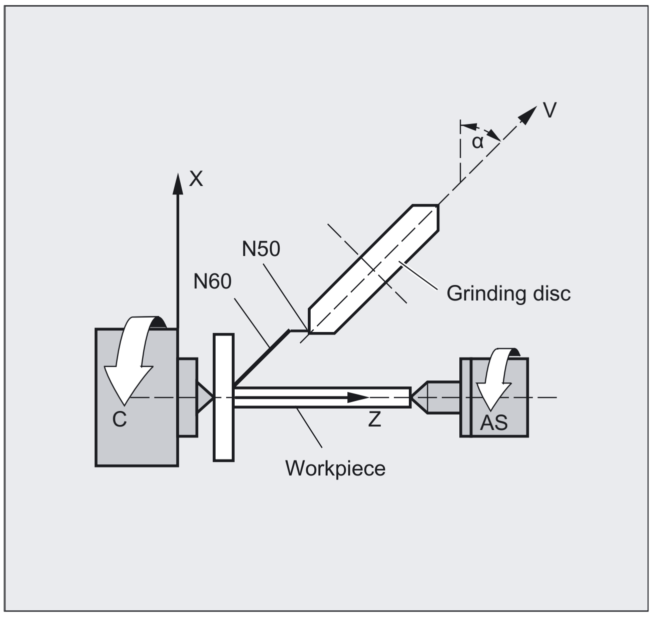
```
G07
G05
```

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.

Parameters

G07	Approach starting position
G05	Activates oblique plunge-cutting

Example



```
N..                                     ;Program angle for inclined axis
N20 G07 X70 Z40 F4000                   ;Approach starting position
N30 G05 X70 F100                        ;Oblique plunge-cutting
N40 ...
```

7.9 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.

Programming

```
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 round 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 round 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 round axis positions available. The round axis positions to be approached can be selected by programming `STAT = <...>`. If

`STAT = 0` is programmed (this is equivalent to the standard 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.

Parameters

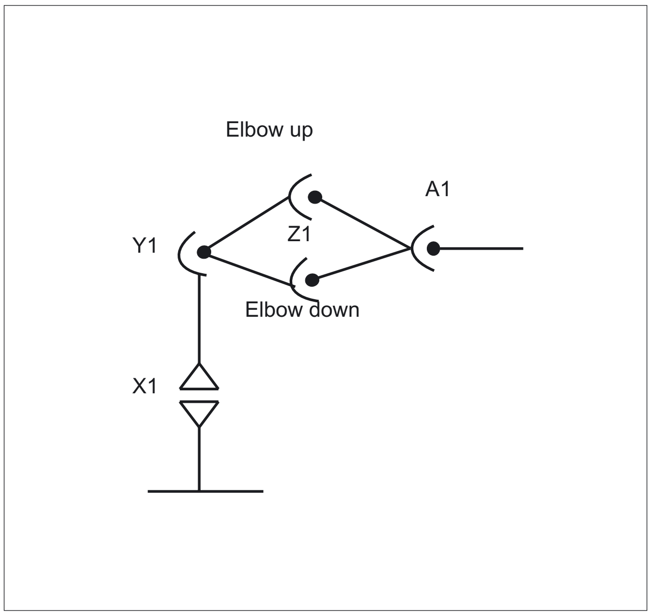
The `PTP` and `CP` commands act in a modal manner. `CP` is the default setting.

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 round axis interpolation.

PTP	**p**oint **t**o **p**oint (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	**c**ontinuous **p**ath (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 blockwise. 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    ;Reorientation without transformation
                                          → Elbow down
N50 X1200 Z400 CP                         ;Transformation active again
N60 X1000 Z500 A20
N70 M30
```

Example PTP transversal at generic 5-axis transformation

Assumption: This is based on a right-angled CA kinematics.

```
TRAORI                                    ;Transformation CA kinematics ON
PTP                                       ;Activate PTP traversal
N10 A3 = 0 B3 = 0 C3 = 1                  ;Round axis positions C = 0 A = 0
N20 A3 = 1 B3 = 0 C3 = 1                  ;Round axis positions C = 90 A = 45
N30 A3 = 1 B3 = 0 C3 = 0                  ;Round axis positions C = 90 A = 90
N40 A3 = 1 B3 = 0 C3 = 1 STAT = 1         ;Round axis positions C = 270 A = -45
```

Select clear approach position of round axis position:

In block N40, by programming `STAT = 1`, the round 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 < ± 360 degrees, this information must be programmed using the command `"TU="`.

The axes traverse by the shortest path:

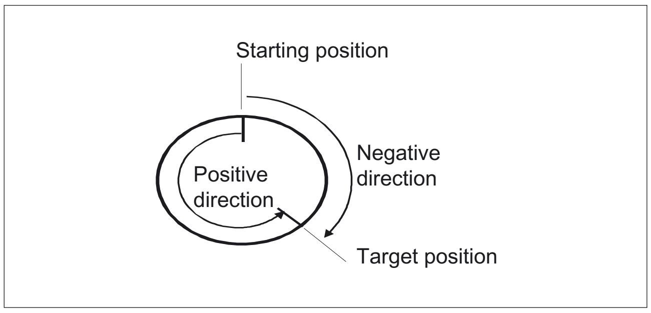
- when no `TU` is programmed for a position,
- with axes that have a traversing range > ±360 degrees.

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.

7.9.1 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 parts 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.

Programming

```
N... TRANSMIT
N... PTPG0
N... G0 ...
...
N... G1 ...
```

Parameters

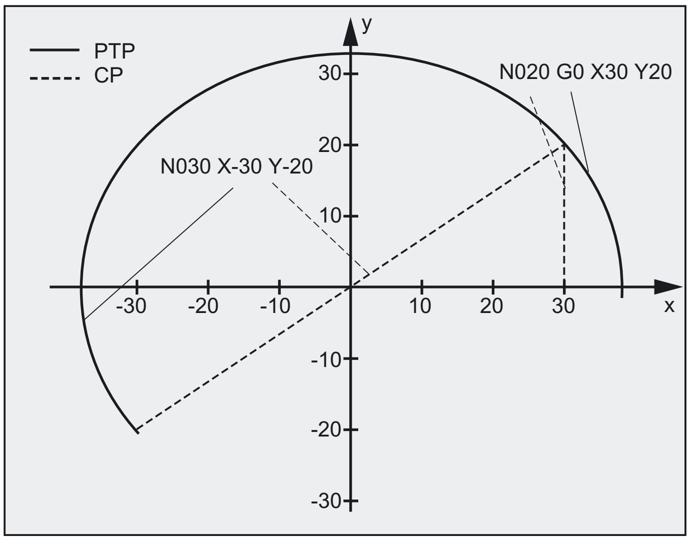
`TRANSMIT`	`Activates the first declared TRANSMIT function`
	`(see section "Milling on turned parts: TRANSMIT")`

`PTPG0`	`**P**oint **t**o **P**oint **G0** (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`	`**p**oint **t**o **p**oint (point to point motion)`
	`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.`

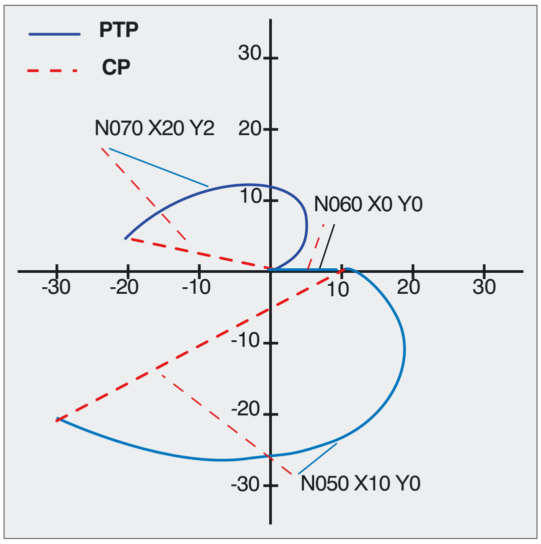
`STAT=`	`Resolving the non-uniqueness with regard to the pole.`
`TU=`	`TU is not relevant for PTP with TRANSMIT`

Example of circumnavigation of the pole with PTP and TRANSMIT



```
N001 G0 X30 Z0 F10000 T1 D1 G90              ;Initial setting absolute
                                             ;dimension
N002 SPOS=0
N003 TRANSMIT                                ;TRANSMIT transformation
N010 PTPG0                                   ;Automatic for each G0 block
                                             ;PTP and then CP again
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



```
N001 G0 X90 Z0 F10000 T1 D1 G90                       ;Initial setting
N002 SPOS=0
N003 TRANSMIT                                         ;TRANSMIT transformation
N010 PTPG0                                            ;Automatic for each G0 block
                                                      ;PTP and then CP again
N020 G0 X90 Y60
N030 X-90 Y-60
N040 X-30 Y-20
N050 X10 Y0
N060 X0 Y0
N070 X-20 Y2
N170 G1 X0 Y0
N160 X10 Y0
N150 X-30 Y-20
M30
```

Description

PTP and PTPG0

`PTPG0` is considered for all transformations that can process `PTP`. `PTPG0` is not relevant is 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

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.

7.10 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).

7.11 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").

Programming

```
TRAFOOF
```

Parameters

`TRAFOOF`	Disables all the active transformations/frames

7.12 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.

Programming

`TRACON(trf, par)`	This activates a chained transformation.
`TRAFOOF`	

Parameters

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).

7.13 Replaceable geometry axes (GEOAX)

Function

The "Replaceable geometry axes" function allows the geometry axis grouping configured via machine data to be modified from the parts program. Here any geometry axis can be replaced by a channel axis defined as a synchronous special axis.

Programming

```
GEOAX(n,channel axis,n,channel axis,…)
```

or

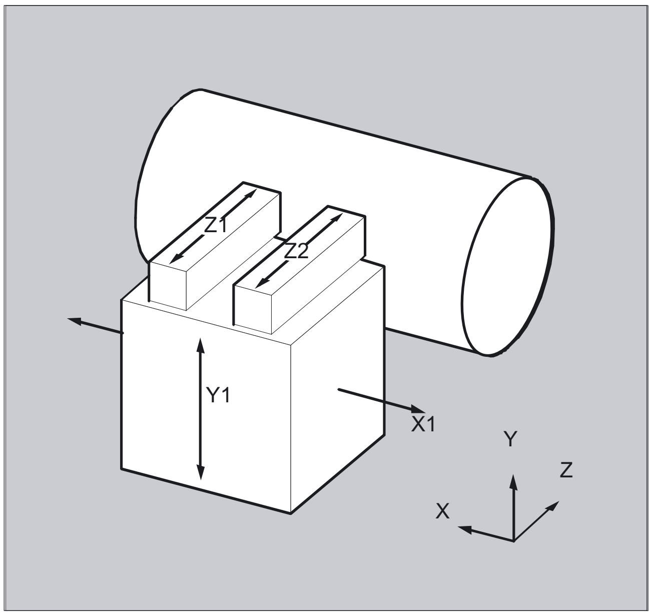
```
GEOAX()
```

Parameters

`GEOAX(n,channel axis,n,channel axis,…)`	Switch the geometry axes.
`GEOAX()`	Call the basic configuration of the geometry axes.
`n`	Number of the geometry axis (n=1, 2 or 3) to be assigned to another channel axis. n=0: remove the specified channel axis from the geometry axis grouping without replacement.
`channel axis`	Name of the channel axis to be accepted into the geometry axis grouping.

Example: two geometry axes changing over alternately

A tool carriage can be traversed over channel axes X1, Y1, Z1, Z2. In the parts program, axes Z1 and Z2 should be used alternately as geometry axis Z. GEOAX is used in the parts program to switch between the axes.



After activation, the connection X1, Y1, Z1 is effective (adjustable via MD).

```
N100 GEOAX (3,Z2)                         ;Channel axis Z2 functions as the Z axis
N110 G1 .....
N120 GEOAX (3,Z1)                         ;Channel axis Z1 functions as the Z axis
```

Example: geometry axis configurations for 6 channel axes

A machine has six channel axes called XX, YY, ZZ, U ,V ,W. The basic setting of the geometry axis configuration via the 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)

N10 GEOAX()	;The basic configuration of the geometry axes is effective.
N20 G0 X0 Y0 Z0 U0 V0 W0	;All the 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), 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 stays as the second geometry axis (Y).
N50 G17 G2 X20 I10 F1000	;Full circle in the X, Y plane. Channel axes XX and V traverse
N60 GEOAX(2,W)	;Channel axis W becomes the second geometry axis (Y).
N80 G17 G2 X20 I10 F1000	;Full circle in the X, Y plane. Channel axes XX and W traverse.
N90 GEOAX()	;Reset to initial state
N100 GEOAX(1,U,2,V,3,W)	;Channel axis U becomes the first (X), V the second (Y), 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 the special ;axis traverses to position 25.
N120 GEOAX(0,V)	;V is removed from the geometry axis grouping. ;U and W are still the first (X) and third geometry axis (Z). ;The second geometry axis (Y) remains unassigned.
N130 GEOAX(1,U,2,V,3,W)	;Channel axis U stays the first (X), V becomes the second (Y), ;W stays the third geometry axis (Z).
N140 GEOAX(3,V)	;V becomes the third geometry axis (Z), which overwrites W and thus ;removes it from the geometry axis grouping. The second geometry ;axis (Y) is still unassigned.

Prerequisites and restrictions

1. It is not possible to switch the geometry axes over during:
   - an active transformation,
   - an active spline interpolation,
   - active tool radius compensation (see PG Fundamentals, section "Tool compensation")
   - active fine tool compensation (see PG Fundamentals, section "Tool compensation")
2. If the geometry axis and the channel axis have the same name, it is not possible to change the particular geometry axis.
3. None of the axes involved in the switchover can be involved in an action that might persist beyond the block limits, as is the case, for example, with positioning axes of type A or with following axes.
4. The `GEOAX` command can only be used to replace geometry axes that already existed at power ON (i.e. no newly defined ones).
5. Using `GEOAX` for axis replacement while preparing the **contour table** (`CONTPRON`, `CONTDCON`) produces an alarm.

Description

Geometry axis number

In the command `GEOAX(n,channel axis...)` the number n designates the geometry axis to which the subsequently specified channel axis is to be assigned.

Geometry axis numbers 1 to 3 (X, Y, Z axis) are permissible for loading a channel axis.

n = 0 removes an assigned channel axis from the geometry axis grouping without reassigning the geometry axis.

After the transition, an axis replaced by switching in the geometry axis grouping is programmable as a special axis via its channel name.

Switching over the geometry axes deletes all the frames, protection zones and operating range limits.

Polar coordinates

As with a change of plane (`G17-G19`), replacing geometry axes with `GEOAX` sets the modal polar coordinates to the value 0.

DRF, NPV

Any existing handwheel offset (DRF) or an external zero offset, will stay active after the switchover.

Exchange axis positions

It is also possible to change positions within the geometry axis grouping by reassigning the axis numbers to already assigned channel axes.

```
N... GEOAX (1, XX, 2, YY, 3, ZZ)          ;Channel axis XX is the first, YY the
N... GEOAX (1, U, 2, V, 3, W)             ;second and ZZ the third geometry axis,
                                          ;Channel axis U is the first, V the
                                          ;second and W the third geometry axis.
```

Deactivating switchover

The command `GEOAX()` calls the basic configuration of the geometry axis grouping.

After POWER ON and when switching over to reference point approach mode, the basic configuration is reset automatically.

Transition and tool length compensation

An active tool length compensation is also effective after the transition. However, for the newly adopted or repositioned geometry axes, it counts as not retracted. So accordingly, at the first motion command for these geometry axes, the resultant travel path comprises the sum of the tool length compensation and the programmed travel path.

Geometry axes that retain their position in the axis grouping during a switchover, also keep their status with regard to tool length compensation.

Geometry axis configuration and transformation change

The geometry axis configuration applicable in an active transformation (defined via the machine data) cannot be modified by using the "switchable geometry axes" function.

If it is necessary to change the geometry axis configuration in connection with transformations, this is only possible via an additional transformation.

A geometry axis configuration modified via `GEOAX` is deleted by activating a transformation.

If the machine data settings for the transformation and for switching over the geometry axes contradict one another, the settings in the transformation take precedence.

Example:

A transformation is active. According to the machine data, the transformation should be retained during a RESET, however, at the same time, a RESET should produce the basic configuration of the geometry axes. In this case, the geometry axis configuration is retained as specified by the transformation.

Tool offsets 8

8.1 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 parts program. However, this T has no reference to the programmed D number.

Machine manufacturer

User cutting edge data can be configured via machine data. Please refer to the machine manufacturer's instructions.

Parameters

Note

Individual values in the offset memory

The individual values of the offset memories P1 to P25 can be read from and written to the program via system variable. 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.

[1]Also applies to milling tools for 3D face milling
[2]Tool type for slotting saw
[3]Reserved: Is not used by SINUMERIK 840/810D

Tool parameter number (DP)	Meaning of system variables	Comment
$TC_DP1	Tool type	For overview see list
$TC_DP2	Tool point direction	Only for turning tools
Geometry	Length compensation	
$TC_DP3	Length 1	Allocation to
$TC_DP4	Length 2	Type and level
$TC_DP5	Length 3	
Geometry	Radius	
$TC_DP6[1] $TC_DP6[2]	Radius 1 / length 1 diameter d	Milling/turning/grinding tool Slotting saw
$TC_DP7[1] $TC_DP7[2]	Length 2 / corner radius, tapered milling tool Slot width b corner radius	Milling tools slotting saw
$TC_DP8[1] $TC_DP8[2]	Rounding radius 1 for milling tools projecting length k	Milling tools slotting saw
$TC_DP9[1.3]	Rounding radius 2	Reserved
$TC_DP10[1]	Angle 1 face end of tool	Tapered milling tools
$TC_DP11[1]	Angle 2 tool longitudinal axis	Tapered milling tools
Wear	Tool length and radius compensation	
$TC_DP12	Length 1	
$TC_DP13	Length 2	
$TC_DP14	Length 3	
$TC_DP15[1] $TC_DP15[2]	Radius 1 / length 1 diameter d	Milling/turning/grinding tool slotting saw
$TC_DP16[1] $TC_DP16[3]	Length 2 / corner radius, tapered milling tool, slot width b corner radius	Milling tools slotting saw
$TC_DP17[1] $TC_DP17[2]	Rounding radius 1 for milling tools projecting length k	Milling / 3D face milling slotting saw
$TC_DP18[1.3]	Rounding radius 2	Reserved
$TC_DP19[1]	Angle 1 face end of tool	Tapered milling tools
$TC_DP20[1]	Angle 2 tool longitudinal axis	Tapered milling tools
Base dimensions/ adapter	Length offsets	
$TC_DP21	Length 1	
$TC_DP22	Length 2	
$TC_DP23	Length 3	
Technology		
$TC_DP24	Clearance angle	only for turning tools
$TC_DP25		Reserved

Comments

Several entries exist for the geometric variables (e.g. length 1 or radius). These are added together to produce a value (e.g. total length 1, total radius) which is then used for the calculations.

Offset values not required must be assigned the value zero.

Tool parameters $TC-DP1 to $TC-DP23 with contour tools

Note

The tool parameters not listed in the table, such as $TC_DP7, are not evaluated, i.e. their content is meaningless.

Tool parameter number (DP)	Meaning	Cutting Dn		Comment
$TC_DP1	Tool type			400 ... 599
$TC_DP2	Length of cutting edge			
Geometry	**Length compensation**			
$TC_DP3	Length 1			
$TC_DP4	Length 2			
$TC_DP5	Length 3			
Geometry	**Radius**			
$TC_DP6	Radius			
Geometry	**Limit angle**			
$TC_DP10	minimum limit angle			
$TC_DP11	maximum limit angle			
Wear	**Tool length and radius compensation**			
$TC_DP12	Wear length 1			
$TC_DP13	Wear length 2			
$TC_DP14	Wear length 3			
$TC_DP15	Wear radius			
Wear	**Limit angle**			
$TC_DP19	Wear min. limit angle			
$TC_DP20	Wear max. limit angle			
Tool base dimension/ adapter	**Tool length offsets**			
$TC_DP21	Length 1			
$TC_DP22	Length 2			
$TC_DP23	Length 3			

Basic value and wear value

The resultant values are each a total of the basic value and wear value (e.g. $TC_DP6 + $TC_DP15 for the radius). The basic measurement ($TC_DP21 – $TC_DP23) is also added to the tool length of the first cutting edge. All the other parameters, which may also impact on effective tool length for a standard tool, also affect this tool length (adapter, orientational toolholder, setting data).

Limit angles 1 and 2

Limit angles 1 and 2 each relate to the vector of the cutting edge center point to the cutting edge reference point and are counted clockwise.

8.2 Language commands for tool management

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 the unit number monitoring data.
- Read the T number of the tool preselected for the spindle.

Programming

```
T="DRILL" or T="123 tools with name"
```
or
```
Return parameter=NEWT("WZ", DUPLO_NR)
```
or
```
DELT("MYTOOL",DUPLO_NR)
```
or
```
Return parameter=GETT("MYTOOL", DUPLO_NO)
```
or
```
SETSPIECE(x,y)
```
or
```
GETSELT (x)
```

Parameters

`T="MYTOOL"`	Select tool with name
`NEWT ("WZ",DUPLO_NR)`	Create new tool, duplo number optional
`DELT ("WZ",DUPLO_NR)`	Delete tool, duplo number optional
`GETT ("WZ",DUPLO_NR)`	Determine T number
`SETPIECE(x,y)`	Set piece number
`GETSELT (x)`	Read preselected tool number (T No.)
`"WZ"`	Tool identifier
`DUPLO_NO`	Number of workpieces
`x, y`	Spindle number, entry optional

Example of the NEWT function

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.

If no duplo number is specified, this is generated automatically by the tool manager.

```
DEF INT DUPLO_NO
DEF INT T_NO
DUPLO_NO = 7
T_NO=NEWT("DRILL", DUPLO_NO)          ;Create new "DRILL" tool with duplo number 7.
                                      ;The created T number will be stored in T_NO.
```

Example of the DELT function

The DELT function can be used to delete a tool without referring to the T number.

Example of the GETT function

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 parameter = –1: the tool name or duplo number cannot be assigned to a tool.

```
T="DRILL"
R10=GETT("DRILL", DUPLO_NO)                     ;Return T number for DRILL with
                                                ;duplo number = DUPLO_NO
```

The "DRILL" must first be declared with `NEWT` or `$TC_TP1[ ]`.

```
$TC_DP1[GETT("DRILL", DUPLO_NO),1]=100          ;Write a tool parameter
                                                ;(system variable) with tool name
```

Example of the SETPIECE function

This function is used to update the piece number 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.

`SETPIECE(x,y)`

```
x                                               ;Number of completed workpieces
Y                                               ;y spindle number, 0 stands for
                                                ;master spindle (default setting)
```

Example of the GETSELT function

This 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.

Example of tool change using tool management

T1:
Tool preselection; i.e. the tool magazine can be put in tool position parallel to the machining.

M6:
Changing to a preselected tool (depending on default setting in the machine data it may also be programmed without M6).

```
T1 M6                                          ;Load tool 1
D1                                             ;Select tool length compensation
G1 X10 …                                       ;Machining with T1
T="DRILL"                                      ;Preselect drill
D2 Y20 …                                       ;Change cutting edge T1
X10 …                                          ;Machining with T1
M6                                             ;Load tool drill
SETPIECE(4)                                    ;Number of completed workpieces
D1 G1 X10 …                                    ;Machining with drill
```

Note

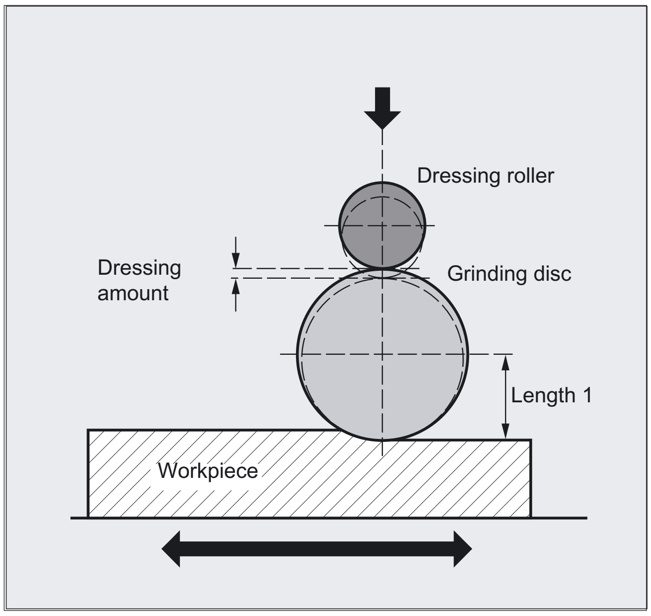
The complete list of all variables for the tool management are contained in

References:
/PGA1/ List of System Variables.

8.3 Online tool compensation (PUTFTOCF, 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.

Programming

`FCTDEF(Polynomial_no., LLimit, ULimit,a0,a1,a2,a3)`

or

`PUTFTOCF(Polynomial_No., Ref_value, Length1_2_3, Channel, Spindle)`

or

`PUTFTOC(Value, Length1_2_3, Channel, Spindle)`

or

`FTOCON`

or

`FTOCOF`

Parameters

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
a_0, a_1, a_2, a_3	Coefficients of polynomial function
Value	Value added in the wear parameter

Example

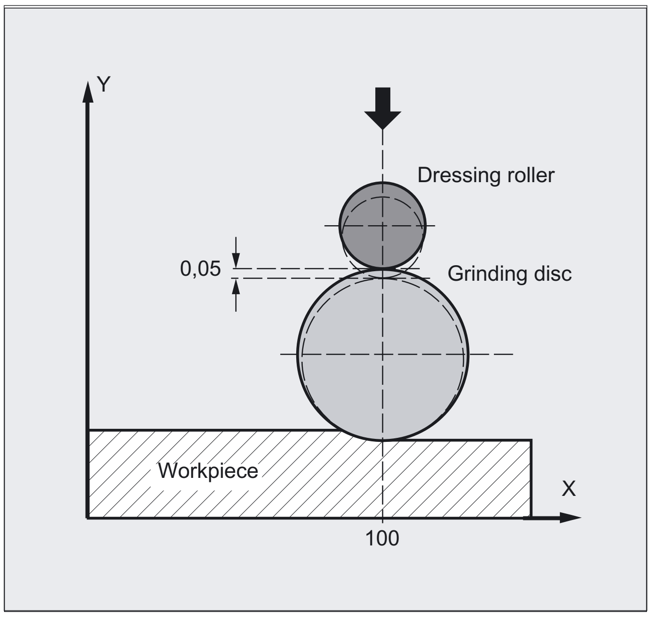
On a surface grinding machine with the following parameters, the grinding wheel is to be dressed by the amount 0.05 after the start of the grinding movement at X100. The dressing amount is to be active with write online offset continuously.

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:

```
%_N_MACH_MPF
…
N110 G1 G18 F10 G90                     ;Initial setting
N120 T1 D1                              ;Select current tool
N130 S100 M3 X100                       ;Spindle ON, traverse against starting
                                        ;position
N140 INIT (2, "DRESS", "S")             ;Select dressing program on channel 2
N150 START (2)                          ;Start dressing program on channel 2
N160 X200                               ;Traverse against target position
N170 FTOCON                             ;Activate online offset
N… G1 X100                              ;Further machining
N… M30
```

Dressing program in channel 2:

```
%_N_DRESS_MPF
…
N40 FCTDEF (1, -1000, 1000, -$AA_IW[V],   ;Define function: Straight
1)
N50 PUTFTOCF (1, $AA_IW[V], 3, 1)         ;Write online offset continuously:
                                          ;Length 3 of the current grinding wheel
                                          ;is derived from the movement of the
                                          ;V axis and corrected in channel 1.
N60 V-0.05 G1 F0.01 G91                   ;Infeed movement for dressing, PUTFTOCF
                                          ;is only effective in this block
…
N… M30
```

Dressing program, modal:

```
%_N_DRESS_MPF
FCTDEF(1,-1000,1000,-$AA_IW[V],1)         ;Define function:
ID=1 DO FTOC(1,$AA_IW[V],3,1)             ;Select online tool offset:
                                          ;Actual value of the V axis is the input
                                          ;value for polynomial 1; the result is
                                          ;added length 3 of the active grinding
                                          ;wheel in channel 1 as the offset value.
WAITM(1,1,2)                              ;Synchronization with machining channel
G1 V-0.05 F0.01, G91                      ;Infeed movement to dress wheel
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

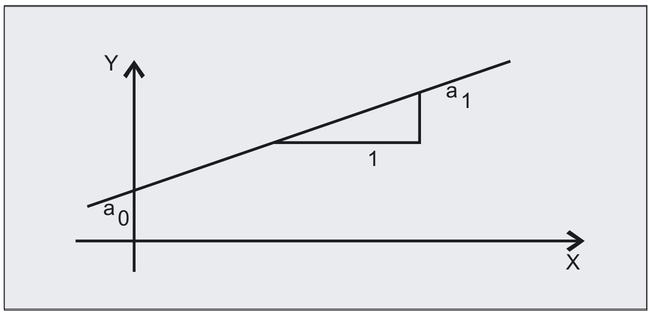
The parameters are defined in a separate block:

```
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 = a0 + a1x) 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.

8.4 Keep tool radius compensation constant (CUTCONON)

Function

The "Keep tool radius compensation constant" function is used to suppress the tool radius compensation for a number of blocks, whereby a difference between the programmed and the actual tool center path traveled set up by the tool radius compensation in the previous blocks is retained as the offset. It can be an advantage to use this method when several traversing blocks are required during line milling in the reversal points, but the contours produced by the tool radius compensation (follow strategies) are not wanted. It can be used independently of the type of tool radius compensation ($2^1/_2$D, 3D face milling, 3D circumferential milling).

Programming

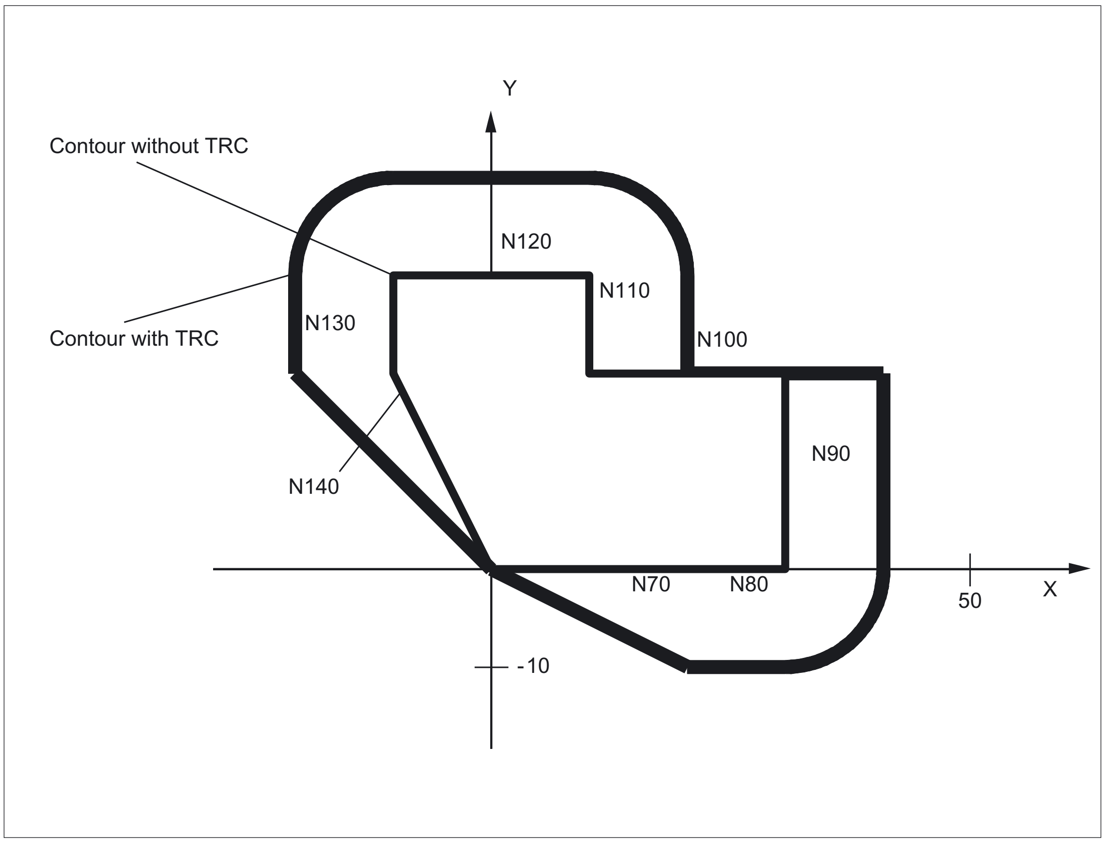
```
CUTCONON
CUTCONOF
```

Parameters

CUTCONON	Activate the tool radius compensation constant function
CUTCONOF	Deactivate the constant function (default setting)

Example

```
N10                                  ;Definition of tool d1
N20 $TC_DP1[1,1] = 110               ;Type
N30 $TC_DP6[1,1]= 10.                ;Radius
N40
N50 X0 Y0 Z0 G1 G17 T1 D1 F10000
N60
N70 X20 G42 NORM
N80 X30
N90 Y20
N100 X10 CUTCONON                    ;Activate compensation suppression
N110 Y30 KONT                        ;Insert bypass circle if necessary on
                                     ;deactivation of contour suppression
N120 X-10 CUTCONOF
N130 Y20 NORM                        ;No bypass circle on deactivation of TRC
N140 X0 Y0 G40
N150 M30
```



Description

Tool radius compensation is normally active before the compensation suppression and is still active when the compensation suppression is deactivated again. In the last traversing block before `CUTCONON`, the offset point in the block end point is approached. All following blocks in which offset suppression is active are traversed without offset. However, they are offset by the vector from the end point of the last offset block to its offset point. These blocks can have any type of interpolation (linear, circular, polynomial).

The deactivation block of the offset suppression, i.e. the block that contains `CUTCONOF`, is offset normally; it starts in the offset point of the start point. One linear block is inserted between the end point of the previous block, i.e. the last programmed traversing block with active `CUTCONON`, and this point.

Circular blocks for which the circle plane is perpendicular to the offset plane (vertical circles), are treated as though they had `CUTCONON` programmed. This implicit activation of the offset suppression is automatically canceled in the first traversing block that contains a traversing motion in the offset plane and is not such a circle. Vertical circle in this sense can only occur during circumferential milling.

8.5 Activate 3D tool offsets (CUT3DC..., CUT3DF...)

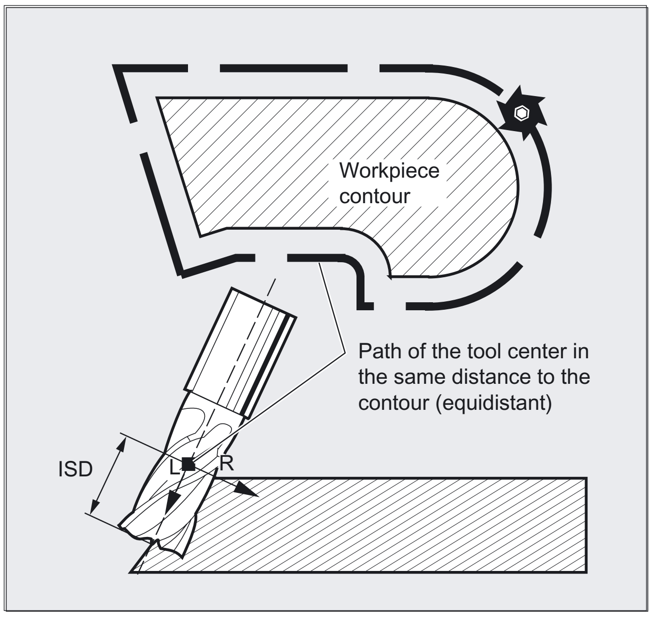
8.5.1 Activate 3D tool offsets (CUT3DC, CUT3DF, CUT3DFS, CUT3DFF)

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. With G41/G42, the left/right-hand compensation is specified in the direction of movement. The approach behavior is always NORM. 3D tool radius compensation is only active when five-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.


Workpiece contour
Path of the tool center in the same distance to the contour (equidistant)
ISD
L
R

Difference between 2 1/2 D and 3D tool radius compensation

In 3D tool radius compensation tool orientation can be changed.

2 1/2 D tool radius compensation assumes the use of a tool with constant orientation.

Programming

```
CUT3DC
```
or
```
CUT3DFS
```
or
```
CUT3DFF
```
or
```
CUT3DF
```

The commands are modal and are in the same group as `CUT2D` and `CUT2DF`. The command is not deselected until the next movement in the current plane is performed. This always applies to `G40` and is independent of the CUT command.

Intermediate blocks are permitted with 3D tool radius compensation. The definitions of the 2 1/2D tool radius compensation apply.

Parameters

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	3D 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=value	Insertion depth

G450/G451 and DISC

A circle block is always inserted at outside corners. `G450/G451` have no effect.

The command `DISC` is not evaluated.

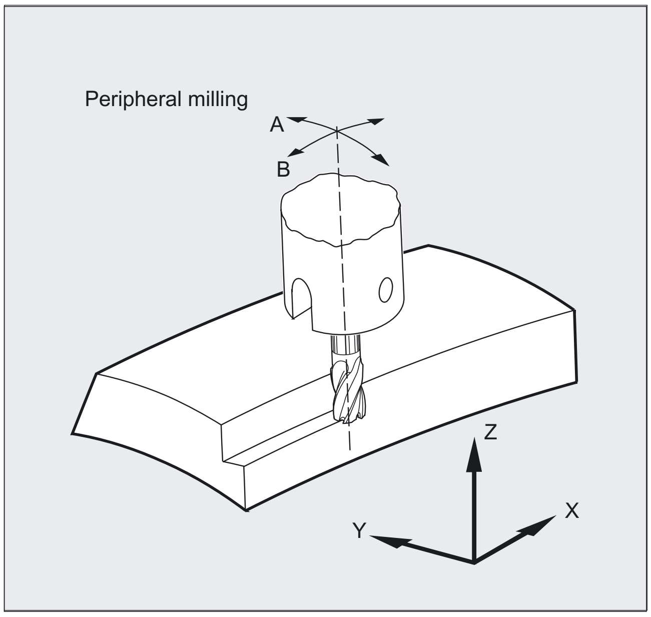
Example

```
N10 A0 B0 X0 Y0 Z0 F5000
N20 T1 D1                          ;Invoke 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 …
```

8.5.2 3D tool radius compensation: peripheral milling, face milling

Peripheral 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. The only decisive factor is the radius at the tool contact point.

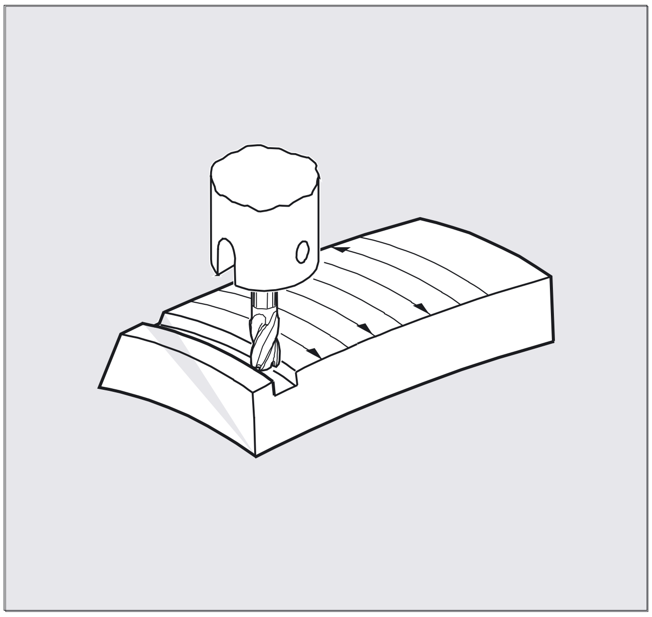


Note

The 3D TRC function is limited to cylindrical tools.

Face milling

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 that are normally performed in CAM. In addition to the NC blocks, the postprocessor writes the tool orientations (when five-axis transformation is active) and the G code for the desired 3D tool offset into the parts program. This feature offers the machine operator the option of using slightly smaller tools than that used to calculate the NC paths.



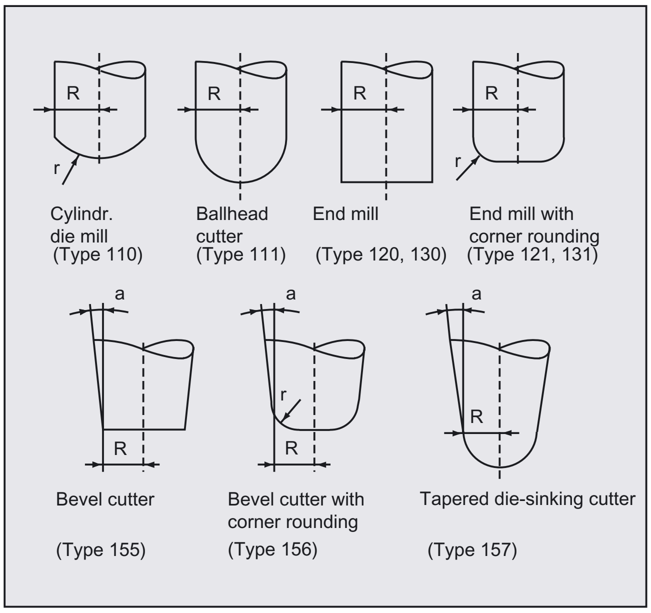
Example:
NC blocks have been calculated with a 10 mm mill. In this case, the workpiece could also be machined with a mill diameter of 9.9 mm, although this would result in a different surface profile.

8.5.3 Tool types/tool change with changed dimensions (G40, G41, G42)

Function

Mill shapes, tool data

The table below gives an overview of the tool shapes, which may be used in face milling operations, as well as tool data limit values. The shape of the tool shaft is not taken into consideration - the tools 120 and 156 are identical in their effect.


R
r
Cylindr. die mill (Type 110)
Ballhead cutter (Type 111)
End mill (Type 120, 130)
End mill with corner rounding (Type 121, 131)
a
Bevel cutter (Type 155)
Bevel cutter with corner rounding (Type 156)
Tapered die-sinking cutter (Type 157)

If a different type number is used in the NC program than the one listed in the table, the system automatically uses tool type 110 die-sinking cutter. An alarm is output if the tool data limit values are violated.

Parameters

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

Tool data	**Tool parameters**		X = is not evaluated
Tool dimensions	Geometry	Wear	
R	$TC_DP6	$TC_DP15	R = shank radius (tool radius)
r	$TC_DP7	$TC_DP16	r = corner radius
a	$TC_DP11	$TC_DP20	a = angle between tool longitudinal axes and upper end of torus surface

Tool length offset

The tool tip is the reference point for length offset (intersection longitudinal axis/surface).

3D tool offset, tool change

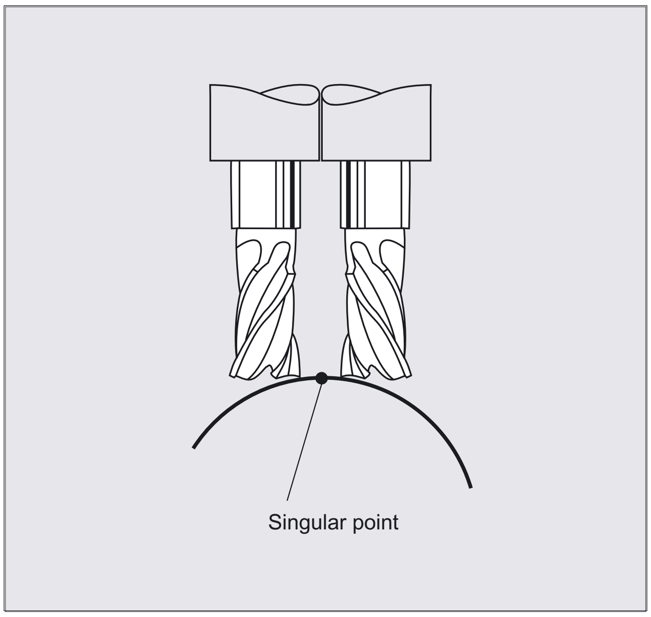
A new tool with changed dimensions (R, r, a) or a different form may only be specified with the programming of `G41` or `G42` (transition G40 to G41 or `G42`, reprogramming of `G41` `G42`). This rule does not apply to any other tool data, e.g., tool lengths, so that tools to which such data apply can be fitted without reprogramming `G41` or `G42` .

8.5.4 Compensation on the path, path curvature, and insertion depth ISD and tool status (CUT3DC)

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.

Programming

Insertion depth (ISD)

`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 peripheral milling operations. This makes it possible to change the position of the machining point on the outer surface of the tool.

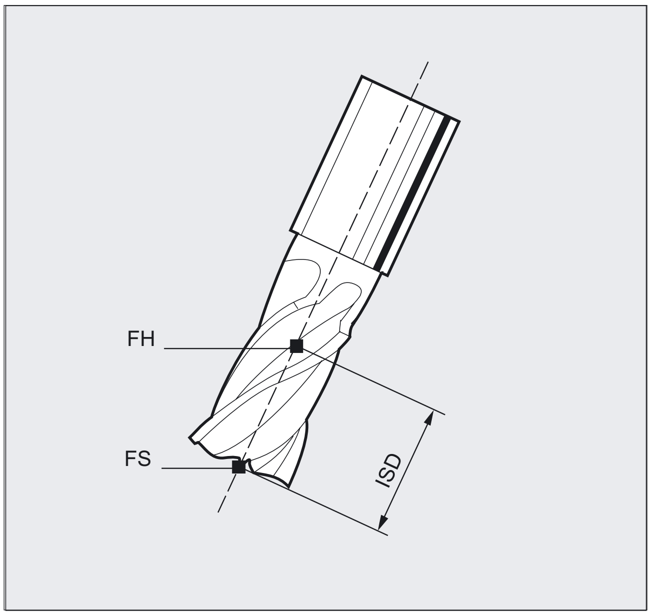
3D tool compensation circumference milling

```
CUT3DC
```

Parameters

`CUT3DC`	`Activate 3D tool offset for circumferential milling, e.g., for pocket milling with oblique side walls.`
`ISD`	`ISD defines the distance between cutter tip FS and cutter construction point FH.`

Point FH is obtained by projecting the programmed machining point onto the tool axis.



Description

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 face end of the mill is located remains unchanged if the insertion depth `ISD` has remained the same. For example, a mill 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 more information on collision monitoring, see
Literature: /PG/ Programming Manual Fundamentals, "Tool Offsets" section.

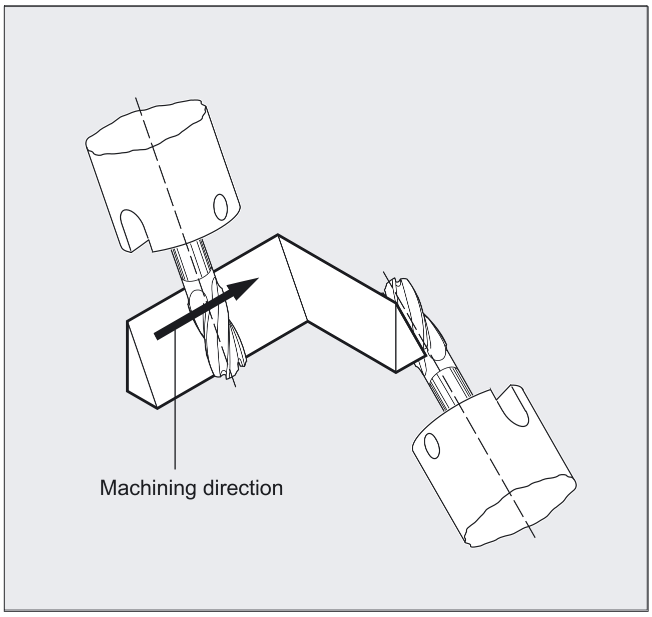
8.5.5 Inside corners/outside corners and intersection procedure (G450/G451)

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.



Programming

```
G450
```

or

```
G451
```

Parameters

`G450`	`Transition circle (tool travels round workpiece corners on a circular path).`
`G451`	`Intersection of equidistant paths (tool backs off from the workpiece corner).`

Description

Intersection procedure for 3D compensation

With 3D circumferential milling, G code `G450/G451` is now evaluated at the outside corners; this means that the 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, as with 2 ½ D radius compensation, both of the curves involved are lengthened and the intersection of both lengthened 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 no such intersection, the corner is handled as previously, that is, a circle is inserted.

References:

Further information for the intersection procedure /FB/ W5, 3D Tool Radius Compensation

8.5.6 3D circumferential milling with limitation surfaces general use

Function

Adaptations 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 in the parts program.

Suitable measures must be taken to replace important information that would be required for optimum correction but is not longer available. Here are some typical compensation methods for critical transitions either

- directly in the parts program or
- while determining the real contour, e.g. by tool infeed.

Applications

In addition to the typical application case 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 tool-independent. Like with conventional tool radius compensation, the entire radius if used to calculate the perpendicular offset to the limitation surface.

8.5.7 Consideration of 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.

Programming

`CUT3DCCD`

or

`CUT3DCC`

Parameters

`CUT3DCCD`	`Activation of 3D tool offset for the circumferential milling with limitation surfaces with a differential tool on the tool center point path:`
	`infeed to the limitation surface.`

`CUT3DCC`	`Activation of the 3D tool offset for circumferential milling with limitation surfaces with 3D radius compensation:`
	`contour on the machining surface.`

Note

Tool radius compensation with G41, G42

If tool radius compensation with `G41, G42` is programmed when `CUT3DCCD` or `CUT3DCC` is active, the option "orientation transformation" must also be active.

Standard tools with corner rounding

Corner rounding with a standard tool is defined by the tool parameter `$TC_DP7`. Tool parameter `$TC_DP16` describes the deviation of the corner rounding of the real tool compared with the standard tool.

Example

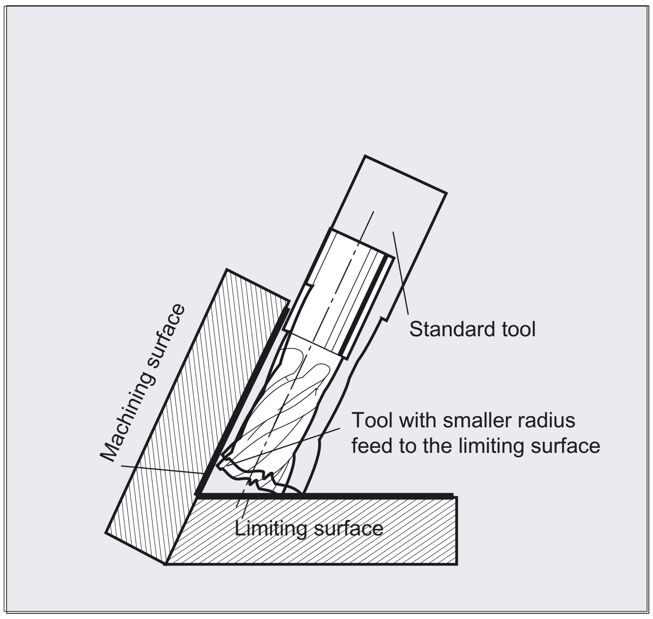
Tool dimensions of a toroidal miller with reduced radius as compared with the standard tool.

Tool type	R = shank radius	r = corner radius
Standard tool with corner rounding	R = $TC_TP6	r = $TC_TP7
Real tool with corner rounding: Tool types 121 and 131 toroidal miller (end mill)	R' = $TC_TP6 + $TC_TP15 + OFFN	r' = $TC_TP7 + $TC_TP6
In this example tool type ($TC_DP1) is evaluated.	either or	$TC_TP15 + OFFN $TC_TP16 negative.
Only cutter types with cylindrical shank are permitted (cylinder or end mill) and toroidal miller (type 121 and 131) and in the limit case of the cylindrical die mill (type 110).	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.		

Description

Tool center point path with infeed up to the limitation surface CUT3DCCD

If a tool with a smaller radius than the suitable standard tool is used machining is continued with a milling cutter that is infed in the longitudinal direction until it reaches the bottom of the pocket. The tool removes as much material from the corner formed by the surface of limitation and the machined surface as possible. This a combined method of machining using circumferential and face milling. By analogy, if the tool has a larger radius it is infed in the opposite direction.



Unlike all other tool compensations of G code group 22, tool parameter `$TC_DP6` specified for `CUT3DCCD` does not affect the tool radius and the resulting compensation.

The compensation is the sum of

- the wear value of the tool radius (tool parameter `$TC_DP15`)

and a

- programmed tool offset `OFFN`.

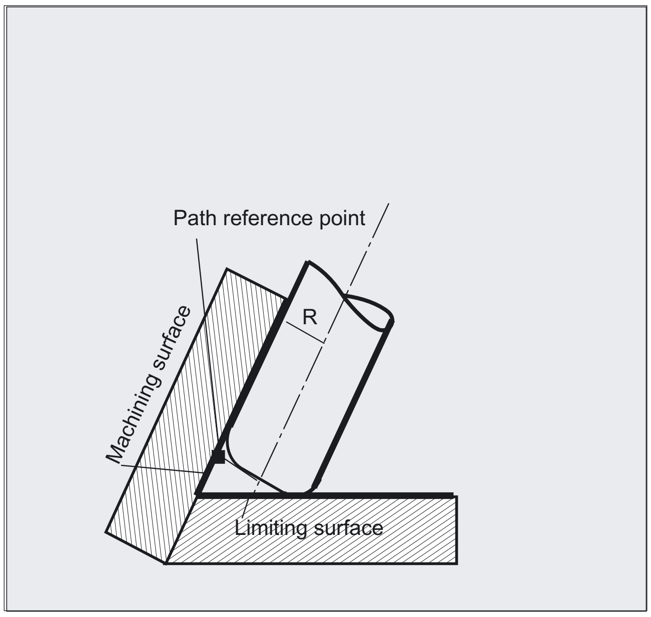
The generated program does not specify whether the surface to be machined is right or left of the path. It is therefore assumed that the radius is a positive value and the wear value of the original tool a negative value. A negative wear value always describes a tool with a reduced diameter.

Using cylindrical tools

If 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 toroidal miller is used (cylinder with rounded corners) tool infeed in the longitudinal direction is required for both acute and obtuse angles.

3D radius compensation with CUT3DCC, contour on the machining surface

If `CUT3DCC` is active with a toroidal miller the programmed path refers to a fictitious cylindrical mill with the same diameter. The resulting path reference point is shown in the following figure for a toroidal miller.



The angle between the machining and limitation surface may change from an acute to an obtuse angle and vice versa even within the same block.

The tool actually used may be either larger or smaller than the standard tool. But the resulting corner radius must not be negative and the sign in front of the resulting tool radius must not change.

In `CUT3DCC` the NC parts program refers to the contour on the machining surface. As with conventional tool radius compensation, the total radius, which is totaled from

- the tool radius (tool parameter `$TC_DP6`)
- the wear value (tool parameter `$TC_DP15`)

and a

- programmed tool offset `OFFN`.

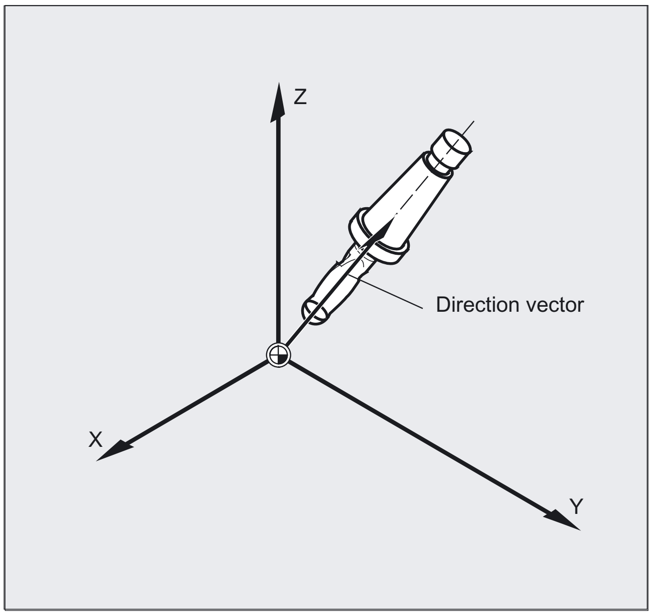
is used. The position of the limitation surface is determined by the difference between the two values

- standard tool dimensions and
- tool radius (tool parameter `$TC_DP6`).

8.6 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.

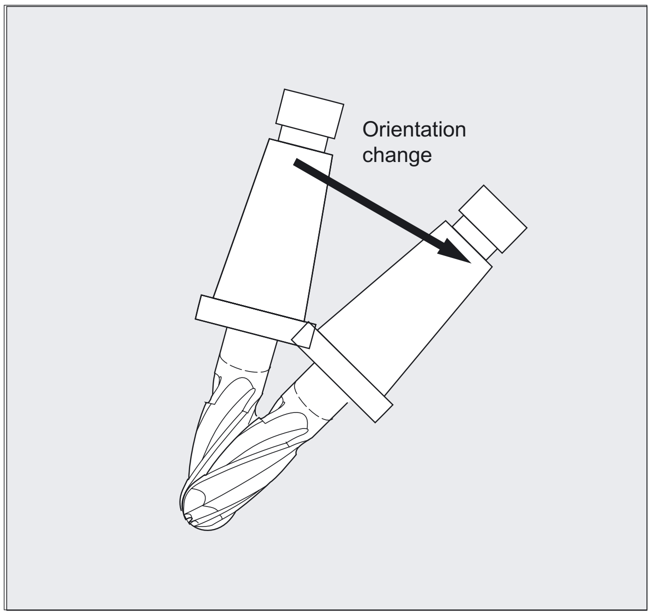
If vector interpolation is active, the smoothed orientation characteristic is also interpolated using vector interpolation. On the other hand, if round axis interpolation is active, the orientation is smoothed directly using round axis movements.

Programming

A change in tool orientation can be programmed by:

- Direct programming of round axes `A, B, C` (round 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`).

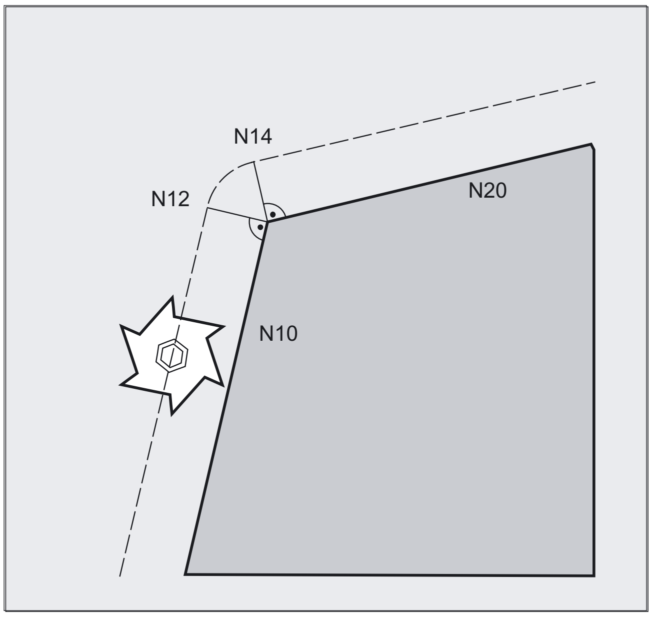


Parameters

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	Rounding of orientation by specifying rounding length with SD $SC_ORI_SMOOTH_DIST.
OST	Rounding of orientation by specifying angle tolerance in degrees for vector interpolation with SD $SC_ORI_SMOOTH_TOL. With round axis interpolation, the specified tolerance is assumed to be the maximum variance of the orientation axes.

ORIC example

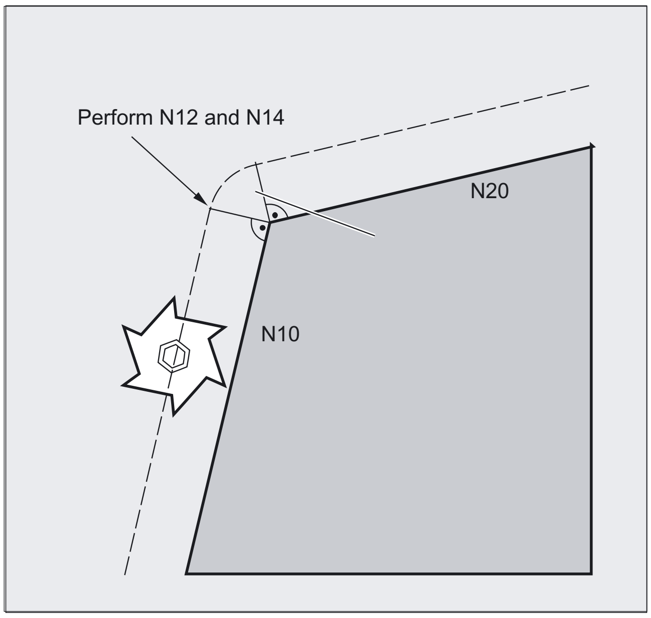
If two or more blocks with orientation changes are programmed between the traversing blocks `N10` and `N20` (e.g., `A2= B2= C2=`) and `ORIC` is active, the inserted circle block is divided according to the size of the angle changes on these intermediate blocks.



```
ORIC
N8 A2=… B2=… C2=…
N10 X… Y… Z…
N12 C2=… B2=…                    ;The circle block inserted at the external corner
N14 C2=… B2=…                    ;is divided among N12 and N14 in accordance with
                                 ;the change in orientation. The circular movement
                                 ;and the orientation change are executed in
                                 ;parallel.
N20 X =…Y=… Z=… G1 F200
```

ORID example

If `ORID` is active, all the 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.



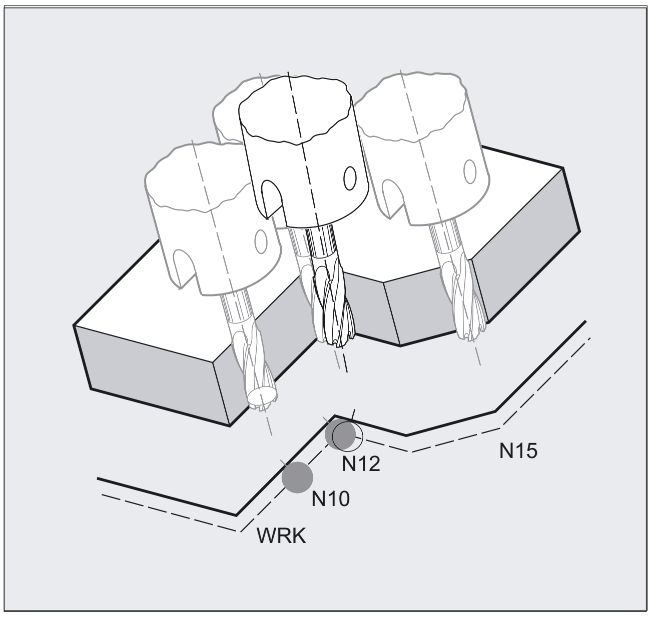
```
ORID
N8 A2=… B2=… C2=…
N10 X… Y… Z…
N12 A2=… B2=… C2=…                 ;The N12 and N14 blocks are executed at the end
                                   ;of N10. The circle block is then executed with
                                   ;the current orientation.
N14 M20                            ;Auxiliary functions, etc.
N20 X… Y… Z…
```

Note

The method by which the orientation is changed at an outer corner is determined by 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 for the change in orientation at an inner corner

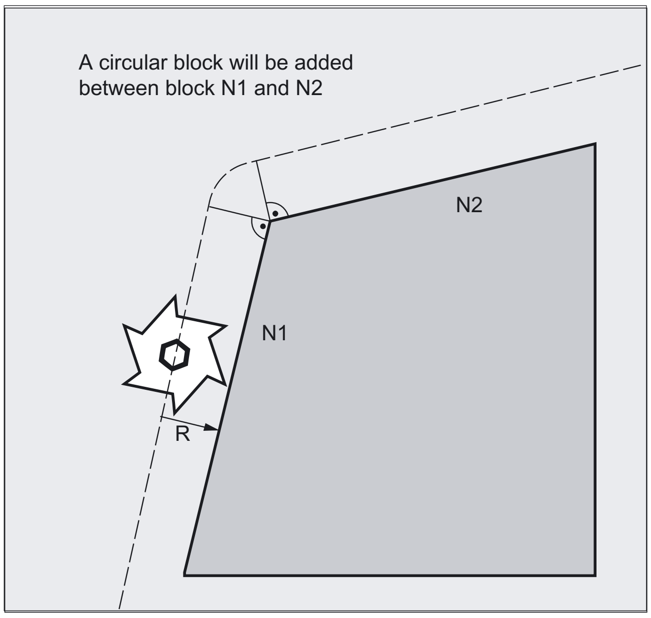


```
ORIC
N10 X …Y… Z… G1 F500
N12 X …Y… Z… A2=… B2=…, C2=…
N15 X Y Z A2 B2 C2
```

Behavior at outer corners

A circle block with the radius of the cutter is always inserted at an outside corner.

The program commands `ORIC` or `ORID` can be used to define whether changes in orientation programmed between blocks `N1` and `N2` are executed before the beginning of the inserted circle block 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.

With `ORID`, the inserted blocks are executed initially without a path movement. 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 movement is divided among the individual inserted blocks according to the values of the orientation changes.

Rounding orientation with OSD and OST

When rounding with G642, the maximum variance for the contour axes and orientation axes cannot vary greatly. The smaller tolerance of the two determines smoothing the shape

- the rounding movement or angle tolerance,
- the orientation characteristics

to a relatively severe extent without having to accept larger contour variances.

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 –> round axis, round axis –> vector). These block transitions can if necessary be rounded with the standard rounding functions G641, G642 and G643.

8.7 Free assignment of D numbers, cutting edge numbers

8.7.1 Free assignment of D numbers, cutting edge numbers (CE address)

Function

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.

References:

/FB1/Function Manual Basic Functions; Tool Offset (W1).

Machine manufacturer

The maximum number of D numbers (cutting edge numbers) and maximum number of cutting edges per tool are defined via the machine data. The following commands only make sense when the maximum number of cutting edges (MD 18105) is greater than the number of cutting edges per tool (MD 18106). See machine manufacturer's specifications.

Note

Besides the relative D number, you can also assign D numbers as 'flat' or 'absolute'
D numbers (-32000) without assigning a reference to a T number
(inside the function 'flat D number structure').

8.7.2 Checking D numbers (CHKDNO)

Function

CKKDNO checks whether the available D numbers assigned are unique. The D numbers of all tools defined within a TO unit may not occur more than once. No allowance is made for replacement tools.

Programming

```
state=CHKDNO(Tno1,Tno2,Dno)
```

Parameters

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 parts 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.

8.7.3 Renaming 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.

Programming

```
d = GETDNO (t,ce)
state = SETDNO (t,ce,d)
```

Parameters

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

```
$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.

8.7.4 Deriving the T number from the specified D number (GETACTTD)

Function

For an absolute D number, GETACTTD determines the associated T number. 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).

Programming

```
status = GETACTTD (Tnr, Dnr)
```

Parameters

Dno	D number for which the T number shall be searched.
Tno	T number found
status	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 contains the value of the first tool found that contains the D number with the value Dno. -5: The Function has not been executed for some other reason.

8.7.5 Invalidate D numbers (DZERO)

Function

This command is used for support during retooling. Offset data sets tagged with this command are no longer verified by the CHKDNO language command. These data sets can be accessed again by setting the D number again with SETDNO.

Programming

```
DZERO
```

Parameters

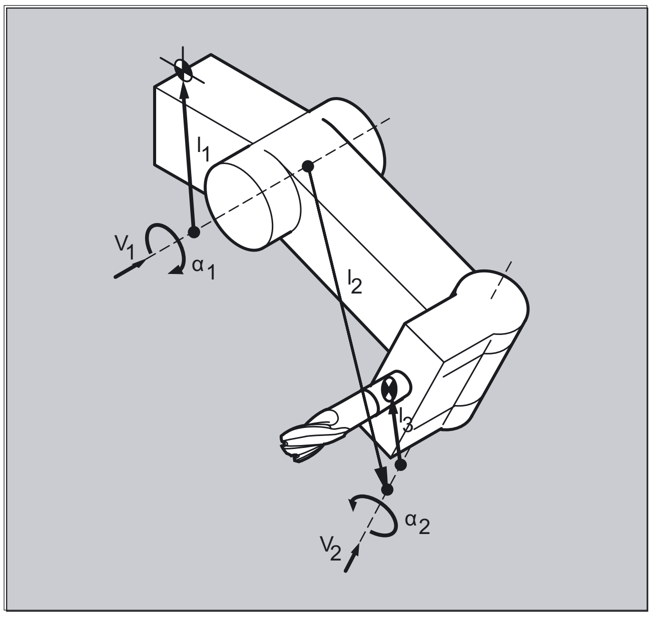
DZERO	Marks all D number of the TO unit as invalid

8.8 Tool holder kinematics

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 α_1, α_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	y 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]

Function of the system variables for orientable toolholders			
α_1 angle of rotation α_2 angle of rotation	$TC_CARR13[m] $TC_CARR14[m]		
l_3 offset vector	$TC_CARR15[m]	$TC_CARR16[m]	$TC_CARR17[m]

Extensions of the system variables for orientable toolholders			
Designation	x component	y component	y component
l_4 offset vector	$TC_CARR18[m]	$TC_CARR19[m]	$TC_CARR20[m]
Axis identifier rotary axis v_1 rotary axis v_2	Axis identifier of the rotary axes v_1 and v_2 (initialized with zero) $TC_CARR21[m] $TC_CARR22[m]		
Kinematic type	$TC_CARR23[m]		
Tool Part Mixed mode	Kinematics type T ->	Kinematics type P ->	Kinematics type M
	Only the tool can rotate (default).	Only the part can rotate	Part and tool can rotate
Offset of the rotary axis v_1 rotary axis v_2	Angle in degrees of the rotary axes v_1 and v_2 on assuming the initial setting $TC_CARR24[m] $TC_CARR25[m]		
Angle offset of the rotary axis v_1 rotary axis v_2	Offset of the Hirth tooth system in degrees for rotary axes v_1 and v_2 $TC_CARR26[m] $TC_CARR27[m]		
Angle increment v_1 rotary axis v_2 rotary axis	Offset of the Hirth tooth system in degrees for rotary axes v_1 and v_2 $TC_CARR28[m] $TC_CARR29[m]		
Min. position rotary axis v_1 rotary axis v_2	Software limit for the minimum position of the rotary axes v_1 and v_2 $TC_CARR30[m] $TC_CARR31[m]		
Max. position rotary axis v_1 rotary axis v_2	Software limits for the maximum position of the rotary axes v_1 and v_2 $TC_CARR32[m] $TC_CARR33[m]		
Toolholder name	A toolholder can be given a name instead of a number. $TC_CARR34[m]		
User: axis name 1 axis name 2 identifier	Intended use in user measuring cycles $TC_CARR35[m] $TC_CARR36[m] $TC_CARR37[m]		
Position	$TC_CARR38[m]	$TC_CARR39[m]	$TC_CARR40[m]
Fine **offset**	Parameters that can be added **to the values in the basic parameters**.		
l_1 offset vector	$TC_CARR41[m]	$TC_CARR42[m]	$TC_CARR43[m]
l_2 offset vector	$TC_CARR44[m]	$TC_CARR45[m]	$TC_CARR46[m]
l_3 offset vector	$TC_CARR55[m]	$TC_CARR56[m]	$TC_CARR57[m]
l_4 offset vector	$TC_CARR58[m]	$TC_CARR59[m]	$TC_CARR60[m]
v_1 rotary axis	$TC_CARR64[m]		
v_2 rotary axis	$TC_CARR65[m]		

Note

Explanations of parameters

"m" specifies the number of the toolholder to be programmed.

`$TC_CARR47` to `$TC_CARR54` and `$TC_CARR61` to `$TC_CARR63` are not defined and produce an alarm if read or write access is attempted.

The start/endpoints of the distance vectors on the axes can be freely selected. The rotation angles α_1, α_2 about the two axes are defined in the initial state of the toolholder by 0°. In this way, the kinematics of a toolholder can be programmed for any number of possibilities.

Toolholders with only one or no rotary axis at all can be described by setting the direction vectors of one or both rotary axes to zero.
With a toolholder without rotary axis the distance vectors act as additional tool offsets whose components cannot be affected by a change of machining plane (`G17` to `G19`).

Parameter extensions

Rotary axis parameters $TC_CARR24 to $TC_CARR33

The system variables have been extended by the entries $TC_CARR24[m] to $TC_CARR33[m] and described as follows:

The **offset** of the rotary axes v_1, v_2	Changing the position of the rotary axis v_1 or v_2 for the initial setting of the oriented toolholder.
The **angle offset/angle increment** of the rotary axes v_1, v_2	The offset or the angle increment of the Hirth tooth system of the rotary axes v_1 and v_2. Programmed or calculated angle is rounded up to the next value that results from phi = s + n * d when n is an integer.
The minimum and maximum position of the rotary axes v_1, v_2	The minimum and maximum position of the rotary axis limit angle (software limit) of the rotary axes v1 and v2.

User parameters $TC_CARR34 to $TC_CARR40

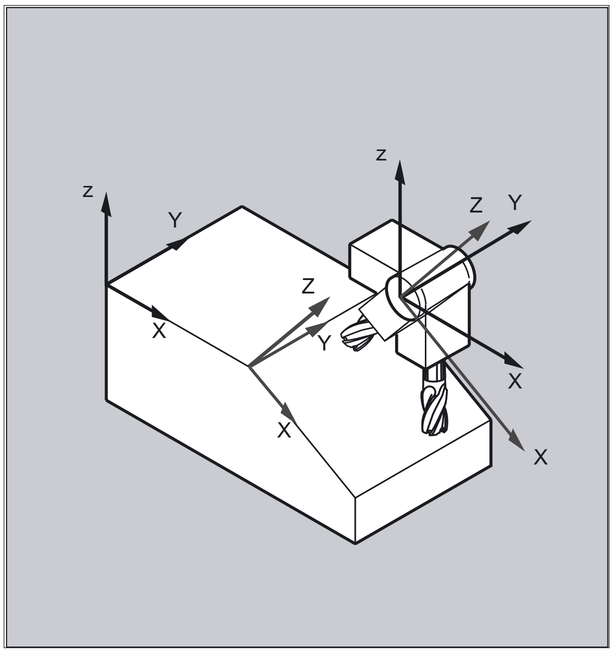
User	contain parameters that are freely available to the user and, up to software version 6.4, were not further interpreted in the NCK or have no meaning.

Fine offset parameters $TC_CARR41 to $TC_CARR65

Fine offset	contain fine offset parameters that can be added to the values in the basic 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.



```
N10 $TC_CARR8[1]=1                     ;Definition of the Y components of the
                                       ;first rotary axis of toolholder 1
N20 $TC_DP1[1,1] = 120                 ;Definition of an end mill
N30 $TC_DP3[1,1]=20                    ;Definition of an end mill with
                                       ;length 20 mm
N40 $TC_DP6[1,1]=5                     ;Definition of an end mill with
                                       ;radius 5 mm
N50 ROT Y37                            ;Frame definition with 37° rotation around
                                       ;the Y axis
N60 X0 Y0 Z0 F10000                    ;Approach start position
N70 G42 CUT2DF TCOFR TCARR=1 T1 D1 X10 ;Set radius compensation, tool length
                                       ;offset in rotated frame,
                                       ;select toolholder 1, tool 1
N80 X40                                ;Execute machining under a 37° rotation
N90 Y40
N100 X0
N110 Y0
N120 M30
```

Requirements

A toolholder can only orientate a tool in all possible directions in space if

- two rotary axes V_1 and V_2 are present.
- the rotary axes are mutually orthogonal.

- the tool longitudinal axis is perpendicular to the second rotary axis V_2.

In addition, the following requirement is applicable to machines for which all possible orientations have to be settable:

- the tool longitudinal axis must be perpendicular to the first rotary axis V_1.

Description

Resolved kinematics

For machines with resolved kinematics (both the tool and the part can rotate), the system variables have been extended to include the entries `$TC_CARR18[m]` to `$TC_CARR23[m]` are described as follows:

The rotatable tool table consisting of:

- the vectoral distance of the second rotary axis V_2 to the reference point of a rotatable tool table I_4 of the third rotary axis.

The rotary axes consisting of:

- the two channel identifiers for the reference to the rotary axes V_1 and V_2, whose position is accessed as required to determine the orientation of the orientable toolholder.

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

`$TC_CARR1[0] = 0` can be used to clear the data of all toolholder data blocks.

The type of kinematics `$TC_CARR23[T] = T` must be assigned one of the three permissible uppercase or lowercase letter (T,P,M) and should not be deleted.

Changing the toolholder data

Each of the described values can be modified by assigning a new value in the parts program. Any character other than T, P or M causes an alarm when you attempt to activate the orientable toolholder.

Reading the toolholder data

Each of the described values can be read by assigning it to a variable in the parts program.

Fine offsets

A permissible fine offset value is not detected unless an orientable toolholder that contains such a value is activated and setting data `SD 42974: TOCARR_FINE_CORRECTION = TRUE`.

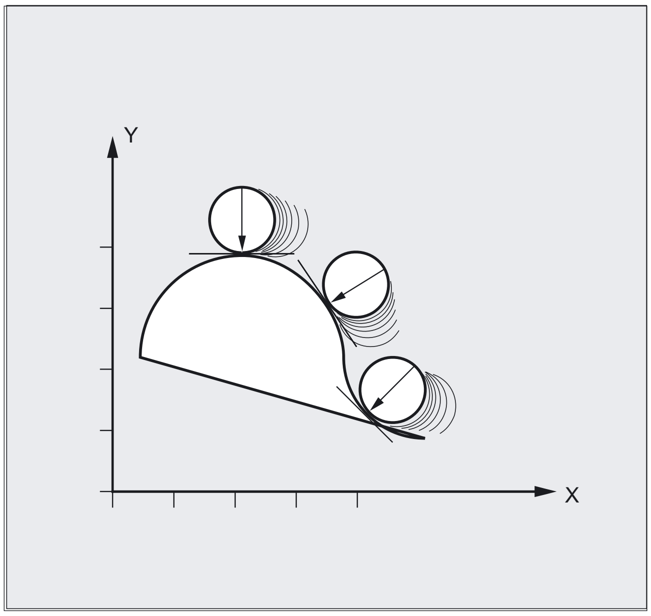
The maximum permissible fine offset is limited to a permissible value in the machine data.

Path traversing behavior 9

9.1 Tangential control (TANG, TANGON, TANGOF, TANGDEL)

Function

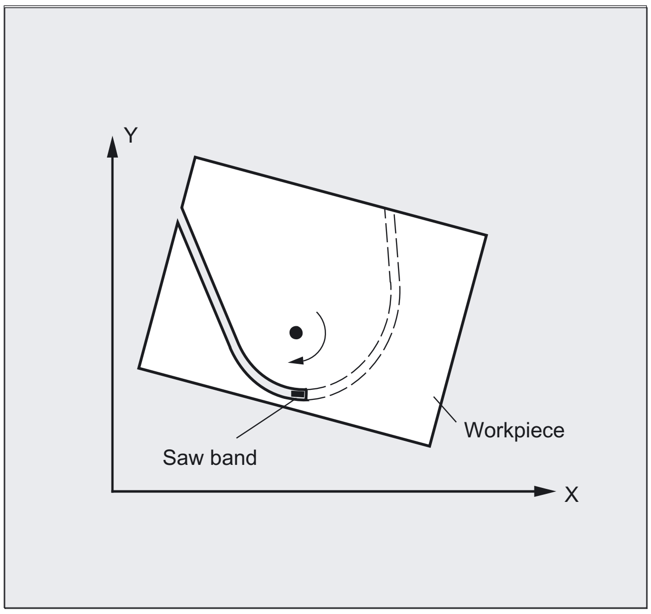
The following axis follows the path of the leading axis along the tangent. This allows alignment of the tool parallel to the contour. The tool can be positioned relative to the tangent with the angle programmed in the TANGON statement.



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.



Programming

`TANG (Faxis,Laxis1,Laxis2,Coupling,CS,Opt)`

or

`TANGON (Faxis,Angle, Dist, Angletol)`

or

`TANGOF (Faxis)`

or

`TLIFT (Faxis)`

or

`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.

Parameters

TANG	Preparatory statement for the definition of a tangential follow-up; default setting: 1 **TANG(C,X,Y,1,"B") means:** Rotary axis C follows geometry axes X and Y. Disable TLIFT
TANGON	Activate tangential control specifying following axis and required offset angle of the following axis and, if necessary, rounding path, angle deviation. **TANGON(C,90) means:** 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.
TANGOF	Deactivate tangential control specifying following axis. The following axis is specified in order to deactivate the tangential control: **TANGOF(C)**
TLIFT	Insert intermediate block at contour corners
TANGDEL	Delete definition of a tangential follow-up. **Example: TANGDEL (FAxis)**
Faxis	Following axis: additional tangential following rotary axis.
Laxis1, Laxis2	Leading axes: path axes, which determine the tangent for the following axis.
Coupling	Coupling factor: relationship between the angle change of the tangent and the following axis. Parameter optional; default: 1
CS	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 time change 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 for plane change

```
N10 TANG(A, X, Y,1)                      ;1. definition of the tang. follow-up
N20 TANGON(A)                            ;Activation of the coupling
N30 X10 Y20                              ;Radius
...
N80 TANGOF(A)                            ;Deactivate 1st coupling
N90 TANGDEL(A)                           ;Delete 1st definition
...
TANG(A, X, Z)                            ;2. definition of the tang. follow-up
TANGON(A)                                ;Activation of the new coupling
...
N200 M30
```

Example of the geometry axis switching and TANGDEL

No alarm is produced.

```
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)                            ;Deactivation of follow-up with Y1
N60 TANGDEL(A)                           ;Delete 1st definition
N70 GEOAX(2, Y2)                         ;Y2 is the new geometry axis 2
N80 TANG(A, X, Y)                        ;2. definition of the tang. follow-up
N90 TANGON(A, 90)                        ;Activation of the follow-up with 2nd def.
...
```

Example of the tangential follow-up with automatic optimization

Automatic optimization using Dist and angle tolerance

```
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 path veloc.
N180 TANG(C,X,Y, 1,,"P")               ;Rounding path 5 mm,
N190 TANGON(C, 0, 5.0, 2.0)            ;Angle tolerance 2 degrees
N210 G1 X1310 Y500                     ;Activation of the follow-up with 2nd def.
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).

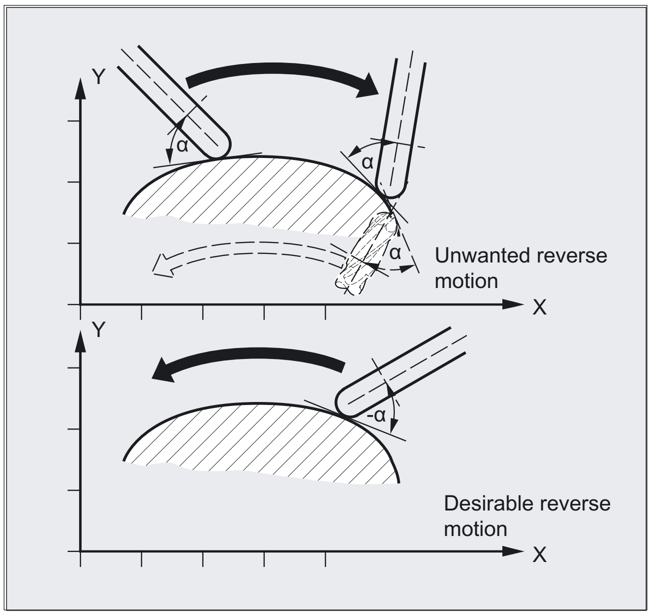
Limit angle using the working area limitation

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.

This is done by limiting the working area of the following axis (G25, G26). The working area limitation 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.

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`.

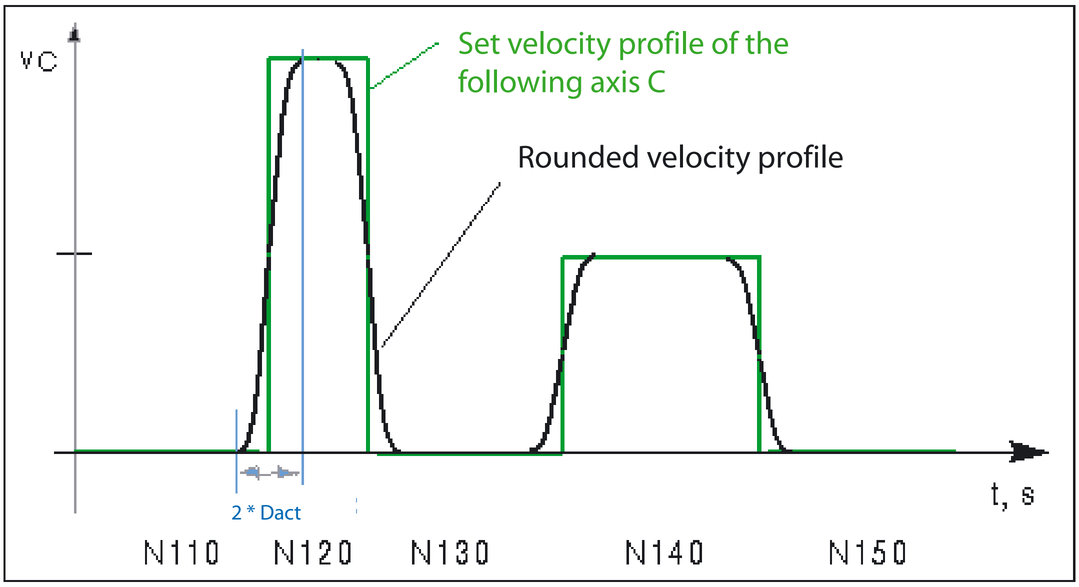
If the follow-up axis was not previously traversed as a path axis it is now traversed as a positioning axis. The velocity is then dependent on the positioning velocity in the machine data.

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

9.2 Coupled motion (TRAILON, TRAILOF)

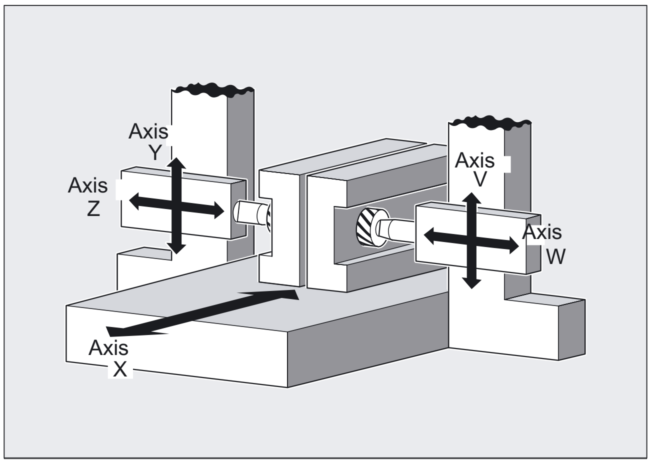
Function

When a defined leading axis is moved, the trailing 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-side machining with 2 coupled axes:
  1st leading axis Y, coupled axis V
  2nd leading axis Z, coupled axis W



Programming

`TRAILON(Faxis,Laxis,Coupling)`

or

`TRAILOF(Faxis,Laxis,LAxis2)`

or deactivate without specification of leading axis

`TRAILOF(FAxis)`

`TRAILON` and `TRAILOF` act modal.

Parameters

`TRAILON`	Activating and defining a coupled-axis grouping **Example:** V = trailing axis, Y = leading axis `TRAILON(V,Y)`

`TRAILOF`	Deactivate coupled axes
	Example:V = trailing axis, Y = leading axis
	`TRAILOF(V,Y)`
	TRAILOF with 2 parameters deactivates the coupling to only 1 leading axis. If a trailing axis is assigned to 2 leading axes, e.g. V=trailing axis and X,Y=leading axes, TRAILOF can be called with 3 parameters to deactivate 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
`Coupling`	Coupling factor = Path of coupled-motion axis/path of trailing axis Default = 1

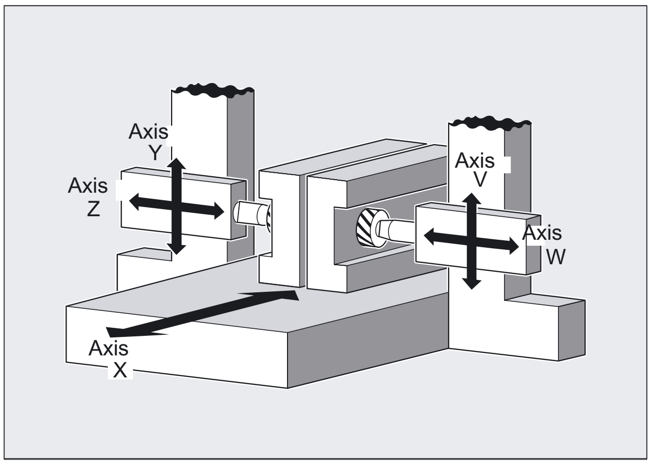
Note

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.



```
…
N100 TRAILON(V,Y)      ;Activate 1st coupled axis grouping
N110 TRAILON(W,Z,-1)   ;Activate 2nd combined axis pair, coupling factor negative:
                       ;Trailing axis traverses in opposite direction to leading
                       ;axis
N120 G0 Z10            ;Infeed Z and W axes in opposite axial directions
N130 G0 Y20            ;Infeed of Y and V axes in same axis directions
…
N200 G1 Y22 V25 F200   ;Superimpose dependent and independent movement of trailing
                       ;axis "V"
…
TRAILOF(V,Y)           ;Deactivate 1st coupled axis grouping
TRAILOF(W,Z)           ;Deactivate 2nd coupled axis grouping
```

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

9.3 Curve tables (CTAB)

9.3.1 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 .

9.3.2 Principal functions curve tables (CTABDEF, CTABEND, CTABDEL)

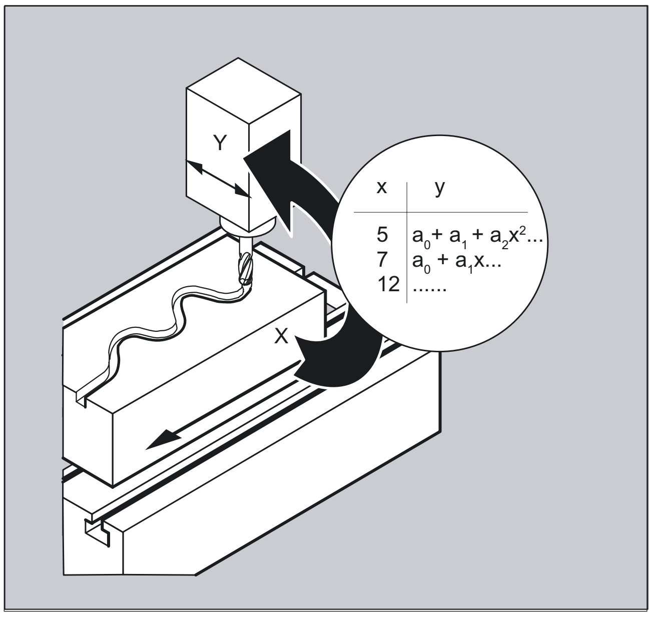
Function

You can use curve tables to program position and velocity relationships between two axes. Curve tables are defined in a parts 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.



Programming

Modal language commands with curve tables

```
CTABDEF(FAxis, LAxis, n, applim, memType)
```

or

```
CTABEND ()
```

or

```
CTABDEL(), CTABDEL(, ,memType)
```

Parameters

Principal functions

CTABDEF ( )	Define beginning of curve table.
CTABEND ()	Define end of curve table.
CTABDEL ()	Deleting all curve tables, **irrespective of the memory type.**
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.

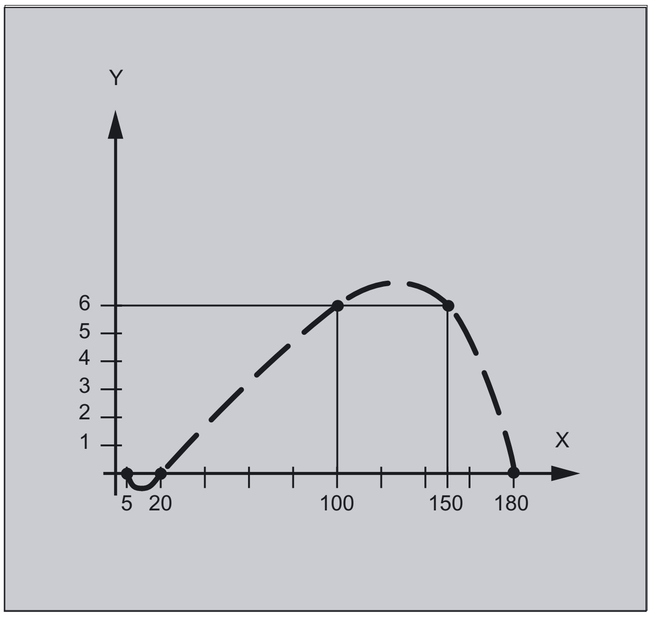
Machine manufacturer
To create curve tables the memory space must be reserved by setting the machine data.

Example of 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:

```
…
CTABDEF(Y,X,1,1)                        ;Definition of a curve table
…
…
IF NOT ($P_CTABDEF)
STOPRE
ENDIF
…
…
CTABEND
```

Example of the definition of a curve table



```
N100 CTABDEF(Y,X,3,0)                   ;Begin of the definition of a non-periodic
                                        ;curve table with the number 3 definition
                                        ;of the curve table
N110 X0 Y0                              ;1. traverse statement defines 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 the definition; The curve table is
                                        ;generated in its internal representation
                                        ;as a polynomial up to the 3rd order;
                                        ;The calculation of the curve definition
                                        ;depends on the modally selected
                                        ;interpolation type
                                        ;(circular, linear, spline interpolation);
                                        ;The parts program state before the
                                        ;beginning of the definition is restored.
```

Example of the 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:

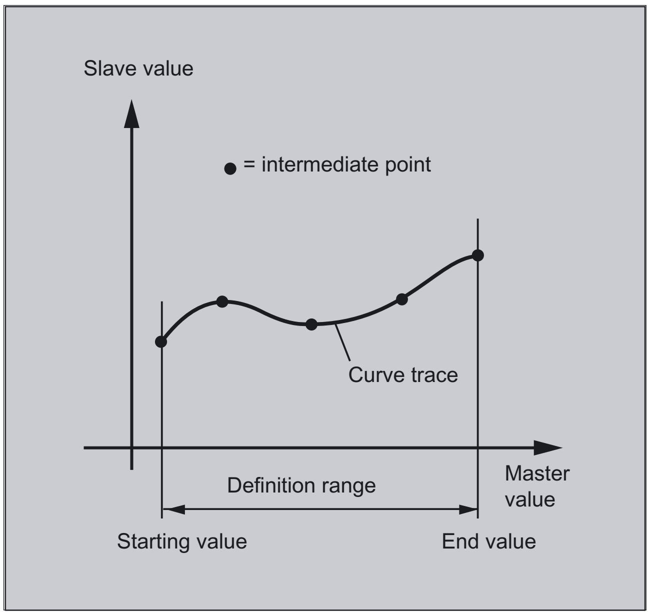
```
N10 DEF REAL DEPPOS
N20 DEF REAL GRADIENT
N30 CTABDEF(Y,X,2,1)                     ;Beginning of definition
N40 G1 X=0 Y=0
N50 POLY
N60 PO[X]=(45.0)
N70 PO[X]=(90.0) PO[Y]=(45.0,135.0,-90)
N80 PO[X]=(270.0)
N90 PO[X]=(315.0) PO[Y]=(0.0,-135.0,90)
N100 PO[X]=(360.0)
N110 CTABEND                             ;End of the definition
                                         ;Test of the curve by coupling Y to X
N120 G1 F1000 X0
N130 LEADON(Y,X,2)
N140 X360
N150 X0
N160 LEADOF(Y,X)
                                         ;Read the table function for
                                         ;master value 75.0
N170 DEPPOS=CTAB(75.0,2,GRADIENT)
                                         ;Positioning of the leading and the
                                         ;following axis
N180 G0 X75 Y=DEPPOS

;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 parts program or a section of a parts program, which is enclosed by `CTABDEF` at the beginning and `CTABEND` at the end.

Within this parts 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 parts 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:

```
...
CTABDEF(Y, X, 1, 0)
X0 Y0
ASPLINE
X=5 Y=10
X10 Y40
...
CTABEND
```

Depending on machine data `MD 20900: 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 specification of an optional memory type, see
"Curve table forms (CTABDEL, ... CTABUNLOCK)".

9.3.3 Curve table forms (CTABDEL, CTABNOMEM, CTABFNO, CTABID, CTABLOCK, CTABUNLOCK)

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.

Programming

Modal language commands with curve tables

`CTABDEL(n, m, memType)`

or

`CTABNOMEM (memType)`

or

`CTABFNO(memType)`

or

`CTABID(n, memType)`

or

`CTABLOCK(n, m, memType) or CTABUNLOCK(n, m, memType)`

or

`CTABDEL(n) or CTABDEL(n, m)`

or

```
CTABLOCK(n) or CTABLOCK(n, m) or CTABLOCK() or
CTABLOCK(, , memType)
```

or

```
CTABUNLOCK(n) or CTABUNLOCK(n, m) or CTABUNLOCK() or
CTABUNLOCK(, , memType)
```

or

`CTABID(n) or CTABID(n, memType) or CTABID(p, memType)`

or

`CTABISLOCK(n)`

or

`CTABEXISTS(n)`

or

`CTABMEMTYP(n)`

or

`CTABPERIOD(n)`

or

`CTABSEGID(n, segType)`

or

```
CTABSEG(memType, segType) or CTABFSEG(memType, segType) or
CTABMSEG(memType, segType)
```

or

`CTABPOLID(n) or CTABMPOL(memType)`

Parameters

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 **defined** curve tables.
CTABFNO(memType)	Number of **possible** tables.
CTABID(n, memType)	Outputs table number entered in memory type as the nth curve table.
CTABLOCK(n, m, memType)	**Enable** deletion and overwrite **lock**.
CTABUNLOCK(n, m, memType)	**Cancel** deletion and overwrite **lock**. 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 **one** curve table.
	Delete **one** curve table range.
CTABDEL(, , memType)	Delete **all** curve tables in the specified memory.
CTABLOCK(n)	**Lock** the delete **and** overwrite:
	Curve table with number n.
CTABLOCK(n, m)	Lock curve tables in the number range n to m.
CTABLOCK()	All existing curve tables.
CTABLOCK(, , memType)	All curve tables **in the** specified memory type.
CTABUNLOCK(n)	**Remove** lock for the delete **and** overwrite: Curve table with number n.
CTABUNLOCK(n, m)	Re-enable curve tables in the number range n to m.
CTABUNLOCK()	All existing curve tables.
CTABUNLOCK(, , memType)	All curve tables **in the** specified memory type.

Uses of other forms for the diagnosis of axis couplings:

CTABID(n, memType)
CTABID(p, memType)
	Outputs table number of the nth/pth curve table **with memory type** memType.

CTABID(n)	Outputs table number of the nth curve table with memory type defined in MD 20905: CTAB_DEFAULT_MEMORY_TYPE **specified** memory type.
CTABISLOCK(n)	Returns the lock status of the curve table **with number n**.
CTABEXISTS(n)	**Checks** curve table with number n.
CTABMEMTYP(n)	**Returns the memory** in which curve table no. n is stored.
CTABPERIOD(n)	Returns the **table periodicity**.
CTABSEG(memType)	Number of **curve segments already used** in the specified memory type.
CTABSEGID(n)	Number of **curve segments** used in **curve table** number n

`CTABFSEG(memType)`	`Number of` **`possible`** `curve segments.`
`CTABMSEG(memType)`	**`Maximum`** `possible number of curve segments.`
`CTABPOLID(n)`	`Number used by` **`curve table`** `number n.` **`Curve table polynomials`**
`CTABSEG(memTyp, segType)`	`Number of type "L" or "P"` **`curve segments used`** `in the memory type.`
`CTABFSEGID(n, segType)`	`Number of type "L" or "P"` **`curve segments`** `used in` **`curve table`** `number n`
`CTABFSEG(memTyp, segType)`	`Number of type "L" or "P"` **`curve segments still possible`** `in the memory type`
`CTABMSEG(memTyp, segType)`	**`Maximum possible`** `number of type "L" or "P"` **`curve segments`** `in the memory type`
`CTABFPOL(memType)`	`Number of` **`curve polynomials still possible`** `in the specified memory type`
`CTABMPOL(memType)`	**`Maximum possible`** `number of` **`curve polynomials`** `in the specified memory type`
`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. It is not possible for there to be tables with the same number in the static and dynamic NC memory.`
`p`	`Entry location (in memType memory area)`
`memType`	`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 with "Processing from external source".

If curve tables are processed from an external source the size of the reload buffer (DRAM) must be selected with `MD 18360: MM_EXT_PROG_BUFFER_SIZE` in such a way the entire curve table definition can be stored simultaneously 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 parts program and power-off if the table has been saved to the static 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.

9.3.4 Behavior at the edges of curve tables (CTABTSV, CTABTSP, CTABMIN, CTABMAX)

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.

Programming

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)
```

or

```
CTABTMAX(n, Faxis)
```

Parameters

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.
CTABMIN ()	Determine the minimum value of a curve table in the complete area or in a defined interval.
CTABMAX ()	Determine the maximum value of a curve table in the complete area or in a defined interval.
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 tables
	Curve table numbers can be freely assigned. They are used exclusively for the unique identification.
degrees	Gradient for incline at start or end of the segment in the 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

```
...
R10=5 R11=20
...
CTABDEF
G1 X=10 Y=20 F1000
R10=R11+5              ;R10=25
X=R10
CTABEND
...                    ;R10=5
```

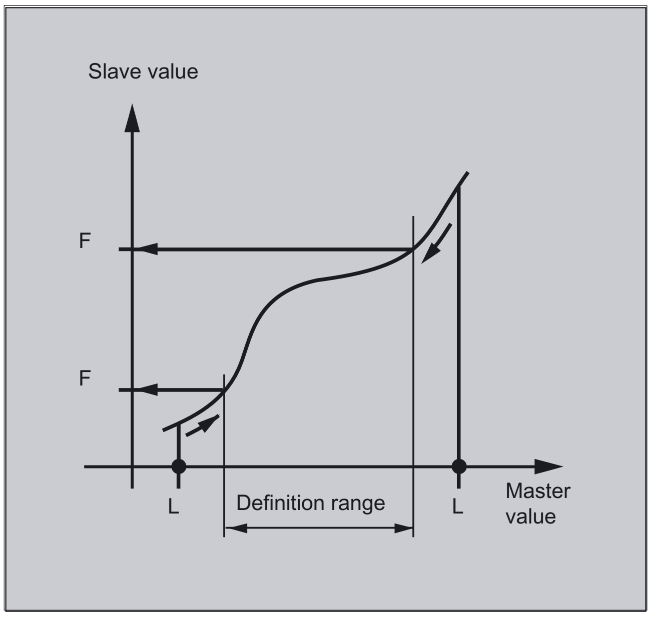
Example of using CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABTMIN, CTABMAX

Determining the minimum and maximum value of a curve table.

```
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)                 ;Begin of the 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
```

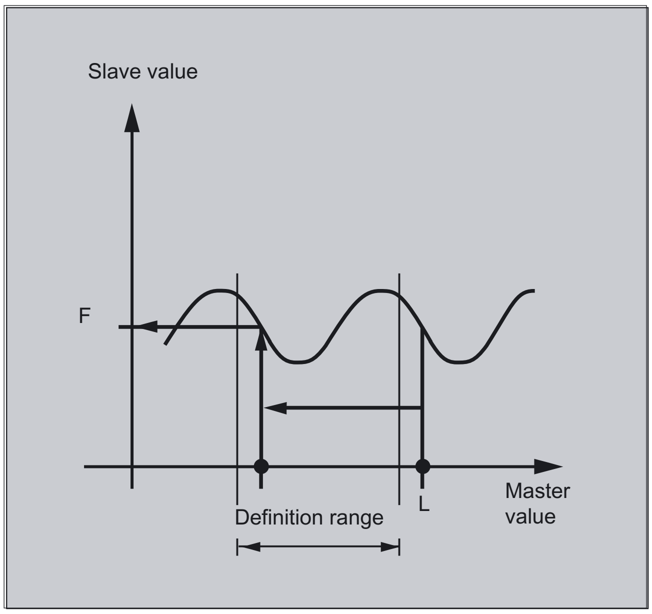
Non-periodic curve table

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

The language commands can be used

- from the parts program or
- directly from synchronous actions.

The internal execution time of the function of

- `CTABINV()P is` **dependent**
- `CTABTSV`, `CTABTEV`, `CTABTSP`, `CTABTEP`,
  (`CTABTMIN`, `CTABTMAX` only if no interval of the master value is specified)
  is **independent**

of the number of table segments.

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".

9.3.5 Access to curve table positions and table segments (CTAB, CTABINV, CTABSSV, CTABSEV)

Function

Reading table positions, CTAB, CTABINV

With CTAB you can read the following value for a master value directly from the parts 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.

Programming

Reading the following value for a master value

```
CTAB(master value, n, degrees, [following axis, leading axis])
```

Reading the master value for a following value

```
CTABINV(following value, approx. master value, n, degrees,
[following axis, leading axis])
```

Reading the start and end values of a table segment

```
CTABSSV(master value, n, degrees, [Faxis]),
CTABSEV(master value, n, degrees, [Faxis])
```

Parameters

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 Axis that is programmed via the curve table.
Laxis	Leading axis Axis that is programmed with the master value.
n, m	Numbers for curve tables. Curve table numbers can be freely assigned. They are used exclusively for the unique identification.
Degrees	Gradient for incline at start or end of segment in curve table
ApproxMasterValue	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

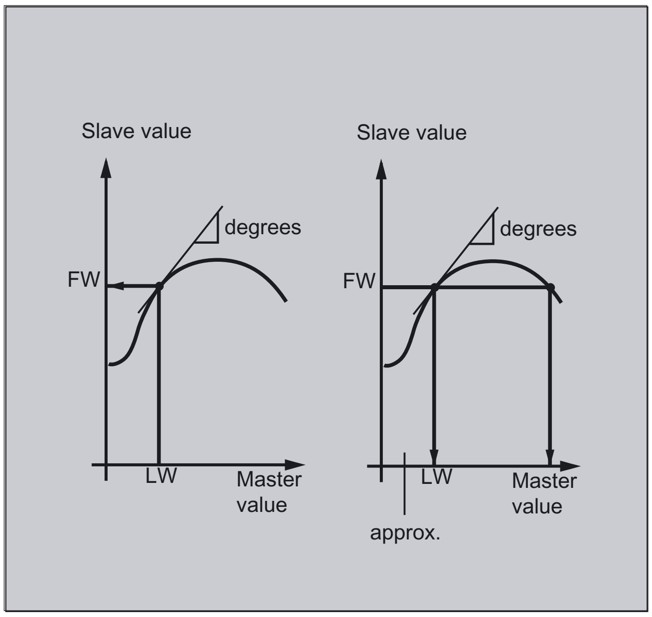
Example of the use of CTABSSV and CTABSEV

Determining the curve segment belonging to master value X = 30.

```
N10 DEF REAL STARTPOS
N20 DEF REAL ENDPOS
N30 DEF REAL GRADIENT
...
N100 CTABDEF(Y,X,1,0)                  ;Begin of the table definition
N110 X0 Y0                             ;Starting position 1st table segment
N120 X20 Y10                           ;End position of the 1st table segment =
N130 X40 Y40                           ;Start position of the
                                       ;2nd table segment ...
N140 X60 Y10
N150 X80 Y0
N160 CTABEND                           ;End of table definition
...
N200 STARTPOS =                        ;Start position Y in segment 2 = 10
 CTABSSV(30.0, 1, GRADIENT)
...                                    ;End position Y in segment 2 = 40
N210 ENDPOS =                          ;Segment 2 belongs to LV X = 30.0.
 CTABSEV(30.0, 1, GRADIENT)
```

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, the 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 parts program or
- directly from synchronized actions

SIMATIC S7. 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.

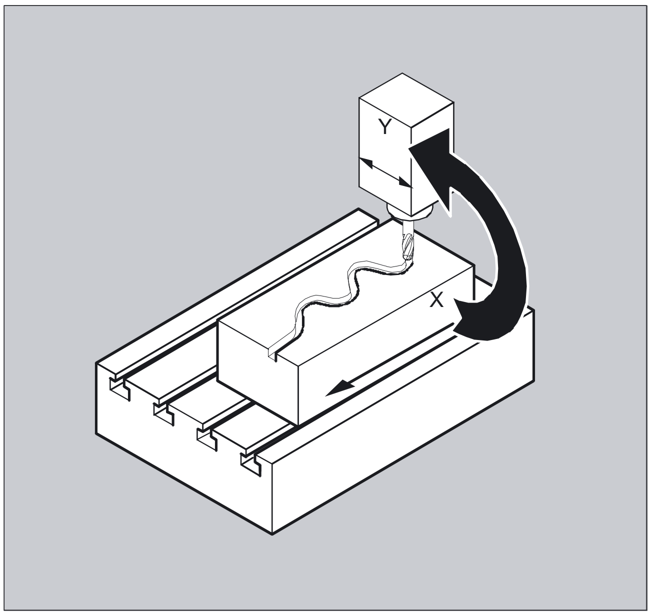
The language commands `CTABSSV` and `CTABSEV` are **not suitable** in the following cases to query programmed segments:

1. Circles or involutes are programmed.
2. Chamfer or rounding with `CHF`, `RND` is active.
3. Corner rounding with `G643` is active.
4. Compressor is, for example, active with `COMPON`, `COMPCURV`, `COMPCAD`.

9.4 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.

Programming

`LEADON(FAxis,LAxis,n)`

or

`LEADOF(FAxis,LAxis)`

or deactivate without details of leading axis

`LEADOF(FAxis)`

The master value coupling can be activated and deactivated both from the parts program and during the movement from synchronized actions, see section "Motion synchronous actions" .

Parameters

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.

Comment

```
; Defines all static synchronized actions.
; **** reset marker
N2 $AC_MARKER[0]=0 $AC_MARKER[1]=0
$AC_MARKER[2]=0 $AC_MARKER[3]=0
$AC_MARKER[4]=0 $AC_MARKER[5]=0
$AC_MARKER[6]=0 $AC_MARKER[7]=0
                                          ; **** E1 0=>1 coupling transfer ON
N10 IDS=1 EVERY ($A_IN[1]==1) AND
($A_IN[16]==1) AND ($AC_MARKER[0]==0)
DO LEADON(X,LV,1) LEADON(YL,LV,2)
LEADON(ZL,LV,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,LV,4) PRESETON(U,0)
$AC_MARKER[5]=1
                                          ; **** E1 0->1 coupling guide head ON
N21 IDS=12 EVERY ($A_IN[1]==1) AND
($A_IN[5]==0) AND ($AC_MARKER[6]==0)
DO LEADON(V,LV,4) PRESETON(V,0)
$AC_MARKER[6]=1
                                          ; **** E1 0->1 coupling greasing ON
N22 IDS=13 EVERY ($A_IN[1]==1) AND
($A_IN[5]==0) AND ($AC_MARKER[7]==0)
DO LEADON(W,LV,4) PRESETON(W,0)
$AC_MARKER[7]=1
```

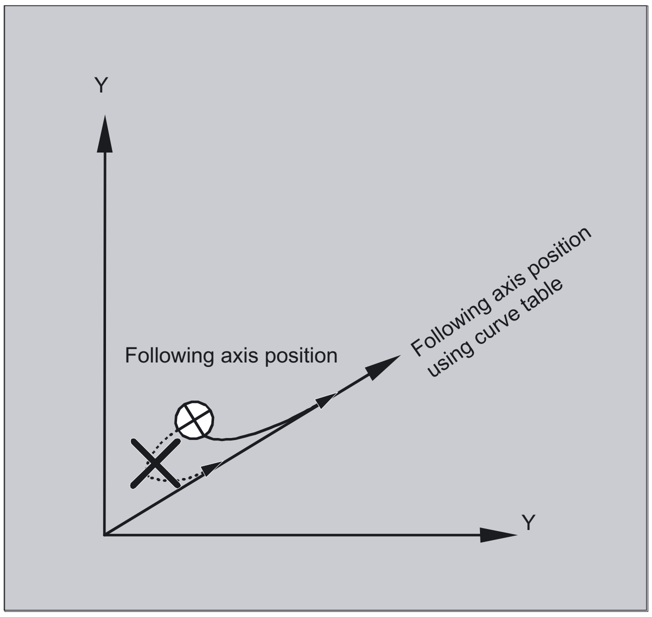
```
                                               ; **** E2 0=>1 coupling OFF
N30 IDS=3 EVERY ($A_IN[2]==1)
DO LEADOF(X,LV) LEADOF(YL,LV)
LEADOF(ZL,LV) LEADOF(U,LV) LEADOF(V,LV)
LEADOF(W,LV) $AC_MARKER[0]=0
$AC_MARKER[1]=0 $AC_MARKER[3]=0
$AC_MARKER[4]=0 $AC_MARKER[5]=0
$AC_MARKER[6]=0 $AC_MARKER[7]=0
....
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).

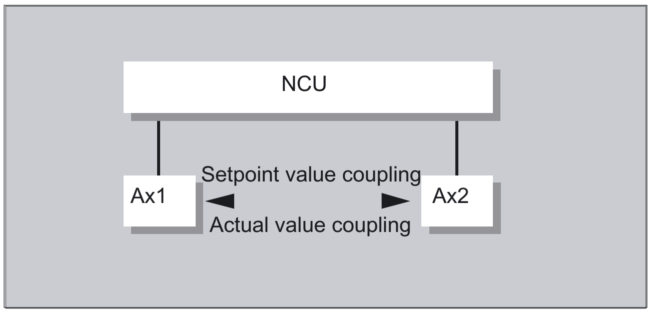


No synchronism

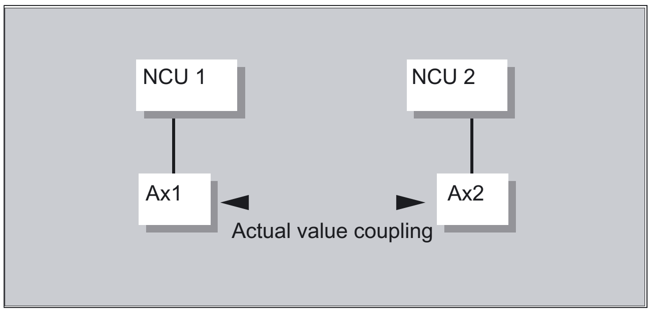
If the following axis position calculated moves away from the current following axis position when the master value coupling is activated, it is not possible to establish synchronization.

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

9.5 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.

Programming

`F… FNORM`

or

`F… FLIN`

or

`F… FCUB`

or

`F=FPO(…,…,…)`

Parameters

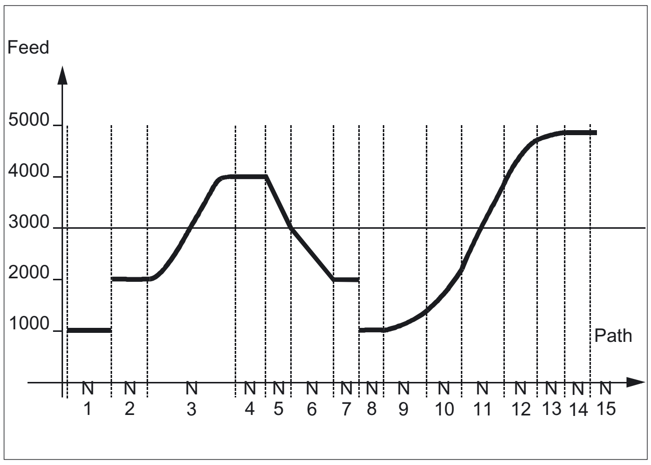
`FNORM`	Basic setting. The feed value is specified as a function of the traverse path of the block and is then valid as a modal value.
`FLIN`	Path velocity profile **linear:** The feed value is approached linearly via the traverse path from the current value at the block beginning to the block end and is then valid as a modal value. The response can be combined with G93 and G94.
`FCUB`	Path velocity profile **cubic:** The blockwise programmed F values (relative to the end of the block) are connected by a spline. The spline begins and ends tangentially with the previous and following defined feedrate and takes effect with G93 and G94. If the F address is missing from a block, the last F value to be programmed is used.
`F=FPO…`	**Polynomial** path velocity profile: The F address defines the feed characteristic via a polynomial from the current value to the block end. The end value is valid thereafter as a modal value.

Feed optimization on curved path sections

Feed polynomial `F=FPO` and feed spline `FCUB` should always be traversed at constant cutting rate `CFC`, thereby allowing a jerk-free setpoint feed profile to be generated. This enables creation of a continuous acceleration setpoint feed profile.

Example of various feed profiles

This example shows you the programming and graphic representation of various feed profiles.

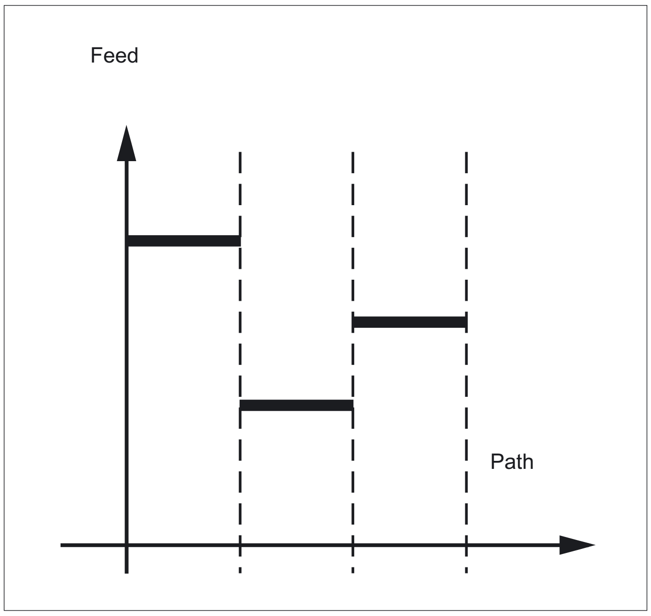


```
N1 F1000 FNORM G1 X8 G91 G64     ;Constant feed profile, incremental dimensioning
N2 F2000 X7                      ;Step change in setpoint velocity
N3 F=FPO(4000, 6000, -4000)      ;Feed profile via polynomial with feed 4000 at
                                 ;block end
N4 X6                            ;Polynomial feed 4000 applies as modal value
N5 F3000 FLIN X5                 ;Linear feed profile
N6 F2000 X8                      ;Linear feed profile
N7 X5                            ;Linear feed applies as modal value
N8 F1000 FNORM X5                ;Constant feed profile with abrupt change in
                                 ;acceleration rate
N9 F1400 FCUB X8                 ;All subsequent, non-modally programmed F values
                                 ;are connected via splines
N10 F2200 X6
N11 F3900 X7
N12 F4600 X7
N13 F4900 X5                     ;Deactivate 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.

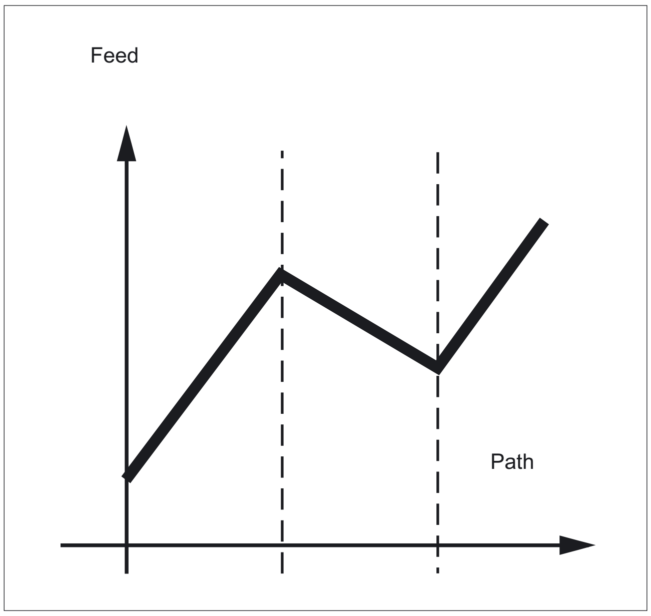


FLIN

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
```

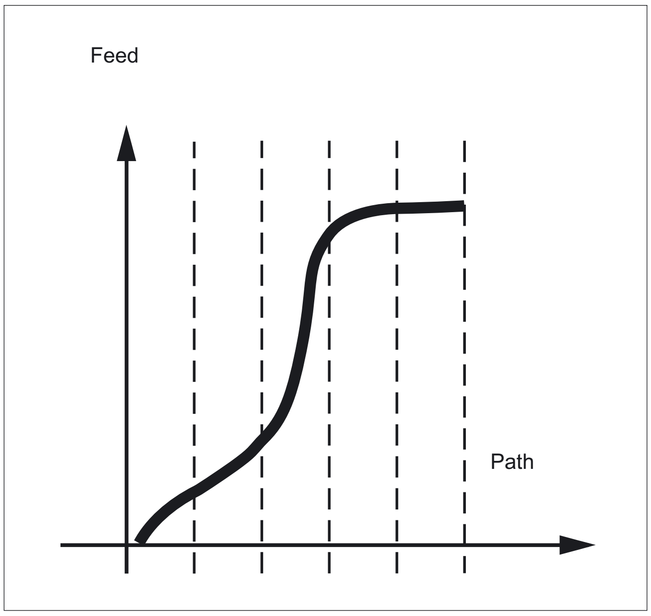


FCUB

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.

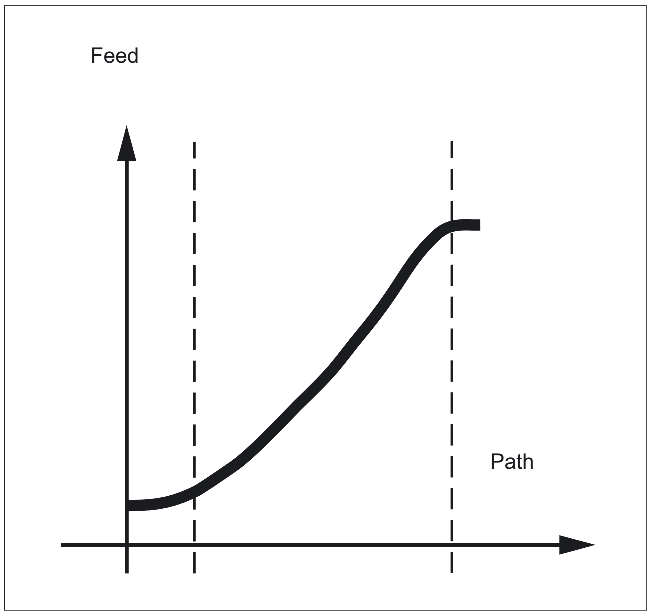
Example:

```
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_COMPESS_VELO_TOL`.

F=FPO(…,…,…)

These blocks are not compressed.

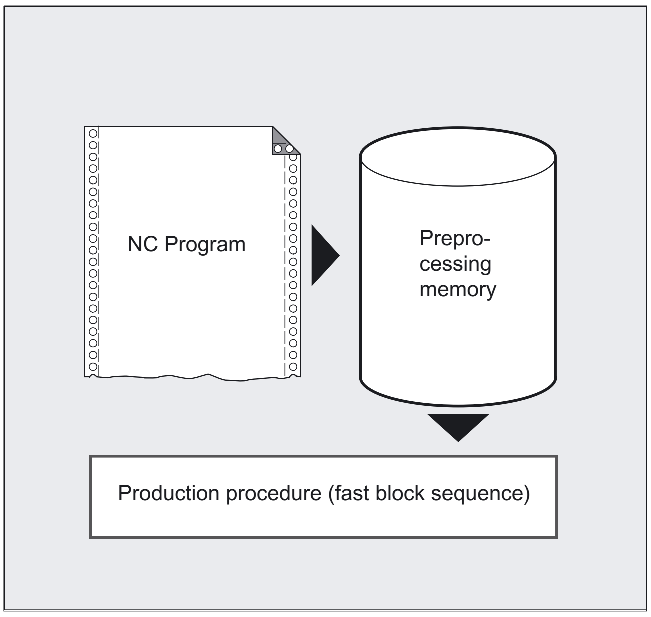
9.6 Program run with preprocessing memory (STARTFIFO, STOPFIFO, STOPRE)

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.



Programming

`STARTFIFO`

or

`STOPFIFO`

or

`STOPRE`

Parameters

`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 of marking a processing section

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 of stopping 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:

```
R10 = $AA_IM[X]        ;Read actual value of X axis
```

Caution

When a tool offset or spline interpolations are active, you should not program the `STOPRE` command as this will lead to interruption in contiguous block sequences.

9.7 Conditionally interruptible program sections (DELAYFSTON, DELAYFSTOF)

Function

Conditionally interruptible parts program sections are called stop delay sections. No **stopping** should occur and the **feed** should not be changed within certain program sections. This essentially means that short program sections used, for example, to machine a thread, should be protected from stop events. Stops do not take effect until the program section has been completed.

Programming

`N... DELAYFSTON`
`N... DELAYFSTOF`
	The commands are programmed separately in a parts program line. **DELAYF**eed **ST**op **ON**/**OF**

Both commands are only permitted in parts programs but not in synchronous actions.

Parameters

`DELAYFSTON`	`Define the beginning of a section in which "soft" stops are delayed until the end of the stop delay section is reached.`
`DELAYFSTOF`	`Define end of a stop delay section.`

Note

For machine data `MD 11550: STOP_MODE_MASK` Bit 0 = 0 (default) a stop delay section is defined implicitly if `G331/G332` is active and a path motion or `G4` is programmed. See note below.

Example of stop events

In a stop delay section, changes in the **feedrate or feed disable** are ignored. They do not take effect until 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
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.

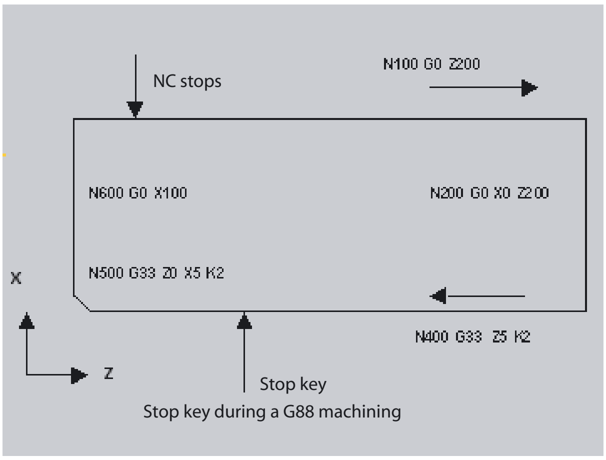
For a list of other responses to stop events, see
/FB1/ Function Manual Basic Functions; Mode Group, Channel, Program Operation, (K1), "Influencing and Impacting on Stop Events" section.

Example of the nesting of stop delay sections in two program levels

```
N10010 DELAYFSTON()            ;blocks with N10xxx program level 1
N10020 R1 = R1 + 1
N10030 G4 F1                   ;stop delay section starts.
...
N10040 subroutine2
...
...                            ;interpretation of subroutine 2
N20010 DELAYFSTON()            ;no effect, restart, 2nd level
...
N20020 DELAYFSTOF()            ;no effect, end in other level
N20030 RET

N10050 DELAYFSTOF()            ;end of stop delay section in same level
...
N10060 R2 = R2 + 2
N10070 G4 F1                   ;stop delay section ends.
                               ;stops now have direct effect
```

Program segment example

The following program block is repeated in a loop:



As shown in the illustration, 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`.

```
...
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
```

Details on SERUPRO type block searches and feeds in conjunction with `G331/G332` feed for tapping without compensating chuck, see

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

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.

9.8 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 parts program sections that the NCK cannot yet re-enter. When a program is aborted the NCK remembers the last processed block that can be traced from the HMI user interface.

Programming

`N... IPTRLOCK`
or
`N... IPTRUNLOCK`
	The commands are programmed separately in a parts program line and permit a programmable interruption pointer

Parameters

IPTRLOCK	Start of untraceable program section
IPTRUNLOCK	End of untraceable program section

Both commands are only permitted in parts programs, but **not** in synchronous actions.

Example

Nesting of untraceable program sections in two program levels with implicit `IPTRUNLOCK`. Implicit `IPTRUNLOCK` in subroutine 1 ends the untraceable section.

```
N10010 IPTRLOCK()
N10020 R1 = R1 + 1
N10030 G4 F1                                ;hold block, the untraceable
...                                         ;program section starts
N10040 subroutine2
...                                         ;interpretation of subroutine 2
N20010 IPTRLOCK ()                          ;no effect, restart
...
N20020 IPTRUNLOCK ()                        ;no effect, end in other level
N20030 RET
...
N10060 R2 = R2 + 2
N10070 RET                                  ;End of untraceable
                                            ;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`

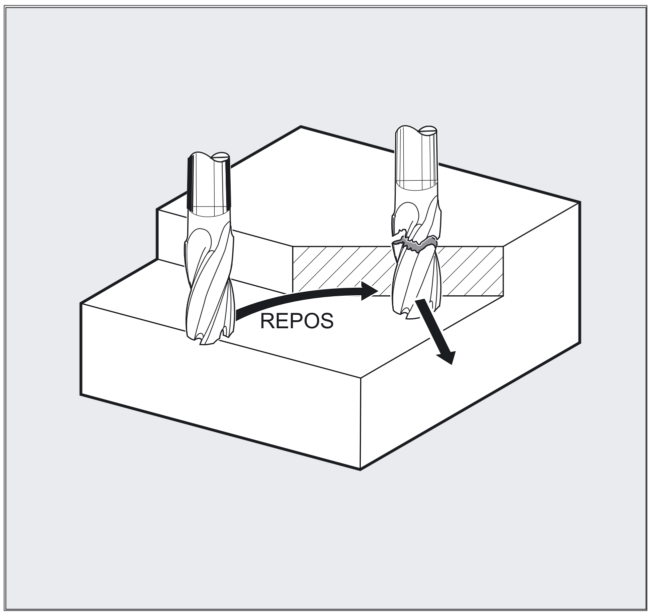
are used to activate the automatic interrupt pointer. If the programmed interrupt pointer and interrupt pointer activated with automatic interrupt pointers overlap, the largest possible untraceable section will be generated.

9.9 Repositioning at contour (REPOSA/L, REPOSQ/H, RMI, RMN, RMB, RME)

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 subroutine 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.

Programming

```
REPOSA RMI DISPR=… or REPOSA RMB or REPOSA RME or REPOSA RMN
```

or

```
REPOSL RMI DISPR=… or REPOSL RMB or REPOSL RME or REPOSL RMN
```

or

```
REPOSQ RMI DISPR=…DISR=… or REPOSQ RMBDISR=… or REPOSQ RME
DISR=… or REPOSQA DISR=…
```

or

```
REPOSH RMI DISPR=… DISR=…or REPOSH RMB DISR=… or REPOSH RME
DISR=… or
REPOSHA DISR=…
```

Parameters

Approach path

REPOSA	Approach along line on all axes
REPOSL	Approach along line
REPOSQ DISR=…	Approach along quadrant with radius DISR
REPOSQA DISR=…	Approach on all axes along quadrant with radius DISR
REPOSH 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

Example of approaching along a straight line, REPOSA, REPOSL

The tool approaches the repositioning point along a straight line.

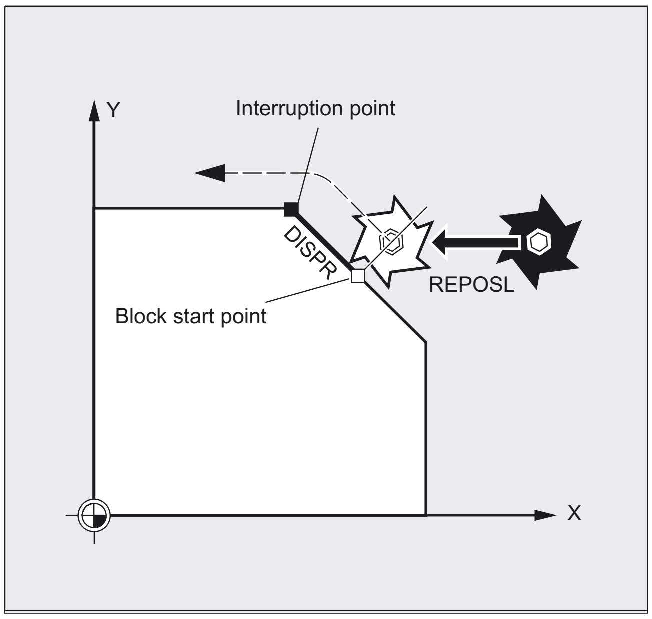
All axes are automatically traversed with command REPOSA. With REPOSL you can specify which axes are to be moved.

Example:

```
REPOSL RMI DISPR=6 F400
```

or

```
REPOSA RMI DISPR=6 F400
```

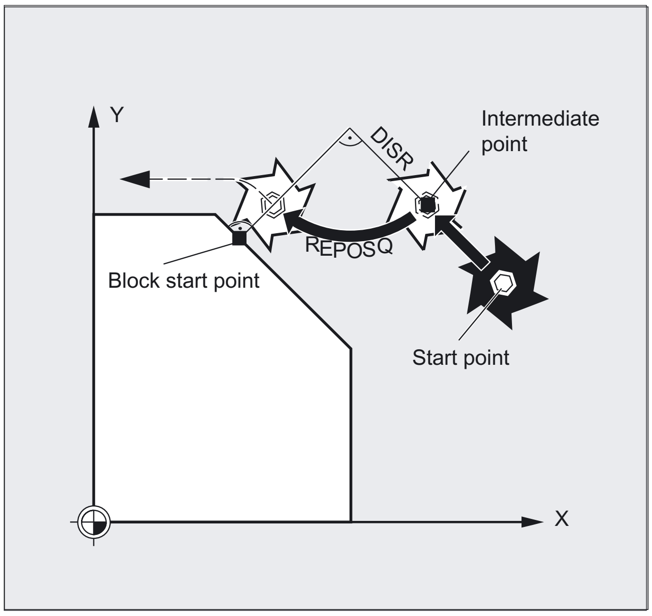


Example of approaching in the quadrant, REPOSQ, REPOSQA

The tool approaches the repositioning point along a quadrant with a radius of `DISR=`.... The control system automatically calculates the intermediate point between the start and repositioning points.

Example:

```
REPOSQ RMI DISR=10 F400
```

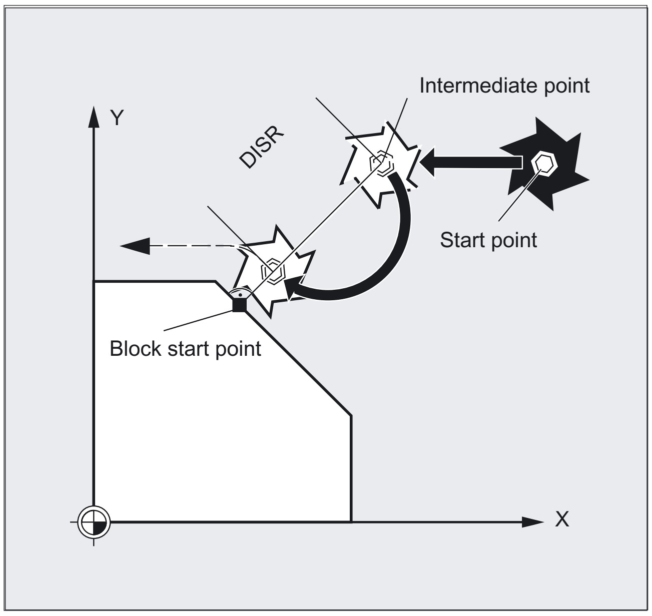


Example of approaching tool along the semi-circle, REPOSH, REPOSHA

The tool approaches the repositioning point along a semi-circle with a diameter of `DISR=...`. The control system automatically calculates the intermediate point between the start and repositioning points.

Example:

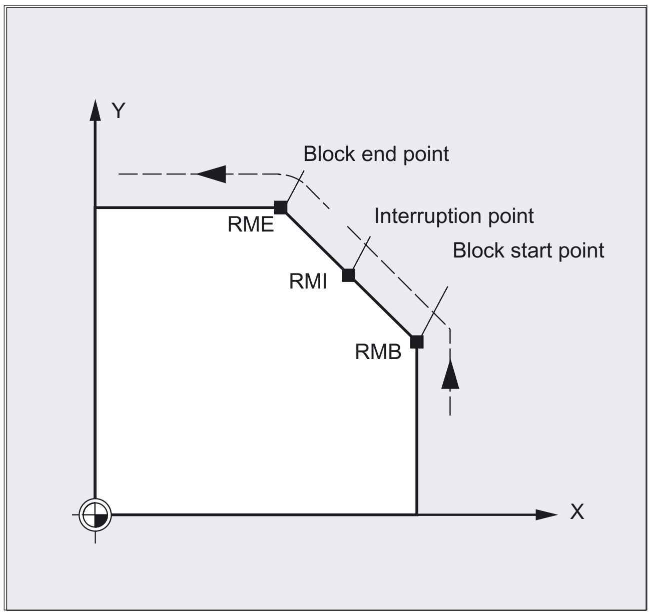
```
REPOSH RMI DISR=20 F400
```



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`).

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.

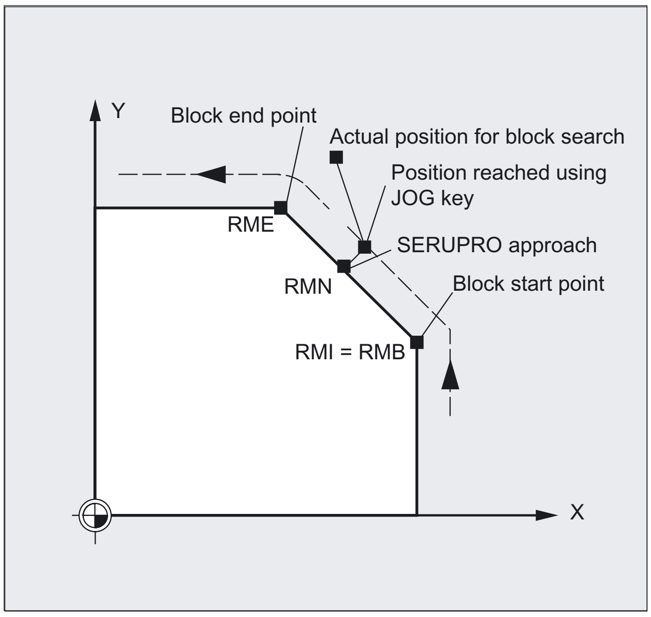
Sample application:

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.



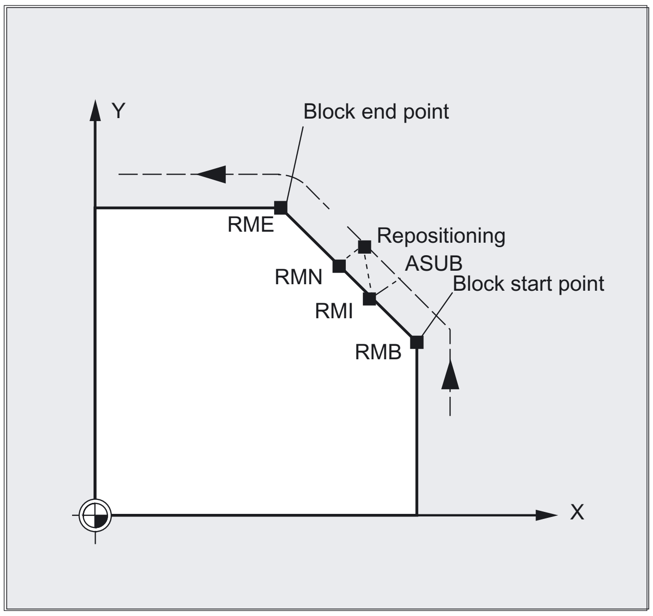
Note

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

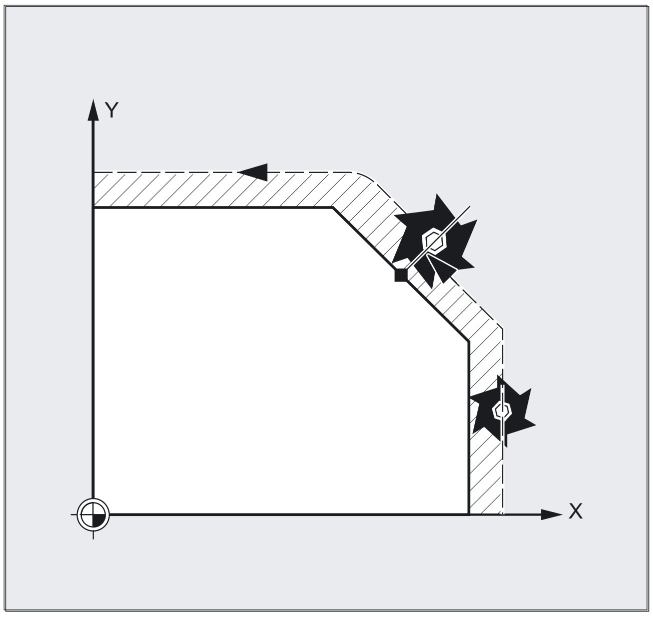
4 `RMN`: Approaching to next path point of the interrupted block

Approaching with a new tool

The following applies if you have stopped the program run due to tool breakage:

When the new D number is programmed, the machining program is continued with modified tool offset values at the repositioning point.

Where tool offset values have been modified, it may not be possible to reapproach the interruption point. In such cases, the point closest to the interruption point on the new contour is approached (possibly modified by DISPR).



Approach contour

The motion with which the tool is repositioned on the contour can be programmed. Enter zero for the addresses of the axes to be traversed.

The `REPOSA`, `REPOSQA` and `REPOSHA` commands automatically reposition all axes. Individual axis names need not be specified.

When the commands `REPOSL`, `REPOSQ` and `REPOSH` are programmed, all geometry axes are traversed automatically, i.e. they 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:

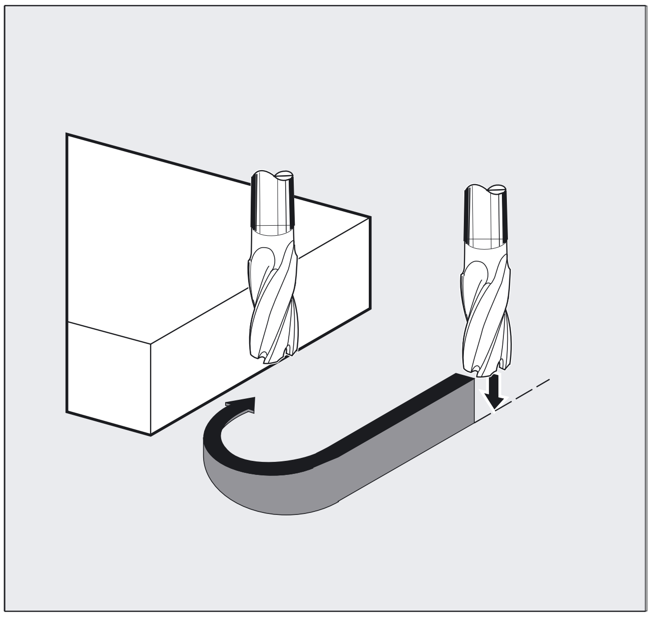
The circle is traversed in the specified working planes `G17` to `G19`.

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 `REPOSL`:

- You have not specified a value for `DISR`.
- No defined approach direction is available (program interruption in a block without travel information).
- With an approach direction that is perpendicular to the current working plane.



Motion synchronous actions 10

10.1 Structure, basic information

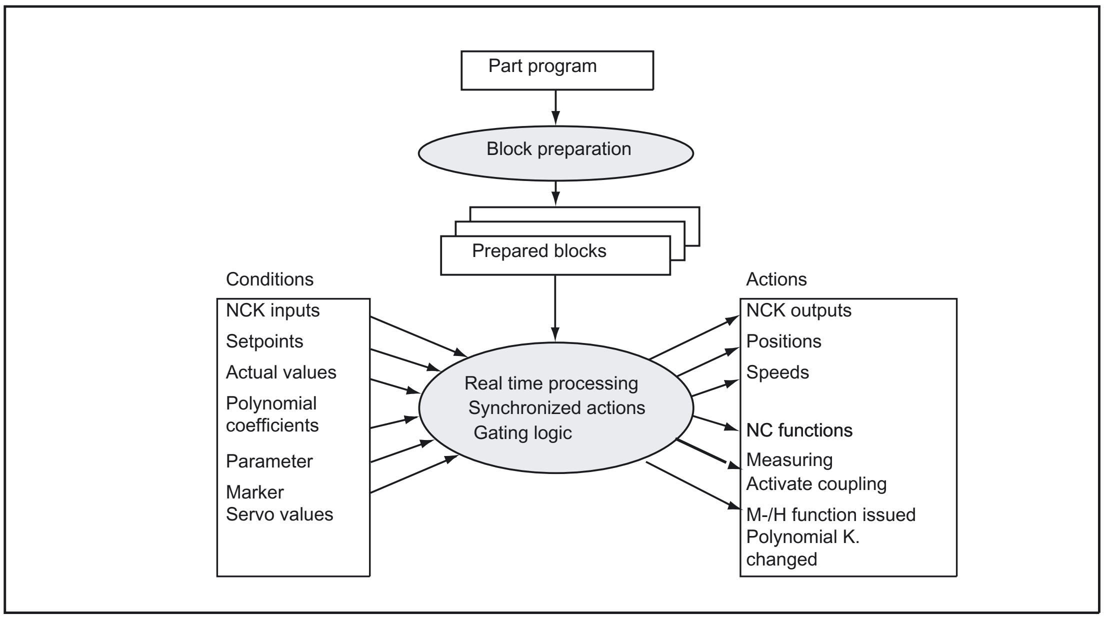
Function

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:



- Optimization of runtime-critical applications (e.g. tool changing)
- Fast response to an external event
- Programming AC controls

- Setting up safety functions
- ....

Programming

```
DO action1 action2 …
KEYWORD condition DO action1 action2 …
ID=n KEYWORD condition DO action1 action2 …
IDS=n KEYWORD condition DO action1 action2 …
```

Command elements

Identification number ID/IDS:

`ID=n`	**Modal** synchronized actions in automatic mode, **local to program**; n = 1... 255
`IDS=n`	**Modal** synchronized actions in each mode, **static**; n = 1... 255
Without ID/IDS	**Non-modal** synchronized actions in automatic mode

Keyword:

No keyword	Execution of the action is not subject to any condition. Cyclical execution in the IPO cycle.
WHEN, WHENEVER, FROM, EVERY,	Querying frequency of the action to be started

Condition:

Main run variable.
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.

DO:

Initiation of the action

Coordination of synchronized actions/technology cycles:

`CANCEL[n]`	Cancel synchronized actions
`LOCK[n]`	Disable synchronized actions
`UNLOCK[n]`	Unlock synchronized actions
`RESET`	Reset technology cycle

Example

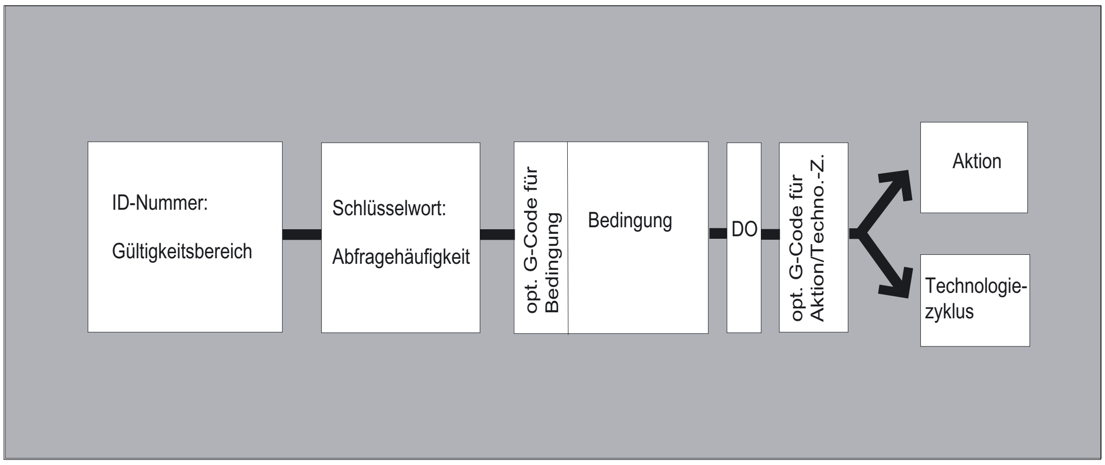
```
WHEN $AA_IW[Q1]>5 DO M172 H510     ;If the actual value of axis Q1 exceeds 5 mm,
                                   auxiliary functions M172 and H510 are output to
                                   the PLC interface.
```

10.1.1 Programming and command elements

Function

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 consist of up to five command elements each with a different task:



Programming

```
ID=n keyword condition DO action 1 action 2 ...
```

Command elements

Identification number ID/IDS	Scope of the **modal** synchronized actions in automatic mode or in each operating mode.
Keyword	Querying frequency none, WHEN, WHENEVER, FROM, EVERY,
Condition	Gating logic for main run variables, the conditions are checked in the interpolation cycle.
DO	Perform when the action or the technology cycle is satisfied.
Action	Action started if the condition is fulfilled, e.g. assign variable.
Technology cycle	A program is called as action if the condition is fulfilled.

Example

ID=1	WHENEVER	$A_IN[1]==1	DO	$A_OUT[1] = 1
Synchronized action no. 1:	whenever	input 1 is set	then	set output 1

10.1.2 Validity range: Identification number ID

Function

The scope of validity of a synchronized action is defined by the identification number:

- no modal ID: Non-modal synchronized actions in automatic mode
- ID=n modal synchronized actions in automatic mode at end of program
- IDS=n modal synchronized actions in each mode static, also beyond end of program

Application

- AC loops in JOG mode
- Logic operations for Safety Integrated
- Monitoring functions, responses to machine states in all modes

Sequence of execution

Synchronized actions that apply modally or statically are executed in the order of their ID(S) numbers (in the interpolation cycle).

Non-modal synchronized actions (without ID number) are executed in the programmed sequence after execution of the modal synchronized actions.

Machine manufacturer

Modal synchronized actions can be protected from modifications or deletions by machine data settings.

Identification number ID

- no modal ID

  The synchronized action is only active in automatic mode. It applies only to the next executable block (block with motion statement or other machine action), is **non-modal**.

  Example:

```
WHEN $A_IN[3]==TRUE DO $A_OUTA[4]=10
G1 X20                                            ;Executable block
```

- ID=n; n=1..255

  The synchronized action applies **modally** in the following blocks and can be deactivated by CANCEL(n) or can be overwritten by programming a new synchronized action with the same ID. The synchronized actions active in the M30 block delay the program end. ID synchronized actions **only** apply in **automatic mode**.

  Example:

```
ID=2 EVERY $A_IN[1]==1 DO POS[X]=0
```

- IDS=n; n=1..255

  The **static** synchronized actions act **modally** in **all modes.** They even remain active beyond the end of the program and can be activated directly after Power On using an ASUB.

  In this way, actions can be activated that are executed regardless of the mode selected in the NC.

  Example:

```
IDS=1 EVERY $A_IN[1]==1 DO POS[X]=100
```

10.1.3 Cyclic checking of the condition

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.

Example

No keyword

`DO $A_OUTA[1]=$AA_IN[X]` ;output the actual value at the analog output

EVERY

`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.

WHENEVER

```
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 logic-gated comparisons
```

Condition

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 parts 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

10.1.4 Actions

Function

In synchronized actions, you can program one or more actions. All actions programmed in a block are active in the same interpolation cycle.

Command elements

`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 of a synchronized action with two actions

```
WHEN $AA_IM[Y] >= 35.7 DO M135              ;If the condition is fulfilled,
$AC_PARAM=50                                ;M135 is output to 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.

```
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

```
DO $AC_PARAM[3] = COS($AC_PARAM[1])
```

* Real-time expressions

```
ID=1 WHENEVER ($AA_IM[Y]>30) AND ($AA_IM[Y]<40)     ;Selection of 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 of the result
                                                    ;of a comparison
```

* Main run variable indexed

```
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 General information on 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.

Syntax of system variables

System variable names always begin with a "$" sign.

Preprocessing variables:

- `$M...` , machine data
- `$S...` , setting data, protection zones
- `$T...` , tool management data
- `$P...` , programmed values, preprocessing data
- `$C...` , cycle variables of the ISO wrapper cycles
- `$O...` , options data
- `R...` , R parameter

Main run variables:

- `$A...` , current main run data
- `$V...` , servo data
- `$R...` , R parameter

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 $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`

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*	Yes1)	–	–	–	–
INT	Yes	Yes	Yes1)	–	–	–	–
BOOL	Yes	Yes	Yes	–	–	–	–

Explanations

*	At type conversion from REAL to INT, fractional values that are >=0.5 are rounded up, others are rounded down (cf. ROUND function). An alarm is output if values are exceeded.
1)	Value <> 0 is equivalent to TRUE; value == 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 for synchronous actions

Function

In addition to the predefined 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 variable protocol.

Configurable parameter ranges

Machine manufacturer

Machine data can be used to add additional channel-specific parameter areas of AXIS, CHAR and STRING data types to the individual GUD modules for the REAL, INT and BOOL data types. These areas can be read and written by the parts program and using 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 variable

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 **REAL**	of data type **INT**	of data type **BOOL**	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 **AXIS**	of data type **CHAR**	of data type **STRING**	in **module**
SYG_AS[ ]	SYG_CS[ ]	SYG_SS[ ]	SGUD module
SYG_AM[ ]	SYG_CM[ ]	SYG_SM[ ]	MGUD module
SYG_AU[ ]	SYG_CU[ ]	SYG_SU[ ]	UGUD module
SYG_A4[ ]	SYG_C4[ ]	SYG_S4[ ]	GUD4 module
SYG_A5[ ]	SYG_C5[ ]	SYG_S5[ ]	GUD5 module
SYG_A6[ ]	SYG_C6[ ]	SYG_S6[ ]	GUD6 module
SYG_A7[ ]	SYG_C7[ ]	SYG_S7[ ]	GUD7 module
SYG_A8[ ]	SYG_C8[ ]	SYG_S8[ ]	GUD8 module
SYG_A9[ ]	SYG_C9[ ]	SYG_S9[ ]	GUD9 module

Note

STRING type variables in synchronized actions have a fixed length of 32 characters.

- Array size corresponding to <value> of 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 can also be undertaken via HMI in the Services operating area in the "Define and activate user data (GUD)" user interface.

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 also 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/Description of Functions; Synchronized Actions

Programming

```
PROC UP(AXIS PAR1=NO_AXIS, AXIS PAR2=NO_AXIS)
IF PAR1 <>NO_AXIS…
```

Subroutine definition

PROC	Subroutine definition
SR	Subroutine name for recognition
PARn	Parameter n
NO_AXIS	Initialization of formula parameter with default axis identifier

Example of the definition of an axis variable in the main program

```
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

```
WHEN ... DO $AC_MARKER[0] = 2
WHEN ... DO $AC_MARKER[0] = 3
WHENEVER $AC_MARKER[0] == 3 DO $AC_OVR=50
```

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.

`$AC_PARAM[n]`	Arithmetic variable for movement synchronized actions (REAL)
`$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 0-n

Example of synchronized action parameter $AC_PARAM[n]

```
$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 parts program and synchronized actions.

Programming

Programming in parts program:

`REAL R[n]`

or

`REAL Rn`

Programming in synchronized actions:

`REAL $R[n]`

or

`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

```
WHEN $AA_IM[X]>=40.5 DO $R10=$AA_MM[Y]      ;Use of R10 in synchronized actions
G01 X500 Y70 F1000
STOPRE                                      ;Preprocessing stop
IF R10>20                                   ;Evaluation of the arithmetic variable
```

```
WHEN $AA_IM[X]>=40.5 DO $R10=$AA_MM[Y]      ;Read access to the R parameter 10
WHEN $AA_IM[X]>=6.7 DO                      ;Read access to the R parameter
$R[$AC_MARKER[1]]=30.6                      ;whose number is contained in marker 1
```

```
SYG_AS[2]=X
SYG_IS[1]=1
WHEN $AA_IM[SGY_AS[2]]>10 DO $R3=$AA_EG_DENOM[SYG_AS[1]], SYG_AS[2]]
WHEN $AA_IM[SGY_AS[2]]>12 DO $AA_SCTRACE[SYG_AS[2]]=1

SYG_AS[1]=X
SYG_IS[0]=1
WHEN $AA_IM[SGY_AS[1]]>10 DO $R3=$$MA_POSCTRL_GAIN[SYG_IS[0]],SYG_AS[1]]
WHEN $AA_IM[SGY_AS[1]]>10 DO $R3=$$MA_POSCTRL_GAIN[SYG_AS[1]]
WHEN $AA_IM[SGY_AS[1]]>15 DO $$MA_POSCTRL_GAIN[SYG_AS[0]], SYG_AS[1]]=$R3
```

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 parts 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

```
ID=1 WHEN $AA_IW[X]>10 DO $$SN_SW_CAM_PLUS_POS_TAB_1[0]=20
                                          $$SN_SW_CAM_MINUS_POS_TAB_1[0]=30
; Alteration of switching positions of software cams. Note: The switching positions
must be changed two to three interpolation cycles before they reach their position.
```

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

`$AC_TIMER[n]`	`Channel-specific timer of data type REAL`
`s`	`Unit in seconds`
`n`	`Index of timer variable`

Setting timers

A timer variable is incremented via value assignment
`$AC_TIMER[n]=value`
n: Number of timer variable
Value: Start value (normally 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

```
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 variable $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 programmed in machine data
`MD 28260: NUM_AC_FIFO`.

The number of values that can be entered in a FIFO variable is defined via machine data `MD 28264: LEN_AC_FIFO`. All FIFO variables are equal in length.

The sum of all FIFO elements is only formed if bit 0 is set in `MD 28266 MODE_AC_FIFO`.

Indices 0 to 5 have a special significance:

n=0:While writing: New value is stored in the FIFO
While reading: the oldest element will be read and removed from the FIFO

n=1:Access to oldest stored element

n=2:Access to latest stored element

n=3:Sum of all FIFO elements

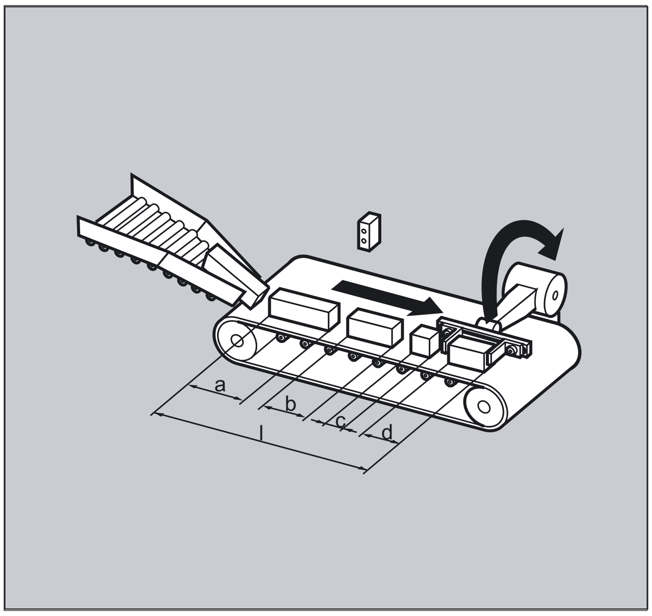
n=4:Number of elements available in FIFO.
Every element in the FIFO can be read and write-accessed. FIFO variables are reset by resetting the number of elements, e.g. for the first FIFO variable: `$AC_FIFO1[4]=0`

n=5:Current write index relative to beginning of FIFO

n=6 to 6+nmax:Access to nth FIFO element:

Example of the circulating stack

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.



```
DEF REAL INTV=2.5                               ;Constant distance between products
                                                ;placed on the belt.
DEF REAL TOTAL=270                              ;Distance between length measurement
                                                ;and removal position.
EVERY $A_IN[1]==1 DO $AC_FIFO1[4]=0             ;Reset FIFO at beginning of process.
EVERY $A_IN[2]==1 DO $AC_TIMER[0]=0             ;If a product interrupts the light
                                                ;barrier, start timing.
EVERY $A_IN[2]==0 DO $AC_FIFO1[0]=$AC_TIMER[0]*$AA_VACTM[B]
;If the light barrier is free, calculate and store in the FIFO the
 product length from the time measured and the velocity of transport.
EVERY $AC_FIFO1[3]+$AC_FIFO1[4]*ZWI>=TOTAL DO POS[Y]=-30
 $R1=$AC_FIFO1[0]
;As soon as the sum of all product lengths and intervals between products is greater
 than or equal to the length between the placement and the removal position, remove
 the product from the conveyor belt at the removal position, read the product length
 out of the FIFO.
```

10.3.11 Information about the block types in the interpolator

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 variable

$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)

$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

Note

`$AC_BLOCKTYPEINFO` always contains the value for the block type in the thousands digit (T) in case there is an intermediate block. The thousands digit is not used in `$AC_BLOCKTYPE` not equal to 0.

T: Thousands digit

H: Hundreds digit

Z: Tens digit

E: Units digit

$AC_SPLITBLOCK	
Value:	Meaning:
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 of counting corner rounding blocks

```
$AC_MARKER[0]=0
$AC_MARKER[1]=0
$AC_MARKER[2]=0
...
;Definition of synchronized actions with which
;corner rounding blocks are counted
;All corner rounding blocks count in $AC_MARKER[0]
ID = 1 WHENEVER ($AC_TIMEC ==0) AND ($AC_BLOCKTYPE==5) DO _
 $AC_MARKER[0]= $AC_MARKER[0] + 1
...
;All corner rounding blocks generated with G641 count in $AC_MARKER[1]
ID = 2 WHENEVER ($AC_TIMEC ==0) AND ($AC_BLOCKTYPEINFO==5001) DO _
 $AC_MARKER[1]= $AC_MARKER[1] + 1
...
;All corner rounding blocks generated with G642 count in $AC_MARKER[2]
ID = 3 WHENEVER ($AC_TIMEC ==0) AND ($AC_BLOCKTYPEINFO==5002) DO _
 $AC_MARKER[2]= $AC_MARKER[2] + 1
...
```

10.4 Actions in synchronized actions

10.4.1 Overview

General information

Actions in synchronized actions consist of value assignments, function or parameter calls, keywords or technology cycles.

Complex executions are possible using operators.

Synchronized actions have been continually updated in several software versions for expressions, usable main run variable and complex conditions in synchronized actions.

The following applications are possible:

- Calculations of complex expressions in the IPO cycle
- Axis movements and spindle controls
- Change and evaluate online setting data from synchronized actions, such as positions, and output times of software cams to PLC or NC peripherals
- Output of auxiliary functions to PLC
- Setting up 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

Applications for motion-synchronous actions	
Synchronized action	Description
DO $V…= DO $A...=	assign (servo values) assign variable (main run variable)
DO $AC…[n]= DO $AC_MARKER[n]= DO $AC_PARAM[n]=	Special main run variable Read or write synchronized action marker Read or write synchronized action parameter
DO $R[n]=	Read or write arithmetic variable
DO $MD...= DO $$SD...=	Read MD value at interpolation time Write SD value in main run
DO $AC_TIMER[n]=Start value	Timers
DO $AC_FIFO1[n] …FIFO10[n]=	FIFO variables
DO $AC_BLOCKTYPE= DO $AC_BLOCKTYPEINFO= DO $AC_SPLITBLOCK=	Interpret the current block (main run variable)
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) DO SYNFCT(polynomial, output, input)	Definition of polynomials Activation of synchronized functions: adaptive control
DO FTOC	Online tool offset
DO G70/G71/G700/G710	Specify measuring system for positioning tasks Dimensions in inches or metric
DO POS[Axis]= / DO MOV[Axis]= DO SPOS[Spindle]=	Start/position/stop command axes Start/position/stop command spindles
DO MOV[Axis]=value	Start/position infinite movements of a command axis

Applications for motion-synchronous actions	
DO POS[Axis]= FA [Axis]=	Axial feed FA
DO $A_WORAREA_PLUS_ENABLE]=	Working area limitation

ID=1 ... DO POS[Axis]= FA [Axis]=
ID=2 ... DO POS[Axis]=
$AA_IM[Axis] FA [Axis]=
	Position from synchronized actions

DO PRESETON(axis, value)	Set actual value (preset from synchronized actions)

ID=1 EVERY $A_IN[1]=1 DO M3 S….
ID=2 EVERY $A_IN[2]=1 DO SPOS=
	Start/position/stop spindles

DO TRAILON(FA, LA, coupling factor) DO LEADON(FA, LA, NRCTAB, OVW)	activate trailing activate leading value coupling
DO MEAWA(axis)= DO MEAC(axis)=	Activate axial measurement Activate continuous measurement
DO [array n, m]=SET(value, value, ...) DO [array n, m]=REP(value, value, ...)	Initialization of array variables with lists of values Initialization of array variables with the same values
DO SETM(flag no.) DO CLEARM(flag no.)	Set wait markers Delete wait markers
DO SETAL(alarm no.)	Set cycle alarm (additional safety function)
DO FXS[axis]= DO FXST[axis]= DO FXSW[axis]= DO FOCON[axis]= DO FOCOF[axis]=	Select travel to fixed stop Change clamping moment Change monitoring window Activate travel with limited moment/force (modal) deactivate FOC (synchronized action acts block-related)
ID=2 EVEREY $AC_BLOCKTYPE==0 DO $R1 = $AC_TANEB	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 by the PLC of the path override specified by 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=	Current IPO computing time Longest IPO computing time Shortest IPO computing time Current IPO computing time in ratio to the IPO cycle Current computing time for synchronized action over all channels Longest computing time for synchronized action over all channels Percentage of the total synchronized action
DO TECCYCLE	Run technology cycle
DO LOCK(n, n, ...) DO UNLOCK(n, n, ...) DO RESET(n, n, ...)	Disable Enable RESET a technology cycle
CANCEL(n, n, ...)	Delete modal synchronized actions with the designation ID(S) in the parts 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

```
WHEN $AA_IW[Q1]>5 DO M172 H510        ;If the actual value of axis Q1 exceeds 5mm,
                                      ;auxiliary functions M172 and H510 are output
                                      ;to the PLC.
```

10.4.3 Set read-in disable (RDISABLE)

Function

With RDISABLE further block execution is stopped in the main program if the condition is fulfilled. 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.

```
...
WHENEVER $A_INA[2]<7000 DO RDISABLE       ;If the voltage 7V is not reached at
                                          ;input 2, the program is stopped
                                          ;(1000= 1V).

N10 G1 X10                                ;When the condition is fulfilled,
                                          ;the read-in disable is active at the
                                          ;end of N10
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.

```
WHEN $AC_DTEB<5 DO STOPREOF            ;Cancel the preprocess stop when distance to
                                       ;block end is less than 5mm.
G01 X100                               ;The preprocessing stop is canceled after
                                       ;execution of the linear interpolation.
IF $A_INA[7]>500 GOTOF MARKE1=X100     ;If the voltage 5V is exceeded at input 7,
                                       ;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.

Programming

Delete distance-to-go for the path
```
DO DELDTG
```
or

axial delete distance-to-go
```
DO DELDTG(axis1, axis2, ...)
```

Example of fast deletion of distance-to-go path

```
WHEN $A_IN[1]==1 DO DELDTG
N100 G01 X100 Y100 F1000        ; When the input is set, the movement is canceled
N110 G01 X…
IF $AA_DELT>50…
```

Example of fast axial deletion of distance-to-go

```
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
```

Description

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).

Programming

```
FCTDEF(polynomial no.,LLIMIT,ULIMIT,a0,a1,a2,a3)
```

Parameter

`Polynomial_No.`	Number of the 3rd order polynomial
`LLIMIT`	Lower limit for function value
`ULIMIT`	Upper limit for function value
a_0, a_1, a_2, a_3	Polynomial coefficient

These values can also be accessed via system variables

`$AC_FCTLL[n]`	Lower limit for function value
`$AC_FCTUL[n]`	Upper limit for function value
`$AC_FCT0[n]`	a_0
`$AC_FCT1[n]`	a_1
`$AC_FCT2[n]`	a_2
`$AC_FCT3[n]`	a_3

Note

Writing system variables

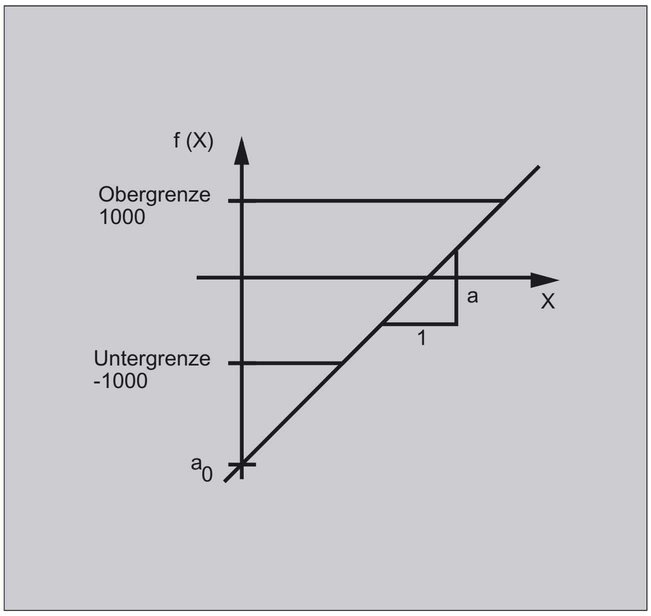
- The system variables can be written from the parts program or from a synchronized action. When writing from parts programs, program STOPRE to ensure that writing is block synchronized.
- The `$AC_FCTLL[n]`, `$AC_FCTUL[n]`, `$AC_FCT0[n]` to `$AC_FCTn[n]` system variables can be changed from synchronized actions

When writing form 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 `a0=$AA_IM[X]` and linear gradient 1 the polynomial is:

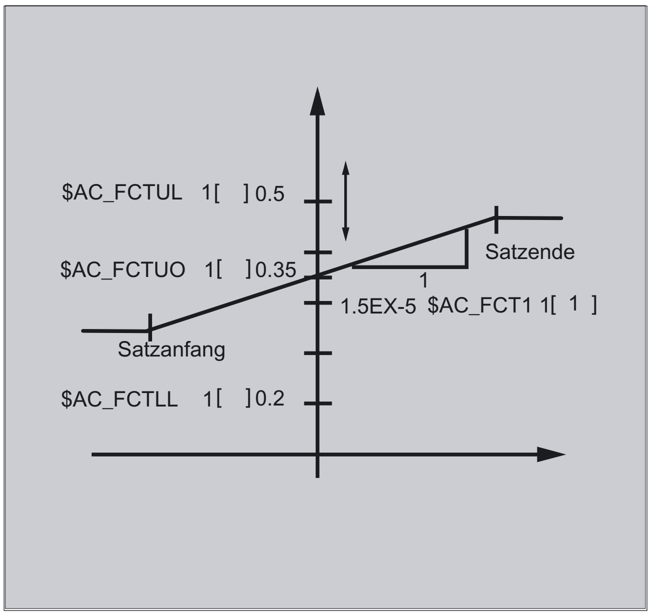
```
FCTDEF(1, -1000,1000,$AA_IM[X],1)
```



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.



```
$AC_FCTLL[1]=0.2                                  ;Definition of the polynomial
                                                  ;coefficient
$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          ;Changing the upper limit online.
ID=2 DO SYNFCT(1,$A_OUTA[1],$AC_VACTW)
;Depending on the path velocity (stored in $AC_VACTW) the laser output control
;is controlled via analog output 1
```

Note

The polynomial defined above is used with SYNFCT.

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

Programming

```
SYNFCT (Polynomial_No., main run variable output, main run variable
input)
```

Parameters

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.

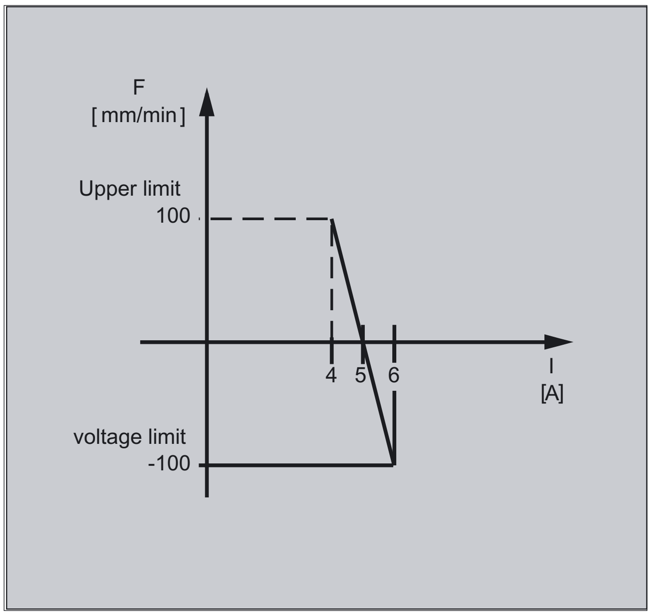
DO SYNFCT	Activation of the evaluation function
Polynomial_No.	With polynomial defined with FCTDEF (see Subsection "Polynomial definition").
Main run variable output	Write main run variable
Main run variable input	Read main run variable

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 = -100mm/1 min A

a_0 = -(-100)*5 =500

$a_2 = a_3 = 0$ (no square and cubic component)

Upper limit = 100

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.

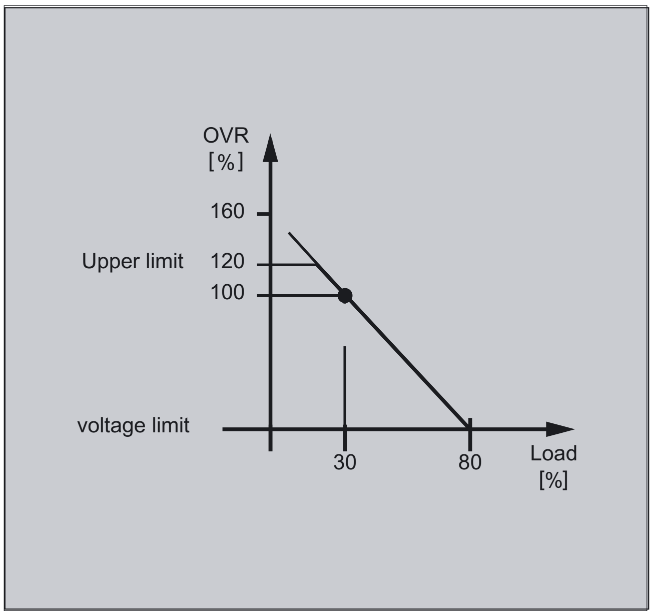
Example of adaptive control (multiplicative)

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%.



1. Polynomial definition

Determination of the coefficients

$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$

$a_1 = -100\%/(80-30)\% = -2$

$a_0 = 100 + (2*30) = 160$

$a_2 = a_3 = 0$ (no square and cubic component)

Upper limit = 120

Lower limit = 0

This means:

```
FCTDEF(2,0,120,160,-2,0,0)
```

2. Activate AC control

```
ID=1 DO SYNFCT(2,$AC_OVR,$AA_LOAD[x])
```

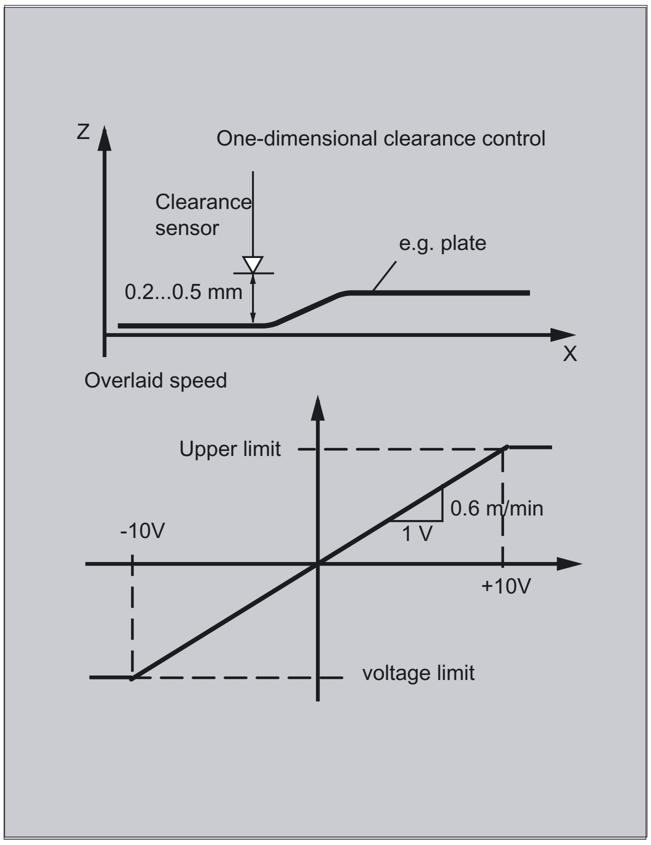
;Read the current axis load (% of the max. drive current) via `$AA_LOAD[x]`,
;calculate the feedrate override with the polynomial defined above.

10.4.8 Clearance control with limited compensation $AA_OFF_MODE

Function

The integrating calculation of the distance values is performed with boundary check

$AA_OFF_MODE = 1



Notice

The loop gain of the overlying control loop depends on the setting for the interpolation cycle.

Remedy: Read MD for interpolation cycle and take it into account.

Note

Limitation of the speed of the overlaid interpolator using MD 32020: JOG_VELO for IPO cycle 12 ms. Formula for speed:

$$\frac{0.120mm}{0.6ms}/mV = 0.6\frac{m}{\min}/V$$

Example

Subroutine: clearance control ON

```
%_N_AON_SPF                                ;Subroutine for clearance control ON
PROC AON
$AA_OFF_LIMIT[Z]=1                         ;Determine limiting value
FCTDEF(1, -10, +10, 0, 0.6, 0.12)          ;Polynomial definition
ID=1 DO SYNFCT(1,$AA_OFF[Z],$A_INA[3])     ;Clearance control active
ID=2 WHENEVER $AA_OFF_LIMIT[Z]<>0          ;Disable axis X when limit value is
 DO $AA_OVR[X] = 0                         ;overshot
RET
ENDPROC
```

Subroutine: clearance control OFF

```
%_N_AOFF_SPF
PROC AOFF                                  ;Subroutine for clearance control OFF
CANCEL(1)                                  ;Cancel clearance control synchronized
                                           ;action
CANCEL(2)                                  ;Cancel limit range check
RET
ENDPROC
```

Main program

```
%_N_MAIN_MPF
AON                                        ;Clearance control ON
...
G1 X100 F1000
AOFF                                       ;Clearance control OFF
M30
```

Position offset in the basic coordinate system

With the system variable `$AA_OFF[axis]` on overlaid movement of each axis in the channel is possible. 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 stored in the setting data
`SD 43350: AA_OFF_LIMIT`.

Using the machine data `MD 36750: AA_OFF_MODE` defines the mode of overlaying distance:

0: Proportional evaluation

1: Integrating evaluation

With system variable `$AA_OFF_LIMIT[axis]` a directional scan to see whether the offset value is within the limits is possible. These system variables can be scanned from synchronized actions and, when a limit value is reached, it is possible to 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.

Programming

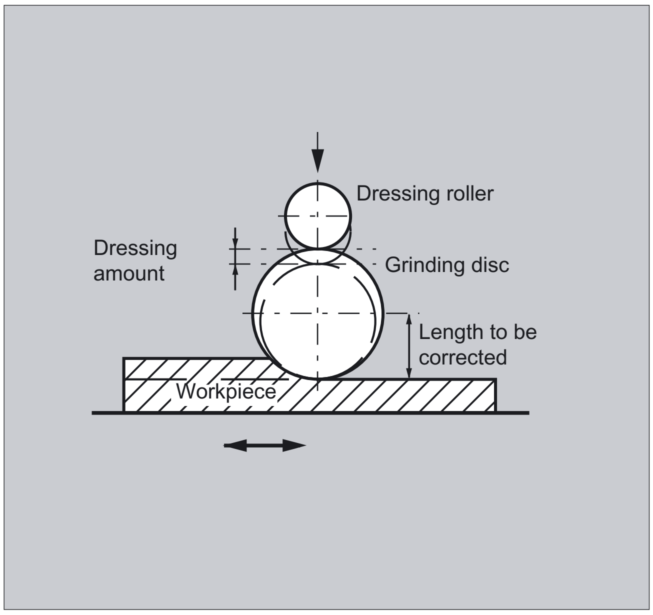
`FTOC(Polynomial_No., RV, Length1_2_3` or `Radius4, channel, spindle)`

Parameters

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.



```
%_N_DRESS_MPF
FCTDEF(1,-1000,1000,-$AA_IW[V],1)          ;Define function:
ID=1 DO FTOC(1,$AA_IW[V],3,1)              ;Select online tool offset:
                                           ;Actual value of the V axis is the input
                                           ;value for polynomial 1; the result is
                                           ;added length 3 of the active grinding
                                           ;wheel in channel 1 as the offset value.
WAITM(1,1,2)                               ;Synchronization with machining channel
G1 V-0.05 F0.01 G91                        ;Infeed movement to dress wheel
G1 V-0.05 F0.02
...
CANCEL(1)                                  ;Deselect online offset
...
```

10.4.10 Online tool length offset ($AA_TOFF[tool direction])

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.

Programming

```
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]
```

Parameter

TOFFON	**T**ool **Off**set **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	**T**ool **Off**set **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	Offset in X direction Offset in Y direction Offset in Z direction

Example of tool length offset selection

```
N10 TRAORI(1)                          ;Transformation ON
N20 TOFFON(Z)                          ;Activation of online tool length offset
                                       ;for the Z tool direction
N30 WHEN TRUE DO $AA_TOFF[Z] = 10      ;For the Z tool direction, a tool
G4 F5                                  ;length offset of 10 is interpolated
N40 TOFFON(X)                          ;Activation of online tool length offset
                                       ;for the X tool direction
N50 ID=1 DO $AA_TOFF[X] = $AA_IW[X2]   ;For the X tool direction, an offset is
G4 F5                                  ;executed subject to the position of
                                       ;axis X2
...
N100 XOFFSET = $AA_TOFF_VAL[X]         ;Assign current offset in X direction for
N120 TOFFON(X, -XOFFSET)               ;the X tool direction, the tool length
G4 F5                                  ;offset will be returned to 0 again
```

Example of tool length offset deselection

```
N10 TRAORI(1)                          ;Transformation ON
N20 TOFFON(X)                          ;Activation of Z tool direction
N30 WHEN TRUE DO $AA_TOFF[X] = 10      ;For the X tool direction, a tool length
G4 F5                                  ;offset of 10 is interpolated
...
N80 TOFFOF(X)                          ;Positional offset of the X tool direction
                                       ;is deleted: …$AA_TOFF[X] = 0
                                       ;No axis is traversed;
                                       ;to the current position in WCS, the
                                       ;positional offset is added in accordance
                                       ;with the current orientation
```

10.4.11 Positioning movements

Function

Axes can be positioned completely unsynchonized 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 parts program, the positioning axis movement has no effect on execution of the parts program.

Programming

```
POS[axis]=value
```

Parameter

DO POS	Start/position command axis
Axis	Name of the axis to be traversed
Value	The value to traverse by (depending on traverse mode)

Example

```
ID=1 EVERY $AA_IM[B]>75 DO POS[U]=100
;Axis U is moved incrementally from the control zero by 100 (inch/mm) or
;to position 100 (inch/mm) independently of the traversing mode.
ID=1 EVERY $AA_IM[B]>75 DO POS[U]=$AA_MW[V]-$AA_IM[W]+13.5
;Axis U moved by a path calculated from main run variables.
```

Example

The program environment affects the positioning travel of the positioning axis
(no G function in the action part of the synchronized action)

```
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=10 mm X=10 mm
N160 G70 Z10 F10                                ;Z=254 mm X=254 mm
N170 G71 Z10 F10                                ;Z=10 mm X=10 mm
N180 M30
```

`G71` in the action part of the synchronized action clearly determines the positioning travel of the positioning axis (metric), whatever the program environment.

```
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=10 mm X=10 mm
N160 G70 Z10 F10                                ;Z=254 mm X=10 mm (X positioned
                                                ;always to 10 mm)
N170 G71 Z10 F10                                ;Z=10 mm X=10 mm
N180 M30
```

If you do not want the axis motion to start at the beginning of the block, the override for the axis can be held at 0 until the appropriate time from a synchronized action.

```
WHENEVER      $A_IN[1]==0 DO $AA_OVR[W]=0
              G01 X10 Y25 F750 POS[W]=1500
              FA=1000
              ;The positioning axis is halted as long as digital input 1 = 0
```

10.4.13 Position in specified reference range (POSRANGE)

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 parts program, the alarm 14091 %1 block %2 is triggered, function not permitted, index: %3 with index 5 called up.

Programming

```
BOOL POSRANGE(Axis, Refpos, Winlimit,[Coord])
```

Parameter:

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) ≤ Actpos(Coord) ≤ 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.

Programming

```
MOV[axis] = value
```

Parameters

`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

```
... 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.

Programming

`GET(axis[,axis{,...}])` Get axis

`RELEASE(axis[,axis{,...}])` Release axis

Parameter

DO RELEASE	Release axis as neutral axis
DO GET	Get axis for axis replacement
Axis	Name of the axis 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:

```
     WHEN TRUE DO RELEASE(Z)         ;Z axis becomes the neutral axis
     WHENEVER($AA_TYP[Z]==1) DO      ;Read-in disable as long as Z axis is program
     RDISABLE                        ;axis
N110 G4 F0.1
     WHEN TRUE DO GET(Z)             ;Z axis returns to status as NC program axis
     WHENEVER($AA_TYP[Z]<>1) DO      ;Read-in disable until Z axis is program axis
     RDISABLE
N120 G4 F0.1
     WHEN TRUE DO RELEASE(Z)         ;Z axis becomes the neutral axis
     WHENEVER($AA_TYP[Z]==1) DO      ;Read-in disable as long as Z axis is program
     RDISABLE                        ;axis
N130 G4 F0.1
N140 START(2)                        ;Start the second channel
```

Program sequence in the second channel:

```
     WHEN TRUE DO GET(Z)              ;Move Z axis to second channel
     WHENEVER($AA_TYP[Z]==0) DO       ;Read-in disable as long as Z axis is in
     RDISABLE                         ;other channel
N210 G4 F0.1
     WHEN TRUE DO GET(Z)              ;Z axis is NC program axis
     WHENEVER($AA_TYP[Z]<>1) DO       ;Read-in disable until Z axis is program axis
     RDISABLE
N220 G4 F0.1
     WHEN TRUE DO RELEASE(Z)          ;Z axis in second channel is neutral axis
     WHENEVER($AA_TYP[Z]==1) DO       ;Read-in disable as long as Z axis is program
     RDISABLE                         ;axis
N230 G4 F0.1
N250 WAITM(10, 1, 2)                  ;Synchronize with channel 1
```

Program sequence in the first channel continues:

```
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       ;Read-in disable as long as Z axis is in
     RDISABLE                         ;other channel
N160 G4 F0.1
N199 WAITE(2)                         ;Wait for end of program in channel 2
N999 M30
```

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:

```
GET(U)                                ;Move 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.

Programming

FA[axis]=feedrate

Example

```
ID=1 EVERY $AA_IM[B]>75 DO POS[U]=100 FA[U]=990
;Define fixed feedrate value
ID=1 EVERY $AA_IM[B]>75 DO POS[U]=100 FA[U]=$AA_VACTM[W]+100
;Calculate feedrate value from main run variables
```

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 parts program or as a positioning axis from a synchronized action.

If the same axis is to be traversed alternately from the parts 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 parts program, preprocessing must be reorganized. This, in turn, causes an interruption in the parts program processing comparable to a preprocessing stop.

Example for traversing X axis alternately from parts program and from synchronized actions

```
N10 G01 X100 Y200 F1000                ;X axis programmed in parts program
…
N20 ID=1 WHEN $A_IN[1]==1 DO           ;Starting positioning from the synchronized
POS[X]=150 FA[X]=200                   ;action if a digital input is set
…
CANCEL(1)                              ;Deselect synchronized action
…
N100 G01 X240 Y200 F1000
;X becomes the path axis; before motion, delay occurs because of axis transfer
;if digital input was 1 and X was positioned from the synchronized action.
```

Example of changing traverse command for the same axis:

```
ID=1 EVERY $A_IN[1]>=1 DO POS[V]=100 FA[V]=560
;Start positioning from the synchronized action if a digital input is >= 1
ID=2 EVERY $A_IN[2]>=1 DO POS[V]=$AA_IM[V] FA[V]=790
;Axis follows, 2nd input is set, i.e. end position and feed for axis V are
;continuously followed during a movement when two synchronized actions are
;simultaneously active.
```

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 parts program and
- all command axes that have been started from a synchronized action

Programming

```
DO PRESETON(axis, value)
```

Parameters

`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 by moved from the parts program and from a synchronized action, only at different times. For this reason, delays can occur in the programming of an axis from the parts program if the same axis has been program in a synchronized action first.

If the same axis is used alternately, transfer between the two axis movements is coordinated. Parts program execution must be interrupted for that.

Example

Moving the control zero of an axis

```
WHEN $AA_IM[a] >= 89.5 DO PRESETON(a4,10.5)
;Offset control zero of axis a by 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 parts 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

```
ID=1 EVERY $A_IN[1]==1 DO M3 S1000          ;Set direction and speed of rotation
ID=2 EVERY $A_IN[2]==1 DO SPOS=270          ;Position spindle
```

Example of setting the direction and speed of rotation/ positioning the spindle

```
ID=1 EVERY $A_IN[1]==1 DO M3 S300           ;Set direction and speed of rotation
ID=2 EVERY $A_IN[2]==1 DO M4 S500           ;Specify new direction and new speed of
                                            ;rotation
ID=3 EVERY $A_IN[3]==1 DO S1000             ;Specify new speed
ID=4 EVERY ($A_IN[4]==1) AND ($A_IN[1]==0)  ;Position spindle
DO SPOS=0
```

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".

Programming

Activate coupled-axis motion

```
DO TRAILON(following axis, leading axis,
coupling factor)
```

Deactivate coupled-axis motion

```
DO TRAILOF(following axis, leading axis,
leading axis 2)
```

Parameters

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

```
$A_IN[1]==0 DO TRAILON(Y,V,1)    ;Activate 1st combined axis pair when the digital
                                 ;input is 1
$A_IN[2]==0 DO TRAILON(Z,W,-1)   ;Activate 2nd coupled axis grouping
G0 Z10                           ;Infeed Z and W axes in opposite axial directions
G0 Y20                           ;Infeed of Y and V axes in same axis directions
...
G1 Y22 V25                       ;Superimpose dependent and independent movement of
                                 ;trailing axis "V"
...
TRAILOF (Y,V)                    ;Deactivate 1st coupled axis grouping
TRAILOF (Z,W)                    ;Deactivate 2nd coupled axis grouping
```

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 be ensured that `TRAILOF` is executed before the channel requests the

axis. 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.

Programming

Activate leading 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)
```

Parameters

Activate axial leading value coupling:	
...DO **LEADON**(FA, LA, NR, OVW)	with: 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`

```
%_N_SCHERE1_MPF
;$PATH=/_N_WKS_DIR/_N_DEMOFBE_WPD
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 following parting
N1500 RELEASE(Y)
N1800 ID=6 EVERY $AA_IM[X]<10 DO LEADON(Y,X,1)
                                        ;Couple Y to X via table 1, for X < 10
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 $R1==1 DO MOV[X]=1      ;Set extruded material axis continuously
FA[X]=$R4                               ;in motion
N2200 M30
```

10.4.23 Measuring (MEAWA, MEAC)

Function

Compared with use in traverse blocks of the parts program, the measuring function can be activated and deactivated as required.

For further information concerning measuring, see special motion commands "Extended measuring function"

Programming

Axial measurement without deletion of distance-to-go

`MEAWA[axis]=(mode, trigger_event_1, ..._4)`

or

Continuous measurement without deleting distance-to-go

`MEAC[axis]=(mode, measurement_memory, trigger_event_1, ..._4)`

Parameters

`DO MEAWA`	`Activate axial measurement`	
`DO MEAC`	`Activate continuous measurement`	
`Axis`	`The name of the axis for which measurement is taken`	

`Mode`	`Specification of the` **`tens`** `decade`
	`0: active measuring system`
	`Number of the measuring systems (depending on the mode)`
	`1: 1. Measuring system`
	`2: 2. Measuring system`
	`3: both measuring systems`
		`Specification of the` **`units`** `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`

`Trigger_event_1 to trigger_event_4`	`: rising edge, probe`
	`-1: falling edge, probe 1 optional`
	`2: rising edge, probe 2 optional`
	`-2: falling edge, probe 2 optional`	

`Measurement memory`	`Number of the FIFO circulating storage`	

10.4.24 Initialization of array variables with SET, REP

Function

Array variables can be initialized or described with particular values in synchronized actions.

Programming

```
DO ARRAY[n,m]=SET(value1, value2, ...)
```

or

```
DO ARRAY[n,m] = REP(value)
```

Initialization starts at the programmed array indexes. For 2D arrays, the second index is incremented first. This is not done with axis indices.

Value assignments of array variables

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.

SET(value list)	Initialization with value lists
REP (value)	Initialization with the same values
Value list	With the number of specified values
Value	With the same value up to the end of the array

SET(value list)

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, a system alarm is triggered.

REP(value)

The array is described from the programmed array indices to the end of the array and repeated using the `REP`parameters.

Example

```
WHEN TRUE DO SYG_IS[0]=REP(0)
WHEN TRUE DO SYG_IS[1]=SET(3,4,5)
Result:
SYG_IS[0]=0
SYG_IS[1]=3
SYG_IS[2]=4
SYG_IS[3]=5
SYG_IS[4]=0
```

10.4.25 Set/delete wait markers with SETM, CLEARM

Function

In synchronized actions, wait markers can be set or deleted for the purpose of coordinating channels, for example.

Programming

```
DO SETM(MarkerNumber)
```

or

```
DO CLEARM(MarkerNumber)
```

Set/delete wait markers for the channel

`SETM(MarkerNumber)`	`Set wait marker for channel`
`CLEARM(MarkerNumber)`	`Clear wait marker for channel`

SETM

The `SETM` command can be written in the parts program and in the action part of a synchronized action. It sets the marker (marker number) for the channel in which the command is applied

CLEARM

The `CLEARM` command can be written in the parts program and in the action part of a synchronized action. It deletes the marker (marker number) for the channel in which the command is applied.

10.4.26 Error responses during SETAL cycle alarms

Function

Incorrect responses can be programmed with synchronized actions by scanning status variables and triggering the appropriate actions.

Some possible responses to error conditions are:

- Stop axis: Override=0
- Set alarm: With SETAL it is possible to set cyclic alarms from synchronized actions.
- Set output
- All actions possible in synchronized actions

Set cycle alarm

```
DO SETAL(AlarmNumber)
```

Cycle alarm range for users: 65000 ... 69999

Example

```
ID=67 WHENEVER ($AA_IM[X1]-$AA_IM[X2])<4.567 DO $AA_OVR[X2]=0
;If the safety distance between axes X1 and X2 is too small, stop axis X2.
ID=67 WHENEVER ($AA_IM[X1]-$AA_IM[X2])<4.567 DO SETAL(61000)
;If the safety distance between axes X1 and X2 is too small, set an alarm.
```

10.4.27 Travel to fixed stop (FXS and FOCON/FOCOF)

Function

The commands for **travel to fixed stop** are programmed with the FXS, FXST and FXSW parts program commands in synchronized actions / technology cycles.
The activation can be made without motion, the moment will be limited immediately. As soon as the axis is moved via a setpoint, the limit stop monitor is activated.

Travel with limited torque/force (FOC):
This function allows torque/force to be changed at any time via synchronized actions and can be activated modally or non-modally.

Parameters

`FXS[axis]`	Selection only in systems with digital drives (FDD, MSD, HLA)
`FXST[axis]`	Modification of clamping torque FXST
`FXSW[axis]`	Change of monitoring window FXSW
`FOCON[axis]`	Activation of the modally effective torque/force limitation
`FOCOF[axis]`	Disable torque/force limitation
`FOCON/FOCOF`	The axis is programmed in square brackets. Permitted are: - Geometry axis identifier - Channel axis identifier - Machine axis identifier

Note

A selection may only be carried out once.

Example of travel to fixed stop (FXS)

Triggered by a synchronized action

```
Y axis:                                     ;Static synchronized actions
Activate:
N10 IDS=1 WHENEVER (($R1==1) AND            ;By setting of $R1=1,
            ($AA_FXS[y]==0)) DO             ;axis Y FXS will be activated, the
            $R1=0 FXS[Y]=1 FXST[Y]=10       ;effective torque is reduced to 10% and
            FA[Y]=200 POS[Y]=150            ;an approach motion started in the
                                            ;direction of the stop
N11 IDS=2 WHENEVER ($AA_FXS[Y]==4) DO       ;Once the stop has been recognized
            FXST[Y]=30                      ;($AA_FXS[Y]==4), the torque is reduced
                                            ;to 30%
N12 IDS=3 WHENEVER ($AA_FXS[Y]==1) DO       ;After reaching the stop
            FXST[Y]=$R0                     ;the torque is controlled depending on
                                            ;R0
N13 IDS=4 WHENEVER (($R3==1) AND            ;Deselect depending
            ($AA_FXS[Y]==1)) DO             ;on R3 and
            FXS[Y]=0                        ;return
            FA[Y]=1000 POS[Y]=0
N20 FXS[Y]=0 G0 G90 X0 Y0                   ;Normal program run:
                                            ;axis Y for
N30 RELEASE(Y)                              ;Enable motion in synchronized action
N40 G1 F1000 X100                           ;Movement of another axis
N50 ......
N60 GET(Y)                                  ;include Y axis again in the path group
```

Example of activating the torque/force limitation (FOC)

```
N10 FOCON[X]                                ;Modal activation of limitation
N20 X100 Y200 FXST[X]=15                    ;X travels with reduced torque (15%)
N30 FXST[X]=75 X20                          ;Change the torque to 75%, X travels
                                            ;with this limited torque
N40 FOCOF[X]                                ;Disable torque limit
```

Multiple selection

If the function is called once more due to faulty programming (`FXS[Axis]=1`) the alarm 20092 "Travel to fixed stop still active" is initiated.

Programming code that scans `$AA_FXS[]` or a separate flag (here R1) in the condition will ensure that the parts program fragment function is not activated more than once.

```
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:

```
N10 G0 G90 X0 Y0
N20 WHEN $AA_IW[X] > 17 DO FXS[X]=1          ;If X reaches a position greater than
                                             ;17 mm
N30 G1 F200 X100 Y110                        ;FXS is activated
```

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 the normal parts program run. The values assigned can be resulted from a calculation.

10.4.28 Determining the path tangent in synchronized actions

Function

The system variable $AC_TANEB (**T**angent **AN**gle at **E**nd of **B**lock), 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 `$AC_TANEB` should not be read for blocks generated by the system (intermediate blocks). The system variable `$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_PLC_OVR` Axial override

`$AC_PLC_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:

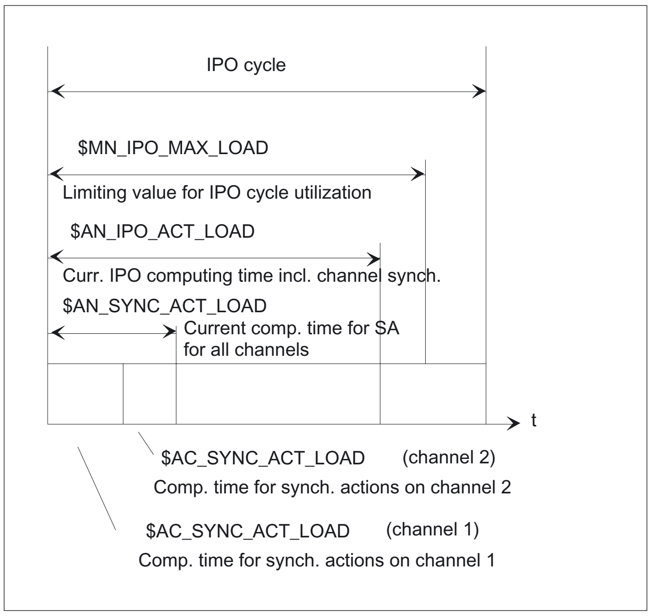
`$AA_OVR * $AA_PLC_OVR` or

`$AC_OVR * $AC_PLC_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.


IPO cycle
$MN_IPO_MAX_LOAD
Limiting value for IPO cycle utilization
$AN_IPO_ACT_LOAD
Curr. IPO computing time incl. channel synch.
$AN_SYNC_ACT_LOAD
Current comp. time for SA for all channels
t
$AC_SYNC_ACT_LOAD (channel 2)
Comp. time for synch. actions on channel 2
$AC_SYNC_ACT_LOAD (channel 1)
Comp. time for synch. actions on channel 1

Parameters

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 subroutines.

It is possible to process several technology cycles or actions in parallel in one channel.

Programming

- 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 non-modal synchronized actions.
- Both IF check structures and GOTO, GOTOF and GOTOB jump instructions can be programmed.

Blocks with DEF and DEFINE instructions in technology cycles

- DEF and DEFINE instructions are read over into technology cycles
- these still result in alarm messages if the syntax is incorrect or incomplete
- can be read over without an alarm message without being applied themselves
- are taken into full consideration with value assignments as parts program cycles

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.
- To provide default parameters with initial values.
- Transfer non-initialized current parameters with a default value.

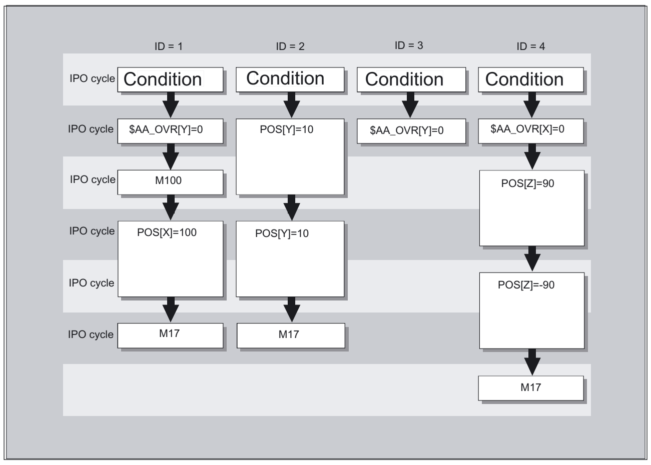
Sequence

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. In the technology cycle, blocks are executed in sequence.

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.

Example

Axis programs are started by setting digital inputs.



```
Main program:                           If
ID=1 EVERY $A_IN[1]==1 DO AXIS_X        ;input 1 is at 1, start axis program X
ID=2 EVERY $A_IN[2]==1 DO AXIS_Y        ;input 2 is at 1, start axis program Y
ID=3 EVERY $A_IN[3]==1 DO $AA_OVR[Y]=0  ;input 3 is at 1, set the override for
                                        ;axis Y to 0
ID=4 EVERY $A_IN[4]==1 DO AXIS_Z        ;input 4 is at 1, start axis program Z
M30

Technology cycle AXIS_X:
$AA_OVR[Y]=0
M100
POS[X]=100 FA[X]=300
M17
Technology cycle AXIS_Y:
POS[Y]=10 FA[Y]=200
POS[Y]=-10
M17
Technology cycle AXIS_Z:
$AA_OVR[X]=0
POS[Z]=90 FA[Z]=250
POS[Z]=-90
M17
```

Examples of different program sequences in the technology cycle

```
PROC CYCLE
N10 DEF REAL "value"=12.3
N15 DEFINE ABC AS G01
```

Both blocks are read over without alarms and without the variable and/or macro being applied

```
PROC CYCLE
N10 DEF REAL
N15 DEFINE ABC G01
```

Both blocks still result in the NC alarm because the syntax is not written correctly.

```
PROC CYCLE
N10 DEF AXIS "axis1"=XX2
```

If axis XX2 is not known, alarm 12080 is output. Otherwise the block is overlooked without alarms and without the variable being applied.

```
PROC CYCLE
N10 DEF AXIS "axis1"
N15 G01 X100 F1000
N20 DEF REAL"value1"
```

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 variable 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 parts 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 parts 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 parts program cycle.

Example of program sequence with query of $P_TECCYCLE in the technology cycle

```
PROC CYCLE
N10 DEF REAL "value1"                  ;is read over in the technology cycle
N15 G01 X100 F1000
N20 IF $P_TECCYCLE==TRUE
N25 "Program sequence for technology cycle (without variable value1)"
N30 ELSE
N35 "Program sequence for parts program cycle (variable value1 is present)"
ENDIF
```

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".

Programming

```
ID=1 WHEN $AA_IW[X]>50 DO TEC(IVAL, RVAL, , SVAL, AVAL)
;A default value is transferred for non-initialized current 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.

Programming

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 parts program and in synchronized actions.

Example

```
TEC (IVAL, RVAL, , SVAL, AVAL)        ;the initial value applies to CVAL and BVAL
```

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.

Programming

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 parts program.

Example of ICYCOF processing mode

```
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.

Programming

```
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 IF check structures

Function

IF check structures can be used in synchronized actions for branches in the processing sequence of technology cycles.

Programming

```
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.

Programming

Unconditional jumps

`GOTO` Label, block number

or

`GOTOF` Label, block number

or

`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 process of a technology cycle can be locked, released again or a technology cycle reset by modal synchronized actions / by other modal synchronized actions.

Programming

`LOCK (n, n, ...)`	Lock synchronized actions, the active action is interrupted
`UNLOCK (n, n, ...)`	Unlock synchronized actions
`RESET (n, n, ...)`	Reset technology cycle
`n`	Identification number of the synchronized action

Locking on the PLC side

Modal synchronized actions can be interlocked from the PLC with the ID numbers **n=1 ... 64**. 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.

Example

Lock synchronized actions, LOCK

```
N100 ID=1 WHENEVER $A_IN[1]==1 DO M130
...
N200 ID=2 WHENEVER $A_IN[2]==1 DO LOCK(1)
```

Unlock synchronized actions, UNLOCK

```
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)
```

Interrupt technology cycle, RESET

```
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 with the identifier ID(S)=n can only be canceled directly from the parts program with CANCEL.

Note

Incomplete movements originating from a canceled synchronized action are completed as programmed.

Programming

`CANCEL(n, n, ...)`	Cancel synchronized action
`n`	Identification number of the synchronized action

Example

```
N100 ID=2 WHENEVER $A_IN[1]==1 DO M130
...
N200 CANCEL(2)                          ;Cancel synchronized action No. 2
```

10.7 Restrictions

Function

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 subroutine ASUB
- Repositioning REPOS
- Deselection with CANCEL

Events

- Power on

  No synchronized actions are ever active during POWER ON. Static synchronized actions can be activated by an asynchronized subroutine (`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 you do not want them to remain active, cancel the synchronized action before end of program by pressing `CANCEL` (see preceding section).

Response following end of program		
Synchronized action/ technology cycle	**Modal and non-modal actions are aborted**	**Static actions (IDS) remain active**
Axis/ positioning spindle	M30 is delayed until the axis / spindle is stationary.	Motion continues
Speed-controlled spindle	End of program: $MA_SPIND_ACTIVE_AFTER_RESET==1 : Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==0 : Spindle stops The spindle remains active if the operating mode changes	Spindle remains active
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	A coupling started from a static synchronized action remains active
Measuring operations	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 by the asynchronous subroutine ASUB**

  ASUB start:
  Modal and static motion-synchronous actions remain active and are also operative in the asynchronous subprogram (ASUB).
  ASUB end:
  If the asynchronized subroutine is not resumed with `REPOS` modal and static motion-synchronized actions that were modified in the asynchronized subroutine remain active in the main program.

- Repositioning REPOS

  After repositioning `REPOS`, the synchronized actions that were active in the interrupted block are reactivated. After `REPOS`, the modal synchronized actions changed from the asynchronous subroutine no longer act for the machining of the remaining block. Polynomial coefficients programmed with `FCTDEF` are not affected by asynchronous subroutines and `REPOS`. No matter where they were programmed, they can be used at any time in the asynchronized subroutine and in the main program after execution of `REPOS`.

- Deselection 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 required, the axis movement can be decelerated before the CANCEL command with axial deletion of distance-to-go:

Example of deselection with CANCEL

```
ID=17 EVERY $A_IN[3]==1 DO POS[X]=15 FA[X]=1500     ;Start positioning axis movement
...
WHEN ... DO DELDTG(X)                               ;End positioning axis movement
CANCEL(1)
```

Oscillation 11

11.1 Asynchronous oscillation

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 asynchronized oscillation

- Asynchronous oscillation is active on an axis-specific basis beyond block limits.
- Block-oriented activation of the oscillation movement is ensured by the parts program.
- Combined interpolation of several axes and superimposing of oscillation paths are not possible.

Programming

The following addresses allow asynchronized 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

Parameters

`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

Hold 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`.

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.

```
WAITP(X,Y,Z)                            ;Initial setting
G0 X100 Y100 Z100                       ;Switch over in positioning axis
                                        ;operation
N40 WAITP(X,Z)
N50 OSP1[Z]=10 OSP2[Z]=100 ->           ;Reversal point 1, reversal point 2
-> OSE[Z]=200 ->                        ;End position
-> OST1[Z]=0 OST2[Z]=-1 ->              ;Stopping time at U1: Exact stop fine;
                                        ;Stopping time at U2: Exact stop coarse
-> FA[Z]=250 FA[X]=1 ->                 ;Feed for oscillating axis, infeed axis
-> OSCTRL[Z]=(4,0) ->                   ;Setting options
-> OSNSC[Z]=3 ->                        ;Three spark-out strokes
N60 OS[Z]=1                             ;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 parts 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.

```
$SA_OSCILL_REVERSE_POS1[Z]=-10
$SA_OSCILL_REVERSE_POS2[Z]=10

G0 X0 Z0
WAITP(Z)

ID=1 WHENEVER $AA_IM[Z] < $$AA_OSCILL_REVERSE_POS1[Z] DO $AA_OVR[X]=0
ID=2 WHENEVER $AA_IM[Z] < $$AA_OSCILL_REVERSE_POS2[Z] DO $AA_OVR[X]=0
                                            ;If the actual value of the oscillation
                                            ;axis has exceeded the reversal point,
                                            ;the infeed axis is stopped.
OS[Z]=1 FA[X]=1000 POS[X]=40                ;Switch on oscillation
OS[Z]=0                                     ;Switch off 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 switch-off function is tested for the block change. 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 in However, 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.

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.

Programming

1. **Define parameters for oscillation**
2. **Define motion-synchronous actions**
3. **Assign axes, define infeed**

Parameters

`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[OscillationAxis]=`	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)
```

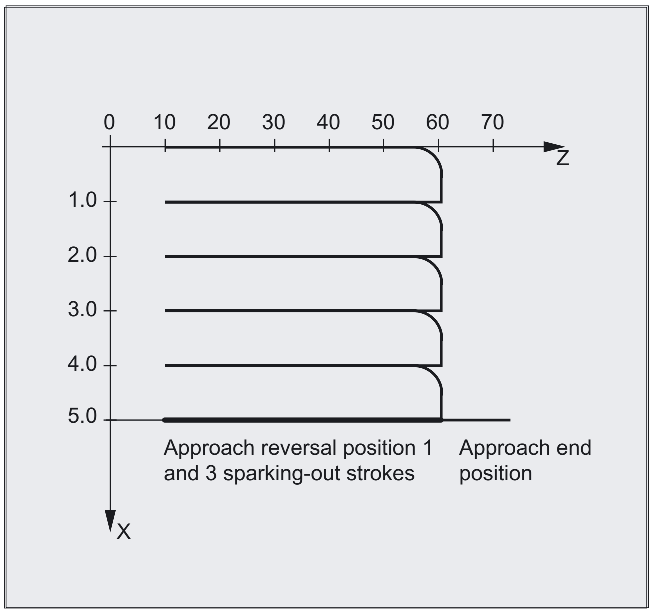
OSCILL	Assign infeed axis or axes for oscillating axis
POSP	Define complete and partial infeeds (see the "File and Program Management" chapter)
Endpos	End position for the infeed axis after all partial infeeds have been traversed.
Partial length	Length of the partial infeed at reversal point/reversal 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

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

```
WHENEVER $AA_IM[Z]<$SA_OSCILL_REVERSE_POS2[Z]DO ->
-> $AA_OVR[X]=0 $AC_MARKER[0]=0
Whenever                 the current position of oscillating axis Z in the MCS is
less than                the start of reversal area 2
then                     set the axial override of infeed axis X to 0%
and                      set the marker with index 0 to value 0.
WHENEVER $AA_IM[Z]>=$SA_OSCILL_REVERSE_POS2[Z] DO $AA_OVR[Z]=0
Whenever                 the current position of oscillating axis Z in the MCS is
greater or equal to      reversal position 2 is
then                     set the axial override of oscillating axis Z to 0%.
WHENEVER $AA_DTEPW[X] == 0 DO $AC_MARKER[0]=1
Whenever                 the distance-to-go of the part infeed
equal to                 is
then                     set the marker with index 0 to value 1.
WHENEVER $AC_MARKER[0]==1 DO $AA_OVR[X]=0 $AA_OVR[Z]=100
Whenever                 the flag with index 0
equal to                 is
then                     set the axial override of infeed axis X to 0% in order to
                         inhibit premature infeed (oscillating axis Z has not yet
                         left reversal area 2 but infeed axis X is ready for a new
                         infeed)
                         set the axial override of oscillating axis Z to 100% (this
                         cancels the 2nd synchronized action).
```

-> must be programmed in a single block

3. Start oscillation

```
OSCILL[Z]=(X) POSP[X]=(5,1,1)  ;Start the axes
                               ;Assign axis X as the infeed axis for oscillating
                               ;axis Z.
                               ;Axis X is to travel to end position 5 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[oscillation 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[InfeedAxis] = (Endpos, Partial length, Mode)
```

The following are declared to the control with the `POSP` command:

- Complete infeed (with reference to end position)
- The length of the partial infeed at the reversal point or in the reversal area
- The partial infeed response when the end position is reached (with reference to mode)

Mode = 0	The distance-to-go to the destination point for the last two partial infeeds is divided into two equal steps (default setting).
Mode = 1	All partial infeeds are of equal size. They are calculated from the complete infeed.

Define motion-synchronized actions

The synchronized-motion actions listed below are used for general oscillation.

You are given example solutions for individual tasks, which you can use as modules for creating user-specific oscillation movements

---

Note

In individual cases, the synchronization conditions can be programmed differentially.

Keywords

WHEN … DO …	when ... , do ...
WHENEVER … DO	whenever ... , do ...

Functions

You can implement the following functions with the language resources described in detail below:

1. Infeed at reversal point.
2. Infeed at reversal area.
3. Infeed at both reversal points.
4. Stop oscillation movement at reversal point.
5. Restart oscillation movement.
6. Do not start partial infeed too early.

The following assumptions are made for all examples of synchronized actions presented here:

- Reversal point 1 < reversal point 2
- Z = oscillating axis
- X = infeed axis

Note

For more details, see the "Motion-synchronous actions" section.

Assign oscillating and infeed axes as well as partial and complete infeed

Infeed in reversal point range

The infeed motion must start within a reversal area before the reversal point is reached.

These synchronized actions inhibit the infeed movement until the oscillating axis is within the reversal area.

The following instructions are used subject to the above assumptions:

Reversal point range 1:

```
WHENEVER $AA_IM[Z]>$SA_OSCILL_RESERVE_POS1[Z]+ii1 DO $AA_OVR[X] = 0
```

Whenever	the current position of oscillating axis in the MCS is
greater than	the start of reversal area 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 current position of oscillating axis in the MCS is
less than	the start of reversal area 2
then	set the axial override of the infeed axis to 0%.

Infeed at reversal point

As long as the oscillation axis has not reached the reversal point, the infeed axis does not move.

The following instructions are used subject to the above assumptions:

Reversal point range 1:

```
WHENEVER $AA_IM[Z]<>$SA_OSCILL_RESERVE_POS1[Z] DO $AA_OVR[X] = 0 →
→ $AA_OVR[Z] = 100
```

Whenever	the current position of oscillating axis Z in the MCS is
greater or less than	the position of reversal point 1
then	set the axial override of infeed axis X to 0%
and	set the axial override of oscillating axis Z to 100%.

Reversal point 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 current position of oscillating axis Z in the MCS is
greater or less than	the position of reversal point 2
then	set the axial override of infeed axis X to 0%
and	set the axial override of oscillating axis Z to 100%.

Stop oscillation movement at the reversal point

The oscillation axis is stopped at the reversal point, the infeed motion starts at the same time. The oscillating motion is continued when the infeed movement is complete.

At the same time, this synchronized action can be used to start the infeed movement if this has been stopped by a previous synchronized action, which is still active.

The following instructions are used subject to the above assumptions:

Reversal point range 1:

```
WHENEVER $SA_IM[Z]==$SA_OSCILL_RESERVE_POS1[Z] DO $AA_OVR[X] = 0 →
→ $AA_OVR[Z] = 100
```

Whenever	the current position of oscillating axis in the MCS is
equal to	reversal position 1 is
then	set the axial override of the oscillation axis to 0%
and	set the axial override of the infeed axis to 100%.

Reversal point range 2:

```
WHENEVER $SA_IM[Z]==$SA_OSCILL_RESERVE_POS2[Z] DO $AA_OVR[X] = 0 →
→ $AA_OVR[Z] = 100
```

Whenever	the current position of oscillating axis in the MCS is
equal to	reversal position 2 is
then	set the axial override of the oscillation axis to 0%
and	set the axial override of the infeed axis to 100%.

Online evaluation of reversal point

If there is a main run variable coded with `$$` on the right of the comparison, then the two variables are evaluated and compared with one another continuously in the IPO cycle.

Note

Please refer to Section "Motion-synchronized actions" for more information.

Oscillation movement restarting

The purpose of this synchronized action is to continue the movement of the oscillation axis on completion of the part infeed movement.

The following instructions are used subject to the above assumptions:

```
WHENEVER $AA_DTEPW[X]==0 DO $AA_OVR[Z]= 100
```

Whenever	the distance-to-go for the partial infeed on infeed axis X in the WCS
equal to	is zero,
then	Set the axial override of the oscillation axis to 100%.

Next partial infeed

When infeed is complete, a premature start of the next partial infeed must be inhibited.

A channel-specific marker (`$AC_MARKER[Index]`) is used for this purpose. It is enabled at the end of the partial infeed (partial distance-to-go ≡ 0) and deleted when the axis leaves the reversal area. The next infeed movement is then prevented by a synchronized action.

On the basis of the given assumptions, the following instructions apply for reversal point 1:

1. Set marker:

```
WHENEVER $AA_DTEPW[X] == 0 DO $AC_MARKER[1]=1
```

Whenever	the distance-to-go for the partial infeed on infeed axis X in the WCS
equal to	is zero,
then	set the marker with index 1 to 1.

2. Delete marker

```
WHENEVER $AA_IM[Z]<> $SA_OSCILL_RESERVE_POS1[Z] DO $AC_MARKER[1] =
0
```

Whenever	the current position of oscillating axis Z in the MCS is
greater or less than	the position of reversal point 1
then	set marker 1 to 0.

3. Inhibit infeed

```
WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0
```

Whenever	marker 1 is
equal to	is
then	set the axial override of the infeed axis to 0%.

Punching and nibbling 12

12.1 Activation, deactivation

12.1.1 Punching and nibbling On or Off (SPOF, SON, PON, SONS, PONS, PDELAYON/OF)

Function

Punching and nibbling activate, deactivate, PON/SON

The punching and nibbling functions are activated with PON and SON respectively. SPOF terminates all functions specific to punching and nibbling operations. Modal commands PON and SON are mutually exclusive, i.e., PON deactivates SON and vice versa.

Punching and nibbling with leader, PONS/SONS

The SONS and PONS commands also activate the punching or nibbling functions.

In contrast to SON/PON - stroke control on interpolation level - PONS and SONS control stroke initiation on the basis of signals on 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 brings about a delay in the output of the punching stroke. The command is modal and has a preparatory function. It is thus generally programmed before PON. Punching continues normally after PDELAYOF.

Programming

```
PONS G... X... Y... Z...
```
or
```
SON G... X... Y... Z...
```
or
```
SONS G... X... Y... Z...
```
or
```
SPOF
```
or
```
PDELAYON
```
or
```
PDELAYOF
```
or

PUNCHACC(S_{min},A_{min}, S_{max}, A_{max})

Parameters

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 PUNCHACC (S_{min}, A_{min}, S_{max}, A_{max})
"S_{min}"	Minimum hole spacing
"S_{max}"	Maximum hole spacing
"A_{min}"	The initial acceleration A_{min} can be larger than A_{max}
"A_{max}"	The final acceleration A_{max} can be less than A_{min}

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.

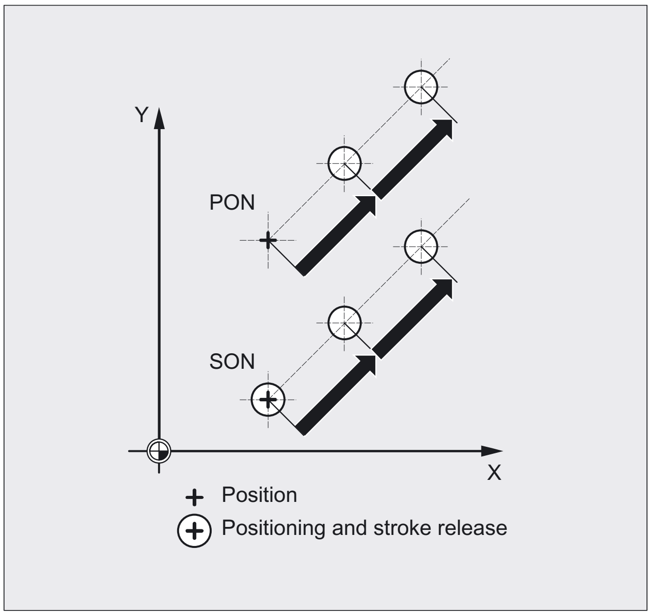
Distance between holes more than 10 mm:

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

Path segmentation

When punching or nibbling is active, SPP and SPN cause the total traversing distance programmed for the path axes to be divided into a number of path sections of equal length (equidistant path segmentation). Each path segment corresponds internally to a block.

Number of strokes

When punching is active, the first stroke is executed at the end of the first path segment. In contrast, the first nibbling stroke is executed at the start of the first path segment. The number of punching/nibbling strokes over the total traversing path is thus as follows:

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.

Programming

```
SPP=
```

or

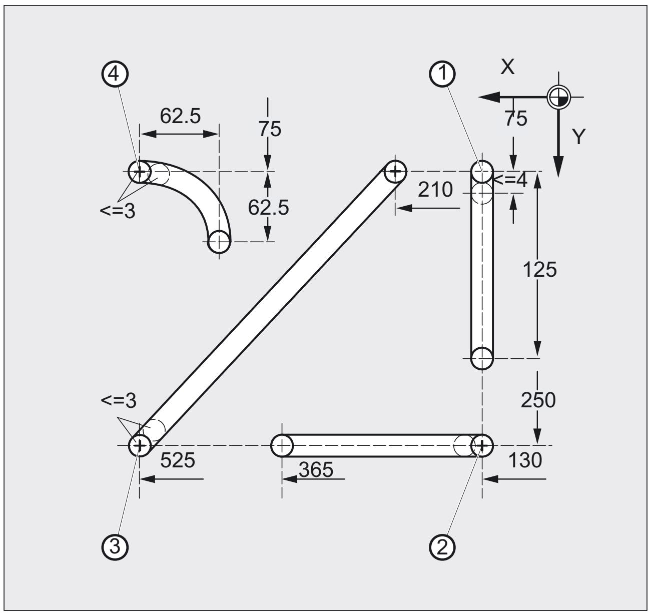
```
SPN=
```

Parameters

`SPP`	`Size of path section (maximum distance between strokes); modal`
`SPN`	`Number of path sections per block; non-modal`

Example 1

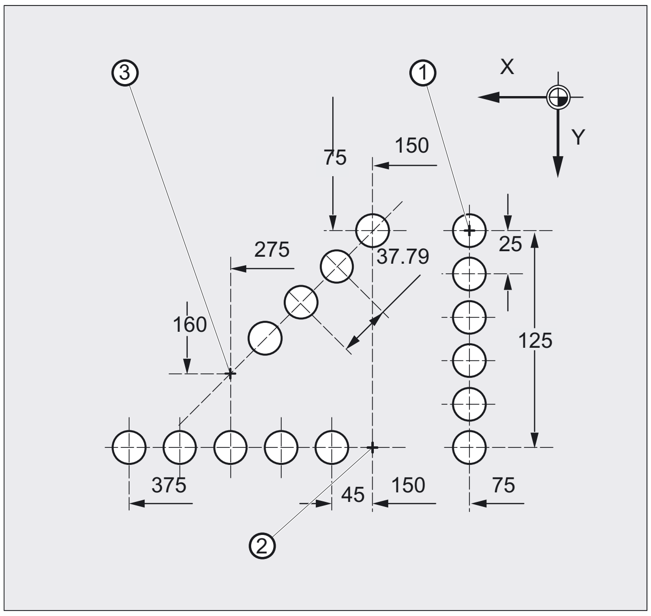
The programmed nibbling paths must be divided automatically into equidistant path segments.



```
N100 G90 X130 Y75 F60 SPOF              ;Position at starting point 1
N110 G91 Y125 SPP=4 SON                 ;Nibbling on, maximum path segment length
                                        ;for automatic path segmentation: 4 mm
N120 G90 Y250 SPOF                      ;Nibbling off, position at
                                        ;starting point 2
N130 X365 SON                           ;Nibbling on, maximum path segment length
                                        ;for automatic path segmentation: 4 mm
N140 X525 SPOF                          ;Nibbling off, position at
                                        ;starting point 3
N150 X210 Y75 SPP=3 SON                 ;Nibbling on, maximum path segment length
                                        ;for automatic path segmentation: 3 mm
N140 X525 SPOF                          ;Nibbling off, position at
                                        ;starting point 4
N170 G02 X-62.5 Y62.5 I J62.5 SPP=3 SON ;Nibbling on, maximum path segment length
                                        ;for automatic path segmentation: 3 mm
N180 G00 G90 Y300 SPOF                  ;Nibbling off
```

Example 2

Automatic path segmentation is to be used to create the individual rows of holes. The maximum path segment length (SPP value) is specified in each case for segmentation purposes.



N100 G90 X75 Y75 F60 PON	;Position at starting point 1; ;punching on; punch one hole
N110 G91 Y125 SPP=25	;Maximum path segmentation length for ;automatic segmentation: 25 mm
N120 G90 X150 SPOF	;Punching off, position at ;starting point 2
N130 X375 SPP=45 PON	;Punching on, maximum path segment length ;for automatic path segmentation: 45 mm
N140 X275 Y160 SPOF	;Punching off, position at ;starting point 3
N150 X150 Y75 SPP=40 PON	;Punching on, the calculated path segment ;length of 37.79 mm is used instead of ;the 40 mm programmed as the path segment
N160 G00 Y300 SPOF	;Punching off; position

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`.

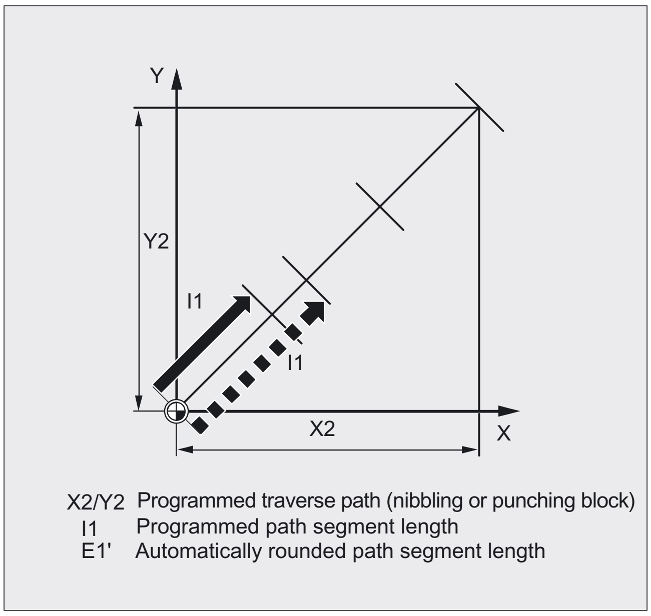
Example:

```
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.



X2/Y2	Programmed traverse path (nibbling or punching block)
I1	Programmed path segment length
E1'	Automatically rounded path segment length

Example:

```
N10 G1 G91 SON X10 Y10
N20 SPP=3.5 X15 Y15
```

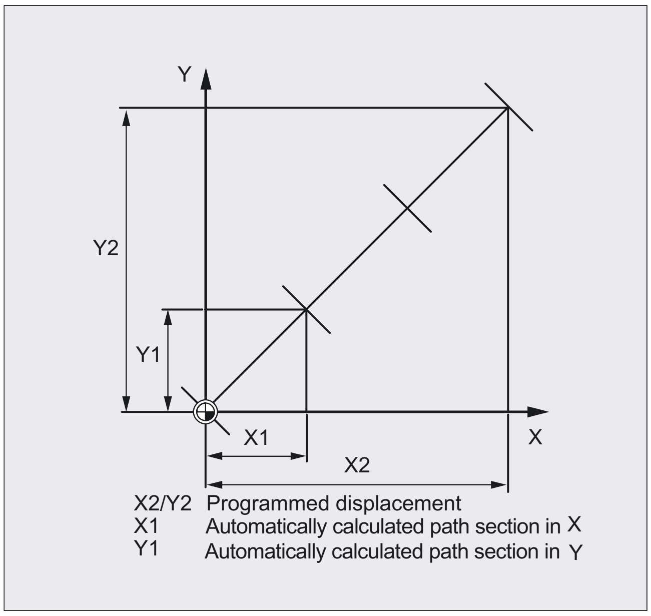
When the total traversing distance is 15 mm and the path segment length 3.5 mm, the quotient is not an integer value (4.28). In this case, the SPP value is reduced down to the next possible integer quotient. The result in this example would be a path segment length of 3 mm.

Number of SPN path segments

`SPN` defines the number of path segments to be generated from the total traversing distance. The length of the segments is calculated automatically. Since `SPN` is non-modal, punching or nibbling must be activated beforehand with `PON` or `SON` respectively.

SPP and SPN in the same block

If you program both the path segment length (`SPP`) and the number of path segments (`SPN`) in the same block, then `SPN` applies to this block and `SPP` to all the following blocks. If `SPP` was activated before `SPN`, then it takes effect again after the block with `SPN`.


Y
Y2
Y1
X1
X2
X
X2/Y2 Programmed displacement
X1 Automatically calculated path section in X
Y1 Automatically calculated path section in Y

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

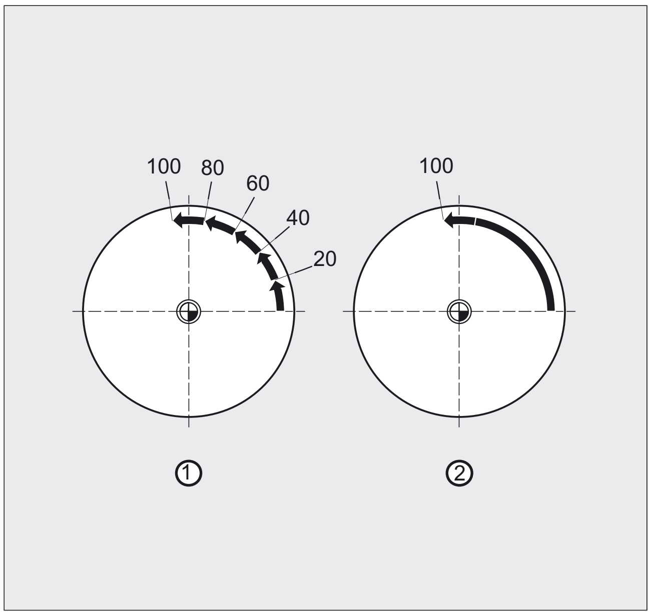
The path traversed by the single axis is distributed evenly among the intermediate blocks generated by `SPP`.

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.



2. Single axis without path segmentation

   The single axis traverses the total distance in the first of the generated blocks.
3. 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.

Additional functions 13

13.1 Axis functions (AXNAME, AX, SPI, AXTOSPI, ISAXIS, AXSTRING)

Function

AXNAME is used, for example, to create generally applicable cycles when the name of the axes are not known (see also the "String functions" section).

SPI is used, for example, when axis functions are used for a spindle, e.g. the synchronized spindle.

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.

Programming

`AXNAME(facing axis)`

or

`AX[AXNAME(String)]`

or

`SPI(n)`

or

`AXTOSPI(X)` or `AXTOSPI(Y)` or `AXTOSPI(Z)`

or

`AXSTRING( SPI(n) )`

or

`ISAXIS(geometry axis number)`

Parameter

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.

SPI extensions

The axis function `SPI(n)` can now also be used for reading and writing frame components, for example, for writing frames with syntax `$P_PFRAME[SPI(1),TR]=2.22`. The additional programming of the axis position using the address `AX[SPI(1)] = <axis position>` can be used to traverse an axis.

AXTOSPI extension

AXTOSPI can be used to convert an axis identifier into a spindle number. If the axis identifier cannot be converted into a spindle number, an alarm message is triggered.

Troubleshooting for AXSTRING[SPI(n)]

For the programming with `AXSTRING[ SPI(n) ]`, the axis index of the axis to which the spindle is assigned will no longer be output as spindle number, but rather the string `"Sn"` will be output.

Example:

`AXSTRING[ SPI(2) ]` returns string `"S2"`

Example

Move the axis defined as a facing axis.

```
OVRA[AXNAME("Transverse axis")]=10          ;Transverse axis
AX[AXNAME("Transverse axis")]=50.2          ;Final position for transverse axis
OVRA[SPI(1)]=70                             ;Override for spindle 1
IF ISAXIS(1) == FALSE GOTOF CONTINUE        ;Does abscissa exist?
AX[$P_AXN1]=100                             ;Move abscissa
CONTINUE:
```

13.2 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 establish 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.

Programming

`STRGINGIS(STRING name) = return value with coding`

In the current configuration, the `(STRING name)` to be checked is always identified using

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.

Parameter

Machine manufacturer

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 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. The STRINGIS NC language command is an integer type variable.
Especially for STRINGIS	**NC cycle names** (an active cycle) **GUD variables** **LUD variables** **Macros** **Label names**
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.

For the behavior of a `STRING` itself, see "String functions".

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.

Scope of NC language	Scope of NC language includes:
	G codes of all existing G code groups such as G0, G1, G2, INVCW, POLY, ROT, KONT, SOFT, CUT2D, CDON, RMB, SPATH
	DIN or NC addresses such as ADIS, RNDM, SPN, SR , MEAS
	NC language functions such as predefined subprograms TANG(Faxis1..n, Laxis1..n, coupling factor).
	NC language procedures (pre-defined procedures **with** return value) such as subprogram with parameter transfer GETMDACT.
	NC language procedures (pre-defined procedures **without** return value) such as deactivate single block suppression SBLOF.
	NC key words such as ACN, ACP, AP, RP, DEFINE, SETMS
	Machine data $MN general, $MA axial, $MC channel-specific as well as all setting data $S... and options data $O... .
	NC system variables $ in the parts program and synchronized actions as well as **NC computing parameters** R.

Return values

Basic information	The return value is coded. The basic information included is sub-divided into y and existing detailed information into x.
Coding:	**Test result,** whether the current configuration includes:
000	The NCK is not aware of the STRING name.
100	The STRING name is a language command but **cannot be programmed**, i.e. this function is inactive.
2xx	The STRING name is a **programmable** 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

Detailed Information	**Significance of the test result:**
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 round 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.

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

T is defined as **auxiliary function** and can always be programmed.	

`400 = STRINGIS("T")`	`;Return value without address`
	`;extension`

`0 = STRINGIS("T3")`	`;Return value with address extension`

Examples of other checks for the programmable scope of NC language 2xx

Xis defined as **axis**	;Axis is a liner 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
INVCW is a defined **G code**	;INVCW is G code evolvent ;interpolation (clockwise).
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
DEFINE is an **NC key word**	;the DEFINE key word exists for ;identification of macros.
205 = STRINGIS("DEFINE")	;DEFINE is present as a key word
the $MC_GCODES_RESET_VALUES is channel-specific **machine data**	;the machine data ;$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.

13.3 ISVAR ( ) function call and read machine 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).

Programming

`ISVAR(variable identifier)`

or

`ISVAR (identifier, [value, value])`

Parameters

Variable identifier	Transfer parameter of type string can be undimensioned, 1-dimensional, or 2-dimensional.
Name of identifier	Identifier with a known variable with or without an array index as machine data, setting data, system variable, or general variable. **Extension:**
	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 of the ISVAR function call

```
DEF INT VAR1
DEF BOOL IS_VAR=FALSE                ;Transfer parameter is a general variable
N10 IS_VAR=ISVAR("VAR1")             ;IS_VAR is TRUE in this case
DEF REAL VARARRAY[10,10]
DEF BOOL IS_VAR=FALSE                ;Different syntax variations
N20 IS_VAR=ISVAR("VARARRAY[,]")      ;IS_VAR is TRUE with a 2-dimensional array
N30 IS_VAR=ISVAR("VARARRAY")         ;IS_VAR is TRUE, variable exists
N40 IS_VAR=ISVAR                     ;IS_VAR is FALSE, array index is not allowed
("VARARRAY[8,11]")
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 allowed
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                    ;IS_VAR is TRUE
("$MC_GCODE_RESET_VALUES[1]"
DEF BOOL IS_VAR=FALSE                ;Transfer parameter is a system variable
N10 IS_VAR=ISVAR("$P_EP")            ;IS_VAR is TRUE in this case
N10 IS_VAR=ISVAR("$P_EP[X]")         ;IS_VAR is TRUE in this case
```

Example of reading a machine data array both 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]`

as previous, this corresponds to

`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`

as previous, this corresponds to

`WHEN TRUE DO $R1 = $MC_EXTERN_GCODE_RESET_VALUES[0]`

and would previously **not be read** with alarm 12400.

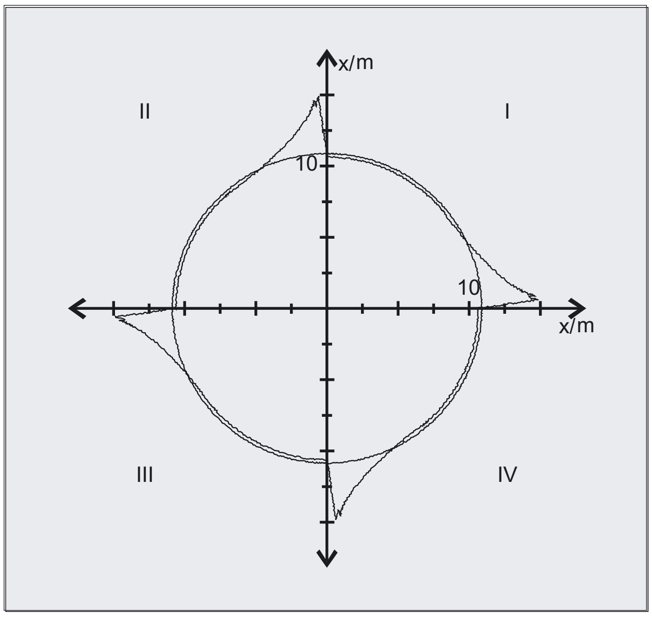
The alarm 12400 will still be issued for

`R1=$MA_POSTCTRL_GAIN`

13.4 Learn compensation characteristics (QECLRNON, QECLRNOF)

Function

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.



Programming

`QECLRNON`

or

`QECLRNOF`

Activate the learning process: QECLRNON

The actual learning process is activated in the NC program with the command `QECLRNON` and specification of the axes:

`QECLRNON (X1, Y1, Z1, Q)`

Only if this command is active are the quadrants changed.

Deactivate the learning process: QECLRNOF

When the learning movements for the desired axes are complete, the learning process is deactivated simultaneously for all axes with `QECLRNOF`.

Parameters

`QECLRNON (axis.1,…4)`	Activate "Learn quadrant error compensation" function
`QECLRNOF`	Deactivate "Learn quadrant error compensation" function
`QECLRN.SPF`	Learning cycle
`QECDAT.MPF`	Sample NC program for assigning system variables and the parameters for the learning cycle
`QECTEST.MPF`	Sample NC program for circle shape 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.

13.5 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

.

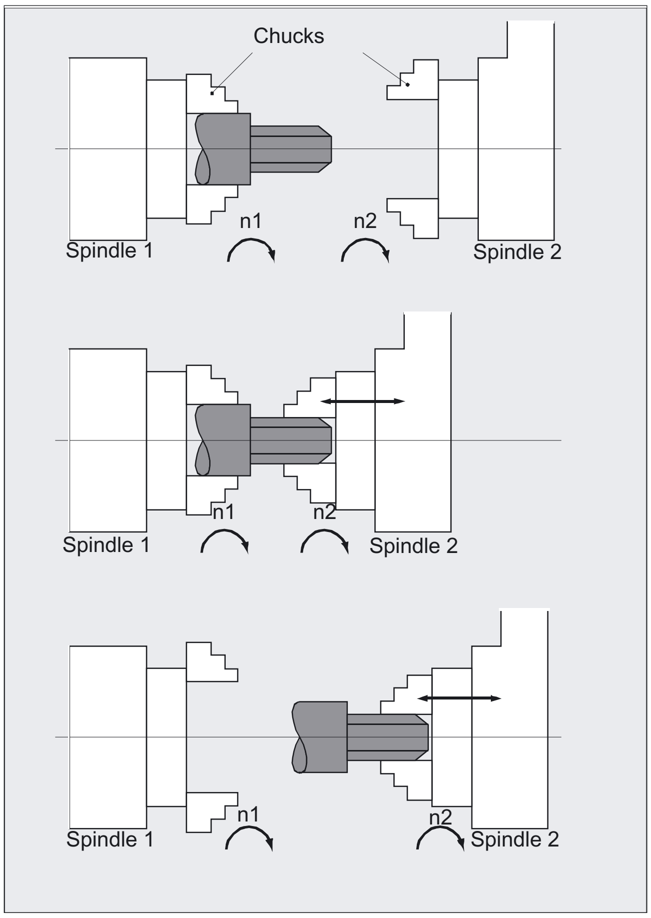
13.5.1 Synchronous spindle (COUPDEF, COUPDEL, COUPON/ONC, COUPOF/OFS, COUPRES)

Function

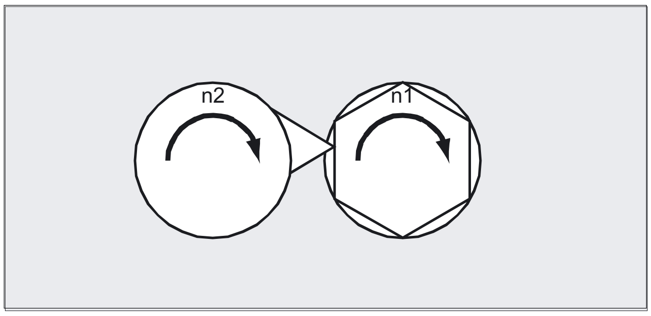
The synchronous spindle function enables turning machines to perform workpiece transfer from spindle 1 to spindle 2 on-the-fly, e.g. for final machining. This avoids downtime caused, for example, by rechucking.

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).



Programming

```
COUPDEF(FS, LS, TFS, TLS, 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 main 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.

Parameters

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 once the switch-off position is crossed: COUPOF(S2, S1, POS_{FS}) Switch-off positions: COUPOF(S2, S1, POS_{FS}, POS_{LS})
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 once the switch-off position is crossed: COUPOFS(S2, S1, POS_{FS})
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
T_{FS}, T_{LS}	Speed ratio parameters for FS = numerator and LS = denominator
	Default setting = 1.0; specification of denominator optional
Block change behavior:	Block change method; Block change is implemented:
"NOC"	Immediately
"FINE"	At "Synchronism fine"
"COARSE"	At "Synchronism coarse"
"IPOSTOP"	in response to IPOSTOP (e.g. after setpoint-based synchronism) (presetting)
	The block change method is modal
Coupling type	Coupling type: Coupling between FS and LS
"DV"	Setpoint linkage (default)
"AV"	Actual value coupling
"VV"	Speed coupling
	The coupling type is modal.
POS_{FS}	Angle offset between leading and following spindles
POS_{FS}, POS_{LS}	Switch-off positions of following and main spindles "The block change is enabled once POS_{FS}, POS_{LS} has been crossed"

Example of working with master and slave spindles.

```
                                        ;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",       ;Def. of coupling, can also be configured
"Dv")
…
N70 SPCON                               ;Include master spindle in position control
                                        ;(setpoint coup.)
N75 SPCON(2)                            ;Include slave spindle in 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
                                        ;positive direction
N220 G4 S50                             ;Dwell time = 50 revolutions of master
                                        ;spindle
N225 FA [S2] = 0                        ;Activate configured speed (MD)
N230 SPOS[2] = IC (-7200)               ;20 rev. With configured speed in
                                        ;negative direction
…
```

```
N350 COUPOF (S2, S1)                    ;Decouple on-the-fly, S=S2=3000
N355 SPOSA[2] = 0                       ;Stop slave spindle at zero degrees
N360 G0 X0 Y0
N365 WAITS(2)                           ;Wait for spindle 2
N370 M5                                 ;Stop slave spindle
N375 M30
```

Example of programming of difference in speed

```
                                        ;Leading spindle = master spindle =
                                        ;spindle 1
                                        ;Following spindle = spindle 2
N01 M3 S500                             ;Master spindle rotates at 500 rpm
N02 M2=3 S2=300                         ;Following spindle rotates at 300 rpm
…
N10 G4 F1                               ;Dwell time of master spindle
N15 COUPDEF (S2, S1, -1)                ;Coupling factor with speed 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 of difference in speed
```

Examples of transfer of a movement for difference in speed

1. Activate coupling during previous programming of following spindle with `COUPON`

```
                                        ;Leading spindle = master spindle =
                                        ;spindle 1
                                        ;Following spindle = spindle 2
N05 M3 S100 M2=3 S2=200                 ;Master 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)                 ;Speed ratio of following spindle to
                                        ;main spindle is 1.0 (presetting)
N20 COUPON (S2, S1)                     ;On-the-fly coupling to main spindle
N10 G4 F5                               ;Following spindle rotates at 100 rpm
```

2. Activate coupling during previous programming of following spindle with `COUPONC`

```
                                        ;Leading spindle = master spindle =
                                        ;spindle 1
                                        ;Following spindle = spindle 2
N05 M3 S100 M2=3 S2=200                 ;Master 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)                 ;Speed ratio of following spindle to
                                        ;main spindle is 1.0 (presetting)
N20 COUPONC (S2, S1)                    ;On-the-fly coupling to main 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`

```
                                         ;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)                  ;Speed ratio of following spindle to
                                         ;main spindle is 1.0 (presetting)
N20 COUPON (S2, S1)                      ;On-the-fly coupling to main spindle
N10 G4 F1                                ;Coupling is closed,
                                         ;S2 remains at 20 degrees
```

4. Activate coupling with following spindle stationary with `COUPONC`

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, T_{FS}, T_{LS}, 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, T_{FS}, T_{LS})

Meaning:

S2 = following spindle, S1 = leading spindle

Speed ratio SR_T

The speed ratio is defined with parameters for `FS` (numerator) and `LS` (denominator).

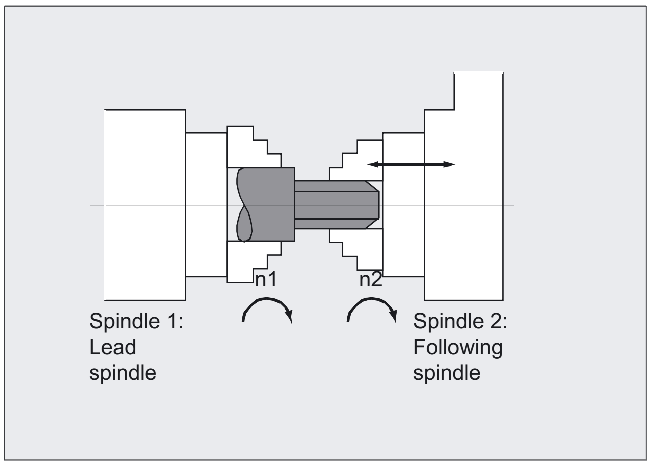
Options:

- 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 `S2` and the leading spindle `S1` 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:

"**NO**C" immediate (default)

"**F**INE" for "fine synchronism"

"**CO**ARSE" for "coarse synchronism"

"**I**POSTOP" 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_{FS}

- Fastest possible activation of coupling with any angle reference between `LS` and `FS`:

```
N ... COUPON(S2, S1)or
N ... COUPON(S2, S1, POSFS)or
N ... COUPON(S2)
```

- Activation with angular offset `POSFS`

  Position-synchronized coupling for profiled workpieces.
  POS_{FS} refers to the 0° position of the lead spindle in the positive direction of rotation
  `POSFS` 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 ±180° 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 Sn=... or Mn=3, Mn=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$_{FS}$ and, where appropriate, POS$_{LS}$ 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$_{FS}$ have been crossed.
  Value range 0° ... 359.999°:

```
COUPOFS(S2, S1, POSFS)
```

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 presettings (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_OFFS[S2]
```

The actual value for the current position offset can be read with:

```
$VA_COUP_OFFS[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.

13.6 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.

13.6.1 Defining an electronic gear (EGDEF)

Function

An EG axis grouping is defined by specifying the following axis and a minimum of one and a maximum of five leading axes with the respective coupling type:

EGDEF(following axis, leading axis1, coupling type1, leading axis2, coupling type 2,...).

Requirements

Preconditions for defining an EG axis grouping: A following axis must not yet be defined for the coupled axes (if necessary, delete any existing one with EGDEL first).

Programming

`EGDEF(C, B, 1, Z, 1, Y, 1)`	B, Z, Y influence C via setpoint

The coupling type does not need to be the same for all leading axes and must be programmed separately for each individual master. The coupling factors are preset to zero when the EG axis grouping is defined.

Note

`EGDEF` triggers preprocessing stop. The gear definition with `EGDEF` should also be used unaltered when one or more leading axes affect the following axis via a **curve table**.

Parameters

EGDEF	Definition of an electronic gear
Following axis	Axis that is influenced by the leading axes
Leading axis 1, ...5	Axes that influence the following axis
Coupling type 1, ...5	Following axis is influenced by: 0: Actual value 1: Setpoint of the respective leading axis

13.6.2 Activate electronic gear (EGON)

Function

There are three variants for the activation command.

Programming

Variant 1:

The EG axis group **without** synchronization-selective will be activated with:
```
EGON(FA, "block change mode", LA1, Z1 ,N1, LA2, Z2, N2, ..LA5,
Z5,N5)
```
Variant 2:

The EG axis group **with** synchronization-selective will be activated with:
```
EGONSYN(FA, "block change mode", SynPosFA,[, LAi, SynPosLAi, Zi,
Ni])
```
Variant 3:

The EG axis grouping is activated selectively **with** synchronization. The **approach mode** is specified with:
```
EGONSYNE(FA, "Block change mode", SynPosFA, approach mode[, LAi,
SynPosLAi, Zi, Ni])
```

Parameters

Variant 1:

FA	Following axis
Block change mode	The following modes can be used:
	"NOC" block change takes place immediately
	"FINE" block change is performed for "synchronism fine"
	"COARSE" block change is performed for "synchronism coarse"
	"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.

Variant 2:

FA	Following axis
Block change mode	The following modes can be used:
	"NOC" block change takes place immediately
	"FINE" block change is performed for "synchronism fine"
	"COARSE" block change is performed for "synchronism coarse"
	"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	Counter for coupling factor i
N1, ... N5	Denominator for coupling factor i
	Coupling factor i = Counter i / Denominator i

Only leading axes previously specified with the `EGDEF` command may be programmed in the activation line. Through the programmed "Synchronized positions" for the following axis (`SynPosFA`) and for the leading axes (`SynPosLA`), positions are defined for which the axis grouping is interpreted as *synchronous*. If the electronic gear is not in the synchronized state when the grouping is switched on, the following axis traverses to its defined synchronized position.

Variant 3:
The parameters are the same as for variant 2 as regards:

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 with respect to programmed synchronized position
	"DCP" Path-optimized with respect to programmed synchronized 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.

Description

Variant 1:

The positions of the leading axes and following axis at the instant the grouping is switched on are stored as "Synchronized positions". The "Synchronized positions" can be read with the system variable `$AA_EG_SYN`.

Variant 2:

If modulo axes are contained in the coupling group, their position values are modulus-reduced. This ensures that the next possible synchronized position is approached (so-called *relative synchronization*: e.g. the next tooth gap). The synchronized position is only approached if "Enable following axis override" interface signal DB(30 + axis number), DBX 26 bit 4 is issued for the following axis. If it is not issued, the program stops at the EGONSYN block and self-clearing alarm 16771 is output until the above mentioned signal is set.

Variant 3:

The tooth distance (deg.) is calculated like this: 360 * Zi/Ni. If the following axis is stopped at the time of calling, path optimization returns responds identically to time optimization.

If the following axis is already in motion, `NTGP` will synchronize at the next tooth gap irrespective of the current velocity of the following axis. If the following axis is already in motion, `NTGT` will synchronize at the next tooth gap depending on the current velocity of the following axis. The axis is also decelerated, if necessary.

Curve tables

If a **curve table** is used for one of the leading axes:

Ni	The denominator of the coupling factor for linear coupling must be set to 0. (Denominator 0 would be illegal for linear couplings.) 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:
/FB3/ 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 parts program can determine the current states of an EG axis grouping and react to them if required.

The system variables for the electronic gear are listed in the Annex. They are identified with names that begin with:

`$AA_EG_ ...`

or

`$VA_EG_ ...`

13.6.3 Deactivate electronic gear (EGOFS)

Function

There are three different ways to deactivate an active EG axis grouping.

Programming

Variant 1:

`EGOFS(following axis)`	The electronic gear is deactivated. The following axis is braked to a standstill. This call triggers a preprocessing stop.

Variant 2:

`EGOFS(following axis, leading axis1, … leading axis5)`	This command parameter setting made it possible to **selectively** remove the influence of the individual leading axes on the following axis' motion.

At least one leading axis must be specified. The influence of the specified leading axes on the slave is selectively inhibited. This call triggers a preprocessing stop. If the call still includes active leading axes, then the slave continues to operate under their influence. If the influence of all leading axes is excluded by this method, then the following axis is braked to a standstill.

Variant 3:

`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 function is only allowed for spindles.

Deleting the definition of an electronic gear

An EG axis grouping must be switched off before its definition can be deleted.

`EGDEL(following axis)`	The defined coupling of the axis grouping is deleted. Additional axis groupings can be defined by means of EGDEF until the maximum number of simultaneously activated axis groupings is reached. This call triggers a preprocessing stop.

13.6.4 Revolutional 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.

13.7 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).

13.7.1 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_ESR_REACTION[X] = 11,
   $MA_ESR_REACTION[Y] = 12,
   $MA_ESR_REACTION[Z] = 12,
   $MA_ESR_REACTION[A] = 10;
   ```
3. Drive configuration:
   `MD 1639: RETRACT_SPEED[X] = 400000H` in pos. direction (max. speed),
   `= FFC00000H` 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
  ```

13.7.2 NC-controlled reactions to retraction

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 parts program. The activate signals must be set for the retraction movement and remain set.

Programming

`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)`
`POLFA(axis, 0/1/2)type)`
	Abbreviated form for single axis retraction

	high-speed deactivation / activation
`POLFA(axis, 0, $AA_POLFA[axis])`	causes **a** preprocessing stop
`POLFA(axis, 0)`	does **not** cause a preprocessing stop
`POLFMASK(axisname1, axisname2, ...)`	Axis selection for the retraction
	unconnected axes
`POLFMLIN(axisname1, axisname2, ...)`	Axis selection for the retraction
	linearly connected axes

Notice

If the type is changed when using the abbreviated forms of `POLFA`, the user must ensure that either the retraction position or the retraction path are assigned a meaningful value. In particular, the retraction position and the retraction path have to be set again after Power On.

Parameters

`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 **identical**, 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 command POLF is modal.
`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 **POLFMASK** command enables the specified axes for retraction - without a connection between axes.
	The command **POLFMASK()** without any axis parameter deactivates fastlift for all axes which were retracted without any connection between axes.
`POLFMLIN,`	The **POLFMLIN** command enables the specified axes for retraction - with a linear connection between axes.
	The command **POLFMLIN()** without any axis parameter deactivates fastlift for all axes which were retracted with a linear connection between axes.
`axisnamei`	Names of the axes that are to travel to positions defined with POLF in case of LIFTFAST. All the axes specified must be in the same coordinate system. Before fastlift to a fixed position can be activated via **POLFMASK** or **POLFMLIN**, a position must be programmed with **POLF** for the selected axes. No machine data is provided for presetting 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 in succession with `POLFMASK`, `POLFMLIN` or `POLFMLIN`, `POLFMASK`, the last definition always applies to each axis.

Caution

The positions programmed with `POLF` and the activation by `POLFMASK` or `POLFMLIN` are deleted when the parts program is started. This means that the user must reprogram in each part program the values for `POLF` and the selected axes in `POLFMASK` or `POLFMLIN`.

For more information on changing the coordinate system, the effect on modulo rotary axes, etc. see
/FB3/ Function Manual, Special Functions; Coupled axes and ESR (M3).

Example of the extended retraction of a single axis:

```
MD 37500: ESR_REACTION[AX1] = 21           ;NC-controlled retraction
...
$AA_ESR_ENABLE[AX1] = 1
POLFA(AX1,1, 20.0)                         ;AX1 becomes the axial retraction
                                           ;position 20.0 assigned (absolutely)
$AA_ESR_TRIGGER[AX1] = 1                   ;Retraction starts here.
```

Requirements

Retract

- the axes selected with `POLFMASK` or `POLFMILIN`,
- 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: ESR_REACTION = 21
- `LFPOS` from the modal 46. G code group.

Enable and start NC-controlled reactions

If system variable `$AC_ESR_TRIGGER = 1` is set, and if a retract axis is configured in this channel (i.e. MD 37500: ESR_REACTION = 21) and `$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.

13.7.3 NC-controlled reactions to stoppage

Function

Stop

The sequence for extended stop (NC-controlled) is specified in the following machine data:

MD 21380: ESR_DELAY_TIME1 and
MD 21381: ESR_DELAY_TIME2.

This axis continues interpolating as programmed for the time duration set in MD 21380. After the time delay specified in MD 21380 has lapsed, controlled braking is initiated by interpolation. The maximum time available for the interpolatory controlled braking is specified in MD 21381; after this time has lapsed, rapid deceleration with subsequent correction is initiated.

Enable and start NC-controlled stop

The NC-controlled stop is

configured: via MD 37500: 22 and 2 times using the two MD, see above;

enabled ($AA_ESR_ENABLE) and

started: System variable $AC_ESR_TRIGGER with $AA_ESR_TRIGGER for single axes.

Example of stopping a single axis:

```
MD 37500: ESR_REACTION[AX1] = 22                ;NC-controlled stop
MD 21380: ESR_DELAY_TIME1[AX1] = 0.3
MD 21381: ESR_DELAY_TIME2[AX1] = 0.06
...
$AA_ESR_ENABLE[AX1] = 1
$AA_ESR_TRIGGER[AX1] = 1                        ;Stopping starts here.
```

13.7.4 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).

13.7.5 Drive-independent stopping

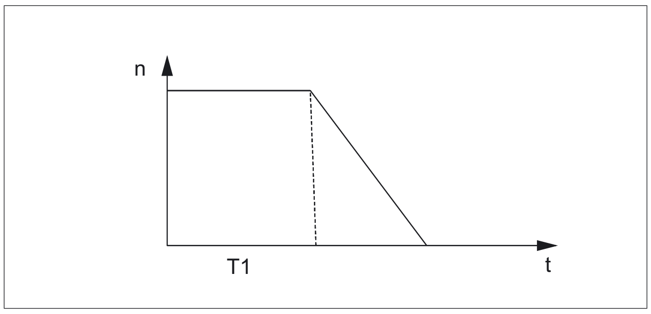
Function

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).

13.7.6 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).

13.8 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).

13.8.1 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 **parts 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.

Parameters

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 `$A_DLB[i]` ;data byte (8 bits)
- INT `$A_DLW[i]` ;data word (16 bits)
- INT `$A_DLD[i]` ;double data word (32 bits)
- REAL `$A_DLR[i]` ;real data (64 bits)

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: -2147483648 to 2147483647

REAL: -4.19e-308 to 4.19e-307

Example

```
$A_DLB[5]=21         The 5th byte in the shared link memory is assigned value 21.
```

13.9 Axis container (AXCTWE, AXCTWED)

Function

On rotary indexing machines/multi-spindle machines, the work-holding axes 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 provided for this purpose.

Only one workpiece clamping axis/spindle is active on the local machining unit at a time. The axis container provides the possible connections to all clamping axes/spindles, of which exactly **one** is **activated** for the machining unit.

Programming

The entries in the axis container can be switched by increment n via the commands:

`AXCTSWE(CTi)`	AXIS CONTAINER SWITCH ENABLE
`AXCTSWED(CTi)`	AXIS CONTAINER SWITCH ENABLE DIRECT

Parameters

AXCTSWE	For each channel, release for a container rotation the axes entered in the container.
AXCTSWED	Under the sole effect of the active channel, the axis container rotates around the stored increment. The axes entered in the container will be enabled when the other channels that have axes in the container are in the RESET state.
CT_i or	The number of the axis container whose contents are to be switched or
e.g., A_CONT1	individual name of axis container set via MD.

Axis container

The following can be assigned via the axis container:

- Local axes and/or
- Link axes (see Fundamentals)

The available axes that are defined in the axis container can be changed by switching the entries in the axis container.

The modification can be triggered by the **parts program**.

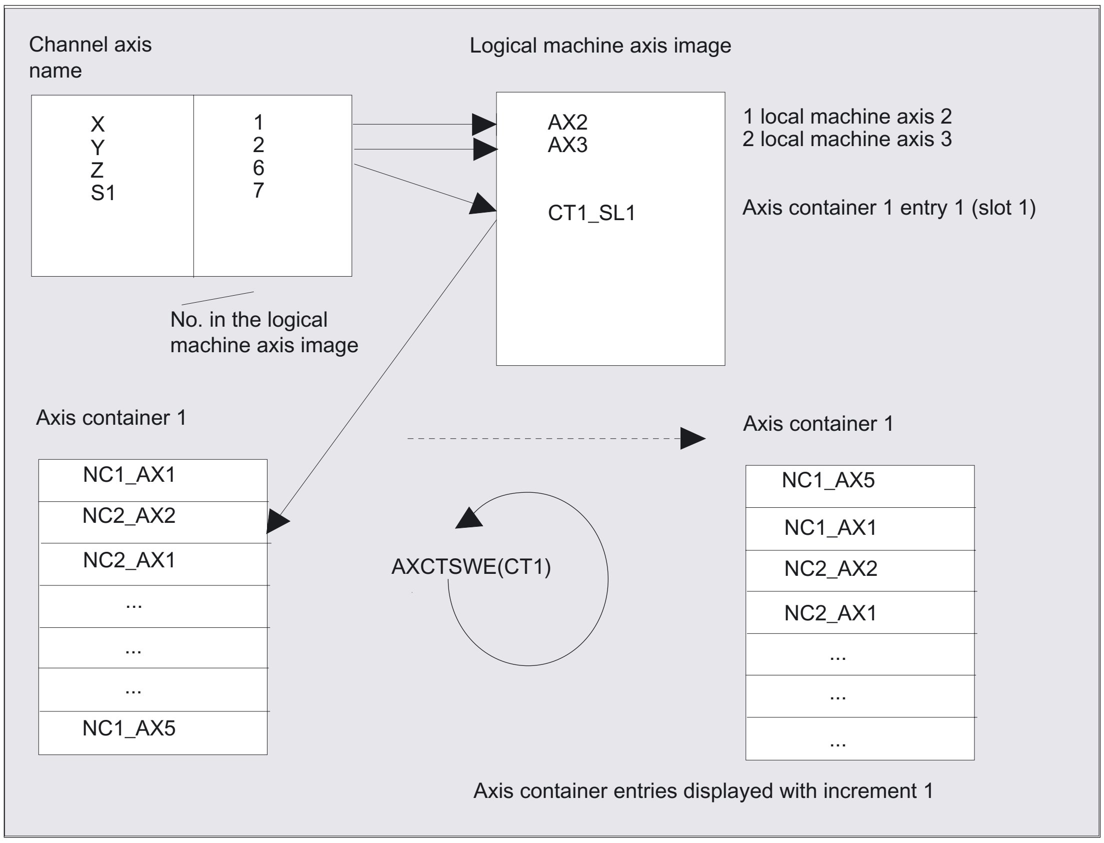
Axis containers with link axes are a NCU-cross device (NCU-global) that is coordinated via the control. It is also possible to have axis containers that are only used for managing local axes.

For detailed information on configuring axis containers, see
/FB2/ Function Manual Extended Functions; Multiple Operator Panels and NCUs (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. Once the control receives the enables from **all** channels for the axes in the container, the container is rotated with the increment specified in the SD.



In the preceding example, after axis container rotation by 1, axis AX5 on NCU1 is assigned to channel axis Z instead of axis AX1 on NCU1.

AXCTSWED ()

The command variant AXCTSWED(CT_i) can be used to simplify startup. Under the sole effect of the active channel, the axis container rotates around the increment stored in the SD. This call may only be used if the other channels, which have axes in the container are in the RESET state.

After an axis container rotation, all NCUs whose channels refer to the rotated axis container via the logical machine axis image are affected by the new axis assignment.

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/GETD. The assignment of the axes cannot be cleared until the axis container revolution is complete.

Machine manufacturer

This behavior can be set using a machine data bit. Please refer to the machine manufacturer's instructions.

Note

Axis container revolution with implicit GET/GETD cannot be used for an axis assigned as a main run axis, e.g., for a PLC axis, as this axis would have to quit main run status for the purpose of axis container revolution.

13.10 Program runtime/Workpiece counter

13.10.1 General

Information about the program runtime is provided to assist the operator on the machine tool.

This information is specified in the respective machine data and can be edited as a system variable in the NC and/or PLC program. This information is also available to the HMI on the operator control panel interface.

13.10.2 Program runtime

Function

Under the program runtime function, timers are provided as system variables, which can be used to monitor technological processes.

These timers can only be read. It can be accessed at any time by the HMI in read mode.

Parameters

The following two timers are defined as NCK-specific system variables and are always active.

System variables

`$AN_SETUP_TIME`	`Time in minutes since the last setup;`
	`is reset with SETUP`
`$AN_POWERON_TIME`	`Time in minutes since the last PowerOn;`
	`is reset with POWERON`

The following three timers are defined as channel-specific system variables and can be activated via machine data.

`$AC_OPERATING_TIME`	`Total execution time in seconds of NC programs in the automatic mode`
`$AC_CYCLE_TIME`	`Execution time in seconds of the selected NC program`
`$AC_CUTTING_TIME`	`Tool operation time in seconds`
`$MC_RUNTIMER_MODE`	`Tool operation time in seconds`

Note

All timers are reset with default values when the control is powered up, and can be read independent of their activation.

Example

```
1. Activate runtime measurement for the active NC program; no measurement
 with active dry run feedrate and program testing:
$MC_PROCESSTIMER_MODE = 'H2'
2. Activate measurement for the tool operating time; measurement also with
active dry run feedrate and program testing:
$MC_PROCESSTIMER_MODE = 'H34'
3. Activate measurement for the total runtime and tool operating time;
 measurement also during program testing:
$MC_PROCESSTIMER_MODE = 'H25'
```

13.10.3 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.

Parameters

The following counters are available:

system variables

$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 reset with default values when the control is powered up, and can be read/written independent of their activation.

Example

```
Activate workpiece counter $AC_REQUIRED_PARTS:
$MC_PART_COUNTER=’H3’                          $AC_REQUIRED_PARTS is active,
                                               display alarm on $AC_REQUIRED_PARTS
                                               == $AC_SPECIAL_PARTS
Activate workpiece counter $AC_TOTAL_PARTS:
$MC_PART_COUNTER='H10'                         $AC_TOTAL_PARTS is active, the
$MC_PART_COUNTER_MCODE[0]=80                   counter is incremented by 1 on each
                                               M02, $MC_PART_COUNTER_MCODE[0] is
                                               irrelevant
Activate workpiece counter $AC_ACTUAL_PARTS:
$MC_PART_COUNTER='H300'                        $AC_TOTAL_PARTS is active, the
$MC_PART_COUNTER_MCODE[1]=17                   counter is incremented by 1 on each
                                               M17
Activate workpiece counter $AC_SPECIAL_PARTS:
$MC_PART_COUNTER='H3000'                       $AC_SPECIAL_PARTS is active, the
$MC_PART_COUNTER_MCODE[2]=77                   counter is incremented by 1 on each
                                               M77
Deactivate workpiece counter $AC_ACTUAL_PARTS:
$MC_PART_COUNTER='H200'                        $AC_TOTAL_PARTS is not active, rest
$MC_PART_COUNTER_MCODE[1]=50                   irrelevant
Activating all counters in examples 1-4:
$MC_PART_COUNTER = 'H3313'                     $AC_REQUIRED_PARTS is active
$MC_PART_COUNTER_MCODE[0] = 80                 Display alarm on $AC_REQUIRED_PARTS
$MC_PART_COUNTER_MCODE[1] = 17                 == $AC_SPECIAL_PARTS
$MC_PART_COUNTER_MCODE[2] = 77                 $AC_TOTAL_PARTS is active, the
                                               counter is incremented by 1 on each
                                               M02
                                               $MC_PART_COUNTER_MCODE[0] is
                                               irrelevant
                                               $AC_ACTUAL_PARTS is active, the
                                               counter is incremented by 1 on each
                                               M17
                                               $AC_SPECIAL_PARTS is active, the
                                               counter is incremented by 1 on each
                                               M77
```

13.11 Interactive window call from parts program, command:

Function

You can use the MMC command to display user-defined dialog windows (dialog displays) on the HMI from the parts program.

The dialog window appearance is defined in a pure text configuration (COM file in cycles directory), while the HMI system software remains unchanged.

User-defined dialog windows cannot be called simultaneously in different channels.

Programming

```
MMC(CYCLES, PICTURE_ON, T_SK.COM, BILD, MGUD.DEF, BILD_3.AWB,
TEST_1, A1", "S")
```

Please see the detailed notes on how to program the MMC command (incl. programming examples) in /IAM/ in manuals AE1, BE1, HE1, IM2, IM4, and IM5, as appropriate for the HMI software used.

Parameters

MMC	Calling the dialog window interactively from the parts program on the HMI.
CYCLES	Operating area in which the configured user dialog boxes are implemented.
PICTURE_ON or PICTURE_OFF	Command: Display selection or display deselection.
T_SK.COM	Com file: Name of the dialog display file (user cycles). The dialog display design is defined here. The dialog screen is used to display user variables and/or comment texts.
DISPLAY	Name of dialog display: The individual displays are selected via the names of the dialog displays.
MGUD.DEF	User data definition file, which is addressed while reading/writing variables.
PICTURE_3.AWB	Graphics file
TEST_1	Display time or acknowledgement variable.
A1	Text variables...",
"S"	Acknowledgement mode: synchronous, acknowledgement via "OK" soft key.

13.12 Influencing the motion control

13.12.1 Percentage jerk correction (JERKLIM)

Function

In critical program sections, it may be necessary to limit the jerk to below maximum value, for example, to reduce mechanical stress. The acceleration mode SOFT must be active. The function only effects path axes.

Programming

```
JERKLIM[axis]= ...
```

Parameters

`JERKLIM`	Percentage change for the greatest permissible jerk relative to the value set in the machine data for the axis.
`Axis`	Machine axis whose jerk limit has to adapted.
`Value range: 1 ... 200`	100 corresponds to: no effect on the jerks. 100 is applied after RESET and parts program start.

Example

In the AUTOMATIC modes, the jerk limit is limited to the percentage of the jerk limit stored in the machine data.

```
N60 JERKLIM[X]=75
```

Meaning: The axis carriage in the X direction must be accelerated/decelerated with only 75% of the jerk permissible for the axis.

Note

Another example in provided in the section "Percentage velocity correction (`VELOLIM`)".

13.12.2 Percentage velocity correction (VELOLIM)

Function

In critical program sections, it may be necessary to limit the velocity to below maximum values, for example, to reduce mechanical stress or enhance finish. The function only effects path and positioning axes.

Programming

```
VELOLIM[axis]= ...
```

Parameters

VELOLIM	Percentage change for the greatest permissible velocity relative to the value set in the machine data for the axis
Axis	Machine axis whose velocity limit has to adapted
Value range: 1 ... 100	100 corresponds to: no effect on the velocity. 100 is applied after RESET and parts program start.

VELOLIM example

In the AUTOMATIC modes, the velocity limit is limited to the percentage of the velocity limit stored in the machine data.

```
N70 VELOLIM[X]=80
```

Meaning: The axis carriage in the X direction must travel at only 80% of the velocity permissible for the axis.

VELOLIM and JERKLIM example

```
N1000 G0 X0 Y0 F10000 SOFT G64
N1100 G1 X20 RNDM = 5 ACC[X] = 20
ACC[Y]=30
N1200 G1 Y20 VELOLIM[X]=5
JERKLIM[Y]=200
N1300 G1 X0 JERKLIM[X]=2
N1400 G1 Y0
M30
```

13.13 Master/slave grouping (MASLDEF, MASLDEL, MASLOF, 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.

Programming

`MASLON(Slv1, Slv2, ..., )`	
`MASLOF(Slv1, Slv2, ..., )`	
`MASLDEF(Slv1, Slv2, ..., master axis)`	Extension for dynamic configuration
`MASLDEL(Slv1, Slv2, ..., )`	Extension for dynamic configuration
`MASLOFS(Slv1, Slv2, ..., )`	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.

Parameters

General

MASLON	Activate a temporary coupling.
MASLOF	Disconnect an active coupling. The extensions for spindles must be observed.

Dynamic configuration extension

MASLDEF	Coupling user-defined using machine data or also create/change from the parts program.
MASLOFS	Disconnect the coupling analog to MASLOF and automatically decelerate the slave spindle.
MASLDEL	Uncouple master/slave axis grouping and clear grouping definition.
Slv1, Slv2, ...	Slave axes led by a master axis.
Master axis	Axis leading slave axes defined in a master/slave grouping.

Example of the dynamic configuration of a master/slave coupling

Dynamic configuration of a master/slave coupling from the parts program:

The axis relevant after axis container rotation must become the master axis.

```
MASLDEF(AUX,S3)                            ;S3 master for AUX
MASLON(AUX)                                ;Coupling in for AUX
M3=3 S3=4000                               ;Clockwise rotation
MASLDEL(AUX)                               ;Clear configuration and
                                           ;disconnect the coupling
AXCTSWE(CT1)                               ;Container rotation
```

Example of the actual-value coupling of a slave axis

Actual-value coupling of a slave axis set to the same value as the master axis with `PRESETON`.

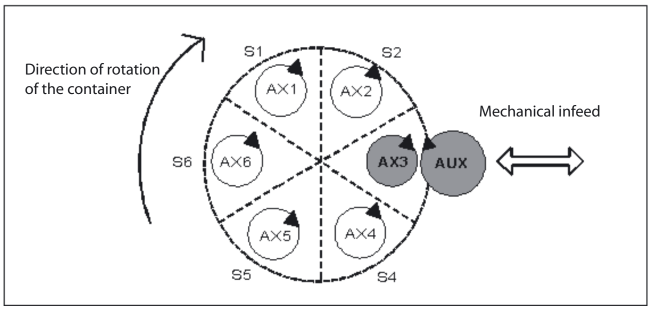
In a permanent master/slave coupling, the actual value on the SLAVE axis is to be changed by `PRESETON`.

```
N37262                                     ;Activate permanent coupling
$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=0
N37263 NEWCONF
N37264 STOPRE
MASLOF(Y1)                                 ;Temporary coupling off
N5 PRESETON(Y1, 0, Z1, 0, B1, 0, C1, 0,    ;Set actual value of the unreferenced
U1, 0)                                     ;slave axes because they are activated
                                           ;on Power on
N37262                                     ;Activate permanent coupling
$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=1
N37263 NEWCONF
```

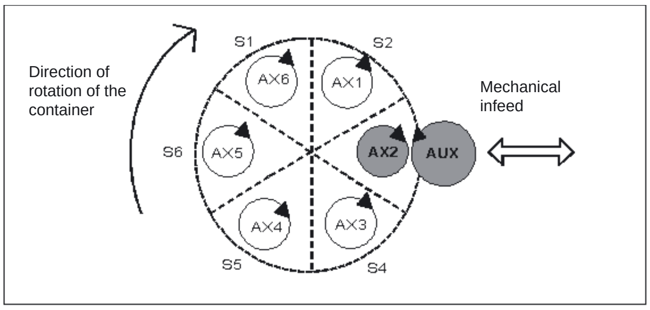
Example of a coupling sequence Position 3 / Container CT1

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:

/FB2/ Function Manual, Extension Functions; Several Operator Panel Fronts and NCUs (B3), Section "Axis container "

Description

General

MASLOF	This statement is executed directly for spindles in speed control mode. The slave spindles rotating at this time retain their speeds until next speed programming.

Dynamic configuration extension

MASLDEF	Definition of a master/slave grouping from the parts program: Previously, the definition was defined exclusively via machine data.
MASLDEL	The statement revokes the assignment of the slave axes to the master axis and disconnects at the same time, analog to MASLOF, the coupling. The master/slave definitions specified in the machine data are retained.
MASLOFS	MASLOFS can be used to decelerate slave spindles automatically when disconnecting the coupling. For axes and spindles in positioning mode, the coupling can only be closed and disconnected while stopped.

Note

For the slave axis, the actual value can be synchronized to the same value of the master axis with `PRESETON`. For this purpose, permanent master/slave coupling must be deactivated briefly to set the actual value of the unreferenced slave axis to the value of the master axis with Power On. After that, the permanent coupling is restored.

The permanent master/slave coupling is activated with MD 37262: MS_COUPLING_ALWAYS_ACTIVE = 1 and does not have any affect on the commands of the temporary coupling.

Coupling characteristics for spindles, SW 6.5 and higher

For spindles in the speed control mode, the coupling characteristics for `MASLON`, `MASLOF`, `MASLOFS` and `MASLDEL` are explicitly specified using the MD 37263: MS_SPIND_COUPLING_MODE.

In the default setting with MD 37263 = 0, the coupling and separation of the slave axes are performed only when the associated axes are stopped. `MASLOFS` corresponds to the `MASLOF`.

For MD 37263 = 1, the coupling statement is performed immediately, and thus also in the motion. The coupling will be closed immediately for `MASLON` and immediately separated for `MASLOFS` or `MASLOF` . The slave spindles turning at this time will be automatically decelerated for `MASLOFS` and for `MASLOF` retain their speed until a new speed programming is made.

14 User stock removal programs

14.1 Supporting function for stock removal

Function

Preprogrammed stock removal programs are provided for stock removal. You can also use the following functions to develop your own stock removal programs.

Note

You can use these functions universally, not just for stock removal.

Prerequisite

Before CONTPRON or CONTDCON is called

- a starting point must be approached which permits collision-free machining,
- tool edge radius compensation with G40 must be deactivated.

Programming

`CONTPRON`

or

`CONTDCON`

with

`INTERSEC`

or

`ISPOINTS`

or

`EXECTAB`

or

`CALCDAT`

Terminate contour preparation

`EXECUTE (ERROR)`

Parameters

CONTDCON	Activate tabular contour decoding (6 columns)
CONTPRON	Activate tabular contour preparation (11 columns)
INTERSEC	Calculate the intersection of two contour elements. (Only for tables created by CONTPRON).
ISPOINTS	Calculate the possible intersections of two contour elements. (Only for tables created by CONTPRON).
EXECTAB	Non-modal processing of the contour elements of a table (Only for tables created by CONTPRON).
CALCDAT	Calculate the radiuses and centers of a circle that consists of 3 or 4 points.
EXECUTE	Terminate contour preparation
ERROR	Variable for error checkback, type INT
	1 = error; 0 = no error

`EXECUTE` deactivates the contour preparation and switches back to the normal execution mode.

Example:

```
N30 CONTPRON(...)
N40 G1 X... Z...
N50 ...
N100 EXECUTE(...)
```

14.2 Contour preparation (CONTPRON)

Function

The blocks executed after CONTPRON describe the contour to be prepared. The blocks are not processed but are filed 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.

Programming

```
CONTPRON (TABNAME, MACH, NN, MODE)
```

Deactivate contour preparations and at the same time switch back to the normal execution mode:

```
EXECUTE (ERROR)
```

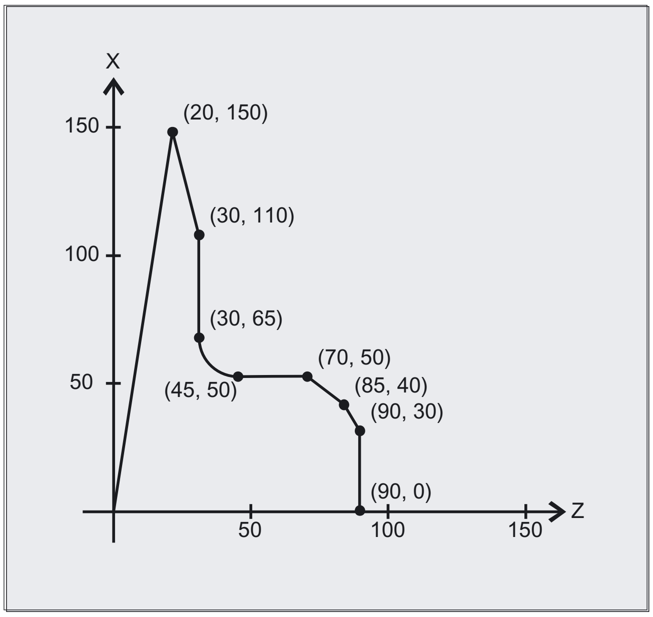
Parameters

CONTPRON	Activate contour preparation
TABNAME	Name of the contour table
MACH	Parameters for type of machining: "G": Longitudinal turning: Inside machining "L": Longitudinal turning: External machining "N": Face turning: Inside machining "P": Face turning: External machining
NN	Number of relief cuts in result variable of type INT
MODE	Machining direction, INT type 0 = Contour preparation forward (default value) 1 = Contour preparation in both directions

Example 1: Creating curve table

Create a contour table with

- name KTAB,
- up to 30 contour elements (circles, straight lines),
- a variable for the number of relief cut elements,
- a variable for error messages.



NC parts program

```
N10 DEF REAL KTAB[30,11]               ;Contour table named KTAB and, for example,
                                       ;a maximum of 30 contour elements
                                       ;parameter value 11 is a fixed quantity
N20 DEF INT ANZHINT                    ;Variable for number of relief cut elements
                                       ;with name ANZHINT
N30 DEF INT ERROR                      ;Variable for acknowledgment
                                       ;0 = no error, 1 = error
N40 G18
N50 CONTPRON (KTAB,"G",ANZHINT)        ;Contour preparation call
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)                    ;Terminate filling of contour table,
                                       ;switch to normal program execution
N140 …                                 ;Continue processing the table
```

Table KTAB

Index Line	Column									
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
7	7	11	0	0	20	150	0	82.40535663	0	0
0	2	11	20	150	30	110	-1111	104.0362435	0	0
1	3	11	30	110	30	65	0	90	0	0
2	4	13	30	65	45	50	0	180	45	65
3	5	11	45	50	70	50	0	0	0	0
4	6	11	70	50	85	40	0	146.3099325	0	0
5	7	11	85	40	90	30	0	116.5650512	0	0
6	0	11	90	30	90	0	0	90	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

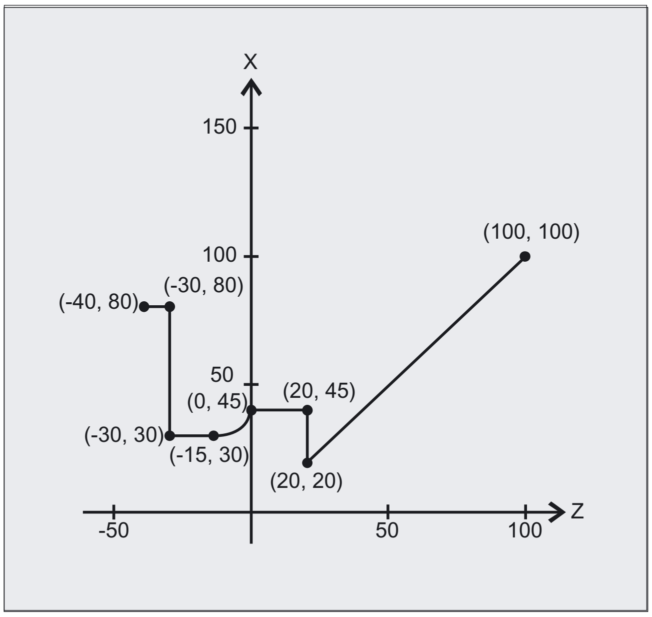
Explanation of column contents

(0)	Pointer to next contour element (to the row number of that column)
(1)	Pointer to previous contour element
(2)	Coding of contour mode for the movement
	Possible values for X = abc
	a = 10^2 G90 = 0 G91 = 1
	b = 10^1 G70 = 0 G71 = 1
	c = 10^0 G0 = 0 G1 = 1 G2 = 2 G3 = 3
(3), (4)	Starting point of contour elements
	(3) = abscissa, (4) = ordinate of the current plane
(5), (6)	Starting point of the contour elements
	(5) = abscissa, (6) = ordinate of the current plane
(7)	Max/min indicator: Identifies local maximum and minimum values on the contour
(8)	Maximum value between contour element and abscissa (for longitudinal machining) or ordinate (for face cutting). The angle depends on the type of machining programmed.
(9), (10)	Center point coordinates of contour element, if it is a circle block.
	(9) = abscissa, (10) = ordinate

Example 2: Creating curve table

Create a contour table with

- name KTAB,
- up to 92 contour elements (circles, straight lines),
- mode: Longitudinal turning, external machining,
- preparation forwards and backwards.



NC parts program

```
N10 DEF REAL KTAB[92,11]               ;Contour table named KTAB and, for example,
                                       ;a maximum of 92 contour elements
                                       ;parameter value 11 is a fixed quantity
N20 CHAR BT="L"                        ;Mode for CONTPRON:
                                       ;longitudinal turning, external machining
N30 DEF INT HE=0                       ;Number of relief cut elements=0
N40 DEF INT MODE=1                     ;Preparation forwards and backwards
N50 DEF INT ERR=0                      ;Error checkback message
...
N100 G18 X100 Z100 F1000
N105 CONTPRON (KTAB, BT, HE, MODE)     ;Contour preparation call
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)                      ;Terminate filling of contour table,
                                       ;switch to normal program execution
...
```

Table KTAB

After contour preparation is finished, the contour is available in both directions.

Index	Column										
Line	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
0	6[1)]	7[2)]	11	100	100	20	20	0	45	0	0
1	0[3)]	2	11	20	20	20	45	-3	90	0	0
2	1	3	11	20	45	0	45	0	0	0	0
3	2	4	12	0	45	-15	30	5	90	-15	45
4	3	5	11	-15	30	-30	30	0	0	0	0
5	4	7	11	-30	30	-30	45	-1111	90	0	0
6	7	0[4)]	11	-30	80	-40	80	0	0	0	0
7	5	6	11	-30	45	-30	80	0	90	0	0
8	1[5)]	2[6)]	0	0	0	0	0	0	0	0	0
	...										
83	84	0[7)]	11	20	45	20	80	0	90	0	0
84	90	83	11	20	20	20	45	-1111	90	0	0
85	0[8)]	86	11	-40	80	-30	80	0	0	0	0
86	85	87	11	-30	80	-30	30	88	90	0	0
87	86	88	11	-30	30	-15	30	0	0	0	0
88	87	89	13	-15	30	0	45	-90	90	-15	45
89	88	90	11	0	45	20	45	0	0	0	0
90	89	84	11	20	45	20	20	84	90	0	0
91	83[9)]	85[10)]	11	20	20	100	100	0	45	0	0

Explanation of column contents and comments for lines 0, 1, 6, 8, 83, 85 and 91

The explanations of the column contents given in example 1 apply.

Always in table line 0:

1) Predecessor: Line n contains the contour end (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)

Permitted traversing commands, coordinate system

The following G commands can be used for the contour programming:

G group 1: G0, G1, G2, G3

also corner and chamfer.

Circular-path programming is possible via CIP and CT.

The Spline, Polynomial, Thread functions produce errors.

It is not permitted to change the coordinate system by activating a frame between CONTPRON and EXECUTE. The same applies to a change between G70 and G71/ G700 and G710.

Changing the geometry axes with GEOAX while preparing the contour table produced an alarm.

Terminate contour preparation

When you call the predefined subroutine EXECUTE (variable), contour preparation is terminated and the system switches back to normal execution when the contour has been described. The variable then indicates:

1 = error

0 = no error (the contour could be prepared without error).

Relief cut elements

The contour description for the individual relief cut elements can be performed either in a subroutine or in individual blocks.

Stock removal independent of the programmed contour direction

The CONTPRON contour preparation has been expanded so that after being called, the contour table is available irrespective of the programmed direction.

14.3 Contour decoding (CONTDCON)

Function

The blocks executed after CONTPRON describe the contour to be decoded. The blocks are not processed but stored, memory-optimized, in a 6-column contour table. Each contour element corresponds to one row in the contour table. When familiar with the coding rules specified below, you can combine DIN code programs from the tables to produce applications (e.g., cycles). The data for the starting point are stored in the table cell with the number 0.

Programming

```
CONTDCON (TABNAME, MODE)
```

Deactivate contour preparations and at the same time switch back to the normal execution mode:

```
EXECUTE (ERROR)
```

Parameters

`CONTDCON`	`Activate contour preparation`
`TABNAME`	`Name of the contour table`

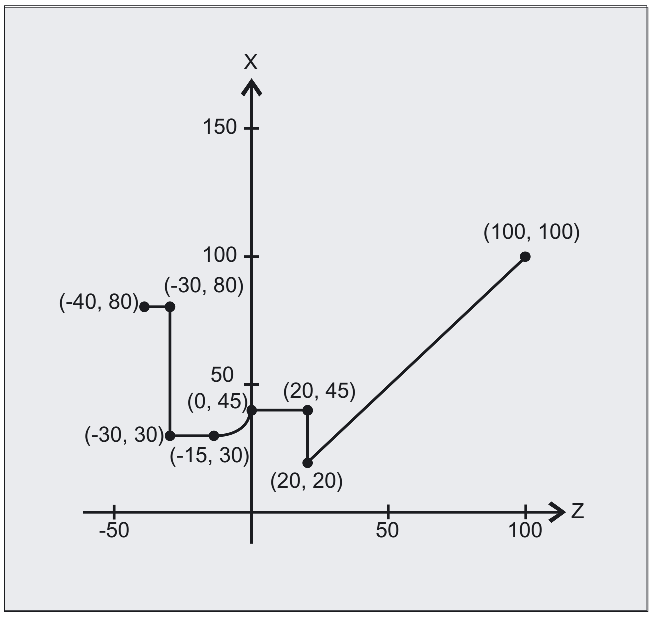
`MODE`	`Direction of machining, type INT`
	`0 = contour preparation (default) according to the contour block sequence`

The G codes permitted for `CONTDCON` in the program section to be included in the table are more comprehensive than for CONTPRON. In addition, feedrates and feed type are also stored for each contour section.

Example of creating a contour table

Create a contour table with

- name KTAB,
- contour elements (circles, straight lines),
- mode: turning,
- preparation forward.



NC parts program

```
N10 DEF REAL KTAB[9,6]                 ;Contour table with name KTAB and 9 table
                                       ;cells.
                                       ;These allow 8 contour sets. Parameter value 6
                                       ;(column number in table) is a fixed size
N20 DEF INT MODE = 0                   ;Default value 0: Only in programmed
                                       ;contour direction. Value 1 is not permitted.
N30 DEF INT ERROR = 0                  ;Error checkback message
...
N100 G18 G64 G90 G94 G710
N101 G1 Z100 X100 F1000
N105 CONTDCON (KTAB, MODE)             ;Call contour decoding
                                       ;MODE may be omitted, see above.
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)                    ;Terminate filling of contour table,
                                       ;switch to normal program execution
...
```

Table KTAB

Column index	0	1	2	3	4	5
Line index	Contour mode	End point abscissa	End point ordinate	Center point abscissa	Center point ordinate	Feed
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 column contents

Line 0 Coding for the **starting point:**		
	Column 0:	
		10^0 (units digit): G0 = 0
		10^1 (tens digit): 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
		10^1 (tens digit): G70 = 0, G71 = 1, G700 = 2, G710 = 3
		10^2 (hundreds digit): G90 = 0, G91 = 1
		10^3 (thousands digit): G93 = 0, G94 = 1, G95 = 2, G96 = 3
		10^4 (ten thousands digit): G60 = 0, G44 = 1, G641 = 2, G642 = 3
		10^5 (hundred thousands digit): G9 = 1
	Column 1:	End point abscissa
	Column 2:	End point ordinate
	Column 3:	Center point abscissa for circular interpolation
	Column 4:	Center point ordinate for circular interpolation
	Column 5:	Feed

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

also corner and chamfer.

Circular-path programming is possible via CIP and CT.

The Spline, Polynomial, Thread functions produce errors.

It is not permitted to change the coordinate system by activating a frame between CONDCRON and EXECUTE. The same applies to a change between G70 and G71/ G700 and G710.

Changing the geometry axes with GEOAX while preparing the contour table produces an alarm.

Terminate contour preparation

When you call the predefined subroutine EXECUTE (ERROR), contour preparation is terminated and the system switches back to normal execution when the contour has been described. The associated variable ERROR gives the return value:

0 = no errors (the contour could be prepared successfully)
1 = error
Invalid commands, incorrect initial conditions, CONTDCON call repeated without EXECUTE( ), too few contour blocks or table definitions too small also produce alarms.

Stock removal in the programmed contour direction

The contour table produced using CONTDCON is used for stock removal in the programmed direction of the contour.

14.4 Intersection of two contour elements (INTERSEC)

Function

INTERSEC calculates the intersection of two normalized contour elements from the contour table generated with CONTPRON.

Programming

```
ISPOINT = INTERSEC (TABNAME1[n1], TABNAME2[n2], ISCOORD, MODE)
```

The status returned by ISPOINT specifies whether or an intersection exists (ISPOINT = TRUE) or an intersect has not been found (ISPOINT = FALSE).

Parameters

INTERSEC	Stock removal function of a REAL type for calculating two contour elements from the contour table produced using CONTPRON
ISPOINT	Variable for the intersection status of the BOOL type: TRUE: Intersection found FALSE: No intersection found
TABNAME1[n1]	Table name and n1. Contour element of the first table
TABNAME2[n2]	Table name and n2. Contour element of the second table
ISCOORD	Intersection coordinates in the active plane G17 - G19
MODE	Machining type: Mode = 0 (default value) or mode = 1 (extension) 0 = intersection calculation in the active plane using parameter 2 1 = intersection calculation regardless of the plane transferred
G17 - G19	Plane of the contour table transferred during activation of CONTPRON

Note

Please note that variables must be defined before they are used.

The values defined with CONTPRON must be observed when transferring the contours:

Parameter 2	Coding of contour mode for the movement
Parameter 3	Contour start point abscissa
Parameter 4	Contour start point ordinate
Parameter 5	Contour end point abscissa
Parameter 6	Contour end point ordinate
Parameter 9	Center point coordinates for abscissa (only for circuit contour)
Parameter 10	Center point coordinates for ordinate (only for circuit 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 CUT (1st element = abscissa, 2nd element = ordinate). If no intersection exists, the program jumps to NOCUT (no intersection found).

```
DEF REAL TABNAME1 [12, 11]              ;Contour table 1
DEF REAL TABNAME2 [10, 11]              ;Contour table 2
DEF REAL ISCOORD [2]                    ;Intersection coordinates when ISPOINT =
                                        1
DEF BOOL ISPOINT                        ;Variable for the intersection status
DEF INT MODE                            ;Defining machining type
…
MODE = 1                                ;Calculation regardless of active plane
N10 ISPOINT=INTERSEC (TABNAME1[16,11],TABNAME2[3,11],ISCOORD, MODE)
                                        ;Call intersection of contour elements
N20 IF ISPOINT==FALSE GOTOF NOCUT       ;Jump to NOCUT
…
```

14.5 Traversing a contour element from the table (EXECTAB)

Function

You can use command EXECTAB to traverse contour elements block by block in a table generated, for example, with the CONTPRON command.

Programming

```
EXECTAB (TABNAME[n])
```

Parameters

TABNAME[n]	Name of table with number n of the element

Example

The contour elements stored in Table KTAB are traversed non-modally by means of subroutine EXECTAB. Elements 0 to 2 are transferred in consecutive calls.

```
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
```

14.6 Calculate circle data (CALCDAT)

Function

Calculation of radius and circle center point coordinates from three or four known circle points. 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.

Programming

```
VARIB = CALCDAT (PT[n,2], NUM, RES)
```

Note

Please note that variables must be defined before they are used.

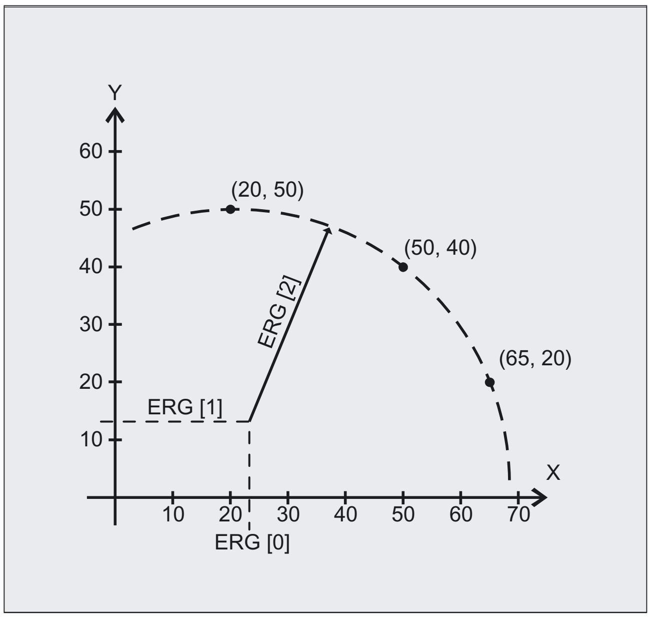
The specified status indicates whether the three or four known points are on a circle (VARIB = TRUE) or not (VARIB = FALSE).

Parameters

CALCDAT	Calculate the radiuses and centers of a circle that consists of 3 or 4 points.
VARIB	Variable for status TRUE = circle, FALSE = no circle
PT [n,2]	Points for calculation n = number of points (3 or 4); 2 = point coordinates
NUM	Number of points used for calculation: 3 or 4
RES [3]	Variable for result: specification of circle center point coordinates and radius; 0 = abscissa, 1 = ordinate of circle center point; 2 = radius

Example

The program determines whether the three points lie along the arc of a circle.



```
N10 DEF REAL PT[3,2]=(20,50,50,40,65,20)          ;Points definition
N20 DEF REAL RES[3]                               ;Result
N30 DEF BOOL STATUS                               ;Variable for the status
N40 STATUS = CALCDAT(PT,3,RES)                    ;Call calculated circle data
N50 IF STATUS == FALSE GOTOF ERROR                ;Jump to error
```

15 Tables

15.1 List of statements

The list of statements summarizes all programming commands and G codes available in the job planning.

Legend:

[1] Default setting at beginning of program (factory settings of the control, if nothing else programmed).

[2] The groups are numbered according to the table in section "List of G functions/preparatory functions".

[3] Absolute end points: modal; incremental end points: non-modal; otherwise modal/non-modal (m, n) depending on syntax of G function.

[4] As arc centers, 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 pitch).

[5] The keyword is not valid for SINUMERIK 810D

[6] The keyword is not valid for SINUMERIK 810D/NCU571

[7] The keyword is only valid for SINUMERIK FM-NC

[8] The OEM can add two extra interpolation types. The names can be changed by the OEM.

[9] Extended address notation cannot be used for these functions.

name	Meaning	Value assignment	Description, comment	Syntax	Modal/ non-modal	Group [2]
:	Block number - main block (see N)	0 ... 9999 9999 integers only, without leading signs	Special identification of blocks rather than N... ;this block should contain all statements for a following complete machining section	e.g. :20		
A	Axis	Real			m,n [3]	
A2 [5]	Tool orientation: Euler angles	Real			s	

A3 [5]	Tool orientation: Direction vector component	Real			s	
A4 [5]	Tool orientation for start of block	Real			s	
A5 [5]	Tool orientation for end of block; normal vector component	Real			s	
ABS	Absolute value	Real				
AC	Input of absolute dimensions	0 ..., 359.9999 °		X=AC(100)	s	
ACC [5]	Axial acceleration	Real, w/o signs			m	
ACN	Absolute dimensions for rotary axes, approach position in negative direction			A=ACN(...) B=ACN(...) C=ACN(...)	s	
ACP	Absolute dimensions for rotary axes, approach position in positive direction			A=ACP(...) B=ACP(...) C=ACP(...)	s	
ACOS	Arc cosine (trigon. function)	Real				
ADIS	Rounding clearance for path functions G1, G2, G3, ...	Real, w/o signs			m	
ADISPOS	Approximate distance for rapid traverse G0	Real, w/o signs			m	
ADISPOSA	Size of the tolerance window for IPOBRKA	Integer, real,		ADISPOSA=.. or ADISPOSA(<axis> [,REAL])	m	
ALF	Angle tilt fast	Integer, w/o signs			m	
AMIRROR	Programmable mirroring (additive mirror)			AMIRROR X0 Y0 Z0AMIRROR ; separate block	s	3
AND	Logical AND					
ANG	Contour angle	Real				
AP	Angle polar	0,..., ± 360°			m,n [3]	
APR	Read/display access protection (access protection read)	Integer, w/o signs				
APW	Write access protection (access protection write)	Integer, w/o signs				
AR	Aperture angle (angle circular)	0, ..., 360°			m,n [3]	

AROT	Programmable rotation (additive rotation)	Rotation about 1st geom. axis: -180º .. 180° 2nd geom. axis: -89.999° ... 90° 3rd geom. axis: -180° .. 180°		AROT X... Y... Z... AROT RPL= ;separate block	s	3
AROTS	Programmable frame rotations with solid angles (additive rotation)			AROT X... Y... AROT Z... X... AROT Y... Z... AROT RPL= ;separate block	s	3
AS	Macro definition	String				
ASCALE	Programmable scaling (additive scale)			ASCALE X... Y... Z... ;separate block	s	3
ASIN	Arc sine (trigon. function)	Real				
ASPLINE	Akima spline				m	1
ATAN2	Arc tangent 2	Real				
ATRANS	Additive programmable shift (additive translation)			ATRANS X... Y... Z... ;separate block	s	3
AX	Integer without sign	Real			m,n [3]	
AXCSWAP	Advance container axis			AXCSWAP(CTn, CTn+1,..)		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	Convert 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 [3]	
B_AND	Bit AND					
B_NOT	Bit negation					

B_OR	Bit OR					
B_XOR	Bit exclusive OR					
B2 [5]	Tool orientation: Euler angles	Real			s	
B3 [5]	Tool orientation: Direction vector component	Real			s	
B4 [5]	Tool orientation for start of block	Real			s	
B5 [5]	Tool orientation for end of block; normal vector component	Real			s	
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[1]	Natural transition to first spline block (begin natural)				m	19
BOOL	Data type: Boolean value TRUE / FALSE or 0 / 1					
BRISK[1]	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 [3]	
C2 [5]	Tool orientation: Euler angles	Real			s	
C3 [5]	Tool orientation: Direction vector component	Real			s	
C4 [5]	Tool orientation for start of block	Real			s	
C5 [5]	Tool orientation for end of block; normal vector component	Real			s	
CAC	Absolute approach of position (coded position: absolute coordinate)		Coded value is table index; table value is approached			
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_MYWPD/subroutine_ID_SPF)		
CANCEL	Cancel modal synchronized action	INT	Cancel with the specified ID. No parameters: All modal synchronized actions are deselected.			
CASE	Conditional program branch					
CDC	Direct approach of position (coded position: direct coordinate)		See CAC			
CDOF [1]	Collision detection OFF				m	23
CDON	Collision detection ON				m	23
CDOF2	Collision detection OFF		For CUT3DC only		m	23
CFC [1]	Constant feed at contour				m	16
CFIN	Constant feed at internal radius only, not at external radius				m	16
CFTCP	Constant feed in tool edge reference 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 module			
CHAR	Data type: ASCII character	0, ..., 255				
CHECKSUM	Forms the checksum over a 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 in direction of movement (chamfer) Chamfer; value = length of chamfer	Real, w/o signs			S	

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
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[1,6]	Compressor OFF				m	30
COMPON[6]	Compressor ON				m	30
COMPCURV	Compressor ON: Polynomials with constant curvature				m	30
COMPCAD	Compressor ON: optimized surface finish				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 software control, FINE/COARSE : block change on "synchronism fine/coarse", IPOSTOP: block change in setpoint-dependent termination of overlaid movement.	COUPDEF(FS, ...)		
COUPDEL	Delete ELG group (couple delete)			COUPDEL(FS,LS)		
COUPOF	ELG group / synchronous spindle pair OFF (couple OFF)			COUPOF(FS,LS, POS_{FS},POS_{LS})		
COUPOFS	Deactivating ELG assembly/synchronized spindle pair with stop of following spindle			COUPOFS(FS,LS,POS_{FS})		
COUPON	ELG group / synchronous spindle pair ON (couple ON)			COUPON(FS,LS, POS_{FS})		

COUPONC	Transfer activation of ELG assembly/synchronized spindle pair with previous programing			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[1,6]	Programmable contour precision OFF				m	39
CPRECON [6]	Programmable contour precision ON				m	39
CPROT	Channel-specific protection zone ON/OFF					
CPROTDEF	Channel specific protection area definition					
CR	Circle radius	Real, w/o signs			S	
CROT	Rotation of the current coordinate system.	FRAME	Max. parameter count: 6			
CROTS	Programmable frame rotations with solid angles (rotations in the indicated axes)			CROT X... Y... CROT Z... X... CROT Y... Z... CROT RPL= ;separate block	S	
CSCALE	Scale factor for multiple axes.	FRAME	Max. parameter count: 2 * axis count$_{max}$			
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			
CTABISLOCK	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 the curve table has been created with number n		Parameter n			
CTABMPOL	Max. number of polynomials still possible in the memory		memType			
CTABMSEG	Max. number of curve segments still possible in the mem.		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 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 beginning of curve table	R10 = CTABTSP(n, degree, Laxis)		

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. of 8 axes			
CUT2D [1]	2½D cutter compensation (cutter compensation type 2 dimensional)				m	22
CUT2DF	2½D tool offset (cutter compensation type 2 dimensional frame); The tool offset acts in relation to the current frame (inclined plane)				m	22
CUT3DC [5]	3D cutter compensation type 3-dimensional circumference milling				m	22
CUT3DCC [5]	Cutter compensation type 3-dimensional circumference milling with limit surfaces				m	22
CUT3DCCD [5]	Cutter compensation type 3-dimensional circumference milling with limit surfaces with differential tool				m	22
CUT3DF [5]	3D cutter compensation type 3-dimensional face milling				m	22
CUT3DFF [5]	3D cutter compensation type 3-dimensional face milling with constant tool orientation dependent on the current frame				m	22
CUT3DFS [5]	3D cutter compensation type 3-dimensional face milling with constant tool orientation independent of the current frame				m	22
CUTCONO[1]	Constant radius compensation OFF				m	40
CUTCONON	Constant radius compensation ON				m	40
D	Tool offset number	1, ... 32 000	Contains the correction data for a specific tool T... ; D0 → correction values for a tool	D...		
DC	Absolute dimensions for rotary axes, approach position directly			A=DC(...) B=DC(...) C=DC(...) SPOS=DC(...)	s	
DEF	Variable definition	Integer, w/o signs				

DEFAULT	Branch in CASE branch		Jump to if expression does not fulfill any of the specified values			
DEFINE	Define macro					
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			
DELT	Delete tool		Duplo number can be omitted			
DIAMCYOF	Radius programming for G90/91: ON. The G-code of this group that was last active remains active for display		Radius programming. last active G-code		m	29
DIAMOF[1]	Diameter programming: OFF (Diametral programming OFF)		Radius programming for G90/G91		m	29
DIAMON	Diametral programming: ON (Diametral programming ON)		Diameter programming for G90/G91		m	29
DIAM90	Diameter program for G90, radius progr. for G91				m	29
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, w/o signs			S	
DISR	Distance for repositioning	Real, w/o signs			S	
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[9]	Velocity-dependent path acceleration			m	21
DRIVEA	Switch on bent acceleration characteristic curve for the programmed axes				
DYNFINISH	Dynamics for smooth-finishing	Technology G group	DYNFINISH G1 X10 Y20 Z30 F1000	m	59
DYNNORM	Normal dynamics as previous		DYNNORM G1 X10	m	59
DYNPOS	Dynamics for positioning mode, tapping		DYNPOS G1 X10 Y20 Z30 F...	m	59
DYNROUGH	Dynamics for roughing		DYNROUGH G1 X10 Y20 Z30 F10000	m	59
DYNSEMIFIN	Dynamics for finishing cut		DYNSEMIFIN G1 X10 Y20 Z30 F2000	m	59
DZERO	Set D number of all tools of the TO unit assigned to the channel invalid				
EAUTO	Definition of last spline section by the last 3 points (end not a knot)			m	20
EGDEF	Definition of an electronic gear (Electronic gear define)	For 1 following axis with up to 5 leading axes			
EGDEL	Delete coupling definition for the following axis (Electronic gear delete)	Stops the preprocessing			
EGOFC	Switch off electronic gear continuous (Electronic gear OFF continuous)				
EGOFS	Switch off electronic gear selectively (Electronic gear OFF selective)				
EGON	Switch on electronic gear (Electronic gear ON)	Without synchronizatio n			
EGONSYN	Switch on electronic gear (electronic gear ON synchronized)	With synchronizatio n			
EGONSYNE	Switch on electronic gearing, stating approach mode (electronic gear ON synchronized)	With synchronizatio n			
ELSE	Program branch, if IF condition not fulfilled				
ENABLE	Interrupt ON				
ENAT [1,7]	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					
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 parts program line to run		Indirect parts 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 the normal program processing			
EXP	Exponential function (e^x)	Real				
EXTCALL	Execute external subroutine		Reload program from HMI in "Processing 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, ..., 99 999. 999	Tool/workpiece path feedrate; unit of measurement in mm/min or mm/rev dependent on G94 or G95	F=100 G1 ...		
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 feedrate for smooth approach and retraction (Feed approach / depart)	Real, w/o signs				

FALSE	Logical constant: Incorrect	BOOL	Can be replaced with integer constant 0			
FCTDEF	Define polynomial function		Is evaluated in SYFCT or PUTFTOCF.			
FCUB[6]	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			s	
FDA	Axial feed for handwheel override (feed DRF axial)	Real, w/o signs			s	
FENDNORM	Corner deceleration OFF				m	57
FFWOF[1]	Feedforward control OFF (feed forward OFF)				m	24
FFWON	Feedforward control ON (feed forward ON)				m	24
FIFOCTRL	Control of preprocessing buffer				m	4
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, w/o signs	The unit set with G93, G94, G95 is applicable (max. rapid traverse)	FL[axis]=...	m	
FLIN[6]	Feed linear variable (feed linear)		Acts on feed with G93 and G94		m	37
FMA	Feed multiple axial	Real, w/o signs			m	
FNORM[1,6]	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					
FP	Fixed point: number of fixed point to be approached	Integer, w/o signs		G75 FP=1	S	
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)		
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				s	
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[1,6]	Online fine tool offset OFF				m	33
FTOCON[6]	Online fine tool offset ON				m	33
FXS	Travel to fixed stop ON	Integer, w/o signs	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 Functions						
G	G function (preparatory function) The G functions are divided into G groups. Only one G function from one group can be written in one block. A G function can either be modal (until canceled by another function from the same group), or non-modal (only effective for the block it is written in).	Only integer, predefined values		G...		
G0	Linear interpolation with rapid traverse (rapid traverse motion)		Motion commands	G0 X... Z...	m	1
G1 [1]	Linear interpolation with feedrate (linear interpolation)			G1 X... Z... F...	m	1
G2	Circular interpolation clockwise			G2 X... Z... I... K... F... ;center and end point G2 X... Z... CR=... F... ;radius and end point G2 AR=... I... K... F... ;aperture angle and ;center point G2 AR=... X... Z... F... ;aperture angle and ;end point	m	1
G3	Circular interpolation counter-clockwise			G3 ... ; otherwise as for G2	m	1
G4	Dwell time preset		Special motion	G4 F... ;dwell time in s or G4 S... ;dwell time in ;spindle revolution. ;separate block	s	2
G9	Exact stop - deceleration				s	11
G17 [1]	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 assignments in	G25 X.. Y.. Z..;separate block	s	3
G26	Upper working area limitation		Channel axes	G26 X.. Y.. Z..;separate block	s	3

G33	Thread interpolation with constant pitch	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 (in Z axis;path larger than in the X axis) G33 Z... X... I... SF=... ;taper thread (in X axis;path larger than in the Z axis)	m	1
G34	Increase in thread pitch (progressive change)		Motion command	G34 Z... K... F_{UP}=...	m	1
G35	Decrease in thread pitch (degressive change)		Motion command	G35 Z... K... F_{DOWN}=...	m	1
G40 [1]	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		s	9
G54	1. Settable zero offset				m	8
G55	2. Settable zero offset				m	8
G56	3. Settable zero offset				m	8
G57	4. Settable zero offset				m	8
G58	Programmable offset		Replacing axially		s	3
G59	Programmable offset		Replacing additively axially		s	3
G60 [1]	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	s	2
G64	Exact stop - continuous-path mode				m	10
G70	Dimension in inches (lengths)				m	13
G71 [1]	Metric dimension (lengths)				m	13
G74	Reference point approach			G74 X... Z...;separate block	s	2
G75	Fixed point approach		Machine axes	G75 FP=.. X1=... Z1=... ;separate block	s	2
G90 [1]	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

G93	Inverse-time feedrate rpm		Execution of a block: Time	G93 G01 X... F...	m	15
G94 [1]	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 ON			G96 S... LIMS=... F...	m	15
G97	Constant cutting speed OFF				m	15
G110	Pole programming relative to the last programmed setpoint position			G110 X.. Y.. Z..	s	3
G111	Polar programming relative to origin of current workpiece coordinate system			G110 X.. Y.. Z..	s	3
G112	Pole programming relative to the last valid pole			G110 X.. Y.. Z..	s	3
G140 [1]	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				s	2
G148	Soft retraction with straight line				s	2
G153	Suppression of current frame incl. base frame				s	9
G247	Soft approach with quadrant				s	2
G248	Soft retraction with quadrant				s	2
G290	Switch to SINUMERIK mode ON				m	47
G291	Switch to ISO2/3 mode ON				m	47
G331	Thread tapping	± 0.001, ..., 2000.00 mm/rev	Motion commands		m	1
G332	Retraction (tapping)				m	1
G340 [1]	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				s	2
G348	Soft retraction with semicircle				s	2
G450 [1]	Transition circle		Corner behavior with tool radius compensation		m	18
G451	Intersection of equidistances				m	18
G460 [1]	Approach/retraction behavior with TRC				m	48
G461	Approach/retraction behavior with TRC				m	48
G462	Approach/retraction behavior with TRC				m	48
G500 [1]	Deactivate all settable frames if G500 does not contain a value				m	8

G505 .... G599	5. ... 99. Settable zero offset			m	8
G601 [1]	Block change at exact stop fine	Only effective with active G60 or 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
G700	Dimension in inches and inch/min (lengths + velocities + system variable)			m	13
G710 [1]	Metric dimension in mm and mm/min (lengths + velocities + system variable)			m	13
G810[1], ..., G819	G group reserved for the OEM				31
G820[1], ..., 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 revolutional feedrate and constant cutting rate or spindle speed			m	15
G961	Constant cutting speed ON	Feed type like for G94	G961 S... LIMS=... F...	m	15
G962	Linear or revolutional feedrate and constant cutting rate			m	15
G971	Constant cutting speed OFF			m	15
G972	Freeze linear or revolutional feedrate and constant spindle speed			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			
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)	Parts program and can also be applied in technology cycles.	GOTO (label, block no.) Labels must be present in the sub-program		
GOTOF	Jump forwards (toward the end of the program)		GOTOF (Label, block no.)		
GOTOB	Jump backwards (toward the start of the program)		GOTOB (Label, block no.)		
GOTOC	Alarm 14080 Suppress jump destination not found.	See GOTO			
GWPSOF	Deselect constant grinding wheel peripheral speed (GWPS)		GWPSOF(T No.)	s	
GWPSON	Select constant grinding wheel peripheral speed (GWPS)		GWPSON(T No.)	s	

H...	Auxiliary function output to the PLC	Real/INT	settable via machine data (machine manufacturer)	H100 or H2=100		
I [4]	Interpolation parameters	Real			s	
I1	Intermediate point coordinate	Real			s	
IC	Incremental dimensioning	0, ..., ±99999.999°		X=IC(10)	s	
ICYCOF	All blocks of a technology cycle are processed in one IPO cycle following ICYCOF.		only within the program level			
ICYCON	Each block of a technology cycle is processed in a separate IPO cycle following ICYCON.		only within the program level			
IDS	Identification of static synchronized actions					
IF	Introduction of a conditional jump in the parts program / technology cycle		Structure: IF-ELSE-ENDIF	IF (condition)		
INDEX	Define index of character in input string	0, ..., INT	String: 1. Parameter 1, character: 2. par.			
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 leading sign	- (2^{31}-1), ..., 2^{31}-1				
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 (Interpolation parameter)	Real				
IPOBRKA	Motion criterion from braking ramp activation		Braking ramp with 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 the interrupt pointer		m	
IPTRUNLOCK	Set end of untraceable program section at current block at time of interruption		Set the interrupt pointer		m	
ISAXIS	Check if geometry axis 1 – 3 specified as parameter	BOOL				
ISD	Insertion depth	Real			m	
ISFILE	Checks 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 number			
ISPOINTS	Possible intersections calculated by ISTAB between two contours on the current plane.	INT	MODE machining type (optional)	STATE=ISPOINTS (KTAB1[n1], KTAB2[n2], ISTAB, [MODE])		
ISVAR	Check whether the transfer parameter contains a variable known in the NC	BOOL	Machine data, setting data and variables as GUDs			
J [4]	Interpolation parameters	Real			s	
J1	Intermediate point coordinate	Real			s	
JERKA	Activate acceleration response set via machine data for programmed axes					
K [4]	Interpolation parameters	Real			s	
K1	Intermediate point coordinate	Real			s	
KONT	Travel round contour on tool offset				m	17
KONTC	Approach/traverse with continuous-curvature polynomial				m	17
KONTT	Approach/traverse with continuous-tangent polynomial				m	17
L	Subroutine number	Integer, up to 7 places		L10	s	
LEAD [5]	Lead angle	Real			m	
LEADOF	Master value coupling OFF (lead off)					
LEADON	Master value coupling ON (lead on)					
LFOF [1]	Interrupt thread cutting OFF				m	41
LFON	Interrupt thread cutting ON				m	41
LFTXT [1]	Tangential tool direction on retraction				m	46
LFWP	Non-tangential tool direction on retraction				m	46
LFPOS	Axial retraction to a position				m	46
LIFTFAST	Rapid lift before interrupt routine call					
LIMS	Spindle speed limitation (Limit Spindle Speed) with G96/G961 and G97	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	0, ..., 9999 9999	Up to 5 unassigned M functions can be assigned by the machine manufacturer			
M0 [10]	Programmed stop					
M1 [10]	Optional stop					
M2 [10]	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 [10]	Subroutine end					
M19	Spindle positions					
M30 [10]	End of program, same effect as M2					
M40	Automatic gear change					
M41... M45	Gear stage 1, ..., 5					
M70	Transition to axis mode					
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					
MCALL	Modal subroutine call		Without subroutine name: Deselection			
MEAC	Continuous measurement without deleting distance-to-go	Integer, w/o signs			S	
MEAFRAME	Frame calculation from measuring points	FRAME				
MEAS	Measure with touch-trigger probe	Integer, w/o signs			S	
MEASA	Measurement with deletion of distance-to-go				s	
MEAW	Measure with touch-trigger probe without deleting distance-to-go	Integer, w/o signs			S	
MEAWA	Measurement without deletion of distance-to-go				s	
MI	Access to frame data: Mirroring					

MINDEX	Define index of character in input string	0, ..., INT	String: 1. parameter, character: 2. par.			
MIRROR	Mirroring, programmable			MIRROR X0 Y0 Z0 ;separate block	s	3
MMC	Calling the dialog window interactively from the parts program on the HMI	STRING				
MOD	Modulo division					
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 leading 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 [1]	Standard setting in starting point and end point with tool offset				m	17
NOT	Logical NOT (negation)					
NPROT	Machine-specific protection zone ON/OFF					
NPROTDEF	Machine-specific protection area definition (NCK-specific protection area definition)					
NUMBER	Convert input string to number		Real			
OEMIPO1[6,8]	OEM interpolation 1				m	1
OEMIPO2[6,8]	OEM interpolation 2				m	1
OF	Keyword in CASE branch					
OFFN	Allowance on the programmed contour			OFFN=5		
OMA1 [6]	OEM address 1	Real			m	
OMA2 [6]	OEM address 2	Real			m	
OMA3 [6]	OEM address 3	Real			m	
OMA4 [6]	OEM address 4	Real			m	
OMA5 [6]	OEM address 5	Real			m	
OFFN	Offset - normal	Real			m	
OR	Logical OR					

ORIC [1,6]	Orientation changes at outside corners are superimposed on the circle block to be inserted (orientation change continuously)				m	27
ORID [6]	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, B2, C2 Additional inputs: Rotational vectors A6, B6, C6 Arc angle of taper in degrees: 0<SLOT<180 deg. Intermediate vectors: A7, B7, C7 contact point of tool: XH, YH, ZH	Parameter settings as follows: Direction vectors normalized A6=0, B6=0, C6=0 Opening angle implemented as travel angle with SLOT=... SLOT=+... at ≤ 180 degrees SLOT = -... at ≥ 180 degrees Intermediate orientation normalized A7=0 B7=0 C7=1	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
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)				m	51
ORIPATH	Tool orientation trajectory relative to the path		Transformation package handling, see /FB/, TE4		m	51
ORIPATHS	Tool orientation relative to the path, blips in the orientation characteristic are smoothed		Relative to the path as a whole		m	51
ORIROTA	Angle of rotation to an absolute direction of rotation				m	54
ORIROTC	Tangential rotational vector in relation to path tangent		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 (rotation sequence XYZ)				m	50
ORIRPY2	Orientation angle via RPY angle (rotation sequence ZYX)				m	50
ORIS [5]	Orientation modification (orientation smoothing factor)	Real	Relative to the path		m	
ORIVECT	Large-radius circular interpolation (identical to ORIPLANE)				m	51

ORIVIRT1	Orientation angle via virtual orientation axes (definition 1)				m	50
ORIVIRT2	Orientation angle via virtual orientation axes (definition 1)				m	50
ORIMKS [6]	Tool orientation in the workpiece coordinate system				m	25
ORIRESET	Initial setting of tool orientation with up to 3 orientation axes		Parameter optional (REAL)	ORIRESET(A,B,C)		
ORIWKS [1,6]	Tool orientation in the workpiece coordinate system				m	25
OS	Oscillation ON / OFF	Integer, w/o signs				
OSB	Oscillating: Start point				m	
OSC [6]	Continuous tool orientation smoothing				m	34
OSCILL	Axis assignment for oscillation- activate oscillation		Axis: 1 – infeed axes		m	
OSCTRL	Oscillation control options	Integer, w/o signs			M	
OSD [6]	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 [1,6]	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 [6]	Tool orientation smoothing at end of block				m	34
OSSE [6]	Tool orientation smoothing at start and end of block				m	34
OST [6]	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	

P	Number of subroutine passes	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 response like CALL			
PDELAYOF [6]	Punch with delay OFF				m	36
PDELAYON [1,6]	Punch with delay ON				m	36
PL	Parameter interval length	Real, w/o signs			S	
PM	Per minute			Feed per minute		
PO	Polynomial	Real, w/o signs			S	
POLF	LIFTFAST position	Real, w/o signs	Geometry axis in WCS, otherwise MCS	POLF[Y]=10 target position of retracting axis	m	
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 **without** a connection between the axes		Selected axes	POLFMASK(AX1, AX2, ...)	m	
POLFMLIN	Enable axes for retraction **with** a linear connection between the axes		Selected axes	POLFMIN(AX1, AX2, ...)	m	
POLY [5]	Polynomial interpolation				m	1
POLYPATH [5]	Polynomial interpolation can be selected for the AXIS or VECT axis groups			POLYPATH ("AXES") POLYPATH ("VECT")	m	1
PON [6]	Punch ON				m	35
PONS [6]	Punch ON in IPO cycle (punch ON slow)				m	35
POS	Position axis			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			Rev. 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 8 axes possible	PRESETON(X,10,Y,4.5)		
PRIO	Keyword for setting the priority for interrupt processing					
PROC	First instruction in a program			Block number - PROC - identifier		
PTP	point to point; point to point motion				m	49
PUTFTOC	Tool offset fine for parallel dressing (continuous dressing)		Channel numbers 1-10 or $MC _CHAN_NAME	PUTFTOC(1,1,2) or PUTFTOC(CH_X, CH_Y)		
PUTFTOCF	PutFineToolCorrectionFunctionDependent: Fine tool correction depending on a function defined by FCtDEF for continuous dressing		Channel numbers 1-10 or $MC _CHAN_NAME	PUTFTOCF(1,1,2) or PUTFTOCF(CH_name)		
PW	Point weight	Real, w/o signs			S	
QECLRNOF	Quadrant error compensation learning OFF					
QECLRNON	Quadrant error compensation learning ON					
QU	Fast additional (auxiliary) function output					

R...	Arithmetic variables also as settable address identifier and with numerical extension	±0.000000 1, ..., 9999 9999	Number of R parameters can be set by MD	R10=3 ;R parameter assignment X=R10 ;axis value R[R10]=6 ;indirect progr.		
RDISABLE	Read-in disable					
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 leading 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 which 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: Linear repositioning with all axes				s	2
REPOSH	Repositioning semicircle: Repositioning in semicircle				s	2
REPOSHA	Repositioning semicircle all axes: Repositioning with all axes; geometry axes in semicircle				s	2
REPOSL	Repositioning linear: Linear repositioning				s	2
REPOSQ	Repositioning quarter-circle: Repositioning in a quadrant				s	2
REPOSQA	Repositioning quarter-circle all axes: Return to contour linear all axes; geometry axes in quarter-circle				s	2

RESET	Reset technology cycle		One or several IDs can be programmed			
RET	Subroutine end		Use in place of M17 – without function output to PLC	RET		
RINDEX	Define index of character in input string	0, ..., INT	String: 1st parameter, character: 2. par.			
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 [1]	Repositioning at interruption point (Repos mode interrupt)				m	26
RMN	Reapproach to nearest path point (Repos mode end of nearest orbital block)				m	26
RND	Round the contour corner	Real, w/o signs		RND=...	s	
RNDM	Modal rounding	Real, w/o signs		RNDM=... RNDM=0: disable modal rounding	m	
ROT	Programmable rotation	Rotation about 1st geom. axis: -180° .. 180° 2nd geom. axis: -89.999°, ..., 90° 3rd geom. axis: -180° .. 180°		ROT X... Y... Z... ROT RPL= ;separate block	s	3
ROTS	Programmable frame rotations with solid angles (rotation)			ROT X... Y... ROT Z... X... ROT Y... Z ROT RPL= ;separate block	s	3
ROUND	Round decimal places	Real				
RP	Polar radius	Real			m,n [3]	
RPL	Rotation in the plane	Real, w/o signs			S	
RT	Parameter for access to frame data: Rotation					

S	Spindle speed or (with G4, G96/G961) other meaning	0.1, ..., 99999999.9	Spindle speed in rpm G4: Dwell time in spindle revolutions G96/G961: cutting speed in m/min	S...: Speed for master spindle S1...: Speed for spindle 1	m, s	
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)					
SCALE50	Programmable scaling (scale)			SCALE X... Y... Z... ;separate block	s	3
SD	Spline degree	Integer, w/o signs			S	
SEFORM	Structuring instruction 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			
SETM	Set one/several markers for channel coordination		Machining in the local channel is not influenced by this.			
SETMS	Reset to the master spindle defined in machine data					
SETMS(n)	Set spindle n as master spindle					
SETPIECE	Set piece number for all tools assigned to the spindle.		Without spindle number: Valid for master spindle			

SF	Starting point offset for thread cutting (spline offset)	0.0000,..., 359.999°			m	
SIN	Sine (trigon. function)	Real				
SOFT	Soft smoothed path acceleration				m	21
SOFTA	Switch on soft axis acceleration for the programmed axes					
SON [6]	Nibbling ON (stroke ON)				m	35
SONS [6]	Nibbling ON in IPO cycle (stroke ON slow)				m	35
SPATH [1]	Path reference for FGROUP axes is arc length				m	45
SPCOF	Switch master spindle or spindle(s) from speed control to position control			SPCON SPCON (n)		
SPCON	Switch master spindle or spindle(s) from position control to speed control			SPCON SPCON (n)		
SPIF1 [1,6]	Fast NCK inputs/outputs for punching/nibbling byte 1 (stroke/punch interface 1)		see /FB/, N4: Punching and nibbling		m	38
SPIF2 [6]	Fast NCK inputs/outputs for punching/nibbling byte 2 (stroke/punch interface 2)		see /FB/, N4: Punching and nibbling		m	38
SPLINE-PATH	Define spline grouping		Max. of 8 axes			
SPOF [1,6]	Stroke OFF, punching, nibbling OFF				m	35
SPN [6]	Number of path sections per block (stroke/punch number)	Integer			s	
SPP [6]	Length of path section (stroke/punch path)	Integer			m	
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, w/o signs			S	
SRA	Sparking-out retraction path with input axial for synchronized action			SRA[Y]=0.2	m	
ST	Sparking-out time for synchronized action	Real, w/o signs			S	

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)		
STAT	Position of joints	Integer			s	
STARTFIFO[1]	Execute; simultaneously fill preprocessing memory				m	4
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					
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)=return value with coding		
STRLEN	Define string length	INT				
SUBSTR	Define index of character in input string	Real	String: 1. Parameter 1, character: 2. par.			
SUPA	Suppress the actual zero offset		incl. programmed offsets, handwheel offsets (DRF), external zero offsets and PRESET offset		s	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)					
T	Call tool (only change if specified in machine data; otherwise M6 command necessary)	1 ... 32 000	Call using T-no.: or with tool identifier:	e.g., 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 [1]	Determine tool length components from the orientation of the current toolholder		Necessary after reset, e.g., through		m	42
TCOFR	Determine tool length components from the orientation of the active frame		Manual setting		m	42
TCOFRX	Determine tool orientation of an active frame during tool selection, tool points in X direction		Tool perpendicular to inclined surface		m	42
TCOFRY	Determine tool orientation of an active frame during tool selection, tool points in Y direction		Tool perpendicular to inclined surface		m	42
TCOFRZ	Determine tool orientation of an active frame during tool selection, tool points in Z direction		Tool perpendicular to inclined surface		m	42
TILT [5]	Tilt angle	Real		TILT=Value	m	
THETA	Angle of rotation		THETA is always vertical to the current tool orientation	THETA=Value THETA=AC THETA=IC Polynomial for THETA PO[THT]=(…)	s	
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				
TOFFOF	Deactivate on-line tool offset				
TOFFON	Activate online tool length compensation (Tool Offset ON)	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 rotations in the 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
TOFROF	Frame rotations in the tool direction OFF			m	53
TOFROT	Z axis parallel to tool orientation	Frame rotations ON Rotation component of programmable frame		m	53
TOFROTX	X axis parallel to tool orientation			m	53
TOFROTY	Y axis parallel to tool orientation			m	53
TOFROTZ	Z axis parallel to tool orientation			m	53
TOLOWER	Convert letters of the string into lowercase				
TOWSTD	Initial setting value for corrections 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 values in machine coordinate system (MCS).			m	56
TOWTCS	Wear values in the tool coordinate system (tool carrier ref. point T at the toolholder)			m	56
TOWWCS	Wear values in workpiece coordinate system (WCS)			m	56
TOUPPER	Convert letters of the string into uppercase				
TR	Parameter for access to frame data: Translation				
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			
TRAFOOF	Deactivate transformation			TRAFOOF( )		
TRAILOF	Synchronous coupled motion of axes OFF (trailing OFF)					
TRAILON	Synchronous coupled motion of axes ON (trailing ON)					
TRANS	Programmable translation			TRANS X. Y. Z.;separate block	s	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	s	
TURN	Number of turns for helix	0, ..., 999			s	
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			
WAITC	Wait until coupling block change criterion for axes / spindles is fulfilled (wait for couple condition)		Up to 2 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; end previous block with exact stop		Channel numbers 1-10 or $MC _CHAN_NAME	WAITM(1,1,2 or WATM(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 WATMC(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	WORK-working area limitation		deselected		m	60
WALCS1	WORK-working-area-limitation group 1		active		m	60
WALCS2	WORK-working-area-limitation group 2		active		m	60
WALCS3	WORK-working-area-limitation group 3		active		m	60
WALCS4	WORK-working-area-limitation group 4		active		m	60
WALCS5	WORK-working-area-limitation group 5		active		m	60
WALCS6	WORK-working-area-limitation group 6		active		m	60
WALCS7	WORK-working-area-limitation group 7		active		m	60
WALCS8	WORK-working-area-limitation group 8		active		m	60
WALCS9	WORK-working-area-limitation group 9		active		m	60
WALCS10	WORK-working-area-limitation group 10		active		m	60
WALIMOF	Working area limitation OFF			; separate block	m	28
WALIMON[1]	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 [3]	
XOR	Logical exclusive OR					
Y	Axis	Real			m,n [3]	
Z	Axis	Real			m,n [3]	

Legend:

[1] Default setting at beginning of program (factory settings of the control, if nothing else programmed).

[2] The groups are numbered according to the table in section "List of G functions/preparatory functions".

[3] Absolute end points: modal; incremental end points: non-modal; otherwise modal/non-modal (m, n) depending on syntax of G function.

[4] As arc centers, 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 pitch).

[5] The keyword is not valid for SINUMERIK 810D

[6] The keyword is not valid for SINUMERIK 810D/NCU571

[7] The keyword is only valid for SINUMERIK FM-NC

[8] The OEM can add two extra interpolation types. The names can be changed by the OEM.

[9] Extended address notation cannot be used for these functions.

List of abbreviations A

A	Output
AS	Automation system
ASCII	American Standard Code for Information Interchange: American coding standard for the exchange of information
ASIC	Application Specific Integrated Circuit: User switching circuit
ASUB	Asynchronous subroutine
AuxF	Auxiliary function
AV	Job planning
BA	Operating mode
BB	Ready to run
BCD	Binary Coded Decimals: Decimal numbers encoded In binary code
BCS	Basic Coordinate System
BIN	Binary files (**Bin**ary Files)
BIOS	Basic Input Output System
BOT	Boot files: Boot files for SIMODRIVE 611 digital
BP	Basic program
C Bus	Communication bus
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CNC	Computerized Numerical Control: Computerized numerical control
COM	Communication
COR	Coordinate rotation
CP	Communications Processor
CPU	Central Processing Unit: Central processing unit
CR	Carriage Return
CRC	Cutter radius compensation
CRT	Cathode Ray Tube picture tube
CSB	Central Service Board: PLC module
CSF	Function plan (PLC programming method)
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
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
DTE	Data Terminal Equipment
DW	Data word
E	Input
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	Feed Drive
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
FM-NC	Function module – numerical control
FPU	Floating Point Unit Floating Point Unit
FRA	Frame block
FRAME	Data record (frame)
FST	Feed Stop: Feed stop
GUD	Global User Data: Global user data
HD	Hard Disk Hard disk
HEX	Abbreviation for hexadecimal number
HHU	Handheld unit
HMI	Human Machine Interface: Operator functionality of SINUMERIK for operation, programming and simulation.

HMS	High-resolution Measuring System
HW	Hardware
I/O	Input/Output
I/R	Infeed/regenerative-feedback unit (power supply) of the SIMODRIVE 611digital
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
IS	Interface signal
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
K1 .. K4	Channel 1 to channel 4
K_{UE}	Speed ratio
K_v	Servo gain factor
LAD	Ladder diagram (PLC programming method)
LCD	Liquid Crystal Display: Liquid crystal display
LEC	Leadscrew error compensation
LED	Light-Emitting Diode: Light emitting diode
LF	Line Feed
LR	Position controller
LUD	Local User Data
MB	Megabyte
MC	Measuring circuit
MCP	Machine control panel
MCS	Machine coordinate system
MD	Machine data
MDI	Manual Data Automatic: Manual input
MLFB	Machine-readable product designation
Mode group	Mode group
MPF	Main Program File: NC parts program (main program)
MPI	Multiport Interface Multiport Interface
MS	Microsoft (software manufacturer)
MSD	Main Spindle Drive
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
NURBS	Non-Uniform Rational B-Spline
OB	Organization block in the PLC
OEM	Original Equipment Manufacturer
OP	Operator Panel
OP	Operator Panel: Operating setup
OPI	Operator Panel Interface
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
PLC	Programmable Logic Controller
PMS	Position measuring system
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
RS-232-C	Serial interface (definition of the exchange lines between DTE and DCE)
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
SR	Subroutine
SRAM	Static RAM (non-volatile)

SSI	Serial Synchronous Interface: Synchronous serial interface
STL	Statement list
SW	Software
SYF	System Files System files
T	Tool
TC	Tool change
TEA	Testing Data Active: Identifier for machine data
TLC	Tool length compensation
TNRC	Tool Nose Radius Compensation
TO	Tool offset
TO	Tool Offset: Tool offset
TOA	Tool Offset Active: Identifier (file type) for tool offsets
TRANSMIT	TRANSform Milling Into Turning: Coordinate conversion on turning machine for milling operations
TRC	Tool Radius Compensation
UFR	User Frame: Work offset
UI	User interface
WCS	Workpiece coordinate system
WOP	Workshop-oriented Programming
WPD	Workpiece Directory: Workpiece directory
ZO	Work offset
ZOA	Zero Offset Active: Identifier (file type) for zero offset data
µC	Micro Controller

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 -> incremental dimension.

Acceleration with jerk limitation

In order to optimize the acceleration response of the machine whilst simultaneously protecting the mechanical components, it is possible to switch over in the machining program between abrupt acceleration and continuous (jerk-free) acceleration.

Address

An address is the identifier for a certain operand or operand range, e.g. input, output etc.

Analog input/output module

Analog input/output modules are signal formers for analog process signals.

Analog input modules convert analog measured values into digital values which can be processed in the CPU.

Analog output modules convert digital values into analog output signals.

Approach machine fixed-point

Approach motion towards one of the predefined -> fixed machine points.

Archiving

Reading out data and/or directories to an external memory device.

A-Spline

The Akima-Spline runs under a continuous tangent through the programmed interpolation points (3rd order polynomial).

Asynchronous subroutine

A parts program which can be started asynchronously to (independently of) the current program status by an interrupt signal (e.g. "rapid NC input" signal).

Automatic

Operating mode of the control (block sequence operation according to DIN): Operating Mode in NC systems in which a -> parts program is selected and continuously executed.

Auxiliary functions

Auxiliary functions can be used to transfer -> parameters to the -> PLC in -> parts programs, where they trigger reactions which are 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 the actual machining, and include for example tool feeders and tool magazines.

Axis address

See -> axis identifier

Axis identifier

Axes are labeled in accordance with DIN 66217 (for a clockwise orthogonal -> coordinate system) with the letters X,Y, Z.

-> Rotary axes which rotate around are labeled with the letters A, B, C. Additional axes parallel to the above can be identified with further address letters.

Axis name

See -> axis identifier

B spline

With the B-Spline, the programmed positions are not interpolation points, as they are just "control points" instead. The generated curve only runs near to the control points, not directly through them (optional 1st, 2nd or 3rd order polynomials).

Backlash compensation

Compensation for mechanical machine backlash, e.g. backlash on reversal for feed screws. Backlash compensation can be entered separately for each axis.

Backup

Saving the memory contents to an external memory device.

Backup battery

A backup battery ensures that the → user program is stored retentively in the → CPU along with specified data areas and bit memory, timers, and counters.

Back-up memory

The backup memory enables buffering of memory areas of the -> CPU without a buffer battery. Buffering can be performed for a configurable number of times, counters, markers and data bytes.

Basic axis

Axis whose setpoint or actual value position forms the basis of the calculation of a compensation value.

Basic Coordinate System

Cartesian coordinate system which is mapped by transformation onto the machine coordinate system.

In the -> parts program, the programmer uses the axis names of the basic coordinate system. The basic coordinate system exists in parallel to the -> machine coordinate system when no -> transformation is active. The difference between the systems relates to the axis identifiers.

Baud rate

Rate of data transfer (Bit/s).

Block

"Block" is the term given to any files required for creating and processing programs.

Block search

For debugging purposes or following a program abort, the "Block search" function can be used to select any location in the part program at which the program is to be started or resumed.

Booting

Loading the system program after power on.

Bus connector

A bus connector is an S7-300 accessory part which is supplied together with the -> I/O modules. The bus connector expands the -> S7-300 bus from the -> CPU or an I/O module to the neighboring I/O module.

C axis

Axis around which the tool spindle describes a controlled rotational and positioning movement.

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.

Channel

A channel is characterized by its ability to execute a -> parts program independently of other channels. A channel exclusively controls the axes and spindles assigned to it. Parts programs run on various channels can be coordinated by -> synchronization.

Channel structure

The channel structure enables the -> programs of the individual channels to be executed simultaneously and asynchronously.

Circular interpolation

The -> tool is required to travel in a circle between defined points on the contour at a specified feedrate while machining the workpiece.

CNC

See -> NC

COM

Component of the NC control for the implementation and coordination of communication.

Compensation axis

Axis with a setpoint or actual value modified by the compensation value

Compensation table

Table containing interpolation points. It provides the compensation values of the compensation axis for selected positions on the basic axis.

Compensation value

Difference between the axis position measured by the position sensor and the desired, programmed axis position.

Connecting cable

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 purpose of continuous-path mode is to prevent excessive deceleration of the -> path axes at the part program block boundaries (in terms of the control, machine and other properties of the operation and the user) and to effect the transition to the next block at as uniform a path speed as possible.

Contour

Outline of the -> workpiece

Contour monitoring

The following error is monitored within a defined tolerance band to ensure contour precision. An impermissibly high following error might be caused by a drive overload, for example. In this case an alarm is triggered and the axes are stopped.

coordinate system

See -> Machine Coordinate System, -> Workpiece Coordinate System

CPU

Central Processor Unit, see -> Programmable Logic Controller

Cycle

Protected subroutine for implementing a repetitious machining operation on the → workpiece.

Data Block

1. Data unit of the -> PLC, which the -> HIGHSTEP programs can access.
2. Data unit of the -> NC: Data blocks contain data definitions for global user data. These data can be initialized directly when they are defined.

Data transmission program PCIN

PCIN is an auxiliary program which is used to send and receive CNC user data via the serial interface, such as e.g. parts programs, tool offsets etc. The PCIN program can be executed under MS-DOS on standard industrial PCs.

Data word

A data unit, two bytes in size, within a -> data block.

Diagnosis

1. Control operating area
2. The control has both a self-diagnostics program and testing aids for service. Status, alarm and service indicators.

Digital input/output module

Digital modules are signal formers for binary process signals.

Dimensions in metric units and inches

Position and gradient values can be entered in the machining program in inches. The control can be set to a basic system regardless of the programmed measuring system (G70/G71).

DRF

Differential Resolver Function: An NC function which generates an incremental zero offset in automatic mode in conjunction with an electronic handwheel.

Drive

The SINUMERIK 840D control system is connected to the SIMODRIVE 611 digital converter system by means of a high-speed digital parallel bus.

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 is used to create, modify, add to, compress, and insert programs/texts/program blocks.

Electronic handwheel

The electronic handwheels can be used to simultaneously traverse selected axes manually. The meaning of the lines on the handwheels is defined by the external zero offset increment weighting.

Exact stop

With a programmed exact stop instruction, the position stated in a block is approached precisely and very slowly, if necessary. In order 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 destination point precisely. The -> part program continues execution at the next block.

External zero offset

Zero 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 assigned. After fast retraction, an interrupt routine can also be executed (SINUMERIK 840D).

Feed override

The programmed velocity is overriden by the current velocity setting made via the → machine control panel or from the → PLC (0 to 200%). The feed velocity can also be offset by applying a programmable percentage factor (1 to 200%) in the machining program.

Finished-part contour

Contour of the finished workpiece. See -> blank.

Fixed machine point

A point defined uniquely by the machine tool, e.g. the 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. Where possible, the control moves these axes in -> rapid traverse.

Frame

A frame is an arithmetic rule that transforms one Cartesian coordinate system into another Cartesian coordinate system. A frame contains the 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 range in the workpiece coordinate system.

Global main program/subroutine

Every global main program/subroutine can only appear once under its own name in the directory, and it is not possible to have the same program name in different directories with different contents as a global program.

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

Helical interpolation is especially suitable for easy machining inside or outside threads with form cutters and for milling lubricating grooves.

The helix consists of two motions:

- A circular movement in one plane
- A linear movement perpendicular to this plane

High-level CNC language

The high-level language offers: -> User-defined variable, -> System variable, -> Macro technique.

High-speed digital inputs/outputs

Digital inputs can be used to start high-speed CNC program routines (interrupt routines), for example. The digital CNC outputs can be used to trigger fast, program-controlled switching functions (SINUMERIK 840D).

HIGHSTEP

Summary of the programming options for the -> PLC in the AS300/AS400 system.

I/O 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

Inch system

Measuring system that 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 "inclined-surface machining" function.

Increment

Traversed distance information via the 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.

Initialization block

Initialization blocks are special -> program blocks. They contain value assignments that are performed before program execution. The primary purpose of initialization blocks is to initialize predefined data or global user data.

Initialization files

It is possible to create an initialization file for each -> workpiece. Various variable assignments which are intended to apply specifically to one workpiece can be stored in this file.

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 that the control reads ahead can be set via system parameters.

Interpolator

Logical unit of the -> NCK which determines intermediate values for the movements to be traversed on the individual axes on the basis of destination positions specified in the parts program.

Interpolatory compensation

The interpolatory compensation allows manufacturing related Leadscrew Error Compensation and Measuring System Error Compensation (LEC, MSEC).

interrupt routine

Interrupt routines are special -> subroutines which can be started on the basis of events (external signals) in the machining process. A parts program block which is currently being worked through is interrupted and the position of the axes at the point of interruption is automatically saved.

Interrupts

All alarms and -> messages are output on the operator panel in plain text with the date and time and a symbol indicating the cancel criterion. The display is divided into alarms and messages.

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 to do this.

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, it is possible to set up the machine. Individual axes and spindles can be moved in this mode using the direction keys. Other functions available in JOG mode are -> reference point approach, -> repositioning and -> preset (setting an actual value).

Key switch

The key switch on the → machine control panel has 4 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.

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 may be limited by values defined in the machine data, the -> PLC or -> setting data.

Linear axis

The linear axis is an axis which, in contrast to a rotary axis, describes a straight line.

Linear interpolation

The tool travels along a straight line to the destination point while machining the workpiece.

Load memory

For the CPU 314 of the -> PLC, the load memory is equal to the -> Work memory .

Look ahead

With the look ahead function, a configurable number of traversing blocks is read in advance in order to calculate the optimum machining velocity.

Machine

Control operating area

Machine axes

Axes which exist physically 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 control the machine tool directly via the PLC.

Machine coordinate system

System of coordinates based on the axes of the machine tool.

Machine zero

A fixed point on the machine tool, which can be referenced by all (derived) measuring systems.

Machining channel

Via a channel structure, parallel sequences of movements, such as positioning a loading gantry during machining, can shorten unproductive times. 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 instructions under a single identifier. The identifier in the program refers to the grouped set of instructions.

Main block

A block prefixed by ":" containing all the parameters required to start execution of a -> parts program.

Main program

Parts program identified by a number or identifier in which further main programs, subroutines or -> cycles may be called.

Mains

The term "network" describes the connection of several S7-300 and other terminal devices, e.g. a programming device, via -> interconnecting cables. A data exchange takes place over the network between the connected devices.

MDI

Control operating mode: Manual Data Automatic. In MDA mode, it is possible to enter individual program blocks or sequences of blocks without reference to a main program or subroutine and to then execute them immediately via the NC start key.

Messages

All messages programmed in the parts program and -> alarms recognized by the system are output on the operator panel in plain text with the date and time and a symbol indicating the cancel criterion. The display is divided into alarms and messages.

Metric system

Standardized measuring system: for lengths in millimeters (mm), meters (m), etc.

Mirroring

Mirroring inverts the signs of the coordinate values of a contour with respect to an axis. It is possible to mirror in relation to more than one axis at a time.

Mode group

At any one time, all axes/spindles are assigned to just one channel. Each channel is assigned to a mode group. The same -> mode is always assigned to the channels in a mode group.

Multipoint interface

The multipoint interface (MPI) is a 9-pole Sub-D interface. A configurable number of devices can be connected to a multipoint interface and then communicate with each other.

- Programming devices
- Operator control and monitoring equipment
- Further automation systems

The parameter block "Multipoint Interface MPI" of the CPU contains the -> parameters which define the properties of the multipoint interface.

Name of identifier

The words according to DIN 66025 are supplemented by the identifiers (names) for variables (computer variable, system variable, user variable), for subroutines, for keywords and words with several address letters. In terms of the block format, these supplements have the same significance as the words. Identifiers must be unique. The same identifier must not be used for different objects.

NC

Numerical Control: NC control incorporates all the components of the of the machine tool control system: -> NCK, -> PLC, HMI, -> COM.

Note

CNC (Computerized Numerical Control) is a more accurate term for the SINUMERIK 840D controls. MARS and Merkur controls.

NCK

Numerical Control Kernel: Component of the NC control which executes -> parts programs and essentially coordinates the movements on the machine tool.

NRK

Numeric Robotic Kernel (operating system of the -> NCK)

NURBS

Internal motion control and path interpolation are performed using NURBS (non-uniform rational B-splines). This provides a uniform internal method for all interpolations in the control (SINUMERIK 840D).

OEM

For machine manufacturers who manufacture their own user interface or wish to integrate their own technology-specific functions in the control, free space has been left for individual solutions (OEM applications) for SINUMERIK 840D.

Offset memory

Data range in the control in which the tool offset data are stored.

Operating mode

An operating concept on a SINUMERIK control. The operating modes -> Jog, -> MDA and -> Automatic are defined.

Oriented spindle stop

Stops the workpiece spindle with a specified orientation angle, e.g. to perform an additional machining operation at a specific position.

Oriented tool retraction

RETTOOL: If machining is interrupted (because of tool breakage, for example), a program command can be used retract the tool with a defined orientation by a defined path.

Overall reset

In the event of an overall reset, the following memories of the → CPU are deleted:

- -→ RAM
- 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

Series of instructions to the NC that act in concert to produce a particular → workpiece. Likewise, this term applies to execution of a particular machining operation on a given → raw part.

Part program block

Part of a → part program that is demarcated by a line feed. There are two types: → main blocks and → subblocks.

Part program management

Part program management can be organized by → workpieces. The size of the user memory determines the number of programs and the amount of data that can be managed. Each file (programs and data) can be assigned a name comprising up to 24 alphanumeric characters.

Path axis

Path axes are all the machining axes in the -> channel which are controlled by the -> interpolator so that they start, accelerate, stop and reach their end positions simultaneously.

Path feed

Path feed acts on -> path axes. It represents the geometrical sum of the feeds on the participating -> geometry axes.

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.

PLC

Programmable Logic Control: Component of → NC: Programmable controller for processing the control logic of the machine tool.

PLC program memory

SINUMERIK 840D: The PLC user program and the user data are stored together with the PLC basic program 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 that defines the position of a point on a plane in terms of its distance from the zero point and the angle formed by the radius vector with a defined axis.

Polynominal 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 which performs an auxiliary movement on a machine tool (e.g. tool magazine, pallet transport). Positioning axes are axes that do not interpolate using → 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 Controller

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 largely eliminated using quadrant error compensation. Quadrant error compensation is parameterized by a circularity 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 speed of an axis. It is used for example to move the tool from rest position to the -> workpiece contour or retract the tool from the contour.

Raw part

Workpiece as it is before it is machined.

Reference point

Machine tool position that the measuring system of the → machine axes references.

Rotary axis

Rotary axes rotate a workpiece or tool 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 montoring 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 malfunction, the machining sequence is interrupted and the drives are stopped and the cause of the malfunction is saved and displayed as an alarm. At the same time, the PLC is informed that a CNC alarm is pending.

Scaling

Component of a → frame that implements axis-specific scale modifications.

Serial V.24 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.

Services

Control operating area

Setting data

Data that communicate properties of the machine tool to the NC control in a manner defined by the system software.

Softkey

A key whose name appears on an area of the screen. The selection of keys displayed is adapted dynamically to the operating situation. The freely assignable function keys (softkeys) are assigned defined functions in the software.

Software limit switch

Software limit switches limit the traversing range of an axis and prevent an abrupt stop of the slide at the hardware limit switch. Two value pairs can be specified for each axis and activated separately by means of the → PLC.

Spline interpolation

With spline interpolation, the controller can generate a smooth curve characteristic from only a few specified interpolation points of a set contour.

SRT

Speed ratio

Standard cycles

Standard cycles are available for frequently recurring machining tasks.

- for drilling/milling technology
- for turning technology

In the "Program" operating area, the available cycles are listed under the "Cycle Support" menu. After selecting the desired machining cycle, the required parameters for the value assignment are displayed in plain text.

Subblock

Block prefixed by "N" containing information for a machining step such as position 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. It is not possible to block every subroutine against unauthorized reading and displaying. → Cycles are a form of subroutines.

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 as geometry axes take for their path.

System memory

The system memory is a memory in the CPU in which the following data are stored:

- Data required by the operating system
- The operands times, counters, markers

System variable

A variable that exists without any input from the programmer of a → part program. It is defined by a data type and variable name preceded by the character **$**.
See → User-defined variable.

TappingRigid

This function allows threads to be tapped without a compensating chuck. By using the method whereby the spindle, as a rotary axis, and the drilling axis interpolate, 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 are 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 utilized tool must be communicated to the control which then takes it into account. The curvature center is maintained equidistantly around the contour offset by the radius of curvature.

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 work offset of an axis.

Traversing range

The maximum permissible traversing range for linear axes is ± 9 decades. The absolute value depends on the selected input and positioning resolutions and the basic unit system used (inches or metric).

User interface

The user interface (UI) is the display medium for a CNC control in the form of a screen. It is laid out with horizontal and vertical softkeys.

User memory

All program and data, such as part programs, subroutines, comments, tool compensations, and work offsets/frames, as well as channel- and program user data can be stored in the shared CNC user memory.

User program

User programs for the S7-300 automation systems are created using the programming language STEP 7. The user program has a modular layout and consists of individual blocks.

The basic block types are:
code modules: these blocks contain the STEP 7 commands.
Data blocks: these blocks contain the constants and variables for the STEP 7 program.

User-defined variable

The user can declare user-defined variables for any use in the -> parts 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 be able to achieve an acceptable traversing velocity on very short traverse movements within a single block, predictive velocity control can be set over several blocks (-> look ahead).

Work offset

Specification of a new reference point for a coordinate system through reference to an existing zero point and a -> frame.

1. Adjustable

   SINUMERIK 840D: A configurable number of adjustable zero offsets is available for each CNC axis. The offsets which can be selected via G functions are effective on an alternating basis.

2. External

   In addition to all the offsets which define the position of the workpiece zero point, an external zero offset can be overlaid by means of the handwheel (DRF offset) or from the PLC.

3. Programmable

   Zero offsets are programmable for all path and positioning axes with the TRANS command.

Working area

Three-dimensional zone into which the tool tip can be moved on account of the physical design of the machine tool. See -> protection zone.

Working area limitation

With the aid of the working area limitation, the traversing range of the axes can be further restricted in addition to the limit switches. One value pair per axis may be used to describe the protected working area.

Working memory

The working area is a RAM area in the -> CPU which is accessed by the processor to access the user program during program execution.

Workpiece

Part to be created/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. In machining operations programmed in the workpiece coordinate system, the dimensions and directions refer to this system.

Workpiece zero

The workpiece zero is the starting point for the → workpiece coordinate system. It is defined in terms of distances to the → machine zero.

Index