Programming Guide 03/2004 Edition

sinumerik

SINUMERIK 840D/840Di/810D
Advanced

SIEMENS

SIEMENS

SINUMERIK 840D/840Di/810D

Programming Guide Advanced

Programming Guide

Valid for

Control	*Software Version*
SINUMERIK 840D powerline	7
SINUMERIK 840DE powerline (export version)	7
SINUMERIK 840Di	3
SINUMERIK 840DiE (export version)	3
SINUMERIK 810D powerline	7
SINUMERIK 810DE powerline (export version)	7

03.04 Edition

SINUMERIK® Documentation

Printing history

Brief details of this edition and previous editions are listed below.

The status of each edition is shown by the code in the "Remarks" column.

Status code in the "Remarks" column:

A .... New documentation.

B .... Unrevised reprint with new Order No.

C .... Revised edition with new status.
If factual changes have been made on the page since the last edition, this is indicated by a new edition coding in the header of that page.

Edition	**Order No.**	**Comment**
02.95	6FC5298-2AB00-0BP0	A
04.95	6FC5298-2AB00-0BP1	C
12.95	6FC5298-3AB10-0BP0	C
03.96	6FC5298-3AB10-0BP1	C
08.97	6FC5298-4AB10-0BP0	C
12.97	6FC5298-4AB10-0BP1	C
12.98	6FC5298-5AB10-0BP0	C
08.99	6FC5298-5AB10-0BP1	C
04.00	6FC5298-5AB10-0BP2	C
10.00	6FC5298-6AB10-0BP0	C
09.01	6FC5298-6AB10-0BP1	C
11.02	6FC5298-6AB10-0BP2	C
03.04	6FC5298-7AB10-0BP0	C

This manual is included in the documentation on CD-ROM (**DOCONCD**)

Edition	Order No.	Comment
03.04	6FC5298-7CA00-0BG0	C

Trademarks

SIMATIC®, SIMATIC HMI®, SIMATIC NET®, SIROTEC®, SINUMERIK®, and SIMODRIVE® are registered trademarks of Siemens AG. The other designations in this publication may also be trademarks, the use of which by third parties may constitute copyright violation.

Further information is available on the Internet under:
http://www.ad.siemens.de/sinumerik

This publication was produced with WinWord V9.0
and Designer V7.0.

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.

We have checked that the contents of this documentation correspond to the hardware and software described. Nonetheless, differences might exist and we cannot therefore guarantee that they are completely identical. The information contained in this document is, however, reviewed regularly and any necessary changes will be included in the next edition. We welcome suggestions for improvement.

Subject to change without prior notice.

Order No. 6FC5298-7AB10-0BP0
Printed in Germany

Siemens Aktiengesellschaft

Preface

Overview of documentation

The SINUMERIK documentation is organized in three parts:

- General Documentation
- User Documentation
- Manufacturer/Service Documentation

Target group

This documentation is intended for the programmer.
It provides detailed information for programming the SINUMERIK 840D/840Di/810D.

Standard scope

The Programming Guide describes the functionality included in the standard scope. Extensions or changes made by the machine tool manufacturer are documented by the machine tool manufacturer.

You can obtain more detailed information on publications about SINUMERIK 840D/840Di/810D or publications that apply to all the SINUMERIK controls (e.g. universal interface, measurement cycles, etc.), from your Siemens branch.

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.

Validity

This Programming Guide is valid for the following controls:

SINUMERIK 840D powerline	SW7
SINUMERIK 840DE powerline (export version)	SW7
SINUMERIK 840DiE	SW3
SINUMERIK 810DiE (export version)	SW3
SINUMERIK 810D powerline	SW7
SINUMERIK 810DE powerline (export version)	SW7

with operator panel fronts OP 010, OP 010C, OP 010S, OP 12 or OP 15 (PCU 20 or PCU 50)

SINUMERIK 840D powerline

From 09.2001, the

- SINUMERIK 840D powerline and the
- SINUMERIK 840DE powerline

will be available with improved performance. A list of the available **powerline** modules can be found in the Hardware Reference Manual

- /PHD/ in Section 1.1

SINUMERIK 810D powerline

From 12.2001, the

- SINUMERIK 810D powerline and the
- SINUMERIK 810DE powerline

will be available with improved performance. A list of the available **powerline** modules can be found in the Hardware Reference Manual

- /PHC/ in Section 1.1

Hotline

Should you have any questions, please consult the following Hotline:
A&D Technical Support Tel.: ++49-(0)180-5050-222
Fax: ++49-(0)180-5050-223
E-mail: adsupport@siemens.com

If you have any questions about the documentation (suggestions, corrections) please send a fax to the following fax address, or e-mail us:
Fax: ++49-(0)0131-98-2176
E-mail: motioncontrol.docu@erlf.siemens.de
Fax form: see the feedback page at the back of this document.

Internet address http://www.ad.siemens.de/sinumerik

Export version

The following functions are not available in the export version:

Function	810DE	840DE
Machining package for 5 axes	–	–
Transformation package handling (5 axes)	–	–
Multiple axes interpolation (> 4 axes)	–	–
Helix interpolation 2D+6	–	–
Synchronized actions stage 2	–	O[1]
Measurement stage 2	–	O[1]
Adaptive control	O[1]	O[1]
Continuous dressing	O[1]	O[1]
Use of the compile cycles (OEM)	–	–
Multidimensional sag compensation	O[1]	O[1]

– Function not available
1) Limited functionality

Structure of the descriptions

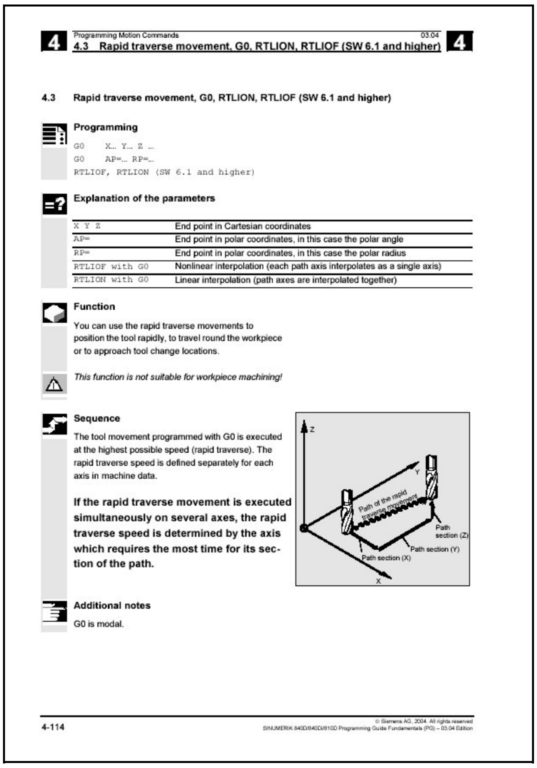
All cycles and programming options have been described – where appropriate and possible – according to the same internal structure. The organization into different information levels allows you to find the information you need quickly.

1. At a glance

If you want to look up a seldom used command or the meaning of a parameter, you can see at a glance how to program the function together with an explanation of the commands and parameters.

This information is always presented at the start of the page.

Note:
To keep this documentation as compact as possible, it is not always possible to list all the types of representation available in the programming language for the individual commands and parameters. The commands are therefore always programmed in the context most frequently used in the workshop.



4 Programming Motion Commands 03.04 4
4.3 Rapid traverse movement, G0, RTLION, RTLIOF (SW 6.1 and higher)

4.3 Rapid traverse movement, G0, RTLION, RTLIOF (SW 6.1 and higher)

Programming

```
G0    X… Y… Z …
G0    AP=… RP=…
RTLIOF, RTLION (SW 6.1 and higher)
```

Explanation of the parameters

X Y Z	End point in Cartesian coordinates
AP=	End point in polar coordinates, in this case the polar angle
RP=	End point in polar coordinates, in this case the polar radius
RTLIOF with G0	Nonlinear interpolation (each path axis interpolates as a single axis)
RTLION with G0	Linear interpolation (path axes are interpolated together)

Function

You can use the rapid traverse movements to position the tool rapidly, to travel round the workpiece or to approach tool change locations.

This function is not suitable for workpiece machining!

Sequence

The tool movement programmed with G0 is executed at the highest possible speed (rapid traverse). The rapid traverse speed is defined separately for each axis in machine data.

If the rapid traverse movement is executed simultaneously on several axes, the rapid traverse speed is determined by the axis which requires the most time for its section of the path.

Additional notes

G0 is modal.

4-114
© Siemens AG, 2004. All rights reserved
SINUMERIK 840D/840Di/810D Programming Guide Fundamentals (PG) – 03.04 Edition

2. Detailed explanations

The theory part contains detailed information on the following:

What is the purpose of the command?

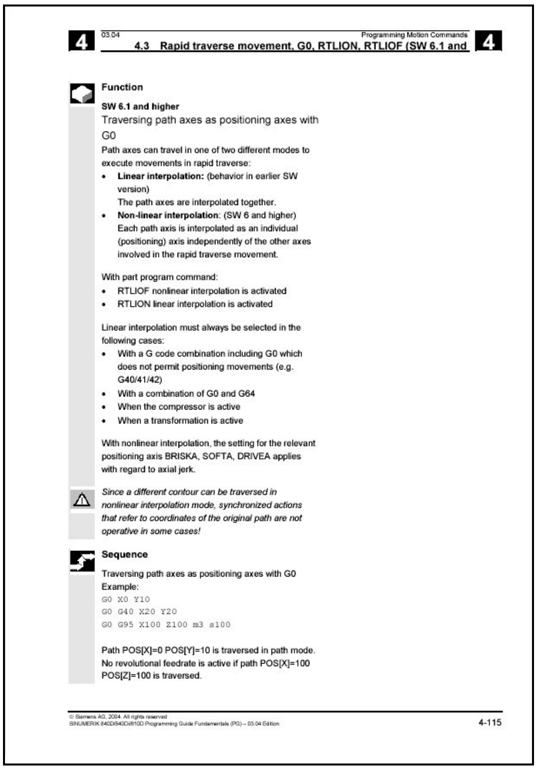
What is the effect of the command?

What is the sequence of command?

What effect do the parameters have?

What else has to be taken into account?

The theory parts are suitable primarily as a guide for NC beginners. Work through the manual carefully at least once to gain an overview of the performance scope and capabilities of your SINUMERIK control.



4 03.04 Programming Motion Commands 4
4.3 Rapid traverse movement, G0, RTLION, RTLIOF (SW 6.1 and

Function

SW 6.1 and higher

Traversing path axes as positioning axes with G0

Path axes can travel in one of two different modes to execute movements in rapid traverse:

- **Linear interpolation:** (behavior in earlier SW version)
  The path axes are interpolated together.
- **Non-linear interpolation**: (SW 6 and higher)
  Each path axis is interpolated as an individual (positioning) axis independently of the other axes involved in the rapid traverse movement.

With part program command:

- RTLIOF nonlinear interpolation is activated
- RTLION linear interpolation is activated

Linear interpolation must always be selected in the following cases:

- With a G code combination including G0 which does not permit positioning movements (e.g. G40/41/42)
- With a combination of G0 and G64
- When the compressor is active
- When a transformation is active

With nonlinear interpolation, the setting for the relevant positioning axis BRISKA, SOFTA, DRIVEA applies with regard to axial jerk.

⚠ *Since a different contour can be traversed in nonlinear interpolation mode, synchronized actions that refer to coordinates of the original path are not operative in some cases!*

Sequence

Traversing path axes as positioning axes with G0
Example:

```
G0 X0 Y10
G0 G40 X20 Y20
G0 G95 X100 Z100 m3 s100
```

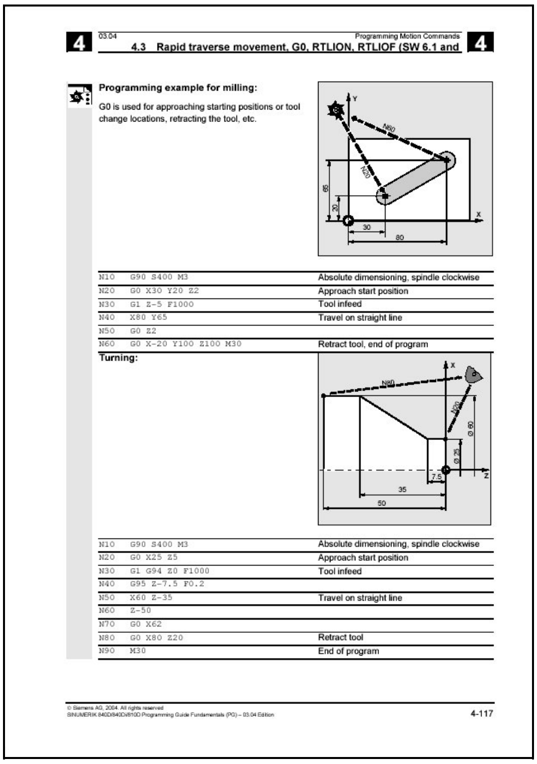
Path POS[X]=0 POS[Y]=10 is traversed in path mode. No revolutional feedrate is active if path POS[X]=100 POS[Z]=100 is traversed.

© Siemens AG, 2004. All rights reserved
SINUMERIK 840D/840Di/810D Programming Guide Fundamentals (PG) – 03.04 Edition 4-115

3. From theory to practice

The programming example shows you how to apply the commands in the program.

You will find an application example for practically all the commands after the theory part.



4 03.04 Programming Motion Commands 4
4.3 Rapid traverse movement, G0, RTLION, RTLIOF (SW 6.1 and

Programming example for milling:

G0 is used for approaching starting positions or tool change locations, retracting the tool, etc.

N10	G90 S400 M3	Absolute dimensioning, spindle clockwise
N20	G0 X30 Y20 Z2	Approach start position
N30	G1 Z-5 F1000	Tool infeed
N40	X80 Y65	Travel on straight line
N50	G0 Z2	
N60	G0 X-20 Y100 Z100 M30	Retract tool, end of program

Turning:

N10	G90 S400 M3	Absolute dimensioning, spindle clockwise
N20	G0 X25 Z5	Approach start position
N30	G1 G94 Z0 F1000	Tool infeed
N40	G95 Z-7.5 F0.2	
N50	X60 Z-35	Travel on straight line
N60	Z-50	
N70	G0 X62	
N80	G0 X80 Z20	Retract tool
N90	M30	End of program

© Siemens AG, 2004. All rights reserved
SINUMERIK 840D/840Di/810D Programming Guide Fundamentals (PG) – 03.04 Edition 4-117

Explanation of the symbols

Sequence of operations

Explanation

Function

Parameters

Programming example

Programming

Additional notes

Cross-references to other documentation and sections

Important information and safety notices

For your information
Your SINUMERIK 840D/840Di/810D is state of the art and is manufactured in accordance with recognized safety regulations, standards and specifications.

Additional devices
SIEMENS offers special add-on equipment, products and system configurations for the focused expansion of SIEMENS controls in your field of application.

Personnel
Only **specially trained, authorized and experienced personnel** should be allowed to work on the control. This applies at all times, even for short periods.

It is necessary to clearly **define** the respective **responsibilities** of the personnel for setting up, operation and maintenance; it is necessary to **supervise** the compliance thereof.

Actions
It must be ascertained that the Instruction Manuals have been read and understood by the persons working on the control **before** installation and start-up of the control. In addition, operation must be conducted under **constant supervision** regarding the overall technical state (faults and damages visible from outside, as well as changes in operation behavior) of the control.

Service
Only **qualified personnel specifically trained** for this purpose should be allowed to perform repairs, and only in accordance with the contents of the maintenance guides. Hereby, all established safety regulations have to be complied with.

Note
The following are considered **not compliant with the usage to the intended purposes** and are therefore **excluded from all liability of the manufacturer**:

Every usage not complying with or going beyond the abovementioned points.

If the control is **not operated in a technically faultless state**, if proper safety precautions are not taken, or if the instructions in the Instruction Manual are not complied with.

If faults which could influence safety of operation are not remedied **before** installation and start-up of the control.

Each **change, jumpering** or **shut-down** of devices on the control which serve for proper functioning, universal usage and active and passive safety.

Unforeseen dangers may result in:

- personal injury and death,
- damage to the control, machine and other property of the company and operator.

The following notes used in the documentation have a special significance:

Notes
This symbol always appears in the documentation if secondary information is given and there is an important fact to be considered.

In this documentation, you will find the symbol shown with reference to an ordering data option. The function described can only be run if the control includes the designated option.

Warnings
The following warnings, of graduated significance, are used in the publication.

Danger
Indicates an imminently hazardous situation which, if not avoided, **will** result in death or serious injury or in substantial property damage.

Notice
Indicates a potentially hazardous situation which, if not avoided, **could** result in death or serious injury or in substantial property damage.

Caution
Used with the safety alert symbol indicates a potentially hazardous situation which, if not avoided, **may** result in minor or moderate injury or in property damage.

Caution
Used without safety alert symbol indicates a potentially hazardous situation which, if not avoided, **may** result in property damage.

Notice
Used without the safety alert symbol indicates a potential situation which, if not avoided, **may** result in an undesirable result or state.

■

Contents

Special Motion Commands 5-189

Frames 6-237

Transformations 7-269

Tool Offsets 8-319

Path Action 9-361

Motion Synchronous Actions 10-417

Oscillation 11-481

Punching and Nibbling 12-501

Additional Functions 13-513

Notes

Flexible NC Programming

1.1 Variables and arithmetic parameters

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 classes
The control uses 3 classes of variable:

User-defined variable	Name and type of variable defined by the user, e.g. arithmetic parameter.
Arithmetic parameter	Special, predefined arithmetic variable whose address is R plus a number. The predefined arithmetic variables are of the REAL type.
System variable	Variable provided by the controller 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 sign	$\pm(2^{31} – 1)$
`REAL`	Real numbers (fractions with decimal point, LONG REAL to IEEE)	$\pm(10^{-300} \dots 10^{+300})$
`BOOL`	Boolean values: TRUE (1) and FALSE (0)	1, 0
`CHAR`	1 ASCII character specified by the code	0 … 255
`STRING`	Character string, number of characters in [...], Max. 200 characters	Sequence of values with 0 ... 255
`AXIS`	Axis identifiers only (axis addresses)	All axis identifiers and spindles in the channel

FRAME	Geometric data for translation, rotation, scaling, mirroring, see Chapter 4.

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

The exact number of arithmetic variables (up to 1000) 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.

Special identifiers of system variables always begin with a "$" sign followed by the specific names.

Summary of system variable types

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

2. 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

User-defined variables
In addition to the predefined variables, the programmer can define and initialize his own variables.
Local variables (LUD) are only valid in the program where they are defined.
Global variables (GUD) are valid in all programs.
SW 4.4 and higher:
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 the machine manufacturer's information.

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

Example:

```
$MN_LUD_EXTENDED_SCOPE=1

PROC MAIN          ;Main program
DEF INT VAR1       ;PUD definition
...
SUB2               ;Subprogram call
...
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
SUB3               ;Subprogram call
...
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
```

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 subprograms.

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.

Programming

DEF INT name
or DEF INT name=value

DEF REAL name
or DEF REAL name1,name2=3,name4
or DEF REAL name[array_index1,array_index2]

DEF BOOL name

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

DEF STRING[string_length] name

DEF AXIS name
or DEF AXIS name[array_index]

DEF FRAME name

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.

Explanation

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. sequence of char
AXIS	Variable type axis, i.e. axis addresses and spindles
FRAME	Variable type frame, i.e. geometric data
name	Variable name

Programming examples

Variable type INT

`DEF INT NUMBER`	This creates a variable of type integer with the name NUMBER. The system initializes the variable with zero.
`DEF INT NUMBER=7`	This creates a variable of type integer with the name NUMBER. The system initializes the variable with zero.

Variable type 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.

Variable type 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.

Variable type 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").

Variable type 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.

Variable type AXIS

`DEF AXIS AXIS_NAME=(X1)`	Variables of type AXIS have the name AXIS_NAME and are assigned the axis identifier of a channel, X1 in this case.

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

Variable type FRAME

`DEF FRAME BEVEL_1`	Variables of type FRAME have names like BEVEL_1.

Other Information

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 of programming with program-local variables

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

Programming example

Query of existing geometry axes

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

Indirect spindle programming

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

1.3 Array definition

Programming

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

Explanation

`INT NAME[n,m]`
`REAL NAME[n,m]`
	Variable type (CHAR, INTEGER, REAL, AXIS, FRAME, BOOL)
	n = array size for 1st dimension
	m = array size for 2nd dimension

`DEF STRING[string_length] NAME[m]`	Data type STRING can only be defined for 1-dimensional arrays
`NAME`	Variable name

The same memory size applies to type BOOL as to type CHAR.
Up to SW3:
The maximum size of an array is set via machine data.

Machine manufacturer

See the machine manufacturer's information.

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

The maximum array size determines the size of the memory blocks 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.

SW 4 and higher:
An array can be larger than a memory block. The MD value for block size should be set such that arrays are fragmented only in exceptional cases.

Default: 256 bytes

Memory requirement per element: See above

Example:

Global user data must contain PLC machine data for switching the controller on/off (definition of BOOL arrays).

Other information

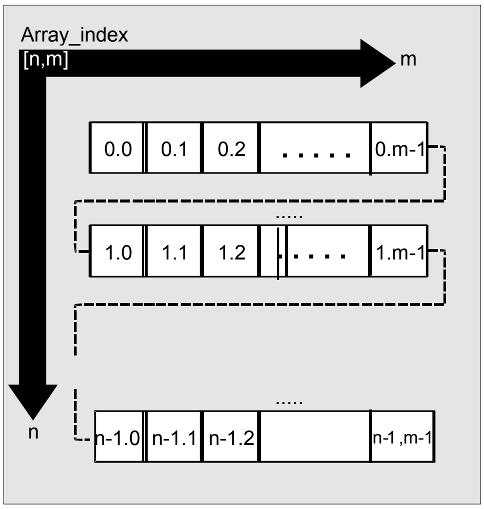
Arrays with up to 2 dimensions can be defined.

Arrays with variables of type STRING can only be 1-dimensional. The string length is specified after the data type String.

Array index

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

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



In the above example, the values have been initialized to double as the index of the array element. This shows the order of the array elements.

Initialization of arrays
The array elements can be initialized during program run or in the array definition.

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

Initialization with value lists, SET

1. Initializing in the array definition

```
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, …)
```

- 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)
```

SET is optional in the array definition.

2. Initializing during the program run

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

- 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, REP

1. Initializing in the array definition

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

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)
```

2. Initializing during the program run

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

- 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.

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

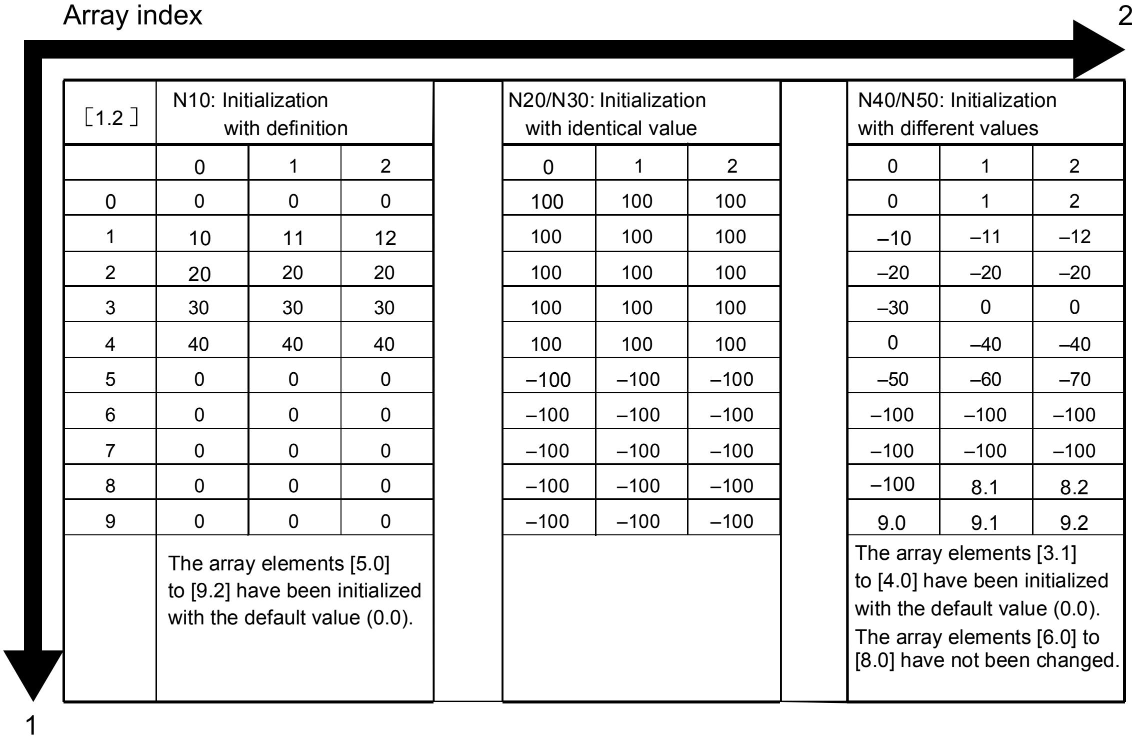
Example:
Initialization of all elements with one value

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

Programming 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)
```



[1.2]	N10: Initialization with definition			N20/N30: Initialization with identical value			N40/N50: Initialization with different 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.		

1.4 Indirect programming

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

All addresses are parameterizable except:

- N – Block number
- G – G command
- L – Subprogram

Indirect programming is not possible for settable addresses.
Example: X[1] in place of X1 is not permissible.

Programming

```
ADDRESS[INDEX]
```

Programming examples

Spindle

`S1=300`	Direct programming

`DEF INT SPINU=1`
`S[SPINU]=300`
	Indirect programming:
	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`
`FA[AXVAR2]=300`
	Indirect programming:
	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`
`$AA_MM[AXVAR3]`
	Indirect programming:
	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	
DEFINE DIM2 AS 5	
DEF INT ARRAY[DIM1,DIM2]	
ARRAY[DIM1-1,DIM2-1]=5	Indirect programming: Array dimensions must be stated as constant values.

Axis assignment with axis variables

X1=100 X2=200	Direct programming
	Indirect programming:
DEF AXIS AXVAR1 AXVAR2	Definition of variables

AXVAR1=(X1) AXVAR2=(X2)
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
	Indirect programming:
DEF AXIS AXVAR1=X	Definition and assignment of the axis name
G2 X100 IP[AXVAR1]=20	Indirect programming of the center

Indirect subprogram call

CALL "L" << R10	Call of the program whose number is in R10

Other Information

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

Indirect G code programming from SW 5

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

Indirect programming of G codes using variables for effective cycle programming

Meaning of parameters

G<group_index>	Integer constants with which the G code group is selected.
<integer/real_variable>	Variable of the integer or real type with which the G code number is selected.

Function

Indirect G code programming (SW 5 and higher)
Indirect programming of G codes using variables facilitates effective cycle programming. Two parameters

• G code groups	integer constant
• G code numbers	variable of integer/real type

are available for this purpose.

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 legal 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.

Other information

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

Programming 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 – G19
```

Run string as parts program line

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

Command EXECSTRING runs a parts program line indirectly

Meaning of parameters

<string_variable>	Parameter of type string is passed with EXECSTRING

Function

EXECSTRING (from SW 6.4)
Parts program command EXECSTRING passes a string as a parameter that already contains the parts program line to run.

Other information

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.

Programming 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

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

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.

Programming example

`R1=10.518 R2=4 VARI1=45`
`X=47.11 Y=R2`
	Assignment of a numeric value

`R1=R3 VARI1=R4`	Assignment of a suitable type variable
`R4=-R5 R7=-VARI8`	Assignment with opposite sign (only permitted for types INT and REAL)

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.

1.6 Arithmetic operations and functions

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.

Use of arithmetic operations requires conventional mathematical notation. Priorities for execution are indicated by parentheses. Angles are specified for trigonometry functions and their inverse functions (right angle = 90°).

Operators/mathematical functions

`+`	Addition
`–`	Subtraction
`*`	Multiplication
`/`	Division **Notice**: (Type `INT`)**/**(Type `INT`)**=**(Type `REAL`); Example: 3/4 = 0.75
`DIV`	Division, for variable type INT and REAL **Notice**: (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

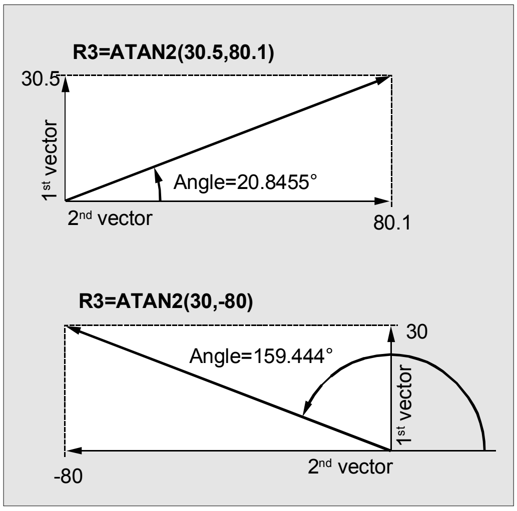
`CSCALE ()`	Change of scale
`CMIRROR ()`	Mirroring

Programming examples

`R1=R1+1`	New R1 = old R1 +1
`R1=R2+R3  R4=R5-R6  R7=R8*R9`	
`R10=R11/R12  R13=SIN(25.3)`	
`R14=R1*R2+R3`	Multiplication or division takes precedence over addition or subtraction
`R14=(R1+R2)*R3`	Parentheses are calculated first
`R15=SQRT(POT(R1)+POT(R2))`	Inner parentheses are resolved first R15 = square root of ($R1^2+R2^2$)

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

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 (–180 < 0 < +180°). The angular reference is always based on the 2nd value in the positive direction.



1.7 Comparison and logic operators

Comparison operators
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 <>.

The result of comparison operations is always of type BOOL.

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

Meaning of comparison operators

==	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Programming example

```
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.

Precision correction on comparison errors

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

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

Function

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.

SW 6.3 and lower
Relative equality considered 10^{-12} for

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

SW 6.4 and higher
Relative equality considered 10^{-12} for

• Greater than	(>)
• Less than	(<)

Programming notes
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.

Compatibility
For compatibility reasons, the check for relative equality with (>) and (<) can be deactivated by setting MD 10280: PROG_FUNCTION_MASK Bit0 = 1.

Programming examples

Precision issues

`N40 R1=61.01 R2=61.02 R3=0.01`	Assignment of initial values
`N41 IF ABS(R2-R1) > R3 GOTOF ERROR`	Jump executed (up to SW 6.3)
`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)`
`R13=TRUNC(R3*1000)`
	Precision correction

`IF ABS(R12-R11) > R13 GOTOF ERROR`	Jump not executed
`M30`	End of program
`ERROR: SETAL(66000)`	

Calculate and evaluate 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 ERROR`	Jump not executed
`M30`	End of program
`ERROR: SETAL(66000)`	

Logic operators

Logic operators are used to link truth values.
AND, OR, NOT, and XOR can only be applied to variables of type BOOL. However, they can also be applied to data types CHAR, INT, and REAL by implicit type conversion.

Spaces must be left between BOOLEAN operands and operators.

For the logic (Boolean) operations, the following applies to data types BOOL, CHAR, INT, and REAL:
0 means FALSE
not equal to 0 means TRUE

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.

Programming example

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

IF NOT R10 GOTOB START
```

NOT is only applied to one operand.

Bit logic operators

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

Meaning of bit logic operators

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

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

Programming example

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

1.8 Priority of the operators

Priority of the operators

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

Example of IF statement:
If (otto==10) and (anna==20) gotof end

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.

1.9 Possible type conversions

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

* 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

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

Other information

If mixed types occur in an expression, type conversion is automatic.

1.10 String operations

Overview

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

Explanation

Type conversion to STRING:

`STRING_ERG = <<bel._Typ`[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:

`bel._Typ`[1] `<< bel. Typ`[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 = STRING_ARRAY [IDX_ARRAY, IDX_CHAR]`	Result type: CHAR

[1] "`bel._Typ`" 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

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

Conversion to STRING

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.

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

FRAME variables cannot be converted.

Example:

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

Conversion from STRING

The NUMBER function converts from STRING to REAL.

If ISNUMBER returns the value FALSE, CALLING NUMBER with the same parameter will trigger 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.

Syntax

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

Semantics:

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.

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

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

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

Examples

`DEF BOOL BOOL_ERG`	
`DEF REAL REAL_ERG`	
`DEF AXIS AXIS_ERG`	
`DEF STRING[32] STRING_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
`STRING_ERG = AXSTRING(X)`	;Now: STRING_ERG == "X"
`AXIS_ERG = AXNAME("X")`	;Now: AXIS_ERG == X

1.10.2 Concatenation of strings

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. Types FRAME and AXIS cannot be used with this operator.

Syntax:

`bel._Typ << bel._Typ`	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).

Syntax:

`<< bel._Typ`	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)
```

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

Programming examples

```
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:"          ;Display: "Index: 2/value: 9.654"
<<VALUE)
NO_MSG:
```

1.10.3 Conversion to lower/upper case

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 MMC, they can be given standard capitalization (upper or lower case):

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

1.10.4 Length of the string

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
```

1.10.5 Look for character/string in the string

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.

`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.

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 ("/").

Programming example

Example of breaking up 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 (INPUT, "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, PROGIDX-PATHIDX-1)`	Returning "_N_MPF_DIR"
`VARIABLE = SUBSTR (INPUT, PROGIDX)`	Returning "_N_EXECUTE_MPF"

1.10.6 Selection of a substring

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.

`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"

1.10.7 Selection of a single character

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

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"
```

Single character access is possible only to user-defined variables (LUD, GUD, and PUD data). This type of access is also possible only for "call-by-value" type parameters in subprogram calls.

Examples:

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
```

Single character access in call-by-reference parameter:

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

1.11 CASE instruction

Programming

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

Explanation of the commands

CASE	Vocabulary word for jump instruction
GOTOB	Jump instruction with jump destination backward (toward the beginning of program)
GOTOF	Jump instruction with forward jump destination (toward the end of program)
GOTO	Jump instruction 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 instruction 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

Function

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

Sequence

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

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 instruction can be used to determine the jump destination.

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

Programming example

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

Explanation

`IF-ELSE-ENDIF`	Selection between 2 alternatives
`LOOP-ENDLOOP`	Endless loop
`FOR-ENDFOR`	Count loop
`WHILE-ENDWHILE`	Loop with condition at beginning of loop
`REPEAT-UNTIL`	Loop with condition at end of loop

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.

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 program loop 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. Count 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 the INT type.

FOR Variable = start value **TO** end value
NC blocks
ENDFOR

4. Program loop with condition at start of the loop WHILE

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

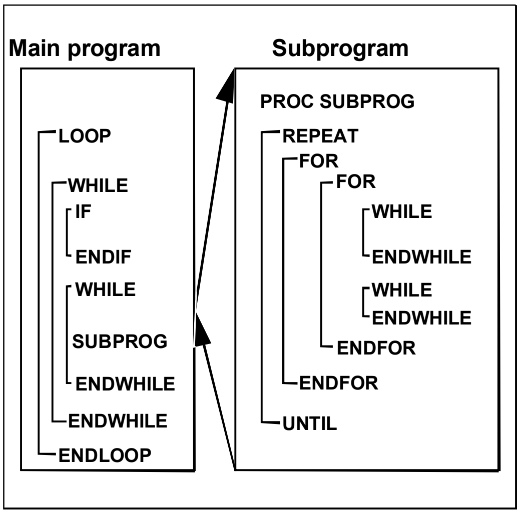
WHILE expression
NC blocks
ENDWHILE

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)

Nesting depth
Check structures apply locally within programs. A nesting depth of up to 8 check structures can be set up on each subprogram level.



Runtime response
In interpreter mode (active as standard), it is possible to shorten program processing times more effectively by using program branches than can be obtained with check structures.
There is no difference between program branches and check structures in precompiled cycles.

Restrictions

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

Check structures are processed interpretively. When a loop end is detected, a search is made for the loop beginning, allowing for the check structures found in the process.
For this reason, the block structure of a program is not checked completely in interpreter mode.
It is not generally advisable to use a mixture of check structures and program branches.
A check can be made to ensure that check structures are nested correctly when cycles are preprocessed.

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

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

Programming example

1. 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-5 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
```

2. Production of a fixed quantity of parts

```
%_N_WKPCCOUNT_MPF

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

1.13 Program coordination

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 instructions (commands) for this program coordination. Each instruction is programmed separately in a block.

Note
Program coordination in the own channel is possible from SW 5.3.

Program coordination instructions

• **Specification with absolute path**	

`INIT ) (n,"/_HUGO_DIR/_N_`*`name`*`_MPF" )`
or
`INIT (n,"/_N_MPF_DIR/_N_`*`name`*`_MPF" )`
	The absolute path is programmed according to the following rules:
	• *Current directory/*_N_*name*_MPF "current directory" stands for the selected workpiece directory or the standard directory /*_N_MPF_DIR*.
	• Selects a particular program for execution in a particular channel: n: Number of the channel, value per control configuration
	• Complete program name

Example:
`INIT(2,"/_N_WKS_DIR/_DRESS_MPF")`
`G01 F0.1`
`START`
 	
`INIT (2,"/_N_WKS_DIR/_N_UNDER_1_SPF")`
	Up to SW 3:
	At least one executable block must be programmed between an **init** command (without synchronization) and an **NC start**.
	With subprogram calls "_SPF" must be added to the path.

• **Relative path specification**	
Example: `INIT(2,"DRESS")`	The same rules apply to relative path definition as for program calls.
`INIT(3,"UNDER_1_SPF")`	With subprogram calls "_SPF" must be added to the program 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)
`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. SETM() can also be programmed independently of a synchronized action.
`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. CLEARM() can also be programmed independently of a synchronized action.

Note
All the above commands must be programmed in separate blocks.
The number of markers depends on the CPU used.

Channel names

Channel names must be converted to numbers via variables (see Chapter 10 "Variables and Arithmetic Parameters").

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

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 instruction START is therefore:

```
START(MACHINE)
```

Example of program coordination

Channel 1:

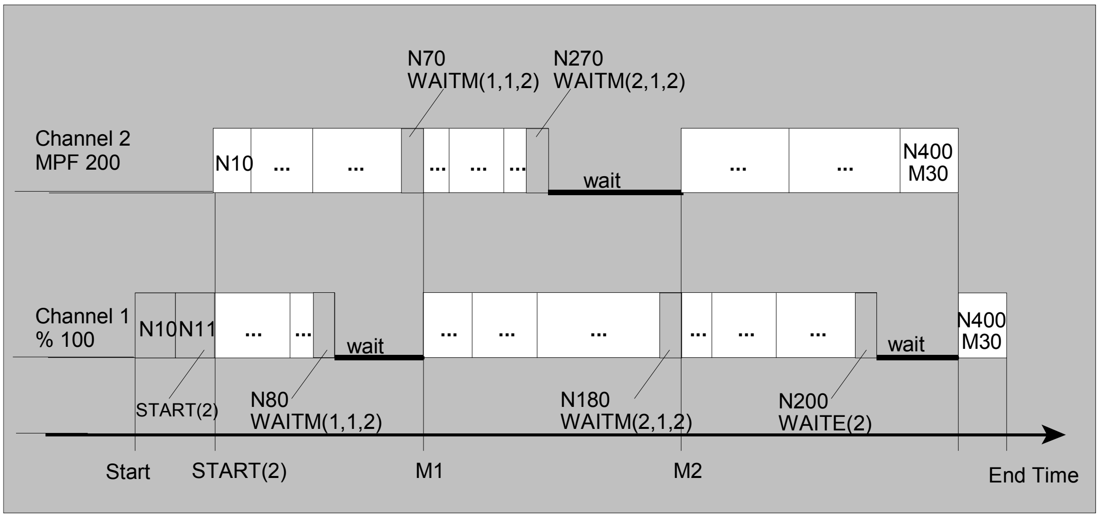
```
%_N_MPF100_MPF
```

`N10 INIT(2,"MPF200")`	
`N11 START(2)` .	Processing in channel 2
`N80 WAITM(1,1,2)` .	Waiting for WAIT marker 1 in channel 1 and in channel 2; further processing in channel 1
`N180 WAITM(2,1,2)` .	Wait for WAIT mark 2 in channel 1 and in channel 2 and execution continued in channel 2
`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`	
 `N70 WAITM(1,1,2)` .	Processing in channel 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 1 and in channel 2 and execution continued in channel 2
`N400 M30`	End of program in channel 2



Example of program from workpiece

```
N10 INIT(2,"/_N_WKS_DIR/_N_SHAFT1_WPD/_N_CUT1_MPF")
```

Example of Init command with relative path definition

```
;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.
```

Other information

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.

Up to SW 3:
WAITE must not be scanned immediately after the START command because a program end would then be detected before the program is started.
Remedy: Programming a dwell time.
Example:

```
N30 START (2)
N31 G4 F0.01
N40 WAITE(2)
```

1.14 Interrupt routine

Programming

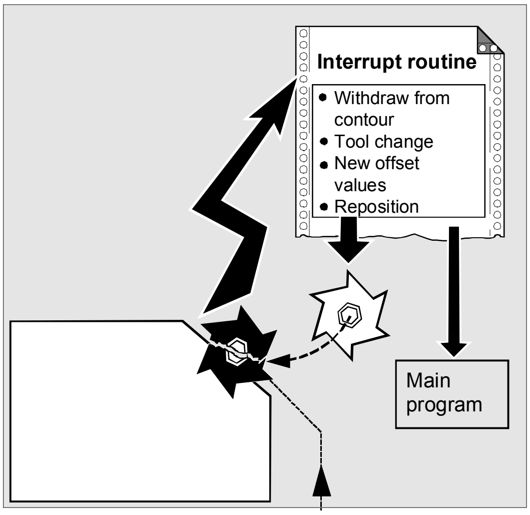
```
SETINT (3) PRIO=1 NAME
SETINT(3) PRIO=1 LIFTFAST
SETINT(3) PRIO=1 NAME  LIFTFAST
G… X… Y… ALF=…
DISABLE (3)
ENABLE (3)
CLRINT (3)
```

Explanation of the commands

`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 subprogram 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

Function

Example: The tool breaks during machining. This triggers a signal that stops the current machining process and simultaneously starts a subprogram – this subprogram is called an interrupt routine. The interrupt routine contains all the instructions 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 Chapter 9, Path Traversing Behavior, Repositioning.

Sequence

Create interrupt routine as subprogram
The interrupt routine is identified as a subprogram 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 instructions 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 subprograms in Chapter 2.

Save interrupt position, SAVE
The interrupt routine can be identified with SAVE in the definition.

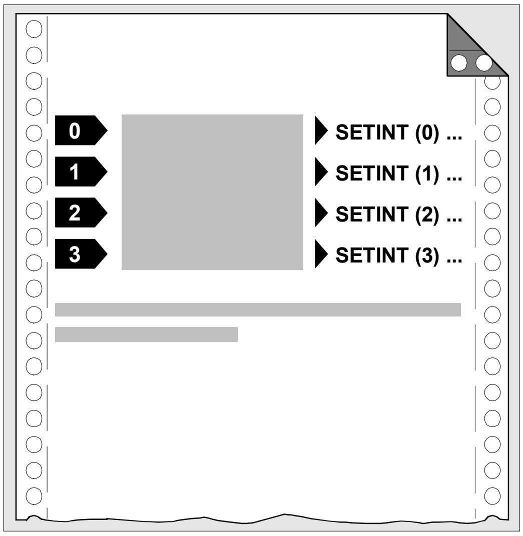
Example:

```
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 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).

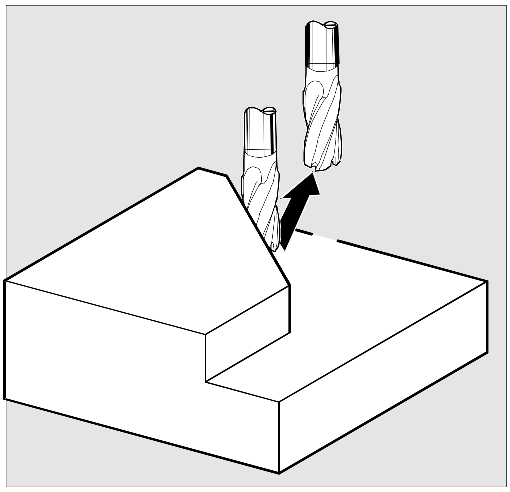
Example:

```
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 subprogram, ASUB, was specified).
- The asynchronized subprogram "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 rapid lift
The distance through which the geometry axes are retracted from the contour on liftfast can be defined in machine data.

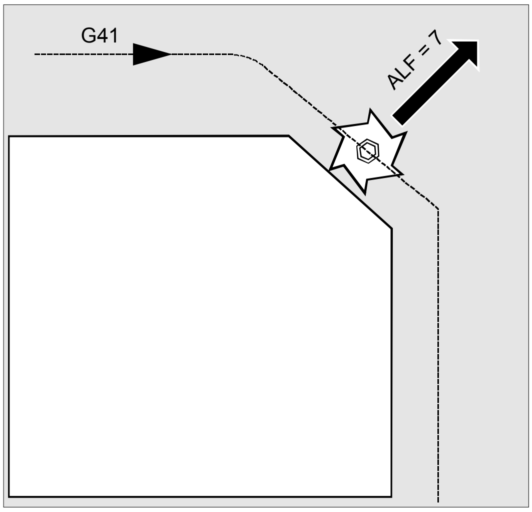
Interrupt routine without LIFTFAST
Decelerates on the path and starts the interrupt routine as soon as motion on the path stops.
This position is stored as the interrupt position and is approached with REPOS with RMI at the end of the interrupt routine.

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.

In SW 5.1 and later, the interrupt routing with LIFTFAST and ALF=0 behaves identically to the interrupt routing 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.

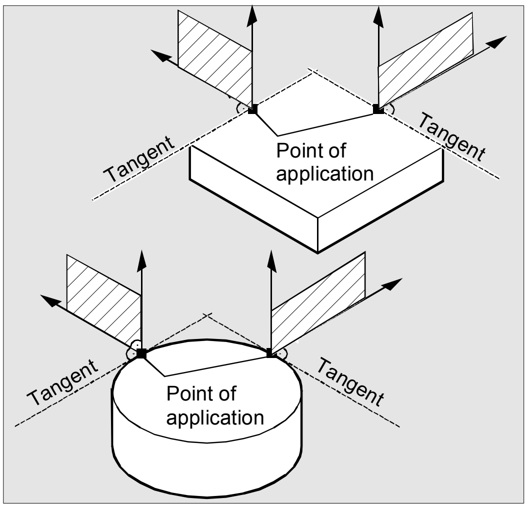
Example:

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

Reference plane for describing the Travel 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.

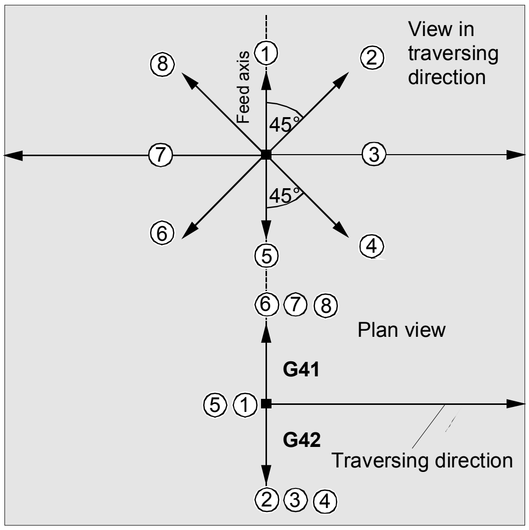
The reference plane is derived from the longitudinal tool axis (infeed direction) and a vector positioned perpendicular to this axis and perpendicular to the tangent at the point of application of the tool.



Code number with traversing directions, Overview

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.



Note:
The following codes should not be used when tool radius compensation is active:
Codes 2, 3, 4 with G41
Codes 6, 7, 8 with G42.
*can **not** be used.*

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

Retraction movement in SW 4.3 and higher
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 from 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 for the retraction movement is the active working plane which is selected by means of 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 (on SW 5 and later)**
  Retraction of the axis declared with POLFMASK to the absolute axis position programmed with POLF.
  See also NC-controlled retraction in Function Description M3.
  ALF has no affect on the lift direction for several axes (SW 6 later) and for several axes in a linear system (SW 7 and later).

The direction is programmed as before in discrete steps of 45 degrees with **ALF** in the plane of the retraction motion.
With **LFTXT**, the retraction is defined in the tool direction for ALF=1.
With **LFWP**, the direction in the working plane is assigned as follows:

• **G17**: X/Y plane	ALF=1	Retraction in X direction
	ALF=3	Retraction in Y direction
• **G18**: Z/X plane	ALF=1	Retraction in Z direction
	ALF=3	Retraction in X direction

- **G19**: Y/Z plane ALF=1 Retraction in Y direction
  ALF=3 Retraction in Z direction

Programming 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 ->`
`-> LIFTFAST`
	When input 1 is enabled, the tool is 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`	

Subprogram

`PROC W_CHANGE SAVE`	Subprogram with storage of current operating state
`N10 G0 Z100 M5`	Tool changing position, spindle stop
`N20 T11 M6 D1 G41`	Change tools
`N30 REPOSL RMB M3`	Repositioning and return to main program

-> programmed in a single block.

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

1.15 Axis replacement, spindle replacement

Explanation of the commands

`RELEASE(axis name, axis name, …)`	Release the axis
`GET(axis name, axis name, …)`	Accept the axis
`GETD  (axis name, axis name, …)`	Direct acceptance of axis
`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, ...

Function

One or more axes or spindles can only ever be used 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: The axis is transferred from channel to channel.

For more information on the functionality of an axis or spindle replacement, see
/FB/, K5 mode groups, channels, axis transfer

Sequence

Preconditions for axis transfer

- 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.

Release axis: RELEASE

When enabling the axis please note:

1. The axis must not 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, leading value coupling, electronic gear) only the leading axis of the group can be enabled.

Accept axis: GET

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

Effects of GET:

Axis transfer 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 transfer has been completed.**

Axis transfer without synchronization:
If the axis does not have to be synchronized no preprocess stop is generated by GET.

```
Example:
N01 G0 X0
N02 RELEASE(AX5)
N03 G64 X10
N04 X20
N05 GET(AX5)                  If synchronization 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 …
```

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.

An axis accepted with GET remains assigned to this axis even after a key or program reset. When a program is started the transferred 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.

Direct axis transfer: GETD
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).

Programming example

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 both channels

`N…`
`N… M30`
	Rest of program after axis transfer

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 both channels
`N150 GET (AX2)`	Accept axis AX2

`N…`
`N… M30`
	Rest of program after axis transfer

Set up variable axis transfer response

The transfer point of axes can be set as follows in MD 10722: AXCHANGE_MASK:

- Automatic axis transfer between two channels then also takes place when the axis has been brought to a neutral state by WAITP (response as before)
- **From SW 5.3**, it will only be possible to transfer all the axes fetched to the axis container by GET or GETD after an axis container rotation.
- **From SW 6.4**, 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.

Programming example

Activating an axis replacement without a preprocessing stop

```
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
N041 G4 F2
N050 M5
N099 M30
```

Traverses the spindle (axis B) immediately after block N023 as the **PLC axis**, e.g., 180 degrees and back 1 degree and back to the neutral axis. So block N040 triggers neither a preprocessing stop nor a reorganization.

1.16 NEWCONF: Setting machine data active (SW 4.3 and higher)

Function

All machine data of the effectiveness level "NEW_CONFIG" are set active by means of the NEWCONF language command. The function corresponds to activating the soft key "Set MD active".
When the NEWCONF function is executed there is an implicit preprocessing stop, that is, the path movement is interrupted.

Explanation

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

Programming example

Milling: Machine drill position with different technologies

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

1.17 WRITE: Write file (SW 4.3 and higher)

Programming

```
WRITE(var int error, char[160] filename, char[200] string)
```

The WRITE command appends a block to the end of the specified file.

Explanation of the parameters

`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. Contains `filename` blank or control character (character with decimal ASCII code <= 32), the WRITE command is aborted with error code 1 "Path not allowed". The file name can be specified with path and file identifier. Path names must be absolute, that is, starting with "/". If the file name does not contain a domain identifier (_N_), it is added accordingly. If there is not 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.

Function

Using the WRITE command, data (e.g. measurement results for measuring cycles) can be appended to the end of the specified file.
The maximum length in KB of the log files is set via MD 11420 LEN_PROTOCOL_FILE. This length is applicable for all files created using the WRITE command.

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

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.

Programming example

```
N10 DEF INT ERROR                        ;
N20 WRITE(ERROR,"TEST1","LOG DATED       ; Write text from LOG DATED
                              7.2.97“)         7.2.97 into file TEST1
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")
```

Other Information

- 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.

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.

1.18 DELETE: Delete file (SW 4.3 and higher)

Programming

```
DELETE(var int error, char[160] filename)
```

The DELETE command deletes the specified file.

Explanation of the parameters

`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, starting 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/

Function

All files can be deleted by means of the DELETE command, irrespective of whether they were created using the WRITE command or not. Files that were created using a higher access authorization can also be deleted with DELETE.

Programming 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                                 ;
...
```

1.19 READ: Read lines in file (SW 5.2 and higher)

Programming

```
READ(var int error, string[160] file, int line, int number, var
string[255] result[])
```

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.

Explanation of the parameters

`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 does not exist (parameter "line" or "number" larger than number of lines in file) 22 Array length of result variable "result" is too small 23 Line range too large (parameter "number" chosen so large that reading went beyond end of 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 three 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 is read before the end of file. 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).

Function

One or several lines can be read from a file with the READ command. The lines read are stored in one array element of an array. The information is available as STRING.

Other Information

- 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.
- 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 error=13.
- 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 variable "result". Read line are cut off, if the line is longer than the string length of the target variable "result". An error message is not output.

Programming example

```
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 identifier
                          RESULT)
...
N30 READ (ERROR, "TESTFILE_MPF", 1, 5,        ; file name without domain and with file
                               RESULT)        identifier
...
N30 READ(ERROR,"_N_TESTFILE_MPF",1,5,         ; file name with domain and file identifier
                                RESULT)
...
N30 READ(ERROR,"/_N_CST_DIR/N_TESTFILE        ; file name with domain and file identifier and
                     _MPF", 1, 5 RESULT)      path specification
^...
N40 IF ERROR <>0                              ; error evaluation
N50   MSG("ERROR"<<ERROR<<" WITH READ COMMAND")
N60   M0
N70 ENDIF
...
```

1.20 ISFILE: File available in user memory NCK (SW 5.2 and higher)

Programming

```
result=isfile(string[160]file)
```

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.

Explanation of the parameters

`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 three 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)

Programming 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
...
```

1.21 CHECKSUM: Creation of a checksum over an array (SW 5.2 and higher)

Programming

```
error=CHECKSUM(var string[16] chksum,string[32]array, int first, int
last)
```

The CHECKSUM function forms the checksum over an array.

Explanation of the parameters

`error`	Error variable for return representation 0 No error 1 Symbol not found 2 No array 3 Index 1 too large 4 Index 2 too large 5 Invalid data type 10 Check sum overflow
`chksum`	Checksum over the array as a string (call-by-reference parameter of type String, with a defined length of 16). The checksum is indicated as a character string of 16 hexadecimal numbers. However, no format characters are indicated. Example: "A6FC3404E534047C"
`array`	Number of the array over which the checksum is to be formed. (call-by-value parameter of type String with a max. length of 32). Permissible arrays: 1 or 2-dimensional arrays of 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)

Function

With CHECKSUM you form a checksum over an array.
Stock removal application:
Check to see whether the initial contour has changed.

Other Information

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.

Programming 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"
```

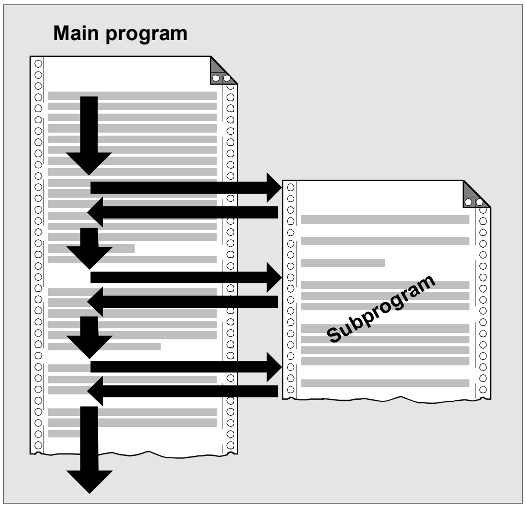
Subprograms, Macros

2.1 Using subprograms

What is a subprogram?

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

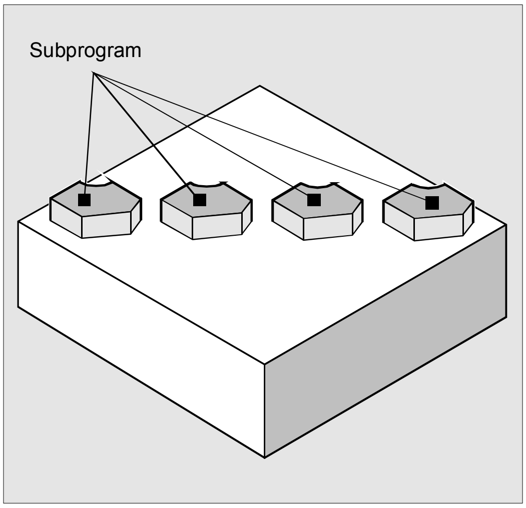
Basically, there is no difference between a main program and a subprogram. The subprogram contains either machining operations or sequences of operations that are to be performed several times.



Use of subprograms

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

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

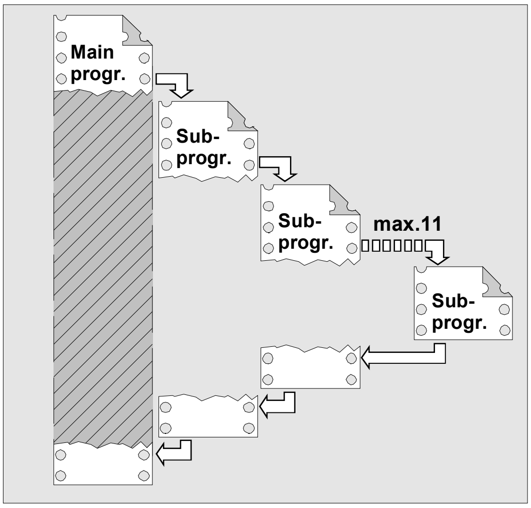


Structure of a subprogram

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

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

Nesting depth



Nesting of subprograms

A subprogram can itself contain subprogram calls. The subprograms called can contain further subprogram calls etc.
The maximum number of subprogram levels or the nesting depth is 12.

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

Restrictions

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

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

2.2 Subprogram with SAVE mechanism

Function

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

The SAVE attribute sets modal G functions to the same value at the end of subroutines that they had at the beginning. If G function group 8 (settable zero offset), G function group 52 (frame rotation of a turnable workpiece), or G function group 53 (frame rotation in tool direction) is changed while doing so, the corresponding frames are restored.

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

From SW 6.1 you can change the response of the settable zero offset and the basic frame via machine data MD 10617: FRAME_SAVE_MASK.
For further information, see
/FB/ K1, General Machine Data

Example:
Subprogram 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 subprogram 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 Subprograms with parameter transfer

Program start, PROC
A subprogram that is to take over parameters from the calling program when the program runs is designated with the vocabulary word PROC.

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

Interruption of continuous-path mode
To prevent continuous-path mode from being interrupted:
Make sure the subprogram does **not** have the SAVE attribute. For more information about the SAVE mechanism, see Section 2.2.

RET must be programmed in a separate NC block.

Example:

```
PROC CONTOUR
N10
…
N100 M17
```

Parameter transfer between main program and subprogram

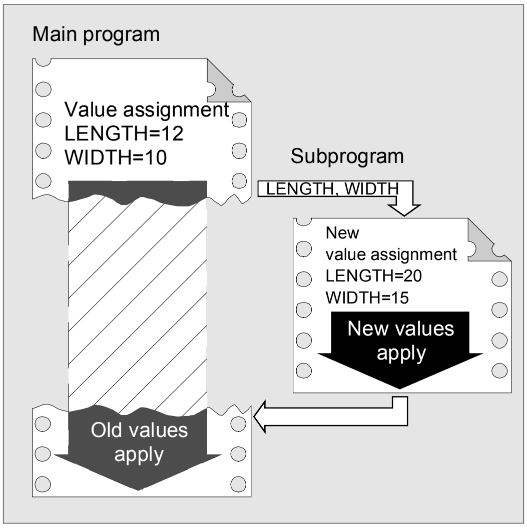
If you are working with parameters in the main program, you can use the values calculated or assigned in the subprogram as well.

For this purpose the values of the **current parameters** of the main program are passed to the **formal parameters** of the subprogram when the subprogram is called and then processed in subprogram execution.

Example:

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

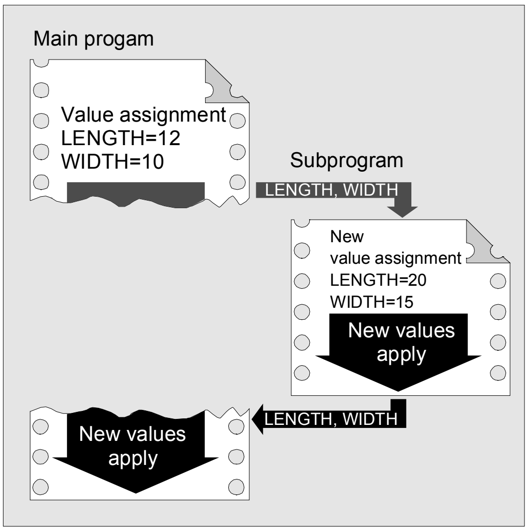
The values assigned in N20 in the main program are passed in N30 when the subprogram is called. Parameters are passed in the sequence stated. The parameter names do not have to be identical in the main programs and subprogram.



Two ways of parameter transfer

Values are only passed (call-by-value)
If the parameters passed are changed as the subprogram runs this does not have any effect on the main program. The parameters remain unchanged in it (see Fig.).

Parameter transfer with data exchange (call-by-reference)
Any change to the parameters in the subprogram also causes the parameter to change in the main program (see Fig.).



Programming

The parameters relevant for parameter transfer must be listed at the beginning of the subprogram 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 vocabulary word 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 vocabulary word 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])
```

Other Information

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

Array definition

The following applies to the definition of the formal parameters:

With two-dimensional arrays the number of fields 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 subprograms with arrays of variable length. However, when defining the variables you must define how many elements it is to contain.

See the Programming Guide "Advanced" for an explanation of array definition.

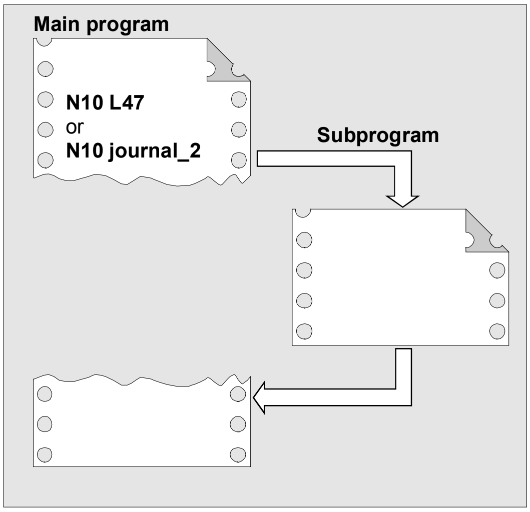
Programming example

Programming with variable array dimensions	
%_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)	Subprogram call
M30	

Creating a drilling pattern using the position table of variable dimension passed	
%_N_DRILLING_PATTERN_SPF	Subprogram
PROC DRILLING_PATTERN(VAR REAL ARRAY[,2],-> -> INT NUMBER)	Parameter transfer
DEF INT COUNTER	
STEP: G1 X=ARRAY[COUNTER,0]-> -> Y=ARRAY[COUNTER,1] F100	Machining sequence
Z=IC(-5)	
Z=IC(5)	
COUNT=COUNT+1	
IF COUNT<NUMBER GOTOB STEP	
RET	Subroutine end

2.4 Calling subprograms: L or EXTERN

Subprogram call without parameter transfer
You call the subprogram in the main program either with address L and the subprogram number or by specifying the subprogram name.

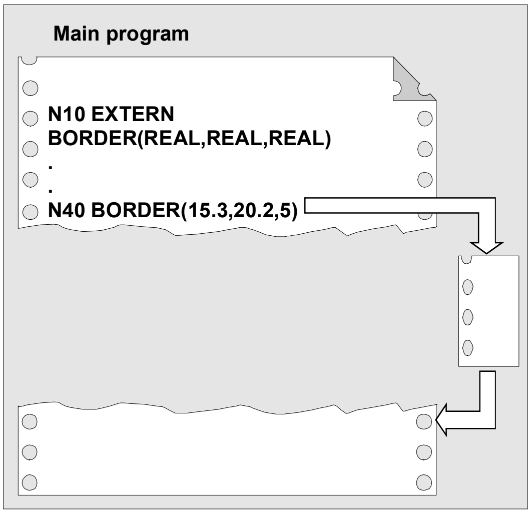


Example:

`N10 L47` or

`N10 SPIGOT_2`

Subprogram with parameter transfer, declaration with EXTERN
Subprograms 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 subprogram and the variable types are declared in the sequence in which they are transferred.



You only have to specify EXTERN if the subprogram is in the workpiece or in the global subprogram directory.
You do not have to declare cycles as EXTERN.

```
EXTERN statement
EXTERN NAME(TYPE1, TYPE2, TYPE3, …) or
EXTERN NAME(VAR TYPE1, VAR TYPE2, …)
```

Example:

```
N10 EXTERN BORDER(REAL, REAL, REAL)
…
N40 BORDER(15.3,20.2,5)
```

N10 Declaration of the subprogram, N40 Subprogram call with parameter transfer.

Subprogram call with parameter transfer

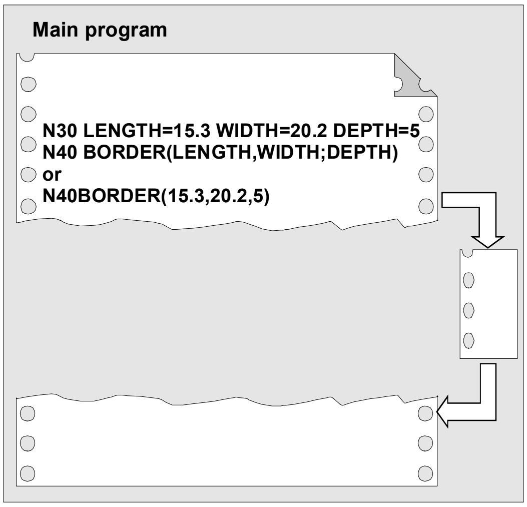
In the main program you call the subprogram by specifying the program name and parameter transfer. When transferring parameters you can transfer variables or values directly (not for VAR parameters).

Example:

```
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)
```



```
N30 LENGTH=15.3 WIDTH=20.2 DEPTH=5
N40 BORDER(LENGTH,WIDTH;DEPTH)
or
N40BORDER(15.3,20.2,5)
```

Subprogram definition corresponds to subprogram call

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

Example:

Definition in the subprogram:

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

Call in the main program:

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

Incomplete parameter transfer

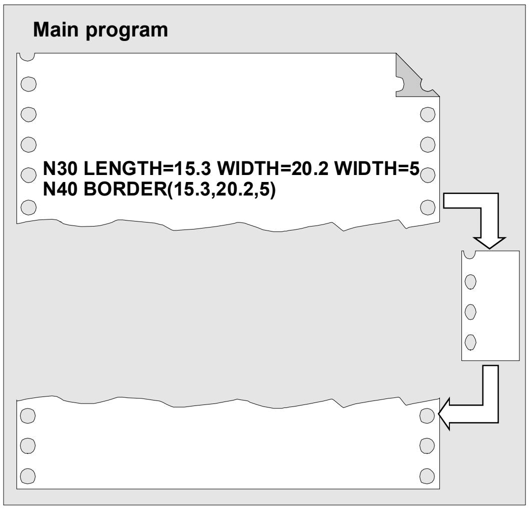
In a subprogram call only mandatory values and parameters can be omitted. In this case, the parameter in question is assigned the value **zero** in the subprogram.

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

Back to the last example:

```
N40 BORDER(15.3, ,5)
```

The mean value 20.2 was omitted here.



```
N30 LENGTH=15.3 WIDTH=20.2 WIDTH=5
N40 BORDER(15.3,20.2,5)
```

Note

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

SW 4.4 and higher:

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

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

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

Example:

Subprogram

```
PROC SUB1 (INT VAR1, DOUBLE VAR2)

IF $P_SUBPAR[1]==TRUE
   ;Parameter VAR1 was not
   ;programmed in the subprogram call
ELSE
   ;Parameter VAR1 was not
   ;programmed in the subprogram call
   ;and was preset by the system
   ;with default value 0
ENDIF
IF $P_SUBPAR[2]==TRUE
   ;Parameter VAR2 was not
   ;programmed in the subprogram call
ELSE
   ;Parameter VAR2 was not
   ;programmed in the subprogram call
   ;and was preset by the system
   ;with default value 0.0
ENDIF
;Parameter 3 is not defined
IF $P_SUBPAR[3]==TRUE -> Alarm 17020
M17
```

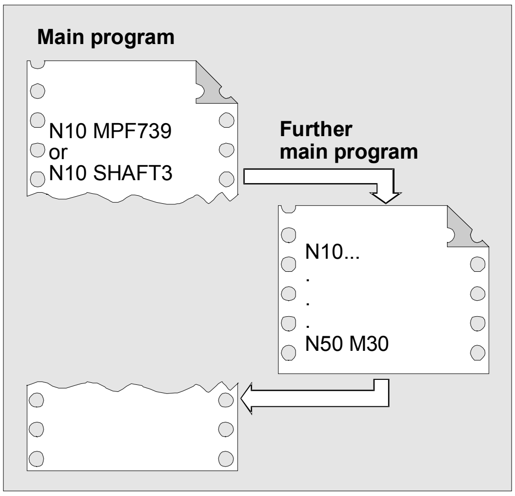
Call main program as subprogram

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

You program the call specifying the program name.

Example:

`N10 MPF739` or
`N10 SHAFT3`



A subprogram can also be started as a main program.

2.5 Parameterizable subprogram return (SW 6.4 and higher)

Programming

Parameterizable subprogram return with the relevant parameters

```
RET (<block_number/label>, <block after block with block_number/label>,
      <number of return levels>), <return to begin. of program>)
```

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

`RET (, , <no_of_return_levels>,`
`      <return to beg. of program>)`
	Subprogram return over two or more levels (jump back the specified number of levels).

Explanation

`<block_number/label>`	1. 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>,`	2. 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 subprogram return goes to the block with <block number/label>.
`<no_of_return_levels>,`	3. 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>,`	4. 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.

Function

Usually, a RET or M17 end of subprogram returns to the calling program and execution of the parts program continues with the lines following the subprogram call.

However, some applications may require program resumption at another position:

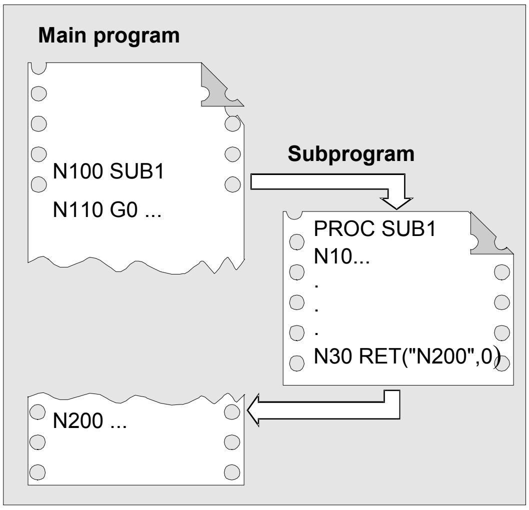
- Continuation of execution after call-up of the cutting cycles in ISO dialect mode, after the contour definition.
- Return to main program from any subprogram level (even after ASUB) for error handling.
- Return over two or more program levels for special applications in compile cycles and in ISO dialect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

The parameterizable command RET can fulfill these requirements 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>

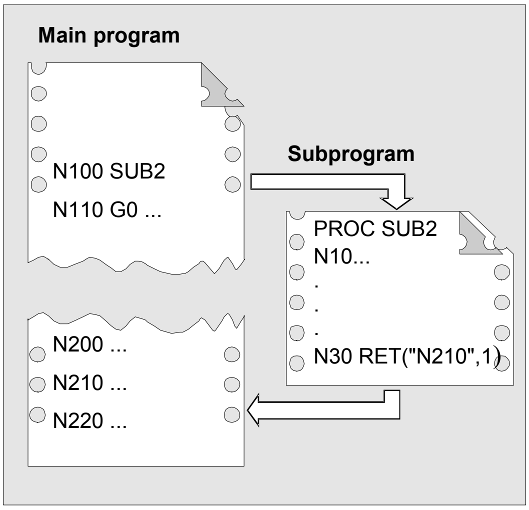
1. <block_number/label>

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



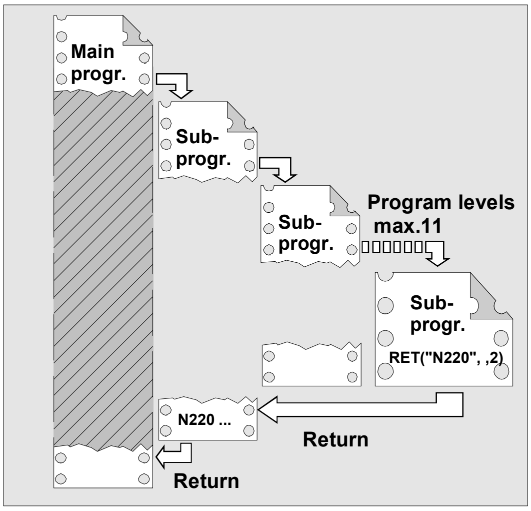
2. <block after block with block number/label>

The subprogram 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 –1 (up to 11)

is programmed, alarm 14091 is output with parameter 5.

Return with SAVE instructions

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

Modal subprogram active on return

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

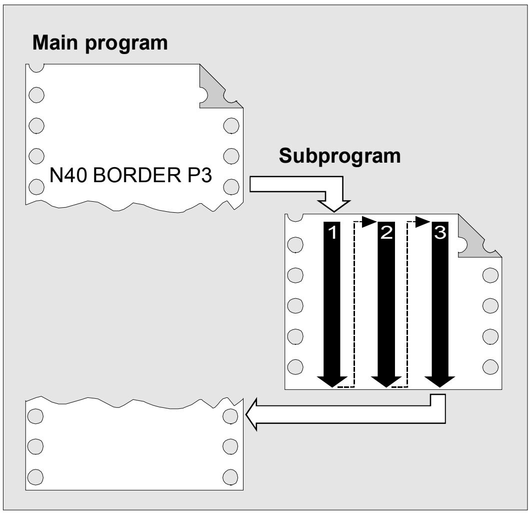
The user must ***always make sure*** *that execution continues with the correct modal settings on return over two or more program levels.*
This is done, for example, by programming an appropriate main block.

Programming example 1

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)     ; Subprogram return
N10900                                   ; Resumption in main program
N10910 MCALL                             ; Deactivate modal subprogram
N10920 G0 G60 G40 M5                     ; Correct further modal settings
```

2.6 Subprogram with program repetition: P



Program repetition, P
If a subprogram is to be executed several times in succession, the desired number of program repetitions can be entered at address P in the block with the subprogram call.

Example:

```
N40 FRAME P3
```

The subprogram FRAME must be executed 3 times in succession.

Value range
P: 1…9999

The following applies to every subprogram call:

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

Subprogram call with program repetition and parameter transfer
Parameters are passed only when the program is called, i.e. on the first run. The parameters remain unchanged for the remaining repetitions.

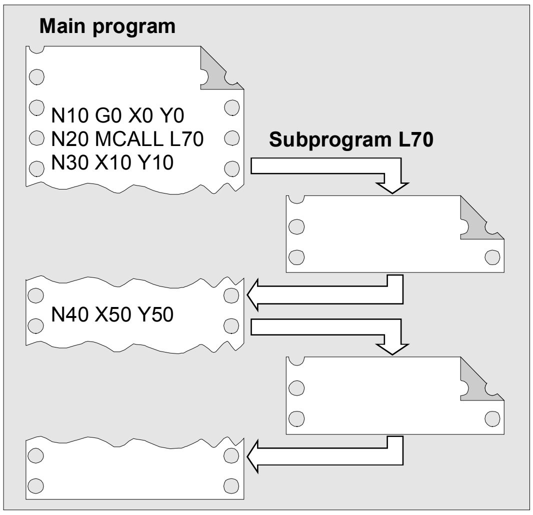
If you want to change the parameters during program repetitions, you must make the appropriate provision in the subprogram.

2.7 Modal subprogram: MCALL

Modal subprogram call, MCALL
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 subprograms that are to be executed at different positions on the workpiece. For example, for drilling patterns.

Examples:

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



In blocks N30 to N40, the program position is approached and subprogram 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 subprogram L80. L70 is called by L80.

⚠ *In a program run, only one MCALL call can apply at any one time. Parameters are only passed once with an MCALL.*

In the following situations the modal subprogram 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.

Deactivating the modal subprogram call

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

2.8 Calling the subprogram indirectly: CALL

Programming

```
CALL <program name>
```

Explanation

`CALL`	Vocabulary word for indirect subprogram call
`<program_name>`	Variable or constant of type string Name of the program containing the program section to run

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

The indirect subprogram call is only possible for subprograms without parameter transfer.

For direct calling of the subprogram, store the name in a string constant.

Example:
Direct call with string constant:

```
CALL "/_N_WCS_DIR/_N_SUBPROG_WPD/_N_PART1_SPF"
```

Indirect call via variable:

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

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

2.9 Repeating program sections with indirect programming (SW 6.4 and higher)

Programming

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

Explanation

`CALL`	Vocabulary word for indirect subprogram 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 ...` Vocabulary word 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

Function

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

Programming 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 Calling up a program in ISO language indirectly: ISOCALL

Programming

```
ISOCALL <program_name>
```

Explanation

`ISOCALL` Subprogram 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.

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 subprogram is called in Siemens mode.

For more information about ISO mode, see /FBFA/, "Description of Functions ISO Dialects"

Example:

Calling up a contour from ISO mode with cycle programming:

```
%_N_0122_SPF                              Contour description in ISO
N1010 G1 X10 Z20                          mode
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
N2300 ...                                 mode
N2400 M30
```

2.11 Calling subprogram with path specification and parameters PCALL

Programming

Subprogram call with the absolute path and parameter transfer

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

Explanation

`PCALL`

Vocabulary word for subprogram call with absolute path name

`<path_name>`

Absolute path name beginning"/", including subprogram names
If no absolute path name is specified, PCALL behaves like a standard subprogram call with a program identifier.
The program identifier is written without the leading _N_ and without an extension

If 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 subprogram.

Function

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

Example:

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

2.12 Extending a search path for subprogram calls with CALLPATH (SW 6.4 and higher)

Programming

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

```
CALLPATH <path_name>
```

Explanation

`CALLPATH` Vocabulary word for programmable search path extension. The CALLPATH command is programmed in a separate parts program line.

`<path_name>` Constant or variable of type string 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_WCS_DIR/_N_WST_WPD) If <path_name> contains the empty string or if CALLPATH is called without parameters, the search path instruction will be reset. The maximum path length is 128 bytes.

2.12 Extending a search path for subprogram calls with CALLPATH

Function

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

Example:

```
CALLPATH ("/_N_WCS_DIR/_N_MYWPD_WPD")
```

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

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

Deselection of the search path extension

The search path extension is deselected by the following results:

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

Other Information

- CALLPATH check whether the programmed path name really exists. An error aborts program execution with correction block alarm 14009.
- CALLPATH call 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 Suppress current block display: DISPLOF

Programming

```
PROC … DISPLOF
```

Function

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

By default the block display is activated.
Deactivation of block display with DISPLOF applies until the return from the subprogram or end of program. If further subprograms are called from the subprogram with the DISPLOF attribute, the current block display is suppressed in these as well. If a subprogram with suppressed block display is interrupted by an unsynchronized subprogram, the blocks of the current subprogram are displayed.

Programming 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                                        ;Subprogram return, the following block
                                           of the calling program is displayed again
```

2.14 Single block suppression: SBLOF, SBLON (SW 4.3 and higher)

Programming

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

Explanation

`SBLOF`	Deactivate single block
`SBLON`	Reactivate single block

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 subprogram or until it is aborted.

At the return command, the decision is made whether to stop at the end of the subprogram.
Return jump with M17:
Stop at the end of the subprogram
Return jump with RET:
No stop at the end of the subprogram

SBLOF is also valid in subroutines which are called.

Example for subroutine without stop in single block

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

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

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

Example:

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

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

Single block disable for unsynchronized subprograms

To run an ASUB in single block mode in one step, the ASUB must contain a PROC instruction 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 instruction 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 instruction, execution stops after the next machine function block of the calling program.

If that is not wanted, SBLON must be programmed in the subprogram before the return (M17). Execution does not stop on a return to a higher-level program with RET.

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 is suppressed in the system ASUP or user ASUP with Bit0 = 1 or Bit1 = 1 of MD 10702: IGNORE_SINGLEBLOCK_MASK, it is possible to re-activate single block stop by programming SBLON in the ASUP.
- The single block stop in the user Asup is suppressed with MD 20117: IGNORE_SINGLEBLOCK_ASUP and can no longer be activated by programming SBLON.
- By selecting SBL3 you can suppress the SBLOF command.
- **SW 6.4 and higher**
  Ignore single block stop in 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.

For more information about block display with/ without single block suppression, see
/FB/, K1 BAG, Channel, Program control "single block"

Programming example 1

A 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.

Programming example 2

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
```

Programming example 3

With MD 10702: IGNORE_SINGLEBLOCK_MASK, Bit 12 = 1 is not stopped as from SW 6.4
In single block type SBL2 (stop at each parts program line) in the SBLON instruction

```
;SBL2 is active
;$MN_IGNORE_SINGLEBLOCK_MASK = `H1000`     ; Set MD 10702: 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                     ; No stopping because MD 10702:
                                           ; Bit 12 = 1
            N120 X1                        ; Stop at this parts program line
            N140 SBLOF
            N150 R0 = 2
            RET
N50 G90 X20                                ; Stop at this parts program line
M30
```

Programming example 4

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 instruction active
N30 X0                                   ; Stop at this block
N40 M30                                  ; Stop at this block
```

2.15 Executing external subprogram: EXTCALL (SW 4.2 and higher)

Programming

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

Explanation

`EXTCALL\`	Keyword for subprogram 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_WCS_DIR/_N_SHAFT_WPD/_N_SHAFT_SPF") or
EXTCALL ("SHAFT")
```

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.

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 subprogram identifier.

Call of an external subprogram
An external subroutine is called by means of parts program command **EXTCALL**.
The

- subprogram names programmed with EXTCALL and
- setting data SD 42700: EXT_PROG_PATH

result in the program path by means of a string of characters comprising

- the content of SD 42700: EXT_PROG_PATH (e.g. /_N_WCS_DIR/_N_WKST1_WPD)
- the character "/" as a separator (if a path has been specified via SD 42700: EXT_PROG_PATH)
- the subprogram path or subprogram identifier specified with EXTCALL.

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

1. current directory / subroutine identifier
2. current directory / subprogram identifier_SPF
3. current directory / subprogram 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": stands for the directory in which the main program was selected.

"subroutine identifier": stands for the subprogram name programmed with EXTCALL.

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 subprogram). The default setting for the size of the load memory is 30 Kbytes. With MD 18360: MM_EXT_PROG_BUFFER_SIZE defines the size of the reload buffer. The number of the reload buffer is parameterized with MD 18362: MM_EXT_PROG_BUFFER_NUM. One reload buffer must be set for each program (main program or subprogram) to run concurrently in "Processing from external source" mode.

Programming examples

1. Program to be reloaded is stored on the **local hard disk of HMI Advanced**:

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

```
N10 PROC MAIN
N20 ...
N30 EXTCALL "ROUGHING"                       ; Call of external subprogram
                                             ; ROUGHING
N40 ...
N50 M30
```

Subprogram "ROUGHING" (located in the HMI Advanced directory structure under workpieces->WST1):

```
N10 PROC ROUGHING
N20 G1 F1000
N30 X=... Y=... Z=...
N40 ...
N90 M17
```

2. The program to be reloaded is located on the **network drive or ATA card of HMI**

EXTCALL Windows path name

Call for **network drive** (HMI Embedded or Advanced)
EXTCALL \\R4711\Workpieces\Contour1.spf

Call for **ATA card** (HMI Embedded) e.g.
EXTCALL C:\Workpieces\Contour2.spf

An absolute path must always be specified in HMI Embedded.

For more information about operation, see
/BEM/ HMI Embedded
/BAD/ HMI Advanced

Other information

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

Subprogram calls – even nested EXTCALL calls – are possible.

SW 6.3 and higher
IF-ELSE-ENDIF constructions are possible.

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

For more information about "Processing from external source", see:
/FB/ K1, BAG, Channel, Program control

2.16 Subprogram call with M/T function

Function

The T/M function can be replaced with a subprogram call by making the appropriate setting in the machine data.
This can be used, for example, to call the tool change routine.
At block search subprogram calls with M/T functions behave like standard subprogram calls.

For more information about "Subprogram call with M/T functions", see:
/FB/ K1, BAG, Channel, Program control

Example 1: `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 subprogram 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
```

Example 2: `Tool change with T function programming`
T function `is replaced by tool change routine` TC_UP_T

```
N10 PROC ROUGHING4
N20 G1 F1000
N30 X=... Y=... Z=...
N40 T1234 ;                                   ; Call TC_UP_T
M30
```

Associated subprogram TC_UP_T:

```
N310 PROC TC_UP_T
```

```
...
N330 IF $C_T_PROG == 1
N340 G53 D0 G0 X=... Y=... Z=...    ;        Approach tool change position
N350 T=$C_T                 ;                Execute tool change
N360 ENDIF
...
N390 M17
```

Extension of T function substitution

As from SW 6.4, T function substitution is extended to permit setting in machine data whether with programming of both:

D numbers or DL numbers and T numbers

- in one block D or DL will be passed as a parameter to the T substitution cycle in accordance with the default setting **or**
- whether it must be executed before the T substitution cycle is called.

In the case of function substitution parameterization of MD 10719: T_NO_FCT_CYCLE_MODE, T with

Val. 0: as before, the D or DL number is passed directly to the cycle (default value).
Val. 1: the D or DL number is calculated directly in the block.

This function is active only if the tool change has been configured with M function (MD 22550: TOOL_CHANGE_MODE = 1); the D or DL values are otherwise always transferred.

2.17 Cycles: Setting parameters for user cycles

Files and paths

Explanation

cov.com	Overview of cycles
uc.com	Cycle call description

Function

Customized cycles can be parameterized with these files.

Sequence

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.

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

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 of 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 subprogram name
;
...
M17
```

```
PROC SPECIALCYCLE (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///Subprogram name)

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

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> /
<comment>)
```

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.

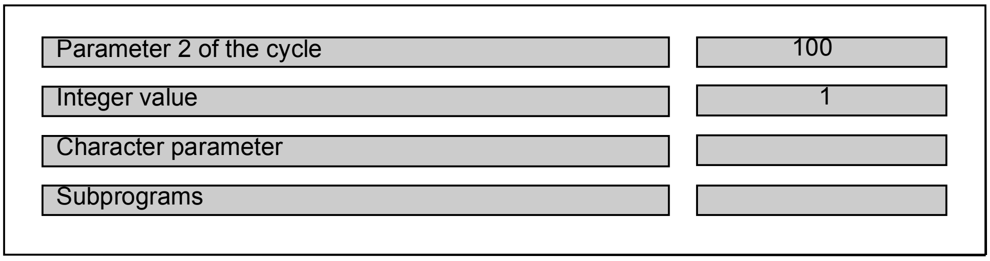
Example:
`(I/*123456/1/Machining type)`
There are no limits for string and character types;

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.

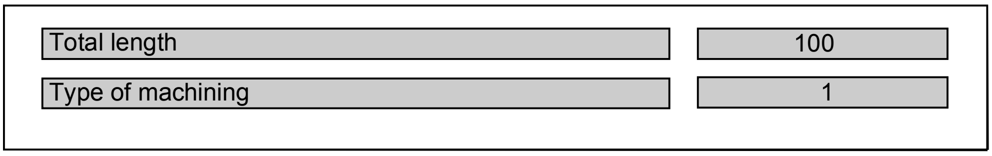
Display example for both cycles

Display screen for cycle `MY_CYCLE_1`



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

Display screen for cycle `SPECIAL CYCLE`



Total length	100
Type of machining	1

2.18 Macros. DEFINE...AS

What is a macro?
A macro is a sequence of individual instructions which have together been assigned a name of their own. G, M and H functions or L subprogram names can also be used as macros.
When a macro is called during a program run, the instructions programmed under the program name are executed one after the other.

Use of macros
Sequences of instructions 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 subprogram and executed.

Programming

Macros are identified with the vocabulary word

```
DEFINE...AS.
```

The macro definition is as follows:

```
DEFINE  NAME AS  <instruction>
```

Example:

Macro definition:

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

Call in the NC program:

```
N20  LINE X10 Y20
```

Activate macro

- **SW 4** and lower
  Macros are active after control POWER ON.
- **SW 5** and higher
  The macro is active when it is loaded into the NC ("Load" soft key).

Three-digit M/G function (SW 5 and higher)

- **SW 4** and higher
  After a three-digit M function is programmed, alarm 12530 is issued.
- **SW 5** and higher
  Supports programming of three-digit M and G functions.
  Example:
  N20 DEFINE M100 AS M6
  N80 DEFINE M999 AS M6

Other Information

Nesting of macros is not possible.
Two-digit H and L functions can be programmed.

Programming example

Example of 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 subprogram (e.g. M106).
`DEFINE G81 AS DRILL(81)`	Emulation of the DIN G function
`DEFINE G33 AS M333 G333`	During thread cutting synchronization is requested with the PLC. The original G function G33 was renamed to G333 by machine data so that the programming is identical for the user.

Example of a global 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                          ;
```

- *Vocabulary words 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.
- *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.
- *With macros you can define any identifiers, G, M, H functions and L program names.*
- *Macro identifiers with 1 letter and 1 digit are permissible (**FM-NC only**).*

■

Notes

File and Program Management

3.1 Overview

Memory structure

The memory structure available to the user is organized in two areas.

1. User memory
The user 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.

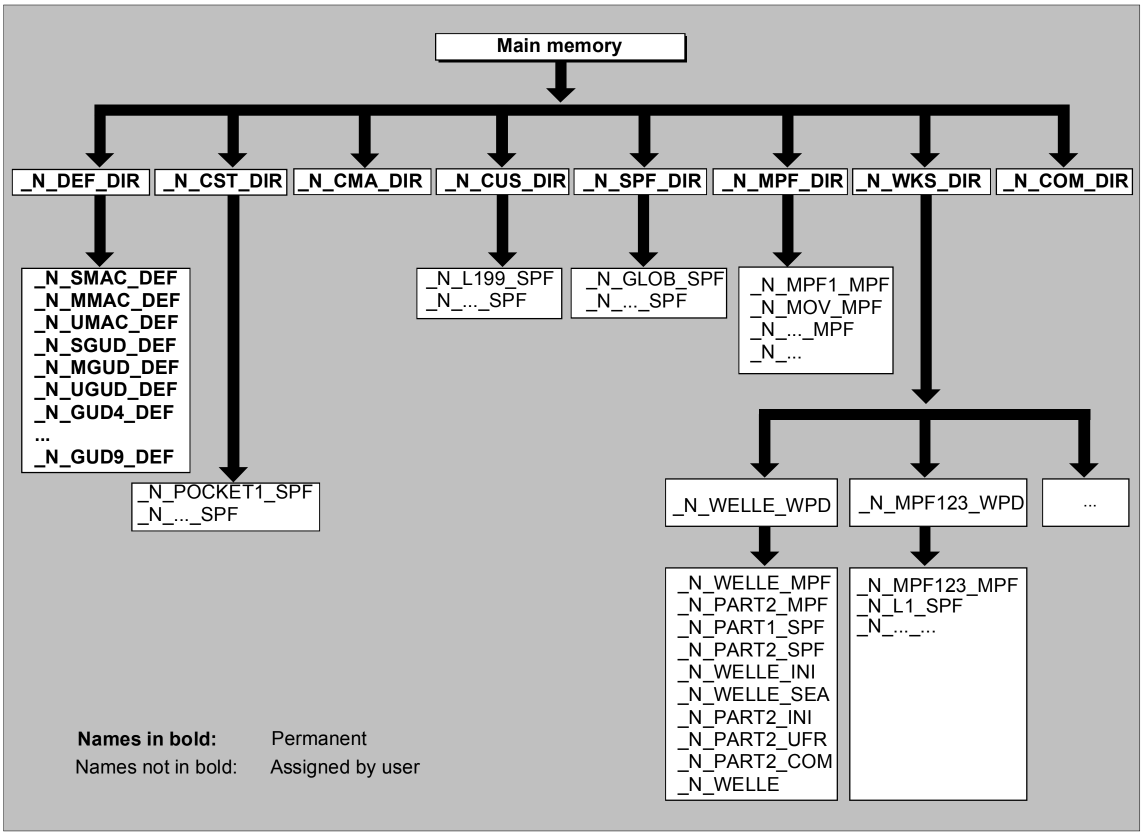
2. Program memory

The files and programs are stored in the program memory and are thus permanently stored (passive file system).
Example:
Main programs and subprograms, macro definitions.

3.2 Program memory

Overview

Main programs and subprograms 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

File types

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

*name*_MPF	Main program
*name*_SPF	Subprogram

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

Workpiece 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, Identifier WPD

To make data and program handling more flexible certain data and programs can be grouped together or stored in individual workpiece directories.
A workpiece directory contains all files required for machining a workpiece.

These can be main programs, subprograms, any initialization programs and comment files.

Initialization programs are executed once on initial part program start after program selection in accordance with machine data MD 11280: WPD_INI_MODE.

Example:
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`	Subprogram
`_N_PART2_SPF`	Subprogram
`_N_SHAFT_INI`	General initialization program for the data of the workpiece
`_N_SHAFT_SEA`	Setting data initialization program
`_N_PART2_INI`	General initialization program for the data for the Part 2 program
`_N_PART2_UFR`	Initialization program for the frame data for the Part 2 program
`_N_SHAFT_COM`	Comment file

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 instruction

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.

Important

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 path for the previous example

```
SHAFT: _/N_SHAFT_MPF is stored in
      /_N_WKS_DIR/_N_SHAFT_WPD

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

SHAFT: _/N_SHAFT_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`.

SW 5 and higher (MMC 102/103 only):

See "Operator's Guide" /BA/ Section on Job list and Selecting program for execution.

Search path with subprogram call

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

Example of subprogram call with absolute path specification:

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

Programs are usually called without specifying a path:

Example:

```
CYCLE1
```

Search path sequence

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 subprograms
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 subprogram call (as from SW 6.4)

CALLPATH command

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

Example:

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

The search path is stored before 5. (user cycle) as programmed.

For further information about the programmable search path for subprogram calls with CALLPATH, see Section 2.12:

3.3 Working memory

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

Generating an initialization program on an external PC

The data area identifier and the data type identifier can be used to determine the areas which are to be treated as a unit when the data are saved.

Example:

`_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.

Saving 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`.
- With `INITIAL` an INI file: `_N_INITIAL_INI` is created over all areas.

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.

Procedure for multi-channel controls

CHANDATA (channel number) for several channels is only permitted in the file `N_INITIAL_INI`.
`N_INITIAL_INI` is the installation file with which all data of the control is initialized.

Example:

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

⚠ *In the parts program, the CHANDATA instruction may only be used for the channel on which the NC program is running, i.e. the instruction 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 instructions.

3.4 Defining user data

Function

User data are defined (GUD) 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`.

- **SW 5** and higher (01.99):
  The user data definition (GUD) can be prepared in the Services operating area of the HMI operator interface. This eliminates 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.

Reserved module 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.

Programming

The GUD variables are programmed with the DEF command:

```
DEF range preprocessing_stop type name[.., ...]=value
```

Explanation

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 preset value, two or more values for arrays, separated by commas REP (w1) , SET(w1, w2, ...), (w1, w2, ...) Initialization values are not possible for type Frame	

Sequence

Defining user data (GUD)

1. Save module _N_INITIAL_INI.
2. Creating a definition file for user data
   - on an external PC (**SW 4** and lower)
   - in the Services operating area (**SW 5** and higher)
3. Load definition file into the program memory of the control.
4. Activate definition files.
5. Back up data.

2. **Creating a definition file for user data**
   Definition files can be prepared on the external PC or in the Services operating area. Predefined filenames exist, too
   (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.

3. **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 RS-232 interface.

Programming example

Example of a 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
```

4. Activating definition files

- **SW 4** and lower
  Before read-in of the `_N_INITIAL_INI`, *save all programs, frames, and machine data because the static memory will be formatted*
  The definition file is only reactivated on read-in of the `_N_INITIAL_INI` file.
- **SW 5** and higher
  When the GUD definition file is loaded into the NC ("Load" soft key), it becomes active. See "Automatic activation ..."

5. Data backup

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

The variable assignments to global user data are also stored in `_N_INITIAL_INI`, the names must be identical with the names in the definition files.

Example of a definition file for global data (machine manufacturer):

```
%_N_MGUD_DEF
;$PATH=/_N_DEF_DIR
;Global data definitions of the machine manufacturer
DEF NCK SYNRW INT QUANTITY          ;Implicit preprocessing stop during
                                    read/write
                                    ;Spec. data available in the control
                                    ;Access from all channels
DEF CHAN INT TOOLTABLE[100]         ;Tool table for channel-spec. image
                                    ;of the tool number at magazine locations
M30                                 ;Separate table created for each channel
```

Other information

Configurable parameter ranges in SW 7.1 and higher
Via machine data it is possible to expand the individual GUD modules by channel-specific parameter ranges that are not also listed in synchronous actions. This GUD parameters then behave like R parameters.
Newly created GUD ranges can be defined as data type REAL, INT, and BOOL; see list of predefined variable names.

List of predefined variable names			
Name of the Synact parameters			
of data type **REAL**	of data type **INT**	of data type **BOOL**	in **module**
SYP_RS[ ]	SYP_IS[ ]	SYP_BS[ ]	SGUD module
SYP_RM[ ]	SYP_IM[ ]	SYP_BM[ ]	MGUD module
SYP_RU[ ]	SYP_IU[ ]	SYP_BU[ ]	UGUD module
SYP_R4[ ]	SYP_I4[ ]	SYP_B4[ ]	GUD4 module
SYP_R5[ ]	SYP_I5[ ]	SYP_B5[ ]	GUD5 module
SYP_R6[ ]	SYP_I6[ ]	SYP_B6[ ]	GUD6 module
SYP_R7[ ]	SYP_I7[ ]	SYP_B7[ ]	GUD7 module
SYP_R8[ ]	SYP_I8[ ]	SYP_B8[ ]	GUD8 module
SYP_R9[ ]	SYP_I9[ ]	SYP_B9[ ]	GUD9 module

For more information, please refer to:
/IAD/, Chapter 6 "Parameterization of the control"

3.5 Defining protection levels for user data (GUD)

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
```

Explanation

`Protection level:`	Access prot.	(**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	Key switch 3
...			...
7	or	17	Keyswitch 0

APW 0-7, APR 0-7
The module variables cannot be written/read via the NC program or in MDA mode.

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 module variables can be written/read via the NC program or in MDA mode.

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.

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 operator interface or from the program.
The access protection applies to **all** variables defined in this module.
When an attempt is made to access protected data, the control outputs an appropriate alarm.

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

a) The file _N_GUD7_DEF has value 3 as write protection. The value 3 is then overwritten with value 2.
b) The file _N_GUD7_DEF has value 0 as write protection. There is no change to it.

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

Sequence

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.

Example of a definition file with access protection write (machine manufacturer), read (keyswitch 2):

Programming example

```
%_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                                      ;
```

3.6 Automatic activation of GUDs and MACs (SW 4.4 and higher)

Function

The definition files for GUD and macro definitions are edited

- in the Services operating area for the MMC 102/103.

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

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

"OK" → You are asked whether you want to save active data.
    "Do you want to keep the previous data of the definitions?"
    "OK" → The GUD blocks of the definition file to be processed are saved while the new definitions are activated and the restored data are loaded again.
    "Do you want to save the previous data in the definitions?"
    "Cancel" → The new definitions are activated and the old data are lost.
    "Cancel" → The changes are rejected in the definition file; the associated data block is not changed.

Unload

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

Load

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.

Additional notes (MMC 103)

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

3.7 Changing the protection data for machine and setting data

Programming

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

Explanation

`REDEF`	**REDEF**inition
`Machine data / setting data`	Machine data or setting data to which a protection level is to be assigned.

`Protection level:`
`APW n`
`APR n`
	Access prot. (**A**ccess **P**rotection)
	for writing (**W**rite)
	for reading (**R**ead)

`n`	Protection level n from 0 (highest level) to 7 (lowest level)

Function

The user **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.
The passwords are required for redefinition by the user. Redefinition is performed in the GUD definition files, e.g. _N_UGUD_DEF.

Programming 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
```

Changing protection levels

Resetting machine/setting data
To undo a change to the protection levels, the original protection levels must be written back again.

Programming 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
```

3.8 Protection levels for NC language commands (SW 7.1 and higher)

Programming

```
REDEF (NC language element) APX value
REDEF (system variable) APW value
REDEF (machine data/setting data) APR value
REDEF (machine data/setting data) APW value
REDEF (machine data/setting data) APR value APW value
```

Explanation

`NC language element`

Language element to which a protection level is to be assigned for execution:

1. Predefined subroutines/ functions (list with same name).
2. Vocab. word "DO" ("DO" statement for synchronous actions).
3. G functions (list of G functions/ preparatory functions).
4. Program identifier for cycle
   The cycle must be stored in one of the cycle directories and contain a PROC instruction.

`system variables`	System variable to which a protection level is to be assigned for write access (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`
`APW, APR`
	Vocabulary word for access protection
	Execute
	Write, read

`value`	Numeric value of the protection level (0 to 7), from 0 (highest level) to 7 (lowest level)
`value 7`	Keyswitch position 0 corresponds to the default setting of all available part program commands

Function

The existing protection level concept for access to machine/setting data and GUDs has been expanded by the part program commands listed above. For this purpose, a protection level 0 to 7 is assigned to a part program command with the REDEF command. This command will now only be executed during part program execution when the corresponding execution right exists.

Effect and application of the REDEF command
The REDEF command has a global scope in all channels and mode groups and can be applied

- to predefined subroutines and functions (predefined subroutines).
- to G codes acc. to the list of G functions/ preparatory functions".
- to write access by the part program or synchronous actions on the system variable. Read access is always possible.
- to "DO" statement for synchronous actions.
- to changing the write or read access to machine and setting data as previously.
- to protection of cycle calls.

For more information, please refer to:
/BAD/, HMI Advanced Operator's Guide, Chapter 2
/IAD/, Installation and Start-up Guide, Parameterization of the control, Chapter "Protection level concept"

Sequence

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

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

Subprogram call in definition files

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

Example:

```
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
```

Note on the REDEF command in SW 7.1 and higher

As soon as this function, protection levels for NC language elements, is active, all previously set protection levels for machine/setting data with REDEF in the GUD definition files must be changed to the new definition files. Now setting protection levels for machine and setting data is only permitted in the above protection level definition files.

The option available **up to SW 6.4** of releasing access to any machine data **in all** definition files with the REDEF command irrespective of the currently valid access right is now denied with alarm 15420. Instead there are now separate definition files for protection levels 0 to 3 (Siemens, machine manufacturer, or end user).

Other information

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 part program command. On the operator interface, the protection level concept of the HMI Advanced/Embedded applies.

3.9 Changing attributes of NC language elements (SW 6.4 and higher)

Programming

```
REDEF NC language element attribute value
```

Explanation

`NC language element`	This includes: GUD R parameters Machine data/setting data Synchronous variables ($AC_PARAM, $AC_MARKER, $AC_TIMER) system vars that can be written from parts prog.: (see Appendix) User frames (G500, etc.) Magazine/tool configurations	
`Attribute`		**Permissible for:**
	`INIPO`	GUD, R parameters, synchronous vars
	`INIRE`	" " "
	`INICF`	" " "
	`PRLOC`	Setting data
	`SYNR`	GUD
	`SYNW`	"
	`SYNRW`	"
	`APW`	Machine and setting data
	`APR`	" " "

`Value`

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

Forms:

Single values	e.g. 5
Value list	e.g. (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) for Variable with 10 elements
`REP` (w1)	where w1: value list to be repeated for variables with more than one element, e.g. REP(12)

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

`n`	necessary parameter protection level 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 instruction.
Cannot be used for R parameters and system variables. Only constants can be assigned. Expressions are not permitted values.

Function

The REDEF instruction available as from SW 6.4 makes the functions described in the previous subsections for defining data objects and protection levels into a general interface for setting attributes and values.

Meanings of the attributes

INIPO

INIt for **P**ower **O**n
The data are overwritten with the default(s) on battery-back restart of the NC.

INIRE

INIt 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

INIt on **N**ew**C**onf request or TP command NEWCONF
On NewConf request or TP command NEWCONF, the data are overwritten with the default values.
INICF also applies to INIRE and INIPO.

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!

Setting a default value:

If with
REDEF <name> INIRE, INIPO; INICF; PRLOC, the behavior of a system variable or GUD is changed, machine data MD 11270: DEFAULT_VALUES_MEM_MASK must be set to 1 (memory for initialization values active). Otherwise, alarm 12261 "Initialization not allowed" is output.

PRLOC

Only **pr**ogram-**lo**cal change
If the data is changed in a parts program, subprogram, 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.

Only possible for GUD:

SYNR — Preprocess stop while reading
SYNW — Preprocess stop during write
SYNRW — Preprocess stop during read and write

APW — Access right during write
APR — 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 position 0)

Constraints on REDEF

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 instruction) 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 ever **for the entire array**:

```
DEF NCK 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 instructions with **R parameters** must be enclosed in parentheses.

```
REDEF R[ ] INIRE
```

INI attributes
Make sure that a sufficiently large **memory for init values** (settable in MD 18150: MM_GUD_VAL_MEM) is available when setting INI attributes for these variables. 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).

Programming example 1

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:

```
REDEF _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.

Programming example 2

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.

Programmable setting data

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
43210	$SA_SPIND_MIN_VELO_G25	G25
43220	$SA_SPIND_MAX_VELO_G26	G26
43230	$SA_SPIND_MAX_VELO_LIMS	LIMS
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

System variables that can be written from the parts program:

Section 15.2 of this description lists 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.

3.10 Structuring instruction SEFORM in the Step editor (SW 6.4 and higher)

Programming

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

Explanation of parameters

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

Function

The SEFORM instruction is evaluated in the Step editor to generate the step view for HMI Advanced The step view is available as from SW 6.3 on HMI Advanced and makes for better readability of the NC subprogram. The SEFORM structuring instruction supports Step editor (editor-based program support) over the three specified parameters.

Other information

- The SEFORM instructions are generated in the Step editor.
- The string passed with the <section name> parameter is stored main-run-synchronously in the OPI variable in a similar way to the MSG instruction. The information remains until overwritten by the next SEFORM instruction. Reset and end of 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

■

Notes

Protection Zones

4.1 Definition of the protection zones CPROTDEF, NPROTDEF

Programming

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

Explanation of the commands

DEF INT NOT_USED	Define local variable, data type integer (see Chapter 10)
CPROTDEF	Channel-specific protection zones (for NCU 572/573 only)
NPROTDEF	Machine-specific protection zones
EXECUTE	End definition

Explanation of the parameters

n	Number of defined protection zone

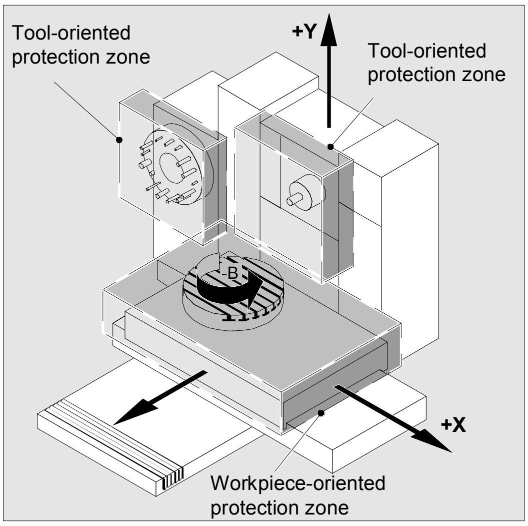
t	TRUE = **Tool**-oriented protection zone
	FALSE = **Workpiece**-oriented protection zone

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

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 which belong to the tool
(e.g. tool, tool carrier).
Workpiece-oriented protection zones:
For parts which belong to the workpiece
(e.g. parts of the workpiece, clamping table, clamp, spindle chuck, tailstock).


Tool-oriented protection zone
+Y
Tool-oriented protection zone
-B
+X
Workpiece-oriented protection zone

Sequence

Defining protection zones
Definition of the protection zones includes the following:

- CPROTDEF for channel-specific protection zones
- NPROTDEF for machine-specific protection zones
- Contour description for protection zone
- Termination of the definition with EXECUTE

You can specify a relative offset for the reference point of the protection zone when the protection zone is activated in the NC part program.

Reference point for contour description

The workpiece-oriented protection zones are defined in the basic coordinate system. The tool-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 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.

A maximum of four contour elements are available for defining one protection zone (max. of four protection zones) with the SINUMERIK FM-NC. 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).

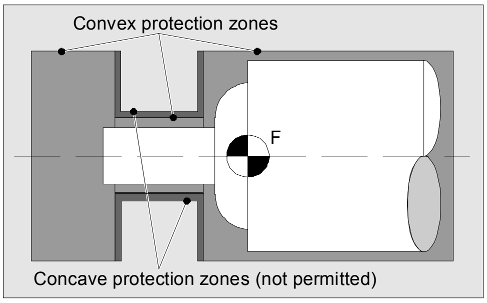
If a full circle describes the protection zone, it must be divided into two half circles. The order G2, G3 or G3, G2 is not permitted. A short G1 block must be inserted, if necessary.

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 **dynamically balanced** protection zones (e.g. spindle chucks) 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

Programming

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

Explanation of the commands and 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
xMov,yMov,zMov	Move defined protection zone on the geometry axes

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.

If no workpiece-oriented protection zone is active, protection zone monitoring does not take place.

Sequence

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 on (pre)activation

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.

Other information

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.

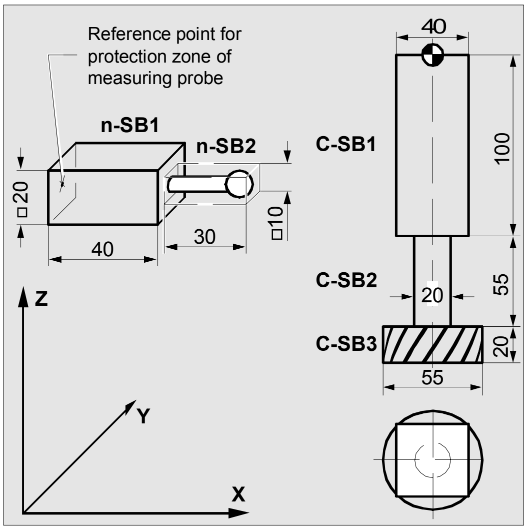
Multiple activation of protection zones

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.

Programming example



Possible collision of a milling cutter with the measuring probe is to be monitored on a milling machine. The position of the measuring probe is to be defined by an offset when the function is activated.

The following protection zones are defined for this:

- A machine-specific and a workpiece-oriented 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 an auxiliary 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
```

4.3 Checking for protection zone violation, working area limitation and software limits

Programming

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

Explanation

Status

0: Function OK; the defined path can be traversed completely.
–1: In _DLIMIT at least one component is negative.
–2: 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-specific 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 an NCK-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 more than one protection zone is violated, the hundreds and thousands digits indicate the protection zone leading to the **greatest** limitation of the defined path.

_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 basic system inch), the WCS-related 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.

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. They must be alone in a block.

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 it is not possible to traverse one of the axes completely, the paths of the other axes are reduced accordingly in return value _MAXDIST so that the resulting end point is on the defined path.

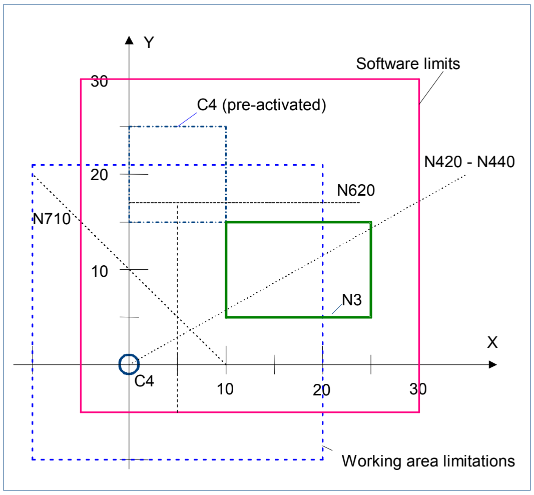
It is permissible for no software limits, working area limitation, or protection zones to be defined for one or more of the axes involved.
All limits are only monitored if the axes involved are **referenced**. Any rotary axes involved are only monitored if they are not modulo axes.

Like in normal traverse mode, monitoring of the software limits and the working area limitations depends on the active settings (interface signals for selecting software limits 1 or 2, G-Code WALIMON / WALIMOF, setting data for individual activation of the working area limits and for definition whether monitoring the working area limitations will consider the radius of the active tool).
On certain kinematic transformations (e.g. Transmit), the position of the machines axes cannot be uniquely determined from the positions in the workpiece coordinate system (WCS).
In normal traverse mode, unique determinability is usually due to the history and the condition that continuous motion of the machine axes must correspond to continuous motion of in the WCS.
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 does not ensure that the distance specified in _DLIMIT[3] is always adhered to during a movement on the defined path. It only ensures that if the end point returned in _MOVDIST is lengthened by this distance, no protection zone will be violated, even though the straight line may pass extremely close to a protection zone.

Programming example

The example in the figure shows software limits and working area limitations in the X-Y plane.
In addition, three protection zones are defined: the two channel-specific protection zones C2 and C4 and the NCK-specific protection zone N3. C2 is a circular, active, tool-related protection zone with a 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.
The following NC program first defines the protection zones and working area limitations as drawn before calling the CALCPOSI function with various parameter settings. 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-2 y0
N90 g3 i2 x2
N100 i-2 x-2
N110 execute(_SB)

N120 cprotdef(4, false, 0)          ; workpiece-related protection
N130 g17 g1 x0 y15                    zone
N140 x10
N150 y25
N160 x0
N170 y15
N180 execute(_SB)

N190 nprotdef(3, false, 0)          ; workpiece-related protection
N200 g17 g1 x10 y5                    zone
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=-10 YY=-10                    ; define working area
N300 g26 xx= 20 yy= 21                    limitations
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] =-27.
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] =-27.
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	–27.000	Pre-activated protection zone C4 not monitored. Defined path can be traversed completely.
540	2222	0.000	–25.000	Because _DLIMIT[1]=2, the traverse path is restricted by the working area limitation.
620	4223	–13.000	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 traverses path in _MOVDIST applies in the translated and rotated coordinate system (WCS).

Other information

You will find details on working area limitations in the Programming Guide PG, and on the software limits in the Function Description A3.

■

Notes

Special Motion Commands

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

Explanation of the commands

`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

Sequence

Via machine data you can enter up to 60 (0 to 59) positions for each of the two axes in the position tables.

For an example of a typical position table see diagram.

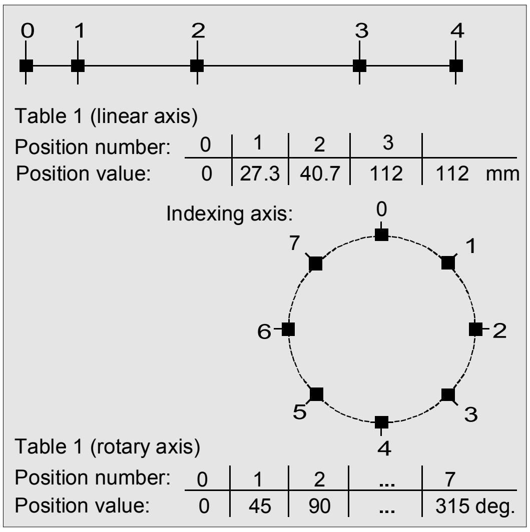


Table 1 (linear axis)

Position number:	0	1	2	3	
Position value:	0	27.3	40.7	112	112 mm

Table 1 (rotary axis)

Position number:	0	1	2	...	7
Position value:	0	45	90	...	315 deg.

Other information

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.

Programming example

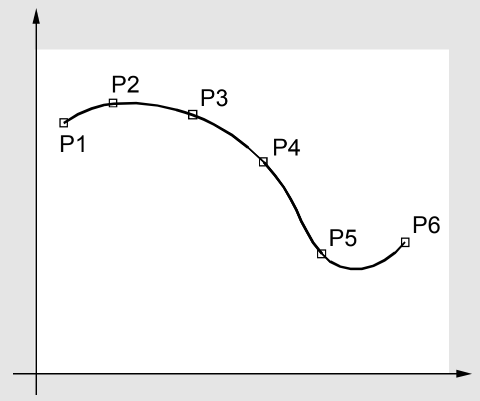
`N10 FA[B]= 300`	Feed for positioning axis B
`N20 POS[B]= CAC (10)`	Approach coded position 10 (absolutely)
`N30 POS[B]= CIC (-4)`	Travel 4 spaces back from the current position

5.2 Spline interpolation, ASPLINE, B-/CSPLINE, BAUTO, BNAT, BTAN

Introduction

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.



P1 to P6: Predefined coordinates

Programming

```
ASPLINE X Y Z A B C
```
or
```
BSPLINE X Y Z A B C
```
or
```
CSPLINE X Y Z A B C
```

Function

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)

Other information

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.

Axes that are to interpolate in the spline grouping are selected with command SPLINEPATH (further details on the following pages).

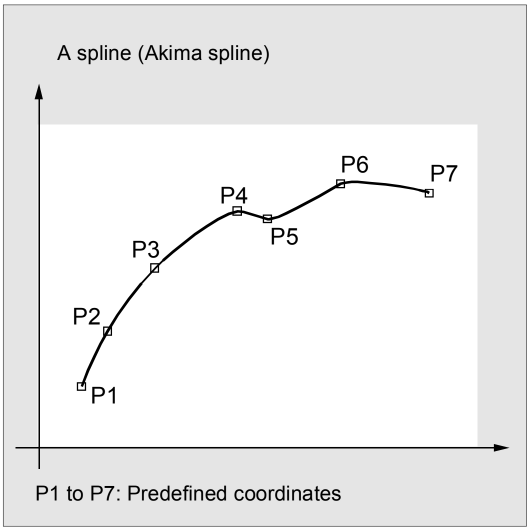
Sequence

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.

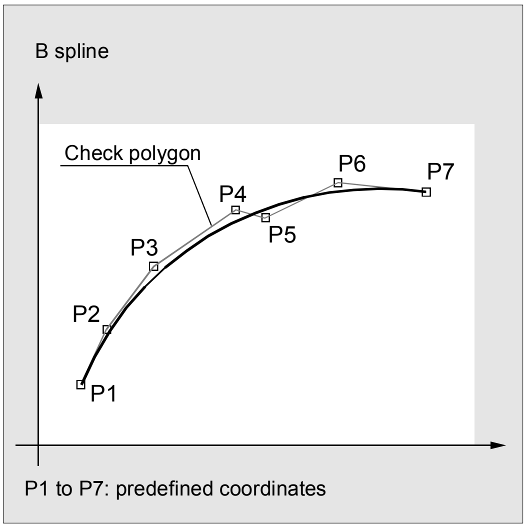
The primary application for this spline type is therefore the interpolation of digitized points. Supplementary conditions can be programmed for akima splines (see below for more information). A polynomial of third degree is used for interpolation.



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.



Programmed supplementary conditions (please see below for more information) have no effect on B splines. The B spline is always tangential to the check polygon at its start and end points.

Point weight:

A weight can be programmed for every interpolation point.

Programming:

`PW = n`

Value range:

0 <= n <= 3; in steps 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:

A third degree polygon is used as standard, but a second degree polygon is also possible.

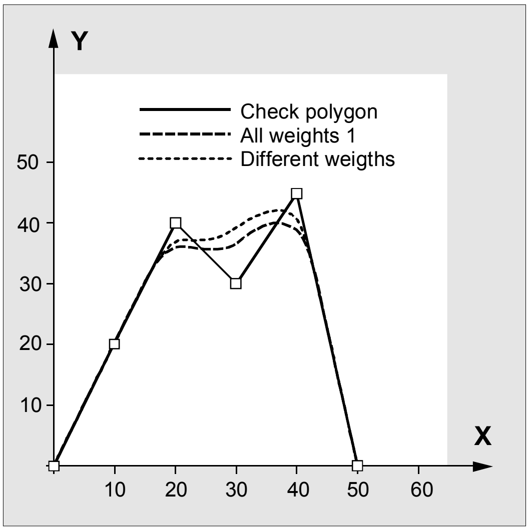
Programming:

`SD = 2`

Distance between nodes:

The distances between nodes are suitably calculated internally. However, the control can also process defined distances between nodes.

Programming:
`PL` = Value range as for path dimension



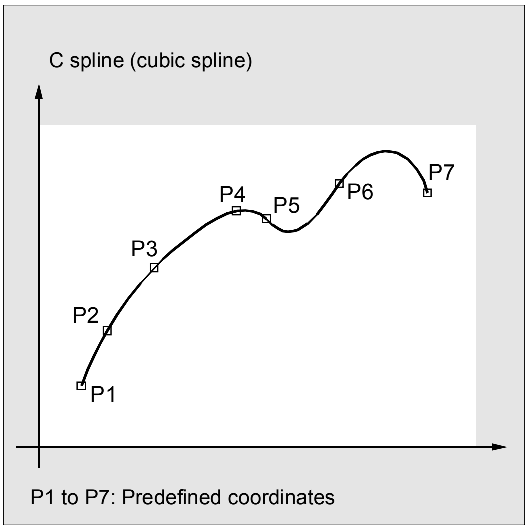
Example of B spline:

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`

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).



Restrictions

The following supplementary conditions apply only to akima and cubic splines (A and C splines).

The transitional response (start and end) of these spline curves can be set via two groups of instructions consisting of three commands each.

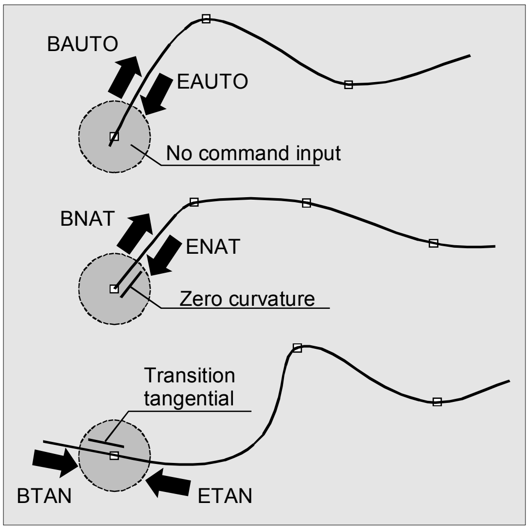
Explanation of the commands

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)

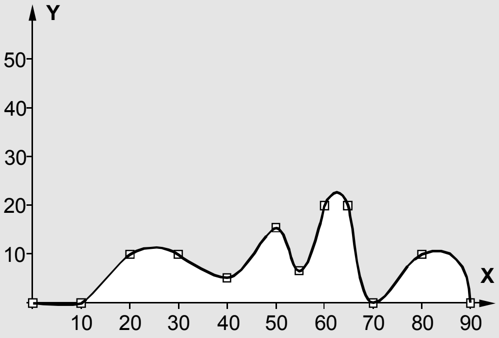
End of spline curve:

`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)



Example

C spline, zero curvature at start and end

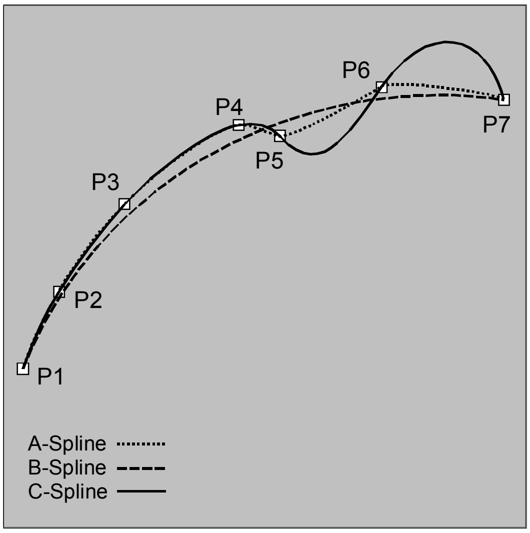


```
N10 G1 X0 Y0 F300
N15 X10
N20 BNAT ENAT                           C spline, at start and end
                                        Zero curvature
N30 CSPLINE X20 Y10
N40 X30
N50 X40 Y5
N60 X50 Y15
N70 X55 Y7
N80 X60 Y20
N90 X65 Y20
N100 X70 Y0
N110 X80 Y10
N120 X90 Y0
N130 M30
```

What does which spline do?

Comparison of three spline types with identical interpolation points:

A spline (akima spline)
B spline (Bezier spline)
C spline (cubic spline)



Spline grouping

Up to eight path axes can be involved in a spline interpolation grouping. The SPLINEPATH instruction defines which axes are to be involved in the spline. 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.

Programming

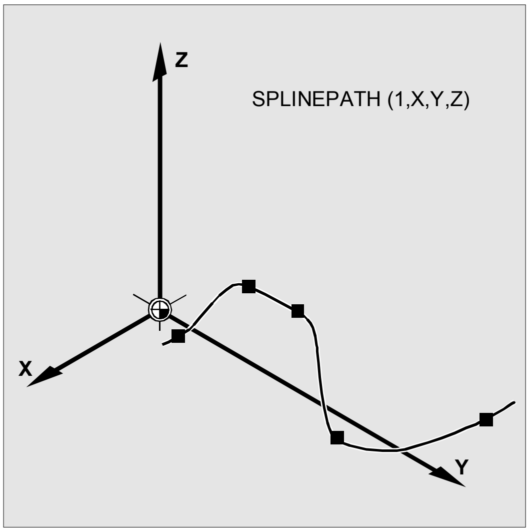
```
SPLINEPATH(n,X,Y,Z,…)
```

Explanation

`SPLINEPATH(n,X,Y,Z,…)`	n = 1, fixed value X,Y,Z,... path axis names

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

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.

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 Compressor COMPOF/ON, COMPCURV, COMPCAD (SW 6.2)

Programming

```
COMPON/COMPCURV/COMPCAD
COMPOF
```

Explanation

`COMPON/COMPCURV/COMPCAD`	Compressor ON
`COMPOF`	Compressor OFF

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.

SW 4.4 and higher:
With G code COMPCURV, the block transitions are with constant acceleration. This ensures both smooth velocity and acceleration of all axes at block transitions.

SW 6.2 and higher
The COMPCAD G code can be used to select a further compression which optimizes the **surface quality and velocity**. The interpolation accuracy can again be specified in machine data. 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 generates polynomial blocks that merge into one another with constant 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.

SW 6.3 and higher

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.
The compressor for orientation function is only implemented if the orientation transformation option is available.
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=<...> A=<...>
      B=<...> F=<...> ; Comment
```

Axis positions as parameter printouts with < ... > parameter printout such as `X=R1*(R2+R3)`

Active orientation transformation (TRAORI) being active, the following kinematics-independent programming options for the tool direction of 5-axis machines are possible:

1. Programming of the direction vector:
   `A3=< ...> B3=< ... > C3=< ... >`
2. Programming of the Euler angle or RPY angle:
   `A2=< ...> B2=< ... > C2=< ... >`

The orientation motion is only compressed when the large circle interpolation is active, i.e. the tool orientation is changed in the plane which is determined by start and end orientation.
Large circle interpolation is performed under the following conditions:

1. Please note that for MD 21104: ORI_IPO_WITH_G_CODE = FALSE, if ORIWKS is active and the orientation is programmed as a vector (with A3, B3, C3 or A2, B2, C2).
2. Please note that for MD 21104: ORI_IPO_WITH_G_CODE = TRUE, if ORIVECT or ORIPLANE is active. The tool orientation can be programmed either as a direction vector or with rotary axis positions.

If one of the G-codes ORICONxx or ORICURVE is active or if polynomials are programmed for the orientation angle (PO[PHI] and PO[PSI]) a large circle interpolation is not performed, i.e., blocks of this type are not compressed.

For **6-axis** machines you can program the tool rotation in addition to the tool orientation. You can program the angle of rotation with the identifier THETA (THETA=<...>).
NC blocks in which additional rotation is programmed, can only be compressed if the angle of rotation changes linear, meaning that you must not program a polynomial with PO[THT]=(...) for the angle of rotation.
NC block structure in general:

```
N... X=<...> Y=<...> Z=<...> A3=<...>
      B3=<...> C3=<...> THETA=<...> F=<...>
```

```
or
```

```
N... X=<...> Y=<...> Z=<...> A2=<...>
      B2=<...> C2=<...> THETA=<...> F=<...>
```

If tool orientation is specified via rotary axis positions, e.g. as:

```
N... X=<...> Y=<...> Z=<...> A=<...>
            B=<...> THETA=<...> F=<...>
```

the compression is performed in two different ways, depending on whether or not large circle interpolation is performed. If large circle interpolation is not performed, the compressed orientation change is represented by axial polynomials for the rotary axes.

Accuracy
You can compress NC blocks only if you allow the contour to deviate from the programmed contour. You can set the maximal 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 max. the tolerance set with the axial MD.

Contour accuracy

It controls the max. geometrical contour deviations (geometry axes) and the tool orientation. It is done via the setting data for:

1. Max. tolerance for the contour
2. Max. angular displacement for tool orientation
3. Max. angular displacement for the angle of tool rotation (only available for 6–axis machines)

With the channel-specific MD 20482 COMPRESSOR_MODE, you can set tolerance specifications:

0: Axis precision: axial tolerances for all axes (geometry axes and orientation axes).

1: Contour precision: Spec. of the contour tolerance (1.), the tolerance for orientation via axial tolerances (a.).

2: Spec. of the max. angular displacement for tool orientation (2.), tolerance for the contour via axial tolerances (a.).

3: Specification of the contour tolerance with (1.) and specification of the max. angular displacement for tool orientation with (2.).

It is only possible to specify a maximum angular displacement for the tool orientation if an orientation transformation (TRAORI) is active.

Activation

You can activate "Compressor for orientations" via one of the following commands:

COMPON, COMPCURV (COMPCAD not possible).

References: /FB3/, F2: "3-axis to 5-axis transformation"

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.

Special features with 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.

As a rule, CAD/CAM systems provide linear blocks that meet the programmed accuracy.
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 activates 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.

Conditions for usage:
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.

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-4.698`
`N13 X16.865 Y-4.72`
`N14 X16.91 Y-4.799`
…	

`N1037 COMPOF` …	compressor OFF

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
```

Not compressed are e.g. extended addresses such as C=100 or A=AC(100).
NC SW 6.3 and higher: Motion blocks with extended addresses are now 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`	

Example "Compressor for orientations"

In the example program below, a circle approached by a polygon definition is compressed.

A synchronous tool orientation moves on the lateral surface of a 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
```

Maximum deviation

the contour	0.05 mm
the orientation	5 degrees

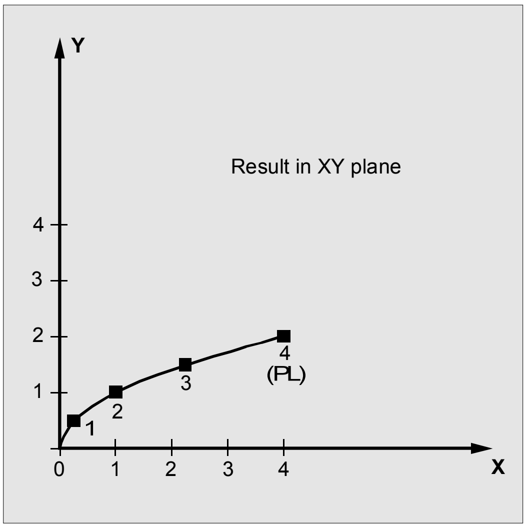
```
$SC_COMPRESS_CONTOUR_TOL = 0.05
$SC_COMPRESS_ORI_TOL = 5
TRAORI
COMPCURV
N100 X0 Y0 A3=0 B3=-1 C3=1
N110 FOR COUNTER = 0 TO NUMBER
N120 ANGLE= 360 * COUNTER /NUMBER
N130  X=RADIUS*COS(ANGLE)Y=RADIUS*
      SIN(ANGLE) A3=SIN(ANGLE)
      B3=-COS(ANGLE) C3=1
N140 ENDFOR
...
```

The movement describes a circle generated from polygons.

while the orientation moves on a taper around the Z axis at an arc angle of 45 degrees.

5.4 Polynomial interpolation – POLY, POLYPATH (SW 5 and higher)

The control system is capable of traveling curves (paths) in which every selected path axis is operating as a function up to SW 5 (polyn., max. 3rd deg.), from SW 6 (polyn., max. 5th deg.).



The equation used to express the polynomial function is generally as follows:
$f(p)= a_0 + a_1p + a_2p^2 + a_3p^3$ (up to SW 5) or
$f(p)= a_0 + a_1p + a_2p^2 + a_3p^3 + a_4p^4 + a_5p^5$ (>= SW 6)
Key:
a_n: Constant coefficients
p: Parameter

By assigning concrete values to these coefficients, it is possible to generate a wide variety of curve shapes such as line, parabola and power functions.

Setting coefficients $a_2 = a_3 = 0$ (up to SW 5) or
$a_2 = a_3 = a_4 = a_5 = 0$ (SW 6 and higher)
for example, results in 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 end of the definition range (PL) and start position

Definition

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.

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).

SW 5 and higher
Subprogram 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

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.

Restrictions

SW 5 and lower

- Polynomials for geometry axes/special path axes can only be programmed if either G0/G1 or POLY is active. Therefore, with circular interpolation it is not possible to traverse additional axes via polynomials.
  As standard, polynomials can only be programmed with PO[...] if the G code POLY is active.

SW 5 and higher

- It is possible to program polynomials **without** the G code POLY 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 subprogram POLYPATH (...), you can select which axes are to be interpolated with polynomial.

SW 6 and higher

- Coefficients a_4 and a_5 are only supported by SW 6 and higher.
- New polynomial syntax with PO
  The syntax used hitherto also remains valid

Example of applicable polynomial syntax with PO

Polynomial syntax used hitherto remains valid	New polynomial syntax (SW 6 and higher)
`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(.. , ..)`

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 (up to SW 5)

POLYPATH ("AXES", "VECT")(SW 5 and higher)

Expansion to polynomials of the 5th degree and new polynomial syntax (SW 6 and higher)

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

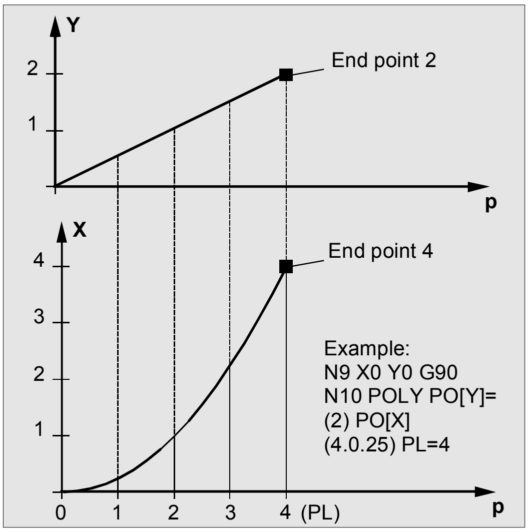
Explanation

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	Coefficients a_2, a_3, a_4, and a_5 are written with their value; range of values 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.

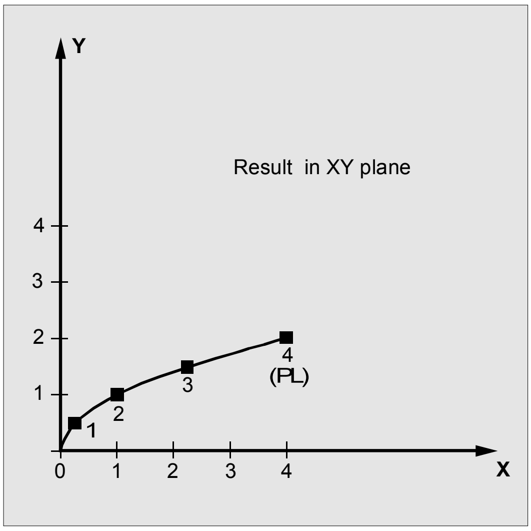
Example

N10 G1 X… Y… Z… F600	
N11 POLY PO[X]=(1,2.5,0.7) -> -> PO[Y]=(0.3,1,3.2) PL=1.5	Polynomial interpolation ON
N12 PO[X]=(0,2.5,1.7) PO[Y]=(2.3,1.7) PL=3 … N20 M8 H126 …	
N25 X70 PO[Y]=(9.3,1,7.67) PL=5	Mixed 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 a curve in the X/Y plane



```
N9 X0 Y0 G90 F100
N10 POLY PO[Y]=(2) PO[X]=(4,0.25) PL=4
```



Special case 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,-10)  PO[Y]=(10)  PO[]=(2,1)`	Geometry axes traverse along quadrant to X0, Y10

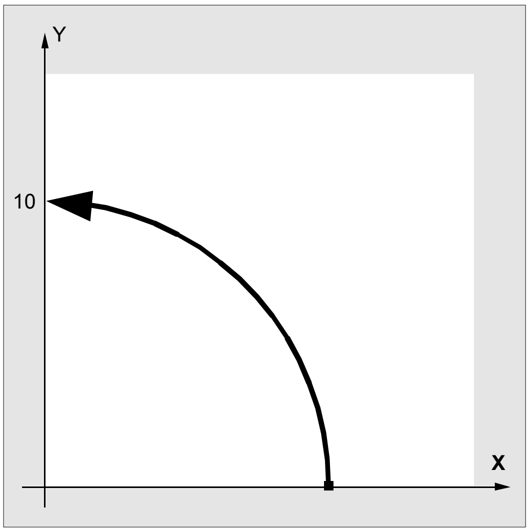
The constant coefficient (a0) of the denominator polynomial is always assumed to be 1, the specified end point is not dependent on G90/G91.

The result obtained from the above example is as follows:

$X(p)=10(1–p^2)/(1+p^2)$ and $Y(p)=20p/(1+p^2)$
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 $(X)=10+0*p–10p^2$
Numerator $(Y)=0+20*p+0*p^2$
Denominator $= 1+2*p+1*p^2$



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.

Other Information

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.

5.5 Settable path reference, SPATH, UPATH (SW 4.3 and higher)

Programming

`SPATH`	Path reference for FGROUP axes is arc length
`UPATH`	Path reference for FGROUP axes is curve parameter

Introduction

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.

Previously, only the first motion control variant was implemented; now SW 4.3 and higher offers a G code (SPATH, UPATH) for selecting and programming the desired response.

Function

During polynomial interpolation - and here we are referring to polynomial interpolation in the stricter sense (POLY), all spline interpolation types (ASPLINE, BSPLINE, CSPLINE) and linear interpolation with compressor (COMPON, COMPCURV) - the positions of all path axes i are preset by means of 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.

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.

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

Example

The example below shows a square with 20 mm side lengths and corners rounded with G643.	
The maximum deviations from the exact contour are defined for each axis by machine data MD 33100: COMPRESS_POS_TOL[...].	
N10 G1 X… Y… Z… F500	
N20 G643	Block-internal corner rounding with G643
N30 XO Y0	
N40 X20 Y0	20 mm edge length for axes
N50 X20 Y20	
N60 X0 Y20	

```
N70  X0   Y0
N100 M30
```

Restrictions

The path reference set is of no importance with

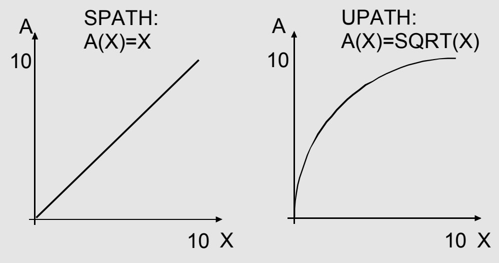
- linear and circular interpolation,
- in thread blocks and
- if all path axes are contained in FGROUP.

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. 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.

Programming example

The following program example shows the difference between both types of motion control. Both times the default setting FGROUP(X,Y,Z) is active.



Different geometry relationships between axes with SPATH and UPATH

```
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:

Control response to power ON, mode change, RESET, block search, REPOS

After a reset, the G code defined via MD 20150: GCODE_RESET_VALUES [44] is active (45th G code group).

The basic setting value for the type of rounding is set in MD 20150: GCODE_RESET_VALUES [9] (10th G code group).

Machine/option data

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 basic setting value for the type of rounding is set in MD 20150: GCODE_RESET_VALUES [9] (10th G code group).

Axial machine data MD 33100: COMPRESS_POS_TOL has been expanded in SW 4.3 and higher: It contains the tolerances for the compressor function and for rounding with G642.

5.6 Measurements with touch trigger probe, MEAS, MEAW

Programming

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

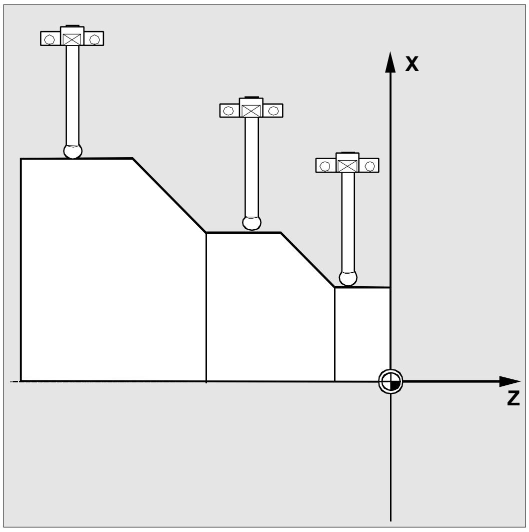
Explanation of the commands

`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

*Max. of two inputs depending on configuration level

Sequence

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. A maximum of 2 probes can be installed.



Measurement result
The measurement result is available under the following variables for these axes:

- 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.

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 performed
1	Measuring job successfully completed (probe has switched state)

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.

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 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.

Example:

```
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.

Measured value recording

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.

Remarks

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.

Other information

The MEAS and MEAW functions are active non-modally.

5.7 Extended measuring function MEASA, MEAWA, MEAC (SW 4 and higher, option)

Programming

`MEASA [axis]=(mode, TE1,…, TE 4)`	Measurement with deletion of distance-to-go
`MEAWA [axis]=(mode, TE 1,…, TE 4)`	Measurement without deletion of distance-to-go
`MEAC [axis]=(mode, meas. memory, TE 1,...TE4)`	Continuous measurement without deleting distance-to-go

Explanation

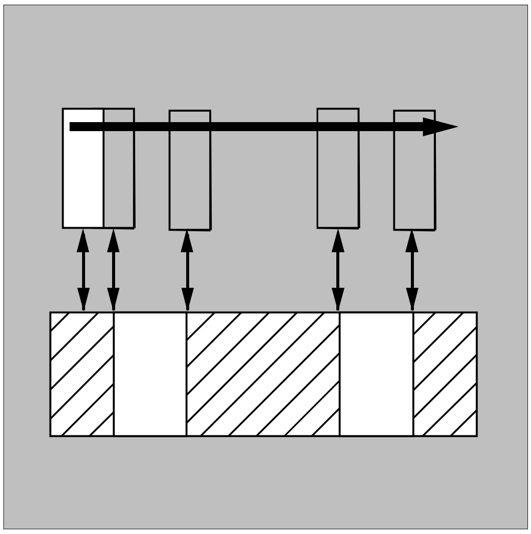
Axis	Name of channel axis used for measurement
Mode	Two-digit setting for operating mode consisting of **Measuring mode** (ones decade) and 0 Abort measuring job 1 Mode **1**: Up to 4 different trigger events can be activated **simultaneously** 2 Mode **2**: Up to 4 trigger events can be activated **successively** 3 Mode **3**: Up to 4 trigger events can be activated **successively**, 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 meas. 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)

Function

Axial measurement is available SW 4 and higher. With this system, measurements can be taken axially with several probes and several measuring systems.

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. MEASA and MEAWA are non-modal commands.

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.



Sequence

The measurements can be programmed in the parts program **or** from a synchronized action (Chapter 10). Please note that only one measuring job can be active at any given time for each axis.

Other information

- 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 feedrates with which no more than one trigger event of the same type and no more than 4 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 4 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 depends on the selected mode.

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.

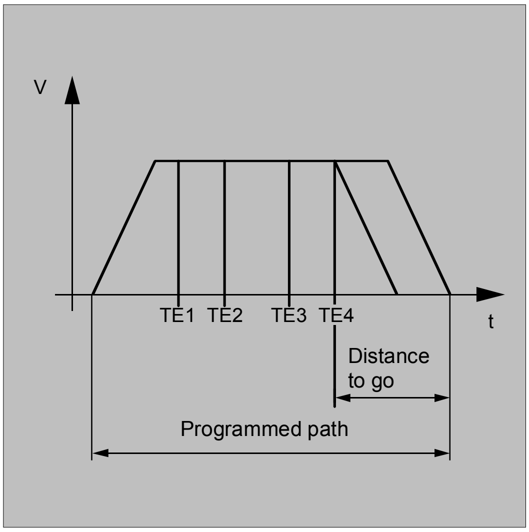
With the second digit, i.e. the **measurement mode**, measuring process is adapted to the capabilities of the connected control system:

- **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.

Other information

No more than 2 trigger events can be programmed if 2 measuring systems are in use.

Measurement with and without delete distance-to-go



When command MEASA is programmed, the distance-to-go is not deleted until all required measured values have been recorded.

The MEAWA function is employed in the case of special measuring tasks where a programmed position must always be approached.

MEASA and MEAWA can be programmed in the same block.

If MEASA/MEAWA is programmed with MEAS/ MEAW in the same block, an error message is output.

- 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

Other Information

No internal preprocessing stop is generated when these variables are read.
A preprocessing stop must be programmed with STOPRE (Section 15.1) at the appropriate position.
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
```

Measuring 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 of measuring system 1 on trigger event 1
$AA_MM2[axis]	or	$AA_MW2[axis]	Measured value of measuring system 2 on trigger event 1
$AA_MM3[axis]	or	$AA_MW3[axis]	Measured value of measuring system 1 on trigger event 2
$AA_MM4[axis]	or	$AA_MW4[axis]	Measured value of measuring system 2 on trigger event 2

Measuring 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.

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:	/FB/ M5, Measurements

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.

References: /PGA/ Chapter 10, Synchronized Actions

Other Information

- 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.

Programming example

Measurement with delete distance-to-go in mode 1
(evaluation in chronological sequence)

a) with one 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`	Preprocessor 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:`	

Programming example

b) with two 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`	Preprocessor stop
`N220 IF $AC_MEA[1] == FALSE gotof END`	Check success of measurement.
`N230 R10 = $AA_MM1[X]`	Store measured value of measuring system 1 on rising edge
`N240 R11 = $AA_MM2[X]`	Store measured value of measuring system 2 on rising edge
`N250 R12 = $AA_MM3[X]`	Store measured value of measuring system 1 on falling edge
`N260 R13 = $AA_MM4[X]`	Store measured value of measuring system 2 on falling edge
`N270 END:`	

Measurement 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`	Preprocessor 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:`	

Programming example

Continuous measurement in mode 1

(evaluation in chronological sequence)

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`	

Measurement with delete distance-to-go after ten measured values

`...`	

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

`N20 MEAC[x]=(1,1,1,-1) G01 X100 F500`	
`N30 MEAC[X]=(0)`	
`N40 R1 = $AC_FIFO1[4]`	No. of measured values
`...`	

The following programming errors are detected and indicated appropriately:

- 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) MEASA[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
  ```

5.8 Special functions for OEM users, G810 to G829

OEM addresses

The meaning of OEM addresses is determined by the OEM user.
Their functionality is incorporated by means of compile cycles. Five OEM addresses are reserved.
The address identifiers are settable.
OEM addresses can be programmed in any block.

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.

Reserved G groups

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 subprograms

OEM users can also set up predefined functions and subprograms with parameter transfer.

5.9 Feedrate reduction with corner deceleration, G62, G621 (SW 6.1 and higher)

Programming

```
FENDNORM
G62 G41
G621
```

Explanation of the commands

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

Function

With automatic corner deceleration the feedrate 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 feedrate reduction
- Override with which the feedrate is reduced
- Detection of a relevant corner

Relevant corners are those whose inside angle is less than the corner parameterized in the setting data.

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 feedrate resulting from:

F * (override for feedrate reduction) * feedrate override

The maximum possible feedrate 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.
Default value FENDNORM deactivates the function of the automatic corner override.

Other Information

This function is not part of the standard scope of SINUMERIK and must be activated for the relevant software versions.
References: /FBA/ Function desc. ISO dialects

5.10 Programmable motion end criterion (SW 5.1 and higher)

Programming

`FINEA [<axis>]`	
`COARSEA [<axis>]`	
`IPOENDA [<axis>]`	
`IPOBRKA(<axis>[, [<value as a percentage>]])`	Multiple specifications are possible
`ADISPOSA(<axis>, [<Int>][,[<Real>]])`	Multiple specifications are possible

Explanation of the commands

`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 of the block change should be as %
`Int`	Mode 0: Tolerance window not active 1: Tolerance window with respect to set position 2: Tolerance window with respect to actual position
`Real`	Size of tolerance window. This value is entered in setting data 43610: ADISPOSA_VALUE synchronized with the main run

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 axes. Depending on the end of motion criterion set, parts program blocks or technology cycle blocks with single axis motion take different times to complete. The same applies for PLC positioning statements via FC15/ 16/ 18.

System variable $AA_MOTEND

The set end of movement criterion can be scanned with system variable `$AA_MOTEND[<axis>]`.

• `$AA_MOTEND[<axis>] = 1`	Movement end with "Exact stop fine"
• `$AA_MOTEND[<axis>] = 2`	Movement end with "Exact stop coarse"
• `$AA_MOTEND[<axis>] = 3`	End of motion with "IPO stop"
• `$AA_MOTEND[<axis>] = 4 (SW 6.2 and higher)`	Block change criterion braking ramp of axis motion
• `$AA_MOTEND[<axis>] = 5 (SW 6.4 and higher)`	Block change in braking ramp with tolerance window relative to "setpoint position"
• `$AA_MOTEND[<axis>] = 6 (SW 6.4 and higher)`	Block change in braking ramp with tolerance window relative to "actual position"

Other Information

The last programmed value is retained after RESET.

References: /FB1/, V1 Feedrates

SW 6.2 and higher

Block change criterion in braking ramp

The percentage value is entered in SD 43600: IPOBRAKE_BLOCK_EXCHANGE synchronized with the main run. 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

From SW 6.4, 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**
- from SW 6.4, its current actual or setpoint 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/, P2 positioning axes
/PG/, Feed rate control and spindle motion

Programming examples

```
...
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

```
...
```

Braking ramp block change condition 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,
              ; as soon as the X-axis starts to brake, the block change occurs
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 X10F100
N70 M30
...
```

Braking ramp block change condition in synchronized actions

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
```

5.11 Programmable servo parameter block (SW 5.1 and higher)

Programming

```
SCPARA [<axis>]= <value>
```

Explanation of the commands

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

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 prevent conflicts between the PLC–user request and NC–user request, a further 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.

The current parameter block can be polled using the system variables `$AA_SCPAR[<axis>]`.

Other Information

- Up to SW 5.1, the servo parameter set can only be selected through the PLC (DB3n DBB9 bits0–2). For G33, G331 and G332, the most suitable parameter block is selected by the control.
- If the **servo parameter block** is to be **changed** both in a parts program and in a synchronized action and the PLC, the PLC application program must be extended.
- **References**: /FB1/V1 Feedrates

Programming example

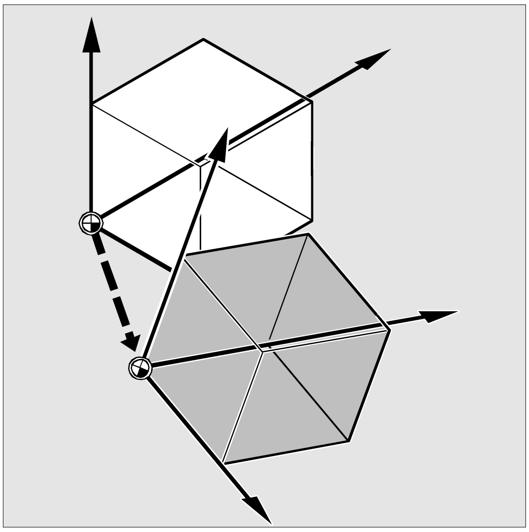
`...`	
`N110 SCPARA[X]= 3`	The 3rd parameter block is selected for axis X.
`...`	

■

Frames

6.1 Coordinate transformation via frame variables

Defining coordinate transformations with frame variables



In addition to the programming options already described in the Programming Guide "Fundamentals", you can also define coordinate systems with predefined frame variables.

Coordinate systems
The following coordinate systems are defined:

MCS:	Machine coordinate system
BCS:	Basic coordinate system
BOS:	Basic origin system
SZS:	Settable zero system
WCS:	Workpiece coordinate system

What is a predefined frame variable?
Predefined frame variables are vocabulary words 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

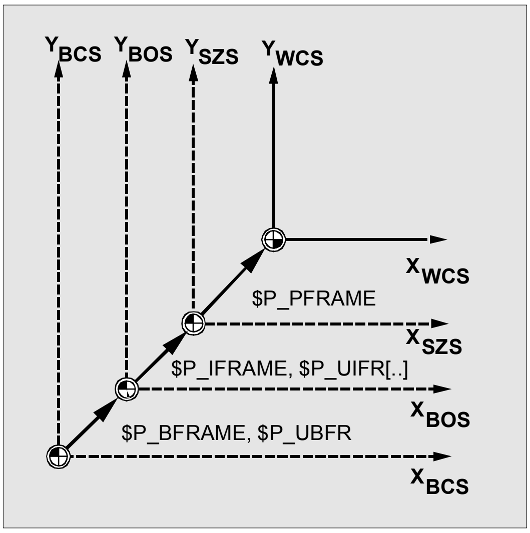
Frame variable/frame relationship

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 out 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

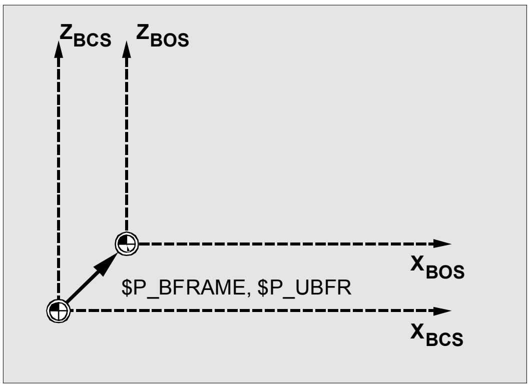
Overview of predefined variables

$P_BFRAME

Current base frame variable that establishes the reference between the basic coordinate system (BCS) and the basic origin system (BOS).

For the base 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

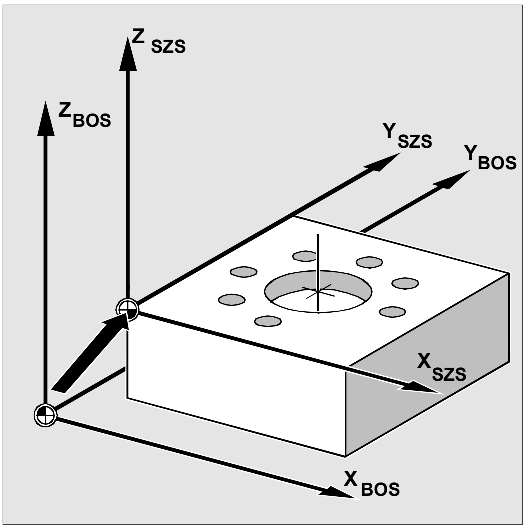


$P_IFRAME

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]`

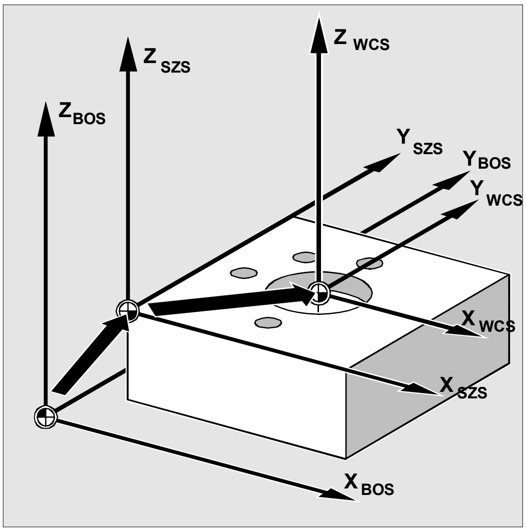
After G54 is programmed, for example, `$P_IFRAME` contains the translation, rotation, scaling and mirroring defined by G54.



$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 frame resulting from the programming of TRANS/ATRANS, ROT/AROT, SCALE/ASCALE, MIRROR/AMIRROR or the assignment of CTRANS, CROT, CMIRROR, CSCALE to the programmable FRAME.



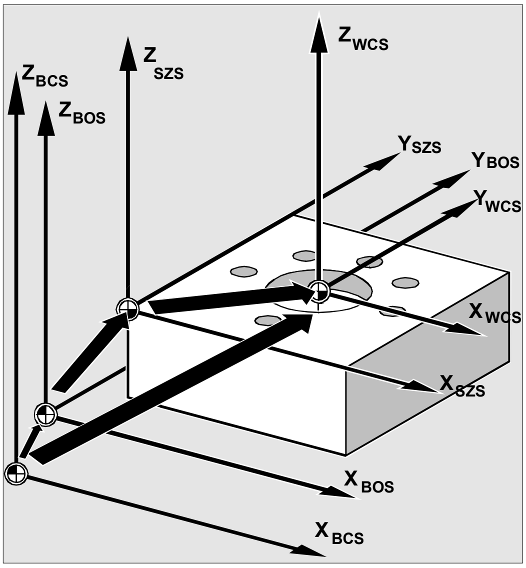
$P_ACTFRAME

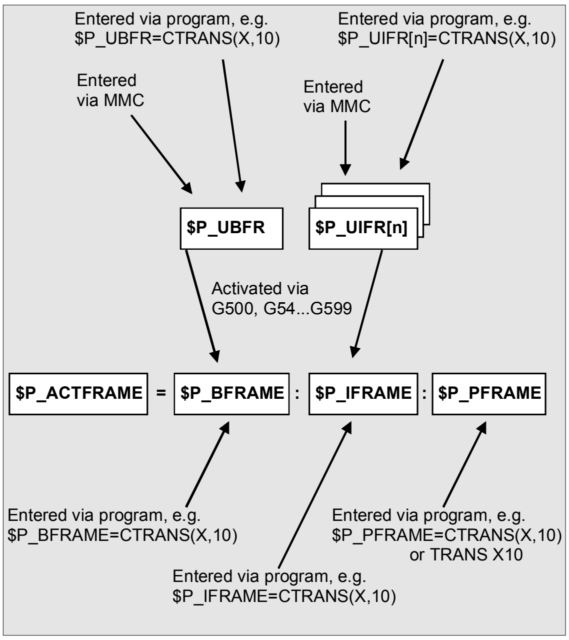
Current total frame resulting from chaining of the current base frame variable `$P_BFRAME`, the current settable frame variable `$P_IFRAME` and the current programmable frame variable `$P_PFRAME`.

`$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
```





Base frame and settable frame are effective after Reset if MD 20110 RESET_MODE_MASK is set as follows:

Bit0=1, bit14=1 --> $P_UBFR (base frame)
effective

Bit0=1, bit5=1 --> $P_UIFR [$P_UIFRNUM]
(settable frame) effective

Predefined settable frames $P_UBFR

The base frame is programmed with $P_UBFR, but it is not simultaneously active in the parts program. The base 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
- instructions 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

Five predefined settable frames are set as standard `$P_UIFR[0]`…`$P_UIFR[4]` or 5 G commands with the same meaning – G500 and G54 to G57 – at whose addresses values can be stored.

`$P_IFRAME=$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

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.

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

Values can be assigned directly, frames can be chained or frames can be assigned to other frames in the NC program.

Direct value assignment

Programming

```
$P_PFRAME=CTRANS (X, axis value, Y, axis value, Z, axis value, …)
$P_PFRAME=CROT (X, angle, Y, angle, Z, angle, …)
$P_PFRAME=CSCALE (X, scale, Y, scale, Z, scale, …)
$P_PFRAME=CMIRROR (X, Y, Z)
```

Programming `$P_BFRAME` is carried out analog to `$P_PFRAME`.

Explanation of the commands

CTRANS	Translation of specified axes
CROT	Rotation around specified axes
CSCALE	Scale change on specified axes
CMIRROR	Direction reversal on specified axis

Function

You can use these functions to assign frames or frame variables directly in the NC program.

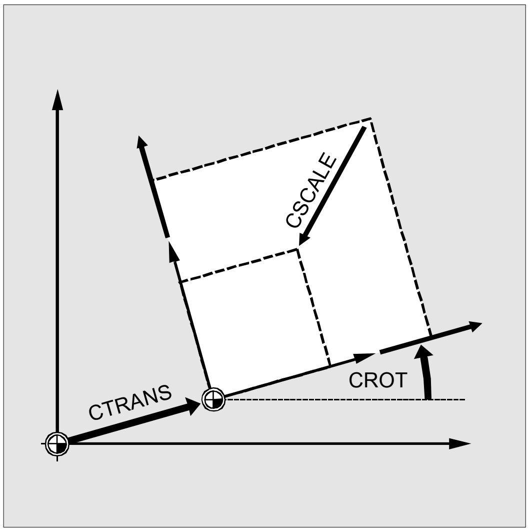
Sequence

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.



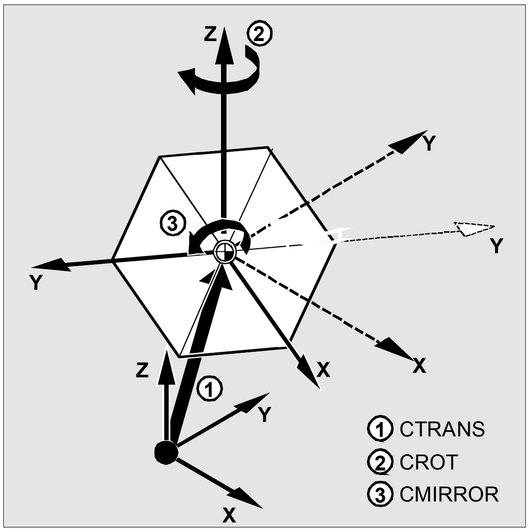
Other information

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`.

Programming 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)
```

Reading and changing frame components

Programming (examples)

`R10=$P_UIFR[$P_UIFRNUM, 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.

Explanation of the commands

`$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.

`TR`
`FI`
`RT`
`SCMI`
	Specify the component to be read or modified:
	TR translation, FI translation fine, RT rotation, SC scale change, MI mirroring.
	The corresponding axis is also specified (see examples).

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.

Sequence

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]`.

Linking complete frames

A complete frame can be assigned to another frame.

Programming (examples)

```
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.

Other information

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°

Frame chaining

Programming (examples)

```
$P_IFRAME=$P_UIFR[15]:$P_UIFR[16]
```

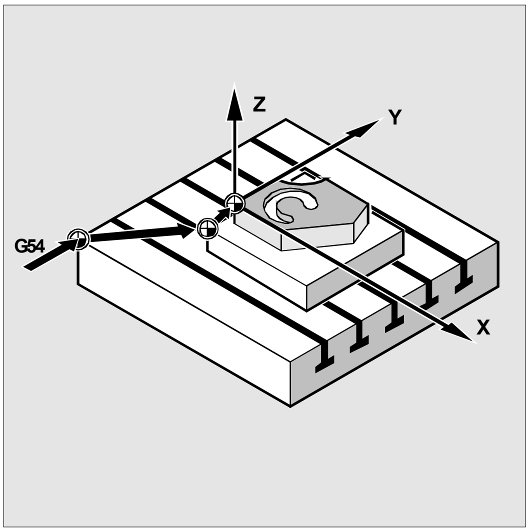
`$P_UIFR[15]` contains, for example, data for zero offsets. The data of `$P_UIFR[16]`, e.g. data for rotations, are subsequently processed additively.

```
$P_UIFR[3]=$P_UIFR[4]:$P_UIFR[5]
```

The settable frame 3 is created by chaining the settable frames 4 and 5.

Function

Frame chaining is suitable for the description of several workpieces, arranged on a pallet, which are to be machined in the same process.



Sequence

The frames are chained in the programmed sequence. The frame components (translations, rotations, etc.) are executed additively in succession.

The frame components can only contain intermediate values for the description of pallet tasks. These are chained to generate various workpiece zeroes.

Please note that the frames must be connected by the colon chain operator **:**.

Defining new frames

Programming

```
DEF FRAME PALLET1

PALETTE1=CTRANS(…):CROT(…)…
```

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.

Sequence

You can use the functions CTRANS, CROT, CSCALE and CMIRROR to assign values to your frames in the NC program.
You will find more information on this subject on the previous pages.

Frame rotation definition

Function

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

- 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 fame rotation. The rotation component is determined by the rotation component of an oriented tool holder.

6.3 Coarse/fine offset

Function

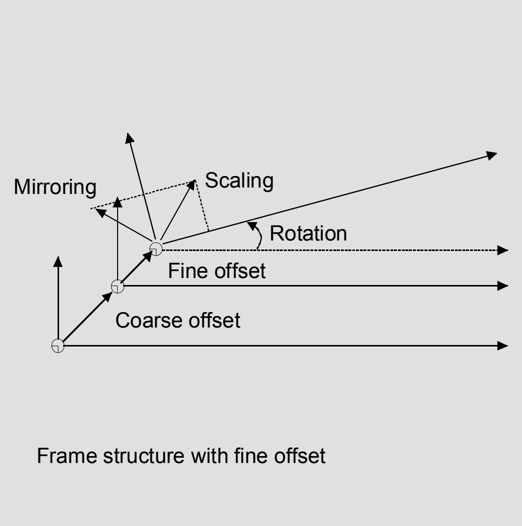
Fine offset

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

Coarse offset

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

Coarse and fine offset add up to the total offset.



Frame structure with fine offset

Programming

```
$P_UBFR=CTRANS(x, 10) : CFINE(x, 0.1) : CROT(x, 45)  ;Concatenation of
                                        ;translation, fine offset
                                        ;and rotation
$P_UIFR[1]=CFINE(x, 0.5, y, 1.0, z, 0.1);of the entire frame is overwritten
                                        ;with CFINE incl. Coarse offset
```

Access to the individual components of the fine offset is achieved through component specification FI.

Programming

```
DEF REAL FINEX                     ;Definition of variable FINEX
FINEX=$P_UIFR[$P_UIFRNUM, x, FI] ;Readout of the fine offset via
                                 ;variables FINEX

FINEX=$P_UIFR[3, X, FI]          ;Readout of the fine offset of the X axis
                                 ;in the 3rd frame via the FINEX variable
```

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

A fine offset changed the operator 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.

Machine manufacturer

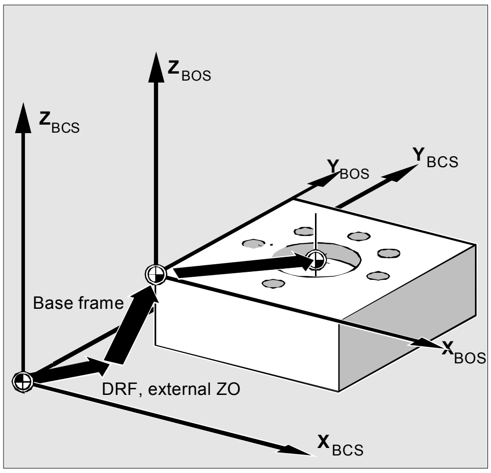
SW 5 and higher
With MD18600: MM_FRAME_FINE_TRANS is used to configure the fine offset for the following variants:
0: The fine offset cannot be entered or programmed. G58 and G59 are not possible.
1: Fine offset for settable frames, base frames, programmable frames, G58 and G59 can be entered/programmed.

6.4 DRF offset

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

The DRF offset is active in the basic coordinate system. See the diagram for the relationships.

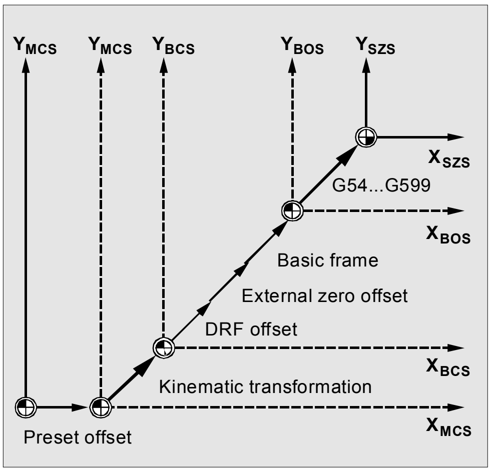
You will find more information in the Operator's Guide.



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

External zero offset
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 offset values, $AA_ETRANS
The offset values are programmed by assigning the axis-specific system variables.

Assigning offset value
`$AA_ETRANS[axis]=R`$_I$

R_I 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.

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 Programming preset offset, PRESETON

Programming

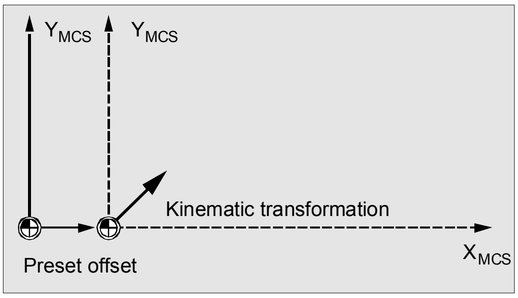
```
PRESETON (AXIS,VALUE,…)
```

Explanation of the commands

`PRESETON`	Preset actual value memory
`Axis`	Machine axis parameter
`Value`	New actual value to apply to the specified axis

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).



Note: Preset mode with synchronized actions should only be implemented with the vocabulary word "WHEN" or "EVERY".

Sequence

The actual values are assigned to the machine coordinate system – the values refer to the machine axes.

Example:

```
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.

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 Section 3.1.

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

Explanation of the commands

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

Other information

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

Example:

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

6.8 Frame calculation from 3 measuring points in space, MEAFRAME

MEAFRAME is an extension of the 840D language used for supporting measuring cycles.
This function is valid in SW 4.3 and higher

Function

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.
The function MEAFRAME calculates the frame from three ideal and the corresponding measured points.
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.

Other information

The frame created by MEAFRAME can be transformed by the ADDFRAME function into another frame in the frame chain as from SW 6.3.
See the application example for chaining MEAFRAME for further corrections.

Programming

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

Explanation of the commands

`MEAFRAME`	Frame calculation of three measured points in space
`IDEAL_POINT`	2-dim. array of real data containing the three coordinates of the ideal points
`MEAS_POINT`	2-dim. array of real data containing the three coordinates of the measured points
`FIT_QUALITY`	Variable of type real 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 an triangle that is congruent to the ideal triangle.

Application

```
; parts program 1
;
DEF FRAME CORR_FRAME
;
; setting measured 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, 9.8);        for test
DEF REAL FIT_QUALITY = 0
;
DEF REAL ROT_FRAME_LIMIT   = 5;        allows max. 5° rotation of the part position
DEF REAL FIT_QUALITY_LIMIT = 3;        allows max. 3 mm distortion between the ideal and ;
                                   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;          limiting the 1st RPY angle
SETAL(65020)
GOTOF NO_FRAME
ENDIF
;
N260 IF CORR_FRAME[Y,RT] > ROT_FRAME_LIMIT;          limiting the 2nd RPY angle
SETAL(65021)
GOTOF NO_FRAME
ENDIF
;
N270 IF CORR_FRAME[Z,RT] > ROT_FRAME_LIMIT;          limiting the 3rd RPY angle
SETAL(65022)
GOTOF NO_FRAME
ENDIF
;
N300 $P_IFRAME=CORR_FRAME;          activate the probe frame via a settable frame
;
; check the frame by positioning the geometry axes at the ideal points
;
N400 X=IDEAL_POINT[0,0] Y=IDEAL_POINT[0,1] Z=IDEAL_POINT[0,2]
N410 SHOW_MCS_POS1[0]=$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, as preset with zero frame (no value set)
;
NO_FRAME:
M0
M30
```

Application example of frame chaining

Chaining of MEAFRAME for offsets

The MEAFRAME( ) function provides an offset frame. If this offset frame is chained with the settable frame $P_UIFR[1] that was active when the function was called, e.g. G54, you obtain a settable frame for further conversion for traversing or machining.

Chaining with ADDFRAME (SW 6.3 and higher)

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 base 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.

Other information

Further information about the FRAME chaining, see:
/FB/, K2, Axes, Coordinate Systems, Frames

6.9 NCU-global frames (SW 5 and higher)

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.
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" is displayed.

NCU-global basic frames: **$P_NCBFR[n]**
You can configure up to 8 NCU-global basic frames.

Machine manufacturer

The number of global base frames is configured via machine data. (See /FB/ K2, Axes, Coordinate Systems, Frames)
Channel-specific base frames can be present at the same time.

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.

NCU-global settable frames: $P_UIFR[n]
All settable frames G500, G54...G599 can be configured either NCU-global or channel-specific.

Machine manufacturer

All settable frames can be reconfigured as global frames via MD 18601 MM_NUM_GLOBAL_USER_FRAMES.
See /FB/ K2, Axes, Coordinate Systems, 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

Function

The number of base frames can be configured in the channel via MD 28081 MM_NUM_BASE_FRAMES.
The standard configuration is designed for at least one base frame per channel. A maximum of eight base frames are supported per channel. In addition to the 8 base frames, there can also be 8 NCU-global base frames in the channel.

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.

$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 from MMC or PLC. These frame variables are saved by the data backup.

First basic frame in the channel
Writing to a predefined variable $P_UBFR will not activate the basic frame with array index 0 simultaneously, but it will be activated only after a G500, G54..G599 command is executed. 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 base frame, an alarm is issued at read/write: "Frame: instruction not permissible".

6.9.2 Frames active in the channel

Function

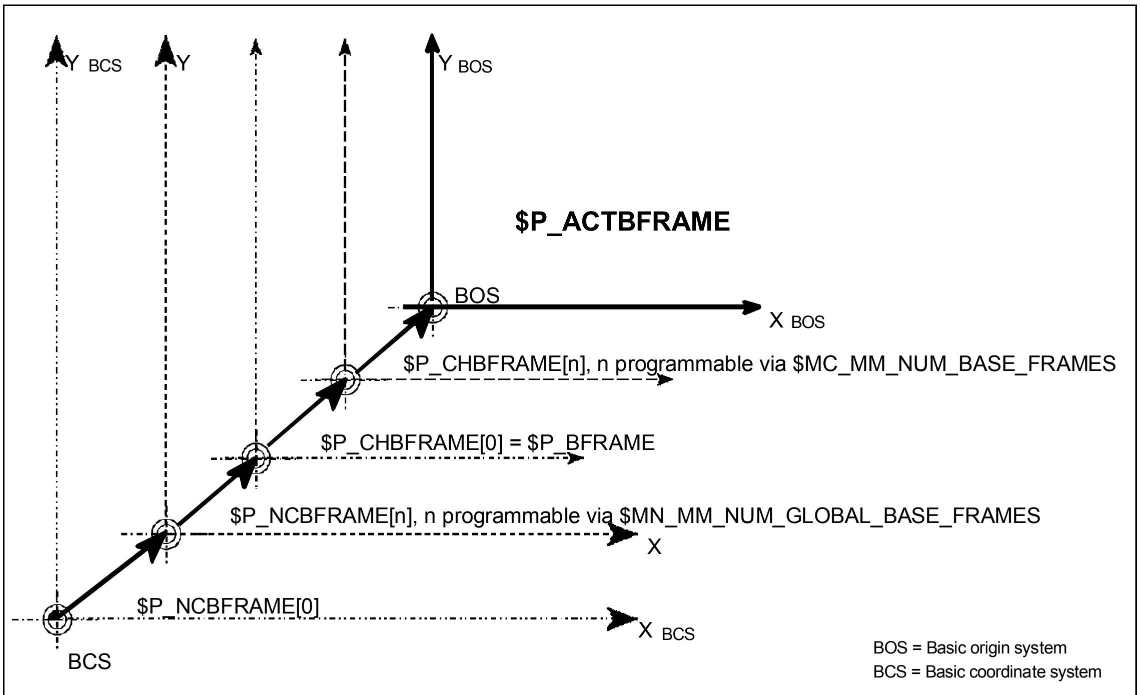
SW 6.1 and higher
Current system frames for
$P_PARTFRAME TCARR and PAROT
$P_SETFRAME preset act. val. mem. & scratching,
$P_EXTFRAME zero offset external,
You can read and write the current system frame in the parts program via these system variables.

$P_NCBFRAME[n]
Current NCU-global basic frames
You can read and write the current global basic frame field elements via system variable $P_NCBFRAME[n]. The resulting total base 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
You can read and write the current channel basic frame field elements via system variable $P_CHBFRAME[n]. 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
You can read and write the current basic frame with array index 0, which applies for the channel, in the parts program via the predefined frame variable $P_BFRAME. The written basic frame is immediately included in the calculation.
$P_BFRAME is identical to $P_CHBFRAME[0]. The system variable always has a valid default value. If there is no channel-specific base frame, an alarm is issued at read/write: "Frame: instruction not permissible".
$P_ACTBFRAME
Total basic frame
Variable $P_ACTBFRAME determines the chained total basic frame. The variable is read-only.

$P_ACTBFRAME corresponds to
$P_NCBFRAME[0] **:** ... **:** $P_NCBFRAME[n] **:**
$P_CHBFRAME[0] **:** ... **:** $P_CHBFRAME[n].


Y BCS
Y
Y BOS
$P_ACTBFRAME
BOS
X BOS
$P_CHBFRAME[n], n programmable via $MC_MM_NUM_BASE_FRAMES
$P_CHBFRAME[0] = $P_BFRAME
$P_NCBFRAME[n], n programmable via $MN_MM_NUM_GLOBAL_BASE_FRAMES
X
$P_NCBFRAME[0]
X BCS
BCS
BOS = Basic origin system
BCS = Basic coordinate system

$P-ACTBFRAME = $P_NCBFRAME[0] : $P_NCBFRAME[n] : $P_CHBFRAME[0] : $P_CHBFRAME[n]

$P_CHBFRMASK and $P_NCBFRMASK

Total basic frame

Via system variables $P_CHBFRMASK and $P_NCBFRMASK, the user can select the basic frames to be included in the calculation of the "total" 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 base frame field element of $P_ACTBFAME 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 = $MN_NCBFRAME_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

You can read and write the current settable frame, which applies in the channel, in the parts program via the predefined frame variable $P_IFRAME. 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.

SW 6.1 and higher
Current system frames for
$P_TOOLFRAME TOROT and TOFRAME
SW 6.3 and higher
$P_WPFRAME workpiece reference points
SW 6.4 and higher
$P_TRAFRAME transformations

The current system frame can be read and written via these system variables in the parts program.

$P_PFRAME
Current programmable frame
$P_PFRAME is the programmable frame which results from programming TRANS/ATRANS, G58/G59, ROT/AROT, SCALE/ASCALE, MIRROR/AMIRROR or from assigning CTRANS, CROT, CMIRROR, CSCALE to the programmable FRAME.
Current, programmable frame variable that establishes the reference between the settable zero system (SZS) and the workpiece coordinate system (WCS).

SW 6.3 and higher
Current system frame for
$P_CYCFRAME Cycles
The current system frame can be read and written in the parts program via these system variables.

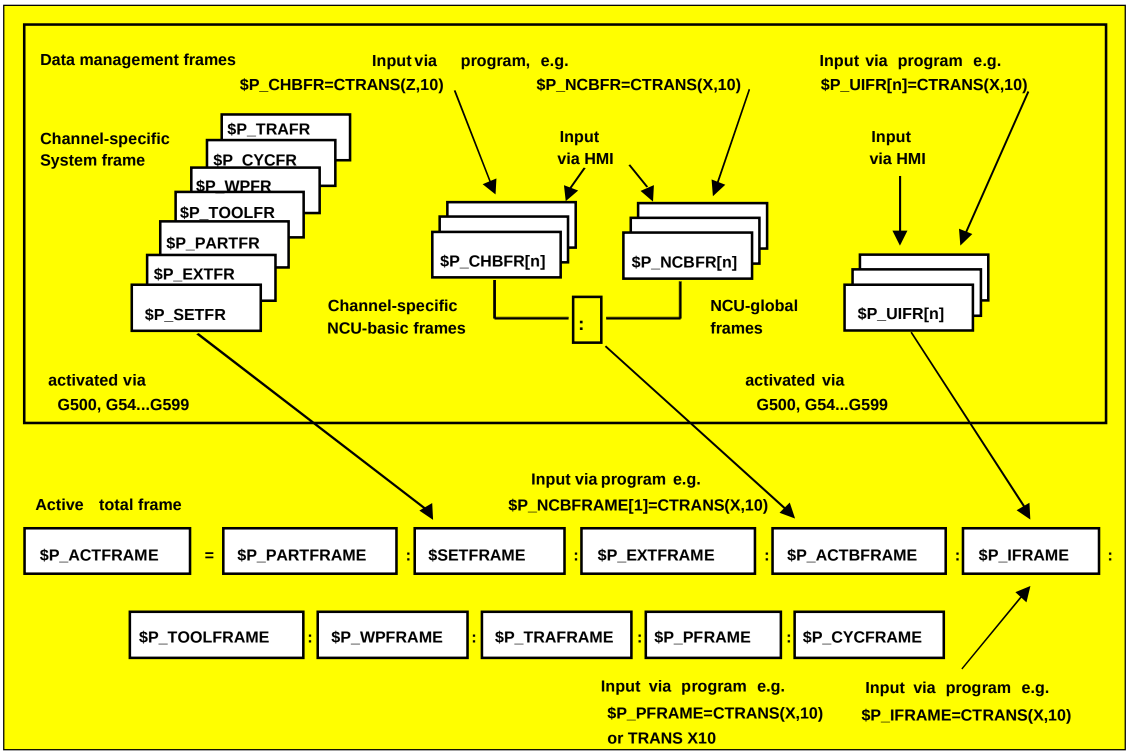
$P_ACTFRAME

Current total frame

The current resulting total frame $P_ACTFRAME now results from chaining all basic frames, the current settable frame and the programmable frame. The current frame is always updated whenever a frame component is changed.

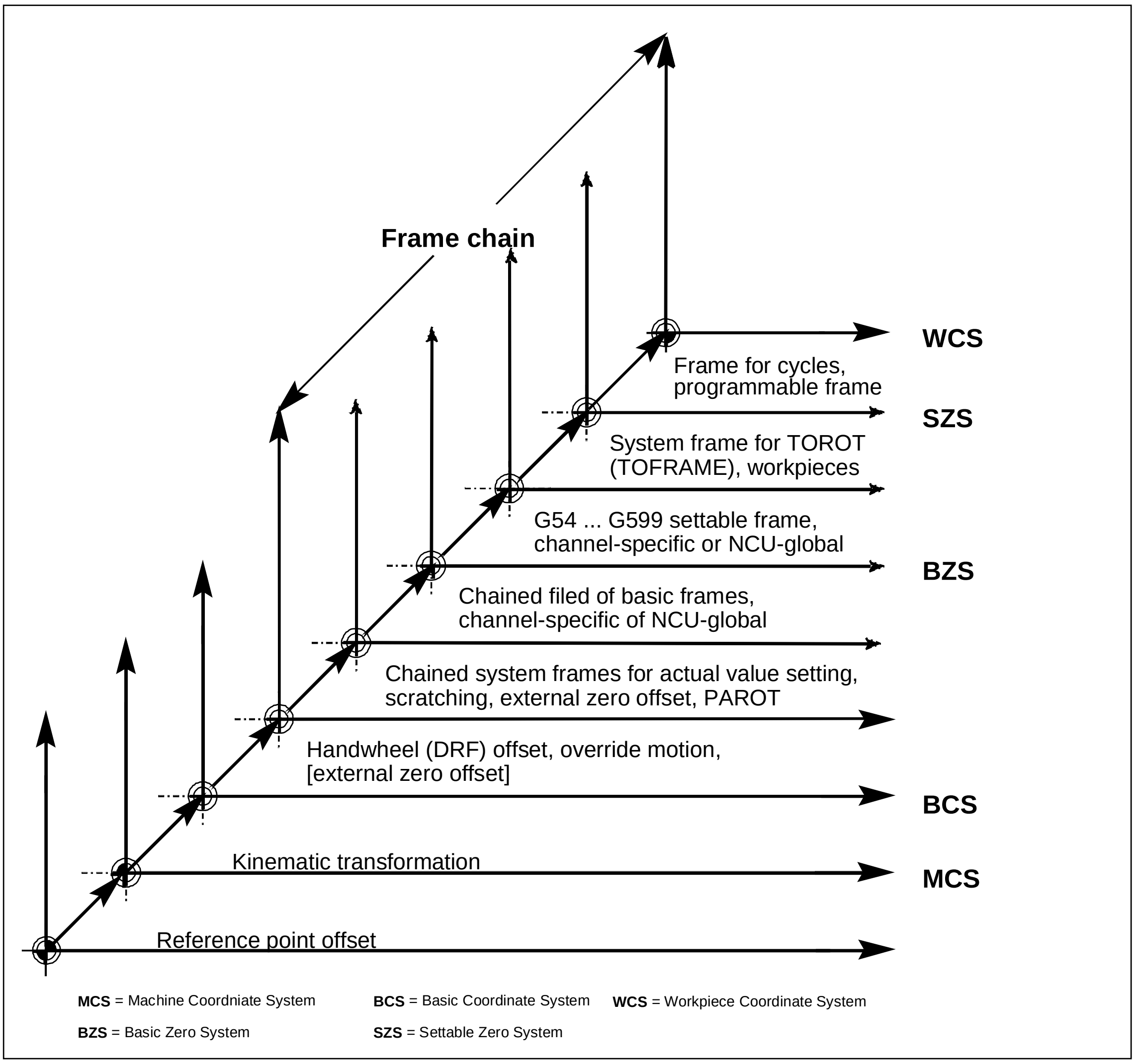
From SW 6.3, $P_ACTFRAME corresponds to
$P_SETFRAME : $P_EXTFRAME : $P_PARTFRAME : $P_ACTBFRAME :
$P_IFRAME : $P_TOOLFRAME : $P_WPFRAME : $P_PFRAME : $P_CYCFRAME

From SW 6.4, $P_ACTFRAME corresponds to
$P_PARTFRAME : $P_SETFRAME : $P_EXTFRAME : $P_ACTBFRAME : $P_IFRAME :
$P_TOOLFRAME : $P_WPFRAME : $P_TRAFRAME $P_PFRAME : $P_CYCFRAME



Frame chaining

The current frame consists of the total basic frame, the settable frame, the system frame, and the programmable frame according to the current total frame mentioned above.


Frame chain
WCS
Frame for cycles, programmable frame
SZS
System frame for TOROT (TOFRAME), workpieces
G54 ... G599 settable frame, channel-specific or NCU-global
BZS
Chained filed of basic frames, channel-specific of NCU-global
Chained system frames for actual value setting, scratching, external zero offset, PAROT
Handwheel (DRF) offset, override motion, [external zero offset]
BCS
Kinematic transformation
MCS
Reference point offset
MCS = Machine Coordniate System
BCS = Basic Coordinate System
WCS = Workpiece Coordinate System
BZS = Basic Zero System
SZS = Settable Zero System

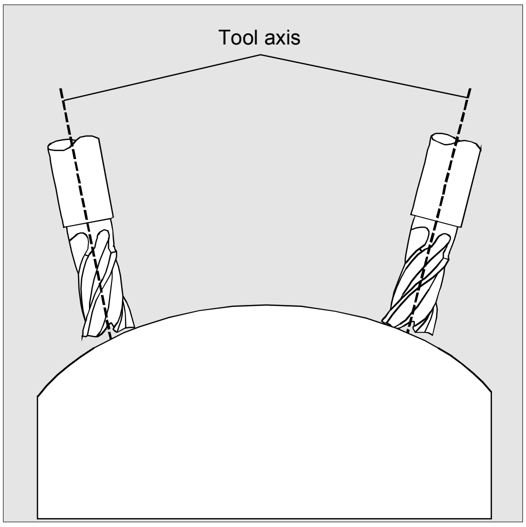
■

Notes

Transformations

7.1 Three, four and five axis transformation: TRAORI

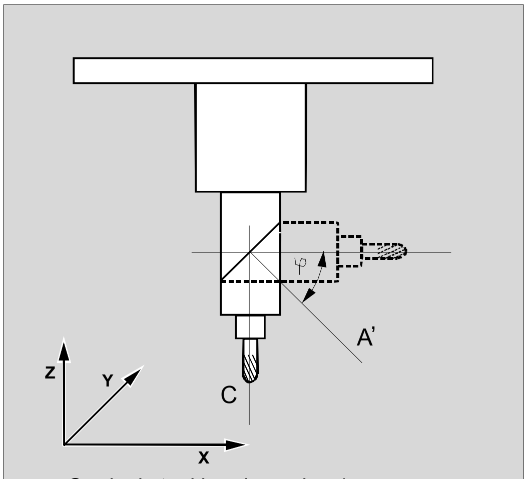
To obtain optimum cutting conditions when machining surfaces with a three-dimensional curve, it must be possible to vary the setting angle of the tool.



The machine design to achieve this is stored in the axis data.

Cardanic tool head
Three linear axes (X, Y, Z) and two orientation axes 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°.

The axis sequence of the orientation axes and the orientation direction of the tool are set up via the machine data subject to the machine kinematics. In the examples shown here, you can see the arrangements in the CA machine kinematics example!

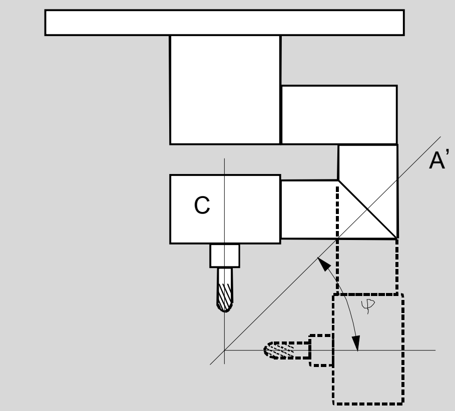


Cardanic tool head, version 1

There are the following possible relationships:
A' is below angle φ to the X axis
B' is below angle φ to the Y axis
C' is below angle φ to the Z axis

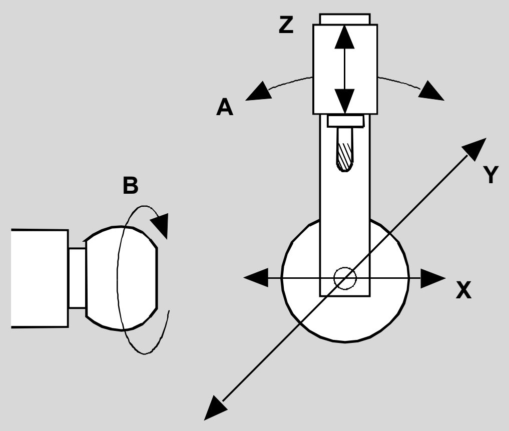
Angle φ can be configured in the range 0° to +89° via machine data.

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 compensation works in the direction of tool orientation.



Cardanic tool head, version 2

Transformation with a 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 are set up via the machine data subject to 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

3-axis and 4-axis transformations

3-axis and 4-axis transformations are special forms of 5-axis transformations.
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.

Configuration and programming for 3-axis and 4-axis transformations are the same as for 5-axis transformations.

Programming

```
TRAORI(n)
TRAORI(n,X,Y,Z,A,B)
TRAFOOF
```

Explanation of the commands

`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 rotary axes (SW 7.1 and higher)
`TRAFOOF`	Deactivate transformation

Other information

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 holder.

Examples of generic transformation:
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 position –45°)

Offset for orientation axes (SW 7.1 and higher)
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 one 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.

References

/FB/ F2, 3-axis to 5-axis transformations

7.1.1 Programming tool orientation

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.

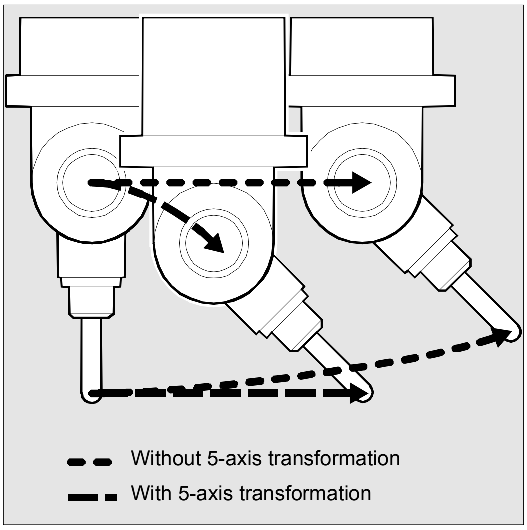
There are three options available when programming tool orientation:

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 angles or RPY angles via A2, B2, C2
   or
   Programming of the direction vector via A3, B3, C3. The direction vector points from the tool tip toward the tool holder.

3. Programming via the lead angle LEAD and the tilt angle TILT (face milling).

In all cases, orientation programming is only permissible if an orientation transformation is active.



Advantage: These programs can be transferred to any machine kinematics.

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
`LEA=D`	Lead angle for programming tool orientation
`TILT=`	Tilt angle for programming tool orientation

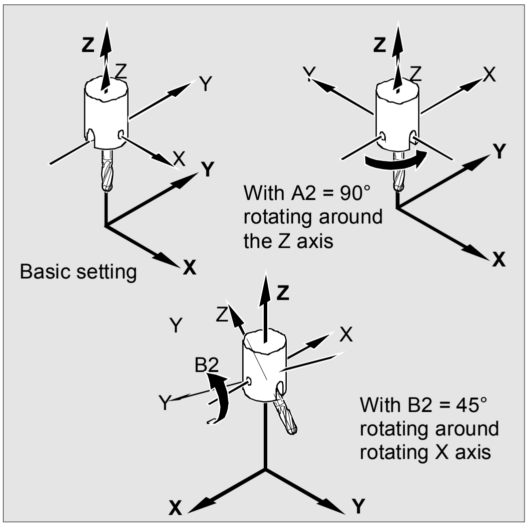
Machine data can be used to switch between Euler and RPY angles.

Programming in Euler angles

The values programmed during orientation programming with A2, B2, C2 are interpreted as Euler angles (in degrees).

The orientation vector results from turning a vector in the Z direction firstly with A2 around the Z axis, then with B2 around the new X axis and lastly with C2 around the new Z axis.

In this case the value of C2 (rotation around the new Z axis) is meaningless and does not have to be programmed.



Basic setting

With A2 = 90° rotating around the Z axis

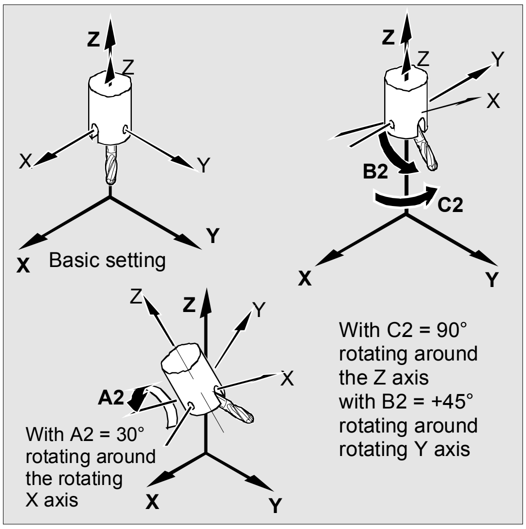
With B2 = 45° rotating around rotating X axis

Programming in RPY angles

The values programmed with A2, B2, C2 for orientation programming are interpreted as an RPY angle (in degrees).

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.

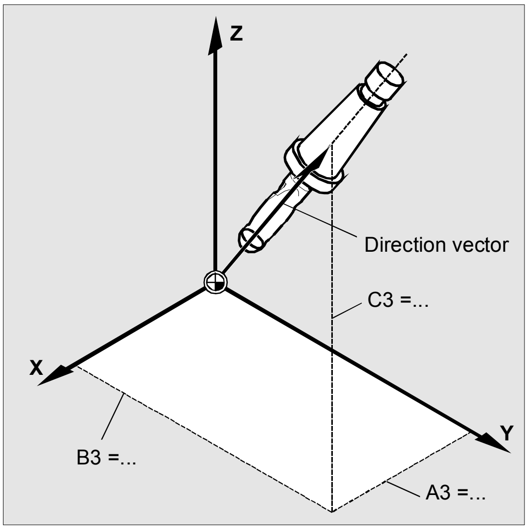
In contrast to Euler angle programming, all three values here have an effect on the orientation vector.


Z
Z
X
Y
X
Y
Basic setting
Z
Z
Y
X
B2
C2
X
Y
With C2 = 90° rotating around the Z axis with B2 = +45° rotating around rotating Y axis
Z
Z
Y
X
A2
With A2 = 30° rotating around the rotating X axis
X
Y

Programming the direction vector

The components of the direction vector are programmed with A3, B3, C3. The vector points towards the toolholder; the length of the vector is meaningless.

Vector components that have not been programmed are set equal to zero.



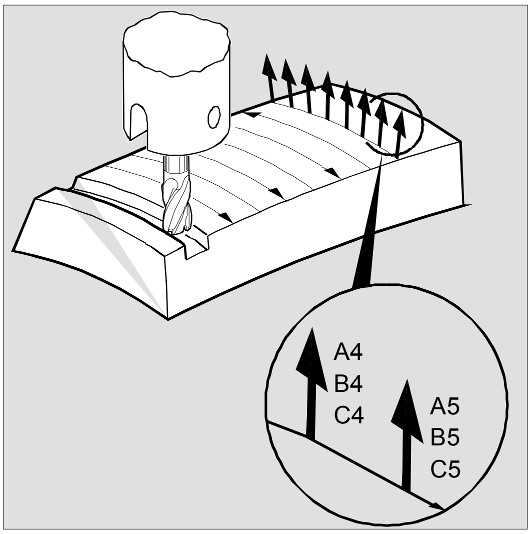
Face milling

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 that are normally performed in CAM.
The fully calculated NC blocks are then read into the control via postprocessors.

Surface description

The path curvature is described by surface normal vectors with the following components:
A4, B4, C4 start vector at block start
A5, B5, C5 end vector at block end



If a block only contains the start vector, the surface normal vector will remain constant throughout the block.

If a block only contains the end vector, interpolation will run from the end value of the previous block via large circle interpolation to the programmed end value.

If both start and end vectors are programmed, interpolation runs between the two directions, also via large circle interpolation. This allows continuously smooth paths to be created.

In the initial setting, surface normal vectors – whatever the active G17 to G19 level – point in the Z direction.

The length of a vector is meaningless.

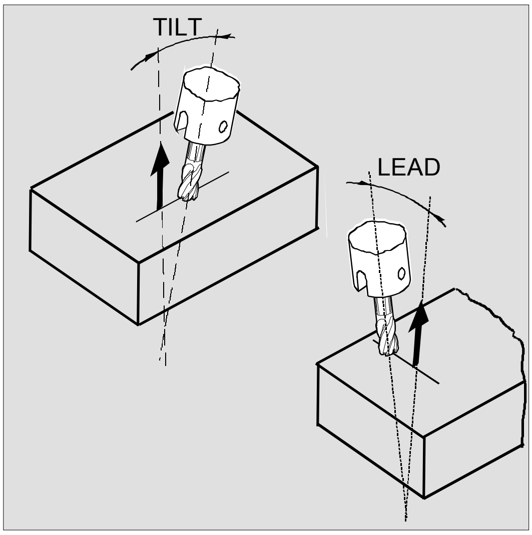
Vector components that have not been programmed are set to zero.
With active ORIWKS (see following pages), the surface normal vectors relate to the active frame and also turn when the frame is turned.

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.

Programming tool orientation with LEAD and TILT

The resultant tool orientation is determined from:

- – the path tangent
- – the surface normal vector
- – the lead angle LEAD
- – the tilt angle TILT at end of block



Explanation of the commands

`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

Behavior at inside corners (for 3D-tool compensation)

If the block at an inside corner is shortened, the resultant tool orientation is also achieved at end of block.

7.1.2 Rotating tool orientation: ORIROTA, ORIROTR, ORIROTT

Programming

`N.. ORIROTA`
or
`N.. ORIROTR`
or
`N.. ORIROTT`
	Define interpolation of the rotational vector (SW 6.1 and higher)

`THETA`	Activating rotation of the orientation vector
`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 rotational vector
`THETA=AC(....)`	Non-modal switchover to absolute dimensions.
`THETA=IC(....)`	Non-modal switchover to incremental dimensions.
PO[THETA]=(d_2, d_3, d_4, d_5)	Interpolate angle of rotation with a 5th order polynomial

Explanation of the commands

`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.
`THETA`	Rotation of the orientation vector
Θe	End angle of rotational vector both absolute with G90 and relative with G91 (incremental dimensioning) is active
`PO[THETA]=(....)`	Polynomial for angle of rotation

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 change in orientation vector.

Tool orientation with orientation aches is described in the following chapters.

Sequence

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.

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 with 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).

7.1.3 Orientation axis reference – ORIWKS, ORIMKS

Programming

```
N.. ORIMKS=
```
or
```
N.. ORIWKS=
```

Explanation of the commands

`ORIMKS`	Rotation in the machine coordinate system
`ORIWKS`	Rotation in the workpiece coordinate system

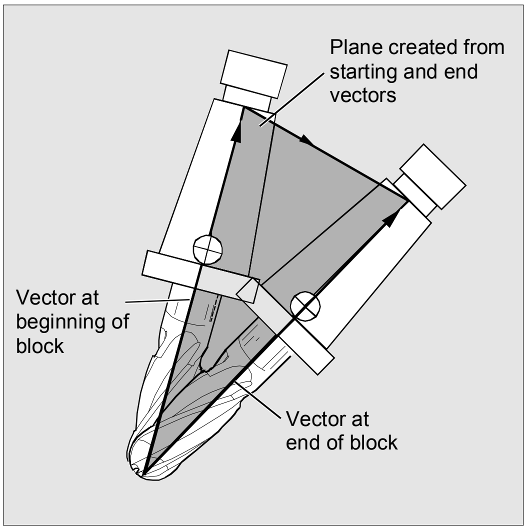
Function

With orientation programming in the workpiece coordinate system via Euler or RPY angles or the orientation vector, ORIMKS/ORIWKS can be used to adjust the course of the rotary motion.

Sequence

With ORIMCS, the movement executed by the tool is dependent 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 tool movement is not dependent 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.


Plane created from starting and end vectors
Vector at beginning of block
Vector at end of block

Other information

ORIWKS is the basic setting. If it is not immediately obvious with a 5-axis program which machine it should run on, 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.

Machine data $MC_ORI_IPO_WITH_G_CODE specifies the active interpolation mode: ORIMKS/ORIWKS or ORIMACHAX/ORIVIRTAX (see Subsection 7.1.4).

7.1.4 Singular positions and handling them

Notes on 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.)

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.

Note on SW 5.2:

As from SW5.2, singular positions will only be handled by MD `$MC_TRAFO5_POLE_LIMIT`
(see Description of Functions Part 3, Subsection 2.8.4).

7.1.5 Orientation axes (SW 5.2 and higher) ORIAXES, ORIVECT, ORIEULER, ORIRPY

Programming

```
N.. ORIAXES or ORIVECT              ; orientation interpolation:
N.. G1 X Y Z  A B C                 ; linear or large circle interpolation
N.. ORIEULER or ORIRPY              ; orientation programming:
N.. G1 X Y Z A2= B2= C2=            ; orientation angle via Euler/RPY
or
N.. ORIVIRT1 or ORIVIRT2            ; orientation angle via virtual
N.. G1 X Y Z A3= B3= C3=            ; orientation axes

N.. ORIPLANE                        ; orientation interpolation (from SW 5.3)
N.. ORICONCW or ORICONCCW           ; extended interpolation (from SW 6.1)
N.. ORICONTO or ORICURVE            ; on taper peripheral surface
N.. G1 X Y Z A6= B6= C6=
N.. ORICONIO
N.. G1 X Y Z A7= B7= C7=
```

Explanation of the commands

`ORIAXES`	Linear interpolation of machine or orientation axes
`ORIVECT`	Large-radius circular interpolation (identical to ORIPLANE)
`ORIMKS` `ORIWKS`	Rotation in the machine coordinate system Rotation in the workpiece coordinate system For description, see Subsection 7.1.2
`G1 X Y Z  A B C`	Programming the machine axis position
`ORIEULER`	Orientation programming via Euler angle
`ORIRPY`	Orientation programming on the basis of RPY angles
`G1 X Y Z A2= B2= C2=`	Angle programming of virtual axes

`ORIVIRT1`
`ORIVIRT2`
	Orientation programming using virtual orientation axes
	(definition 1), definition acc. to MD $MC_ORIAX_TURN_TAB_1
	(definition 2), definition acc. to MD $MC_ORIAX_TURN_TAB_2

`G1 X Y Z A3= B3= C3=`	Directional vector programming of directional axis
	Orientation interpolation (SW 5.3 and higher)
`ORIPLANE`	Interpolation in the plane (large circle interpolation)
`ORICONCW`	Interp. on a peripheral surface of the cone in clockwise direction
`ORICONCCW`	Interpolation on a peripheral surface of the cone in counterclockwise direction
`ORICONTO`	Interpolation a peripheral surface of cone with tangential transition
`G1 X Y Z A6= B6= C6=`	Programming of rotary axis of the cone (scaled vector)

`ORICONIO`
`G1 X Y Z A7= B7= C7=`
	Interpolation on a cone peripheral surface with intermediate orientation setting (programmed as scaled vector)

`ORICURVE`
`XH YH ZH with`
`polynomial`
`PO[XH]=(xe, x2, x3..)`
	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.

Function

The orientation axis function describes the orientation of the tool in space. This introduces an additional third degree of freedom that describes the rotation around itself. This is necessary for 6-axis transformations.

MD `$MC_ORI_DEF_WITH_G_CODE` specifies how the programmed angles A2, B2, C2 are defined:
The definition is made according to MD `$MC_ORIENTATION_IS_EULER` (default) or the definition is made according to 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
```

SW 6.3 and higher
In JOG mode, the Cartesian manual travel function can, for SINUMERIK 840D with the "Handling transformation package" and for Sinumerik 810D powerline from SW 6.1 set up separately the translation of the geometry axes in the reference systems MCS, WCS and TCS.

Feed programming

FORI1	Feedrate for swiveling the orientation vector on the large arc
FORI2	Feedrate for overlaid rotation about the swiveled orientation vector

With orientation movements, the programmable feed corresponds to an angular velocity [degrees/min].

Effectiveness of the feed via G code:

When programming ORIAXES, the feed for an orientation axis can be limited via the FL[ ] instruction (feed limit).

When programming ORIVECT, the feed must be programmed with FORI1 or FORI2. FORI1 and FORI2 must only be programmed once in the NC block. Traversal always takes the shortest path during this programming.
The lowest of the two feedrates is applied for overlaid turning and swiveling motions. With orientation movements, the feed corresponds to an angular velocity [degrees/min].

If geometry axes and orientation axes traverse a common path, the traversing movement is determined from the smallest feed.

Extended interpolation SW 6.1 and higher
With extended orientation it is possible to execute a change in orientation along the peripheral surface of a cone in space. The program must include the following:

- Initial orientation with preceding block
- Final orientation either with directional vector with A3, B3, C3 or in Euler angle or RPY angle with A2, B2, C2.
- Angle of rotation of cone as a vector with A6, B6, C6
- Aperture angle PSI with identifier NUT
- Intermediate orientation: A7, B7, C7

Reference notes:
SINUMERIK 840D/FM-NC Description of Functions (Part 3), "Handling transformation package".
/FB/ F2, 3-axis to 5-axis transformations

7.1.6 Cartesian PTP travel (SW 5.2 and higher)

Programming

```
N.. TRAORI
N.. STAT=`B10` TU=`B100` PTP
N.. CP
```

Explanation of the commands

`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 This makes it possible to clearly approach axis angles between –360 degrees and +360 degrees.

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.

Sequence

The commands PTP and CP effect the changeover between Cartesian traversal and traversing the machine axes. These are modal. CP is the default setting.

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 by 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.

Reference notes:
The various transformations are included in the document:
SINUMERIK 840D/FM-NC Description of Functions (Part 3), "Handling transformation package".

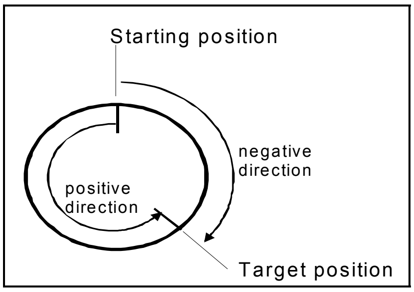
The positional bits to be programmed for "STAT" are included in the document:
SINUMERIK 840D/FM-NC Description of Functions (Part 3), "3-axis to 5-axis transformation".

Programming the axis angle (TU=)
To be able to clearly approach by axis angles < ±360 degrees, this information must be programmed using the command "TU=".
The command is non-modal.

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



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.

Other information

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_RESET_VALUES[48]. The default traversal mode setting is "CP".

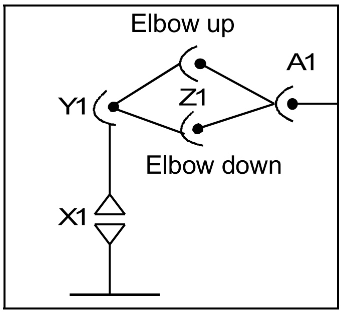
Repos

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 motion, there must be no overrides in the BCS.

Programming 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`	

7.1.7 Online tool length offset: TOFFON, TOFFOF (SW 6.4 and higher)

Programming

```
N.. TRAORI
N.. TOFFON(X,25)
N.. WHEN TRUE DO $AA_TOFF[X]
```

Explanation of the commands

`TOFFON`	**T**ool **Off**set **ON** (activate online tool length compensation) When activating, an offset value can be specified for the relevant direction of compensation and this is immediately recovered.
`TOFFOF`	**T**ool **Off**set **OF** (reset online tool length compensation) The relevant compensation values are reset and a preprocessing stop is initiated.
`X, Y, Z`	Direction of compensation for the specified offset value

Note:
The online tool length compensation is an **option** and must be enabled beforehand. This function is only practical in conjunction with an active orientation transformation or an active orientable toolholder.

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.

Application
The online tool length compensation function can be used for:

- orientation transformation TRAORI
- orientable toolholder TCARR

Other information

Block preparation

During block preparation in preprocessing, the current tool length offset active in the main run is also taken into consideration. To allow extensive use to be made of the maximum permissible axis velocity, it is necessary to stop block preparation with a STOPRE preprocessing stop while a tool offset is 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 size of the difference between the current compensation active in the interpolator and the compensation active at the time the block was prepared, can be queried in the $AA_TOFF_PREP_DIFF[ ] variable.

Adjusting machine data and setting data

The following machine data is available for online tool length compensation:

- MD 20610: ADD_MOVE_ACCEL_RESERVE reserved for velocity planning
- 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 compensation
- MD 21196: TOFF_ACCEL acceleration of online tool length compensation
- Setting data for presetting limit values
  SD 42970: TOFF_LIMIT upper limit of tool length compensation value

Programming example

Tool length compensation 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
...
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
      G4 F5                              ; offset is executed subject to the
                                         ; position of axis X2
...
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
```

Tool length compensation 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
```

References

/FB/ F2, 3-axis to 5-axis transformations

7.2 Milling on turned parts: TRANSMIT

Programming

`TRANSMIT` or `TRANSMIT(n)`
`TRAFOOF`

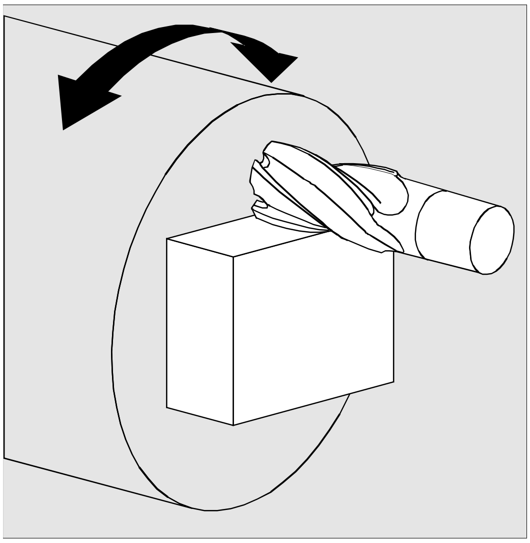
Explanation of the commands

`TRANSMIT`	Activates the first declared TRANSMIT function
`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

An active TRANSMIT transformation is also disabled if one of the remaining transformations is activated in the particular channel (e.g. TRACYL, TRAANG, TRAORI).

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 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 perpendicular to rotary axis
  The linear axes are 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.



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.

Pole

Up to **SW 3.x**

Movements through the pole (origin of Cartesian coordinate system) are disabled, i.e. a movement which traverses the pole is stopped in the pole followed by the output of an alarm. In the case of a cutter center offset, the movement is terminated accordingly at the end of the non-approachable area.

SW 4 and higher

There are two options for traversing through the pole:

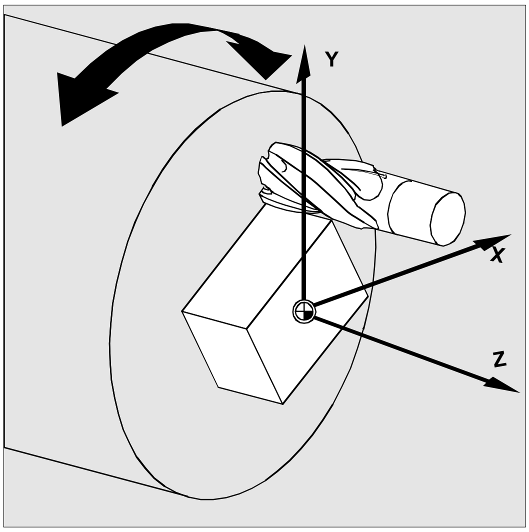
1. Traversal along linear axis
2. Traverse to the pole, rotate the rotary axis at the pole and traveling away from the pole

Make the selection using MD 24911 and 24951.

References

/FB/ M1 Kinematic transformations

Programming 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`
`N50 ATRANS X-2 Y10`
	Set frame

`N60 G1 X10 Y-10 G41 OFFN=1`
`N70 X-10`
`N80 Y10`
`N90 X10`
`N100 Y-10`
	Square roughing; allowance 1 mm

`N110 G0 Z20 G40 OFFN=0`
`N120 T2 D1 X15 Y-15`
`N130 Z10 G41`
	Tool change

`N140 G1 X10 Y-10`
`N150 X-10`
`N160 Y10`
`N170 X10`
`N180 Y-10`
	Square finishing

`N190 Z20 G40`
`N200 TRANS`
`N210 TRAFOOF`
	Deselect frame

`N220 G0 X20 Z10 SPOS=45`	Approach start position
`N230 M30`	

7.3 Cylinder surface transformation: TRACYL

Programming

`TRACYL(d)` or `TRACYL(d,t)`
`TRAFOOF`

Explanation of the commands

`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.
`TRAFOOF`	Transformation OFF (BCS and MCS are once again identical).
`OFFN`	Offset contour normal: Distance of the groove side from the programmed reference contour

An active TRACYL transformation is likewise deactivated if one of the other transformations is activated in the relevant channel (e.g. TRANSMIT, TRAANG, TRAORI).

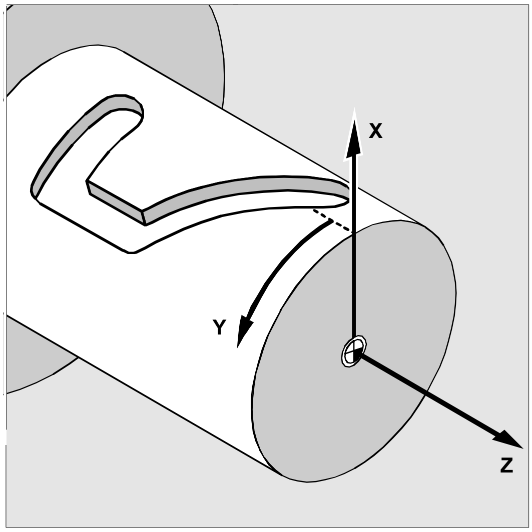
Function

Cylinder surface transformation TRACYL
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.



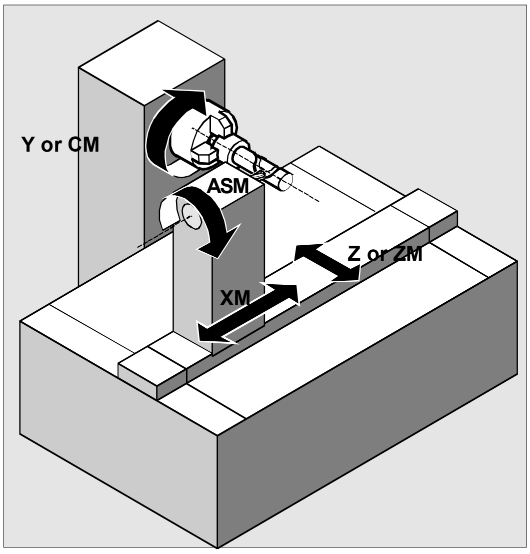
Workpiece coordinate system

There are two instances of cylinder surface coordinate transformation:

- without groove side offset (TRAFO_TYPE_n=512)
- with groove side offset (TRAFO_TYPE_n=513)

Without groove side offset:
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 perpendicular to rotary axis

The linear axes are positioned perpendicular to one another. The infeed axis cuts the rotary axis.

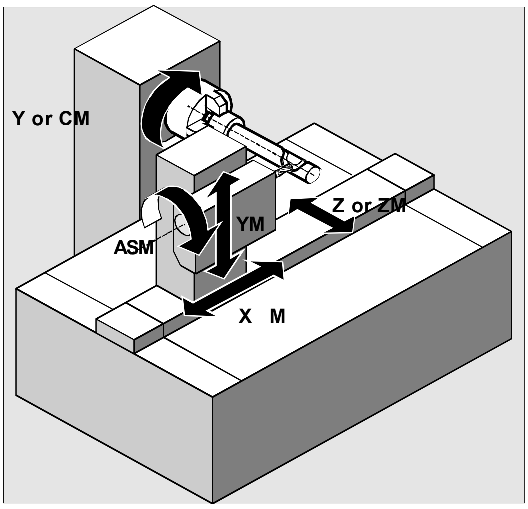


Machine coordinate system

With groove side offset:
Kinematics as above, but in addition
– 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.

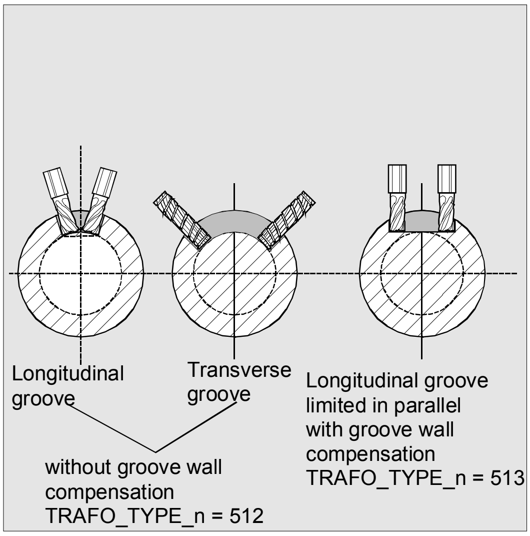


Machine coordinate system

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 groove
Transverse groove
without groove wall compensation TRAFO_TYPE_n = 512
Longitudinal groove limited in parallel with groove wall compensation TRAFO_TYPE_n = 513

Offset contour normal OFFN (513)

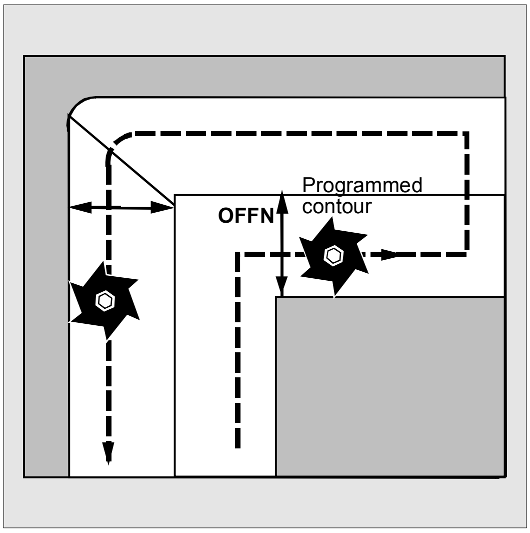
To mill grooves with TRACYL, the following is programmed:

- groove center line in the part program,
- **half the groove width** via OFFN.

OFFN is does not become effective until tool radius correction is selected to avoid damaging the groove wall. **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 situations:

- 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 acts 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.

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.

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.

Axis utilization

The following axes cannot be used as a positioning axis or a reciprocating axis:

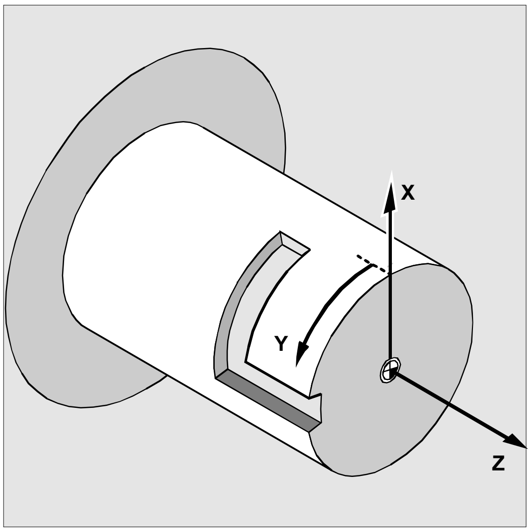
- the geometry axis in the peripheral direction of the cylinder peripheral surface (Y axis)
- the additional linear axis for groove side compensation (Z axis).

Tool definition

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 compensation**	
`$TC_DP3[1,1]=8.`	Length offset vector	Calculation acc. to type
`$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`		
`$TC_DP11[1,1]=0`	Angle for cone milling tools	
Wear	**Tool length and radius compensation**	
`$TC_DP12[1,1]=0`	Remaining parameters to $TC_DP24=0	Base dimensions/ adapter

Programming example



`N10 T1 D1 G54 G90 F5000 G94`	Tool selection, clamping compensation

`N20 SPOS=0`
`N30 G0 X25 Y0 Z105 CC=200`
	Approach start position

`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`	

7.4 Inclined axis: TRAANG

Programming

`TRAANG(α)` or `TRAANG(α,n)`
`TRAFOOF`

Explanation of the commands

`TRAANG`	If angle α is omitted or zero is entered, the transformation is activated with the parameterization of the previous selection. On the first selection, the presettings according to the machine data apply.
`TRAANG(α)`	Activates the first specified inclined axis transformation
`TRAANG(α,n)`	Activates the n-th specified transformation Oblique axis, n must not be more than 2. TRAANG(α,1) corresponds to TRAANG(α).
α	Angle of the inclined axis
`TRAFOOF`	Transformation off

If α (angle) is omitted or zero is entered, the transformation is activated with the parameterization of the previous selection. On the first selection, the presettings according to the machine data apply. (response up to SW < 6.4, for later versions, see below).

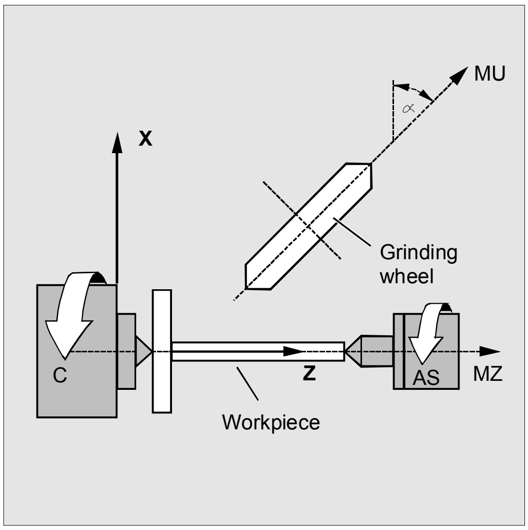
An active TRAANG transformation is also disabled if one of the remaining transformations is activated in the particular channel.
(e.g. TRACYL, TRANSMIT, TRAORI).

(response from SW < 6.4)
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 presettings according to the machine data apply.
An angle α = 0 (e.g. TRAANG(0), TRAANG(0,n)) is a valid parameter setting and no longer corresponds to omitting the parameter as it did in former versions.
Permissible values for α are:
-90 degrees < α < + 90 degrees

Function

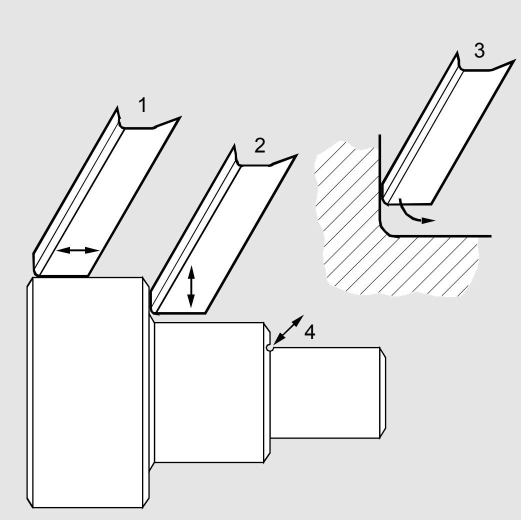
The inclined axis function is intended for grinding technology and facilitates the following performance:

- 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.



The following machining operations are possible:

1. Longitudinal grinding
2. Face grinding
3. Grinding of a specific contour
4. Oblique plunge-cut grinding.



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 velocity 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.

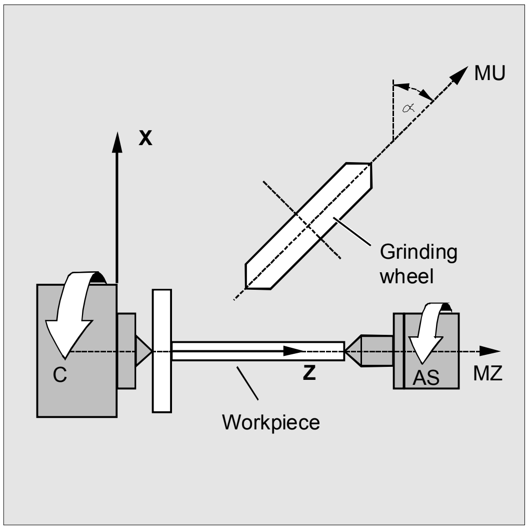
Axis configuration

To be able to program in the Cartesian coordinate system, the control must be told the relationship 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.

Programming example



`N10 G0 G90 Z0 MU=10 G54 F5000 ->`
`-> G18 G64 T1 D1`
	Tool selection, clamping compensation
	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 ->`
`-> OST2[Z]=-2 FA[Z]=5000`
`N60 OS[Z]=1`
`N70 POS[X]=4.5 FA[X]=50`
`N80 OS[Z]=0`
	Reciprocation, until dimension reached
	(for reciprocation, see chapter 9)

`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

7.4.1 Programming an inclined axis: G05, G07 (SW 5.3 and higher)

Programming

```
G07
G05
```

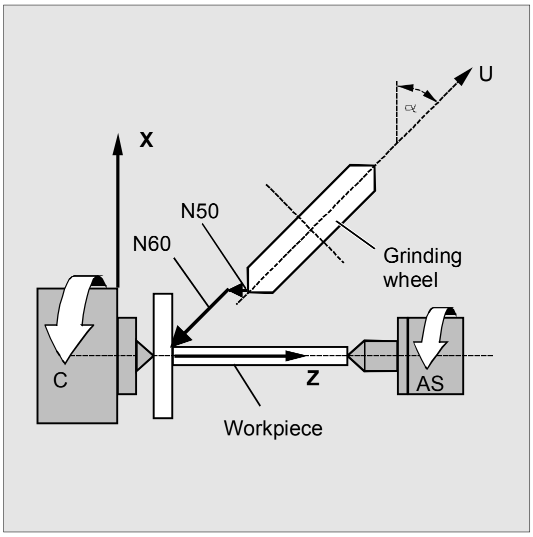
Explanation of the commands

`G07`	Approach starting position
`G05`	Activates oblique plunge-cutting

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.
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/ F2: 3-5 axis transformation, Chapter 2 "Cartesian PTP Traversing".

Programming example



`N..`	Program angle for inclined axis
`N20 G07 X70 Z40 F4000`	Approach starting position
`N30 G05 X70 F100`	Oblique plunge-cutting
`N40 ...`	

7.5 Constraints when selecting a transformation

Transformations can be selected via a parts program or MDA. Please note the following

- 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 compensation 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.
- DRF offsets in the axes involved in the transformation must not change between processing in preprocessing and in main run (SW 3 and earlier).
- 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 of tool length compensation and a tool radius compensation selection/deselection must not be programmed in the same block.

Frame change
All instructions 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).

7.6 Deselecting a transformation: TRAFOOF

Programming

```
TRAFOOF
```

Explanation of the commands

`TRAFOOF`	Disables all the active transformations/frames

Function

The TRAFOOF command disables all the active transformations and frames.

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 previous section "Constraints when selecting a transformation").

7.7 Chained transformations

In SW 5 and higher **two** transformations can be chained such that the motion parts for the axes from the first transformation are input data for the chained second transformation. The motion parts from the second transformation act on the machine axes.

- In SW 5 the chain may encompass **two** transformations.
- The **second** transformation must be "**inclined axis**" (TRAANG).
- The first transformation can be:
  - orientation transformations (TRAORI),
    incl. universal milling head
  - TRANSMIT
  - TRACYL
  - TRAANG

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.

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.

Other information

Information on configuring the machine data of the transformations can be found in the descriptions of the functions: M1 and F2.

Machine manufacturer (MH7.1)

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.

The commands available for chained transformations are:
TRACON for activating and
TRAFOOF for deactivating it.

Switch on

Programming

`TRACON(trf, par)`	This activates a chained transformation.

Explanation of parameters

`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 defaults are to act 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).

Function

This activates the chained transformation. If another transformation was previously activated, it is implicitly disabled by means of TRACON().

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.

Switch off

Programming

```
TRAFOOF
```

Function

The command deactivates the last active (chained) transformation.

7.8 Replaceable geometry axes, GEOAX

Programming

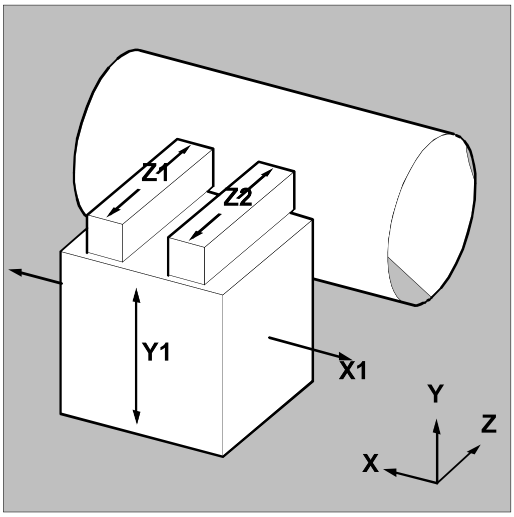
```
GEOAX(n,channel axis,n,channel axis,…)

GEOAX()
```

Explanation of the 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.

Function

The "switchable 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.



Example:
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

Sequence

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, ZO:
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 second and ZZ the third geometry axis,

```
N... GEOAX (1, U, 2, V, 3, W)
```

Channel axis U is the first, V the second and W the third geometry axis.

Prerequisites and restrictions

1. It is not possible to switch the geometry axes over during:
   - an active transformation,
   - an active spline interpolation,
   - an active tool radius compensation, (Programming Guide Fundamentals: Ch. 8)
   - an active fine tool compensation (Programming Guide Fundamentals: Ch. 8)
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.

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.

Other information

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.

Programming example

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.

■

Notes

Tool Offsets

8.1 Offset memory

Structure of the offset memory
Every data field can be invoked with a T and D number (except "Flat D No."); in addition to the geometrical data for the tool, it contains other information such as the tool type.

SW 4 and higher
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.

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.

The individual values of the offset memories P1 to P25 can be read from and written to the program via system variable.

Tool parameters Number (DP)	Meaning of system variables	Remarks
`$TC_DP 1`	Tool type	For overview see list
`$TC_DP 2`	Tool point direction	only for turning tools
Geometry	**Length compensation**	
`$TC_DP 3`	Length 1	Calculation acc. to type
`$TC_DP 4`	Length 2	and plane
`$TC_DP 5`	Length 3	
Geometry	**Radius**	
`$TC_DP 6`	Radius	
`$TC_DP 7`	Slot width b for slotting saw, rounding radius for milling tools	
`$TC_DP 8`	Projection k	For slotting saw only
`$TC_DP 11`	Angle for cone milling tools	
Wear	**Tool length and radius compensation**	
`$TC_DP 12`	Length 1	
`$TC_DP 13`	Length 2	
`$TC_DP 14`	Length 3	
`$TC_DP 15`	Radius	
`$TC_DP 16`	Slot width b for slotting saw, rounding radius for milling tools	
`$TC_DP 17`	Projection k	For slotting saw only
`$TC_DP 20`	Angle for cone milling tools	
Base dimensions/ adapter	**Length offsets**	
`$TC_DP 21`	Length 1	
`$TC_DP 22`	Length 2	
`$TC_DP 23`	Length 3	
Technology		
`$TC_DP 24`	Clearance angle	For turning tools

Other information

All other parameters are reserved.

Machine manufacturer

User cutting edge data can be configured via MD.

8.2 Language commands for tool management

Explanation of the commands

`T="MYTOOL"`	Select tool with name
`NEWT ("MYTOOL",DUPLO_NO)`	Create new tool, duplo number optional
`DELT ("MYTOOL",DUPLO_NO)`	Delete tool, duplo number optional
`GETT ("MYTOOL",DUPLO_NO)`	Determine T number
`SETPIECE (x,y)`	Set piece number
`GETSELT (x)`	Read preselected tool number (T No.)
`"WZ"`	Tool identifier
`DUPLO_NR`	Number of workpieces
`x`	Spindle number, entry optional

If you use the tool manager you can create and call tools by name, e.g. T="DRILL" or T="123".

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.

```
Return parameter=NEWT("WZ", DUPLO_NO)
```

If no duplo number is specified, this is generated automatically by the tool manager.

Example:

`DEF INT DUPLO_NO`	
`DEF INT T_NO`	
`DUPLO_NO = 7`	
`T_NO=NEWT("DRILL", DUPLO_NO)`	Create new tool "DRILL" with duplo number 7. The T number created is stored in T_NO.

DELT function

The DELT function can be used to delete a tool without referring to the T number.

```
DELT("MYTOOL",DUPLO_NR)
```

GETT function

The GETT function returns the T number required to set the tool data for a tool known only by its name.

```
Return parameter=GETT("MYTOOL", DUPLO_NO)
```

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.

Examples:

`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

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)

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 for tool change with tool management

T1 Tool preselection; i.e.; the tool magazine can be put in tool position in parallel with machining.

M6 Changing to a preselected tool (depending on default setting in the machine data it may also be programmed without M6).

Example:

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

A complete list of all variables required for tool management is given in the list of system variables in the Appendix.

8.3 Online tool offset PUTFTOCF, PUTFTOC, FTOCON, FTOCOF

Programming

```
FCTDEF(Polynomial_no., LLimit, ULimit,a0,a1,a2,a3)
PUTFTOCF (Polynomial_no., Ref_value, Length1_2_3, Channel, Spindle)
PUTFTOC (Value, Length1_2_3, Channel, Spindle)
FTOCON
FTOCOF
```

Explanation of the commands

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

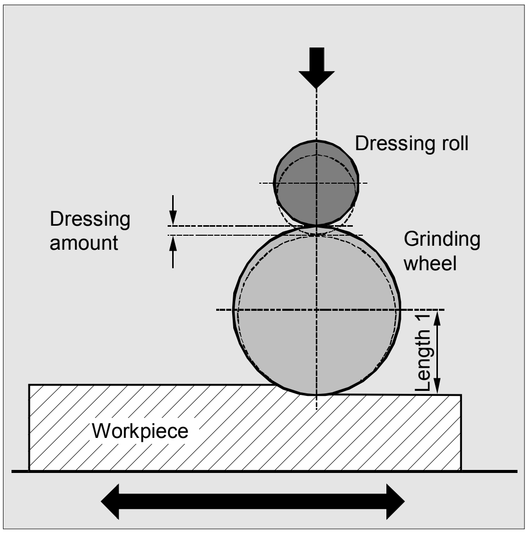
Explanation of the parameters

Polynomial_No.	Values 1 to 3: up to 3 polynomials are possible at one time; polynomial up to 3 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

8.3 Online tool offset PUTFTOCF, PUTFTOC, FTOCON, FTOCOF

Function

The function makes immediate allowance for tool offsets resulting from machining by means of online tool length compensation (e. g. CD dressing: The grinding wheel is dressed parallel to machining). The tool length compensation can be changed from the machining channel or a parallel channel (dresser channel).



Online tool offset can be applied only to grinding tools.

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 with reference to the tool data with GWPSON or TMON as long as it is not the active grinding wheel (see Programming Guide "Fundamentals").
An offset is always applied for the wear parameters for the current tool side or for the left-hand tool side on inactive tools.

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) and centerless grinding (CLGON).

Sequence

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 degree, 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

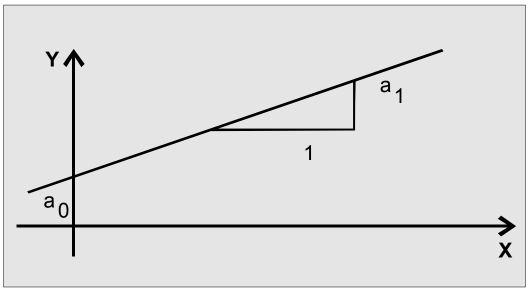
The parameters are defined in a separate block:

FCTDEF(Polynomial_no., LLimit, ULimit,a_0,a_1,a_2,a_3)

The polynomial can be a 1st, 2nd or 3rd degree 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

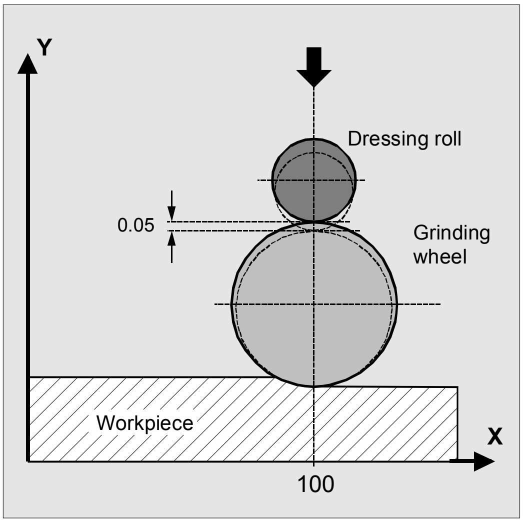
The target channel can only receive online tool offsets when FTOCON is active.

- 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 on 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.

Programming example

Task

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	Continue machining
N…M30	

Dressing program in channel 2:

%_N_DRESS_MPF	
…	
N40 FCTDEF (1, -1000, 1000, -$AA_IW[V], 1)	Define function: Straight
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
...	

8.4 Maintain tool radius compensation at constant level, CUTCONON (SW 4 and higher)

Programming

```
CUTCONON
CUTCONOF
```

Explanation

`CUTCONON`	Activate the tool radius compensation constant function
`CUTCONOF`	Deactivate the constant function (default setting)

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).

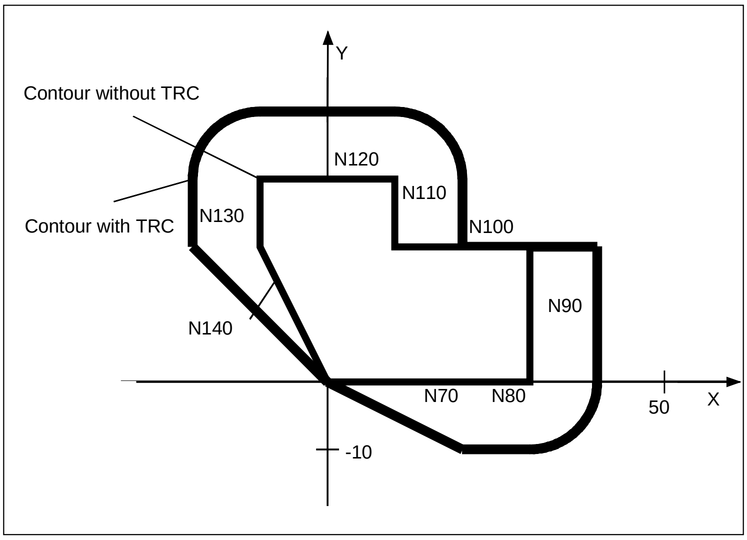
Sequence

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.

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`	



Other information

1. If tool radius compensation active (G40), CUTCONON has no effect. No alarm is produced. The G code remains active, however. This is significant when tool radius compensation is to be activated in a later block with G41 or G42.
2. It is permissible to change the G code in the 7th G code group (tool radius compensation; G40 / G41 / G42) with active CUTCONON. A change to G40 is active immediately.
   The offset used for traversing the previous blocks is traveled.
3. If CUTCONON or CUTCONOF is programmed without traversing in the active compensation plane, the effect is delayed until the next block that has such a traversing motion.

Further information: /FB/, W1 Tool Offset

8.5 Activate 3D tool offsets CUT3DC, CUT3DF, CUT3DFS/CUT3DFF

Explanation

`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).
`CUT3DCC`	3D tool offset for circumferential milling with limitation surfaces.
`CUT3DCCD`	3D tool offset for circumferential milling with limitation surfaces with a differential tool.
`G40 X Y Z`	To deactivate: Linear block G0/G1 with geometry axes
`ISD=Value`	Insertion depth

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.

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 response is always controlled with NORM.

Example

`N10 A0 B0 X0 Y0 Z0 F5000`	
`N20 T1 D1`	Tool call, call tool offset values
`N30 TRAORI(1)`	Transformation selection
`N40 CUT3DC`	3D tool radius compensation selection
`N50 G42 X10 Y10`	Tool radius compensation selection
`N60 X60`	
`N70 …`	

Other information

Intermediate blocks are permitted with 3D tool radius compensation. The definitions for the 2 1/2D tool radius compensation apply.

3D tool radius compensation is only active when five-axis transformation is selected.

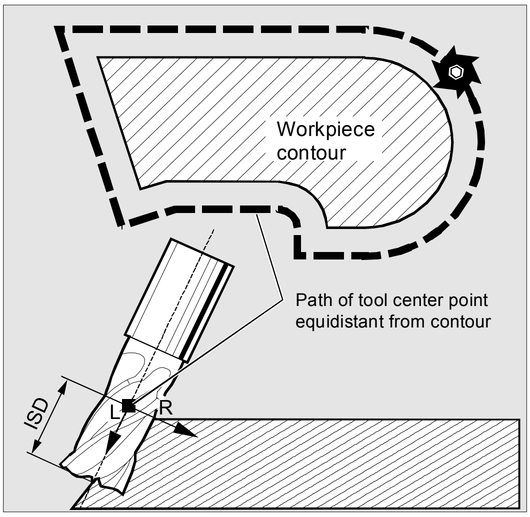
A circle block is always inserted at outside corners. G450/G451 have no effect.

The command DISC is not evaluated.

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.

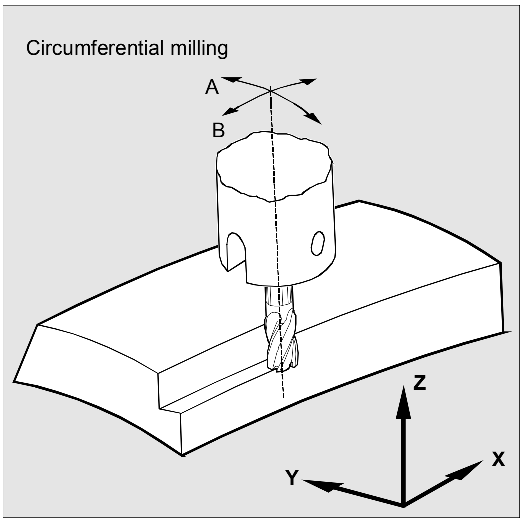


3D tool radius compensation is also called 5D tool radius compensation, because in this case 5 degrees of freedom are available for the orientation of the tool in space.

8.5.1 3D tool radius compensation: Circumferential milling, face milling

Circumferential milling

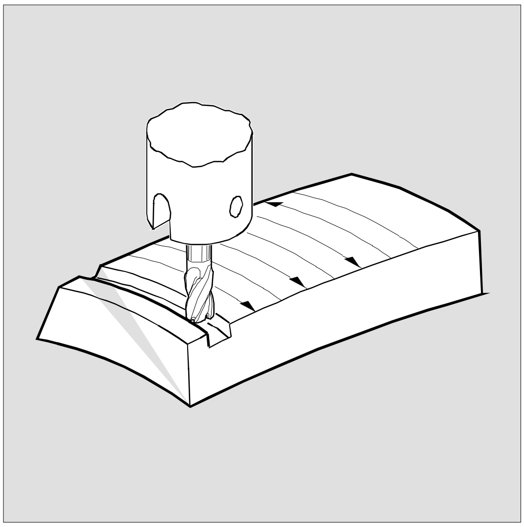
The type of milling used here is implemented by defining a path (guide line) and the corresponding orientation. In this type of machining, the shape of the tool on the path is not relevant. The only decisive factor is the radius at the tool contact point.



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.

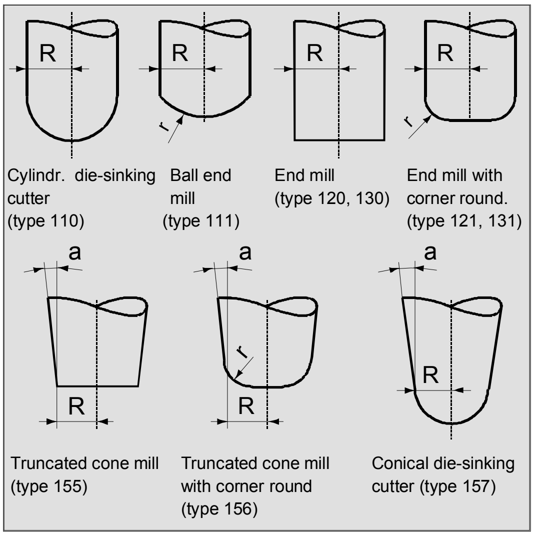
8.5.2 Tool types/tool change with changed dimensions G40/41/42

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 155 are identical in their effect.

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.


R
R
R
R
r
r
Cylindr. die-sinking cutter (type 110)
Ball end mill (type 111)
End mill (type 120, 130)
End mill with corner round. (type 121, 131)
a
a
a
r
R
R
R
Truncated cone mill (type 155)
Truncated cone mill with corner round (type 156)
Conical die-sinking cutter (type 157)

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
Truncated cone mill	155	>0	X	>0

Tool data	Tool parameters		
Tool dimensions	Geometry	Wear	X = is not evaluated
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 compensation

The tool tip is the reference point for length compensation (intersection longitudinal axis/surface).

3D tool offset, tool change

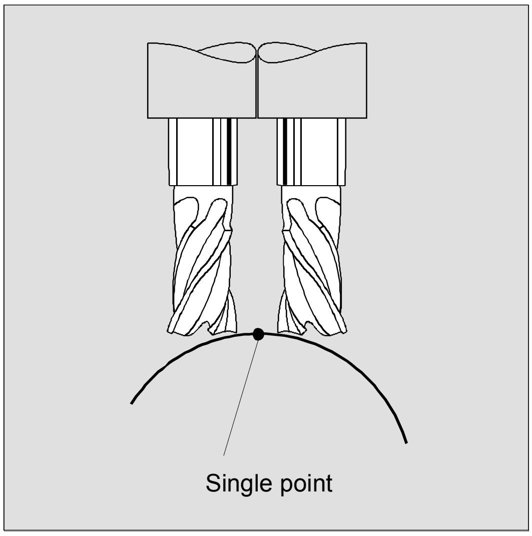
A new tool with changed dimensions (R, r, a) or a different shape may be specified only through programming G41 or G42 (transition G40 to G41 or G42, reprogramming of G41 of G42).

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.

8.5.3 Compensation on the path, path curvature, and insertion depth ISD

Compensation on path

With respect to face milling, it is advisable to examine what happens when the contact point "jumps" on the tool surface as shown in the example on the right where a convex surface is being machined with a vertically positioned tool.
The application shown in the example should be regarded as a borderline case.



This borderline case is monitored by the control that detects abrupt changes in the machining point on the basis of angular approach motions between the tool and normal surface vectors. The control inserts linear blocks at these positions so that the motion can be executed.

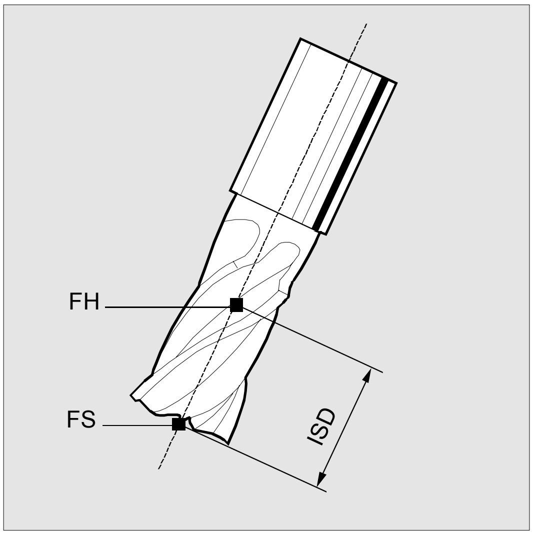
These linear blocks are calculated on the basis of permissible angular ranges for the side angle stored in the machine data.
The system outputs an alarm if the limit values stored in the machine data are violated.

Path curvature

Path curvature is not monitored. In such cases, it is also advisable to use only tools of a type that do not violate the contour.

Insertion depth ISD

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.

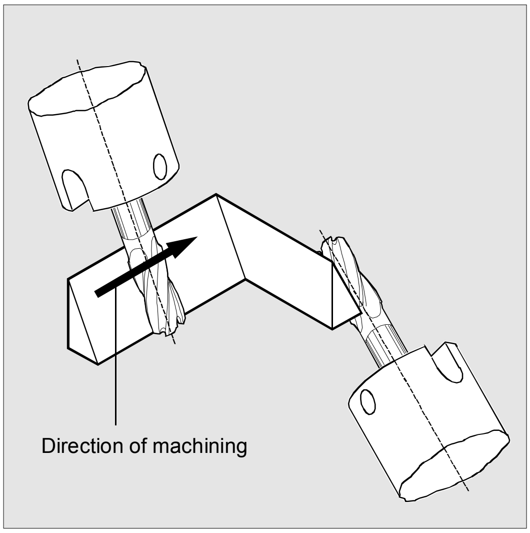


ISD specifies the distance between the cutter tip (FS) and the cutter reference point (FH). Point FH is obtained by projecting the programmed machining point onto the tool axis. ISD is only evaluated when 3D tool radius compensation is active.

8.5.4 Inside corners/outside corners and intersection procedure, G450/G451 (SW 5 and higher)

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.



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.
As from SW 5, the intersection procedure is especially advantageous for 3D programs typically generated by CAD. They often consist of short straight blocks (to approximate smooth curves), where the transitions are almost tangential between adjacent blocks.
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.

For more information about intersection procedure, see
/FB/ W5, 3D Tool Radius Compensation

8.5.5 3D circumferential milling with limitation surfaces, CUT3DCC, CUT3DCCD

Function

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 part 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 part program or
- while determining the real contour, e.g. by tool infeed.

In addition to the typical application case, those cases are dealt with in which the standard tool does not describe the center point path but the contour on the machining surface. In this case, the limitation surface depends on the tool. Like with conventional tool radius compensation, the entire radius if used to calculate the perpendicular offset to the limitation surface.

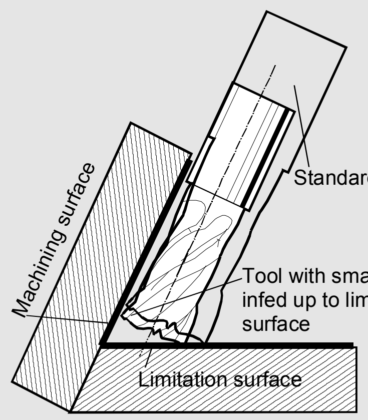
3D radius compensation with standard tools:

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.

Pocket milling with oblique side walls with 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.

For more information on collision monitoring, see /PG/ Fundamentals, "Tool Offsets"

Tool center point path with infeed up to the limitation surface CUT3DCCD: (SW 6.4 and higher)
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.

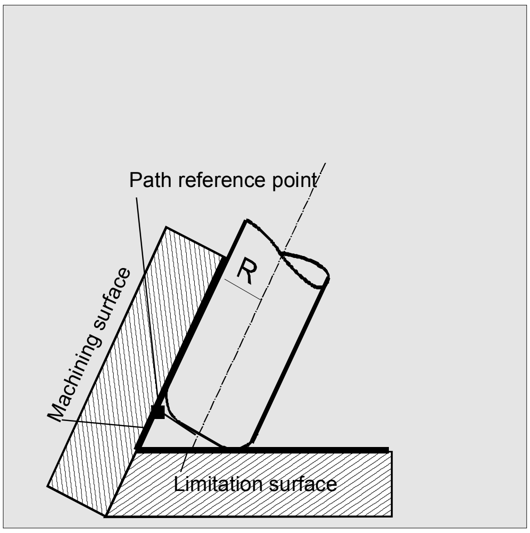


Using cylindrical tools

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: (SW 6.4 and higher) 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 illustration in the right 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.

Standard tools with corner rounding

Corner rounding with a standard tool is described by 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.	both and $TC_TP15 + OFFN $TC_TP16 are negative	
Only milling cutter types with cylindrical shank (cylindrical or end mill) or toroidal millers are permitted (type 121 and 131) and in borderline cases, the cylindrical die-sinking cutter (type 110) Up to SW 6.4 the **following** milling tools are not permitted: with tapered shank (type 155 - 157) with cylindrical shank and rounded tip (type 111)	On these permitted miller types the corner radius r is equal to shank radius R. All other permitted tool types are interpreted as cylindrical cutters and the dimensions specified for the corner rounding are not evaluated.	

The following tools types are permitted:
Numbers 1 – 399 with the exception of numbers 111 and 155 to 157.

Sequence

Tool center point path with infeed up to the limitation surface CUT3DCCD: (SW 6.4 and higher)

Unlike all other tool compensations of G code group 22, tool parameter $TC_DP6 does not affect the tool radius and the resulting compensation. The compensation is the sum of

- the wear value
  (tool parameter $TC_DP15)

and a

- programmable offset OFFN for calculating the vertical offset from the limitation surface

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.

If tool radius compensation with G41, G42 is programmed when CUT3DCCD or CUT3DCC is active, the option "orientation transformation" must also be active.

3D radius compensation with CUT3DCC: (SW 6.4 and higher) contour on the machining surface

In CUT3DCC the NC part 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

- programmable offset OFFN for calculating the vertical offset from the limitation surface

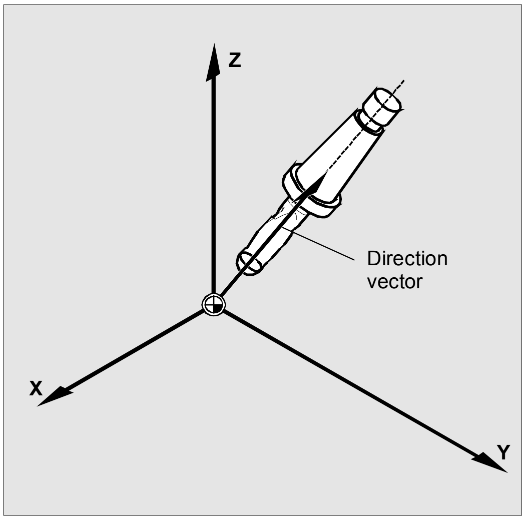
is used. The position of the limitation surface is determined by the difference between the two values

- standard tool dimensions and
- tool radius (tool parameter $TC_DP6)

8.6 Tool orientation, ORIC, ORID, OSOF, OSC, OSS, OSSE

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.

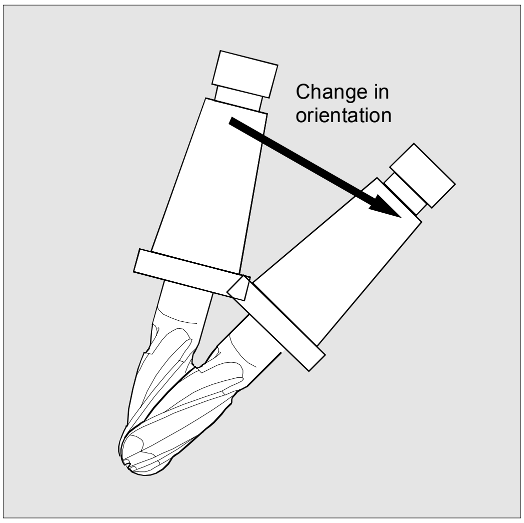


Programming tool orientation
A change in tool orientation can be programmed by:

- Direct programming of the rotary axes
- Euler or RPY angle
- Direction vector
- LEAD/TILT (face milling)

The reference coordinate system is either the machine coordinate system (ORIMKS) or the current workpiece coordinate system (ORIWKS).

A change in orientation can be controlled by the following:

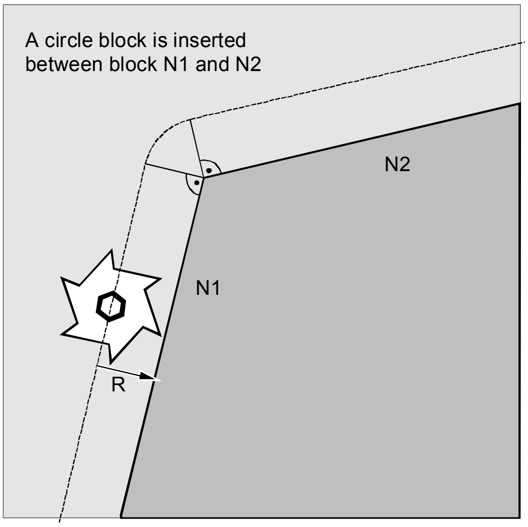


`ORIC`	Orientation and path movement in parallel
`ORID`	Orientation and path movement consecutively
`OSOF`	No orientation smoothing
`OSC`	Orientation constantly
`OSS`	Orientation smoothing only at beginning of block
`OSSE`	Orientation smoothing at beginning and end of block
`ORIS`	Speed of orientation change with active orientation smoothing in degrees per mm; applies to OSS and OSSE

Behavior at outer corners

A circle block with the radius of the cutter is always inserted at an outside corner.

The program commands ORIC and 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.



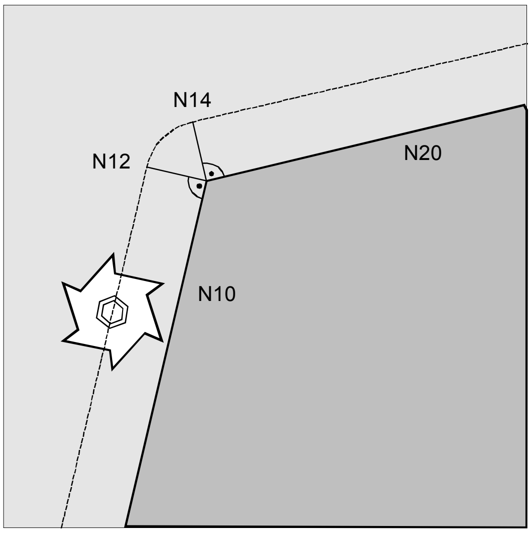
If an orientation change is required at outside corners, this can be performed either at the same time as interpolation or separately together with the path movement.

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.

Programming example for ORIC

If two or more blocks with orientation changes (e.g. A2= B2= C2=) are programmed between traversing blocks N10 and N20 and ORIC is active, the inserted circle block is divided among these intermediate blocks according to the values of the angle changes.



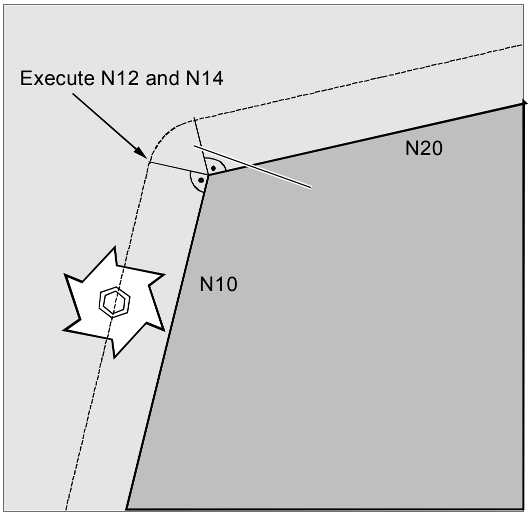
`ORIC`	
`N8 A2=… B2=… C2=…`	
`N10 X… Y… Z…`	

`N12 C2=… B2=…`
`N14 C2=… B2=…`
	The circle block inserted at the external corner 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`	

Programming example for ORID

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=…`	Blocks N12 and N14 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…`	

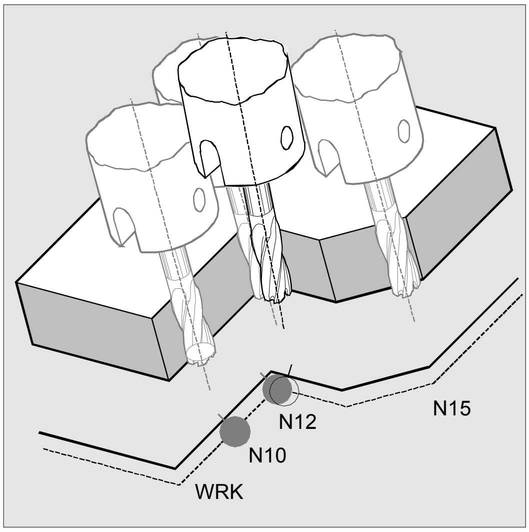
The method by which the orientation is changed at an outer corner is determined by the program command that is active in the first traversing block of an outer corner.

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.

Programming example

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
```

8.7 Free assignment of D numbers, cutting edge number CE (SW 5 and higher)

SW 5 and higher, you can use the D numbers 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: FB, W1 (Tool Offset)

Machine manufacturer (MH 8.12)

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.

Other information

Besides the relative D number, you can also assign D numbers al 'flat' or 'absolute' D numbers (-32000) without assigning a reference to a T number (inside the function 'flat D number structure').

8.7.1 Checking D numbers (CHKDNO)

Programming

```
state=CHKDNO(Tno1,Tno2,Dno)
```

Explanation of the 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.	

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.

8.7.2 Renaming D numbers (GETDNO, SETDNO)

Programming

```
d = GETDNO (t,ce)

state = SETDNO (t,ce,d)
```

Explanation of the 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).

Function

GETDNO

This command returns the D number of a particular cutting edge (ce) of a tool with tool number t.
If no D number matching the input parameters exists, d=0. If the D number is invalid, a value greater than 32000 is returned.

SETDNO

This commands 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.

Example: (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.

Other information

You must assign unique D numbers. Two different cutting edges of a tool must not have the same D number.

8.7.3 Deriving the T number from the specified D number (GETACTTD)

Programming

```
status = GETACTTD (Tno, Dno)
```

Explanation of the 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.

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).

8.7.4 Invalidating D numbers

Programming

```
DZERO
```

Explanation

`DZERO`	Marks all D number of the TO unit as invalid

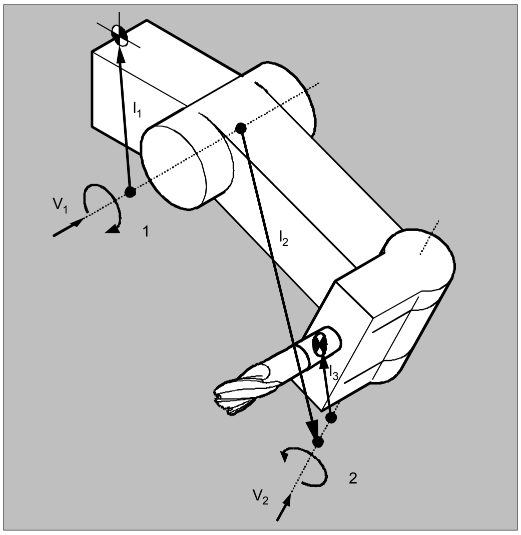
Function

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.

8.8 Tool holder kinematics

The toolholder kinematics with max. two rotary axes is programmed by means of 17 system variables $TC_CARR1[m] to $TC_CARR17[m]. The description of the toolholder consists of:

- The vectorial distance between the first rotary axis and toolholder reference point l_1, the vectorial distance between the first and second rotary axis l_2, the vectorial distance between the second rotary axis and tool reference point l_3.
- The direction vectors of both rotary axes v_1, v_2.
- The rotational angles α_1, α_2 around both axes. The rotation angles are counted in viewing direction of the rotary axis vectors, positive, in clockwise direction of rotation.



Resolved kinematics as of SW 5.3

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 vector distance between the second rotary axis v_2 and the reference point of a rotatable tool table $\mathbf{l}_4$ 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. These positions are accessed as required to determine the orientation of the orientable toolholder.

The type of kinematics with one of the values T, P or M:

- Type of kinematics T: Only tool can rotate.
- Type of kinematics P: Only part can rotate.
- Type of kinematics M: Tool and part can rotate.

Function of the system variables for orientable tool holders

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]
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]
l_4 offset vector	$TC_CARR18[m]	$TC_CARR19[m]	$TC_CARR20[m]
Axis identifier of rotary axis v_1 of rotary axis v_2	Axis identifier of rotary axes v_1 and v_2 (default is zero) $TC_CARR21[m] $TC_CARR22[m]		
Kinematic type Default T	$TC_CARR23[m]		
	Type of kinematics T ⇨	Type of kinematics P ⇨	Type of Kinematics M
	Only the **T**ool can be rotated	Only the **P**art can be rotated	Part and tool **M**ixed mode can be rotated
Offset of rotary axis v_1 rotary axis v_2	Angle in degrees of rotary axes v_1 and v_2 when assuming the initial setting $TC_CARR24[m] $TC_CARR25[m]		
Angle offset for rotary axis v_1 rotary axis v_2	Offset of 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	Increment of Hirth tooth system in degrees for rotary axes v_1 and v_2 $TC_CARR28[m] $TC_CARR29[m]		
Minimum position rotary axis v_1 rotary axis v_2	Software limit for minimum position for rotary axes v_1 and v_2 $TC_CARR30[m] $TC_CARR31[m]		
Maximum position rotary axis v1 rotary axis v_2	Software limits for maximum position for rotary axes v_1 and v_2 $TC_CARR32[m] $TC_CARR33[m]		

Parameters of the rotary axes SW 6.1 and higher

The system variables are extended by the entries $TC_CARR24[m] to $TC_CARR33[m] and described as follows:

The **offset** of the rotary axes

- Changing the position of rotary axis v_1 or v_2 during initial setting of the orientable toolholder.

The **angle offset/angle increment** of the rotary axes

- Offset or angle increment of Hirth tooth system of 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 position/maximum position** of the rotary axis

- Limit angle (software limit) for rotary axis v_1 and v_2.

System variable expansions for orientable tool holders SW 6.4 and higher			
Designation	x component	y component	y component
Tool holder name	A tool holder 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]		
	x component	y component	y component
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]		

Other information

"m" specifies the number of the tool holder to be programmed.

The start/endpoints of the distance vectors on the axes can be freely selected. The rotation angles α_1, α_2 around the two axes are defined in the initial state of the toolholder by 0°. In this way, the kinematics of a toolholder can be programmed for any number of possibilities.

Toolholders with only one or no rotary axis at all can be described by setting the direction vectors of one or both rotary axes to zero. With a toolholder without rotary axis the distance vectors act as additional tool offsets whose components cannot be affected by a change of machining plane (G17 to G19).

Clearing the toolholder data
The data of all toolholder data sets is cleared via $TC_CARR1[0] = 0.

SW 5.3 and higher
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.

SW 6.4 and higher
$TC_CARR34 to $TC_CARR40
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.

$TC_CARR41 to $TC_CARR65 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.

$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.

Restrictions

A tool holder can only orientate a tool in all possible directions in space if

- two rotary axes v_1 exist v_2.
- the rotary axes are mutually orthogonal.
- the tool longitudinal axis is perpendicular to the second rotary axis $v_{2.}$

SW 5.3 and higher

In addition, the following requirement is applicable to machines for which all possible orientations have to be settable:

- Tool orientation must be perpendicular to the first rotary axis v_1.

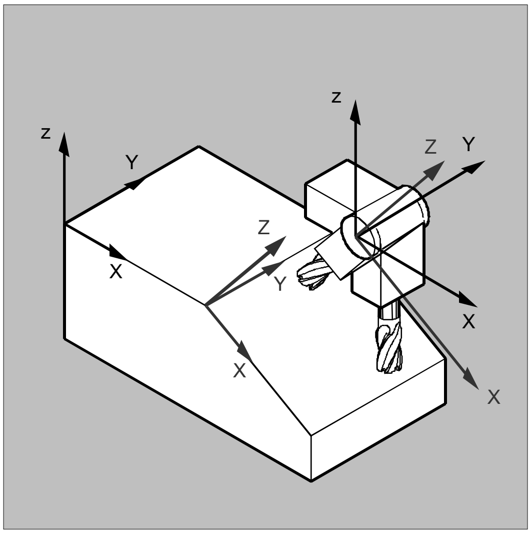
Fine offsets (SW 6.4 and higher)

A illegal 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.

Programming 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`	with length 20 mm
`N40 $TC_DP6[1,1]=5`	and 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 compensation in rotated frame, select toolholder 1, tool 1
`N80 X40`	Execute machining under a 37° rotation
`N90 Y40`	
`N100 X0`	
`N110 Y0`	
`N120 M30`	

Path Action

9.1 Tangential control TANG, TANGON, TANGOF, TLIFT, TANGDEL

Programming

```
TANG (Faxis,Laxis1,Laxis2,Coupling,CS,Opt)
TANGON (Faxis,Angle, Dist, Angletol)
TANGOF (Faxis)
TLIFT (Faxis)
TANGDEL (FAxis)
```

Explanation of the commands

TANG	Preparatory instruction for the definition of a tangential follow-up
TANGON	Activate tangential control specifying following axis and offset angle and, if necessary, rounding path, angle deviation
TANGOF	Deactivate tangential control specifying following axis
TLIFT	Insert intermediate block at contour corners
TANGDEL	Delete definition of a tangential follow-up

Explanation of the parameters

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	Identifier for coordinate system "B" = basic coordinate system; data optional; default ["W" = workpiece coordinate system]
Opt	Optimization: "S" Standard (up to and including SW version 6), Default "P" automatic adaptation of time characteristic of tangential axis and 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"

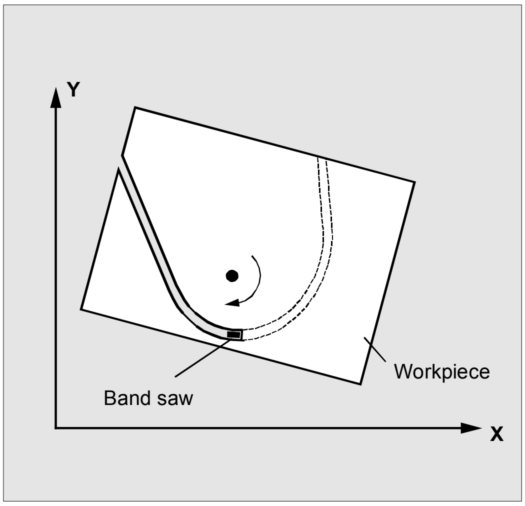
Function

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 dresser tool on a grinding wheel (see diagram below)
- Positioning of a cutting wheel for glass or paper working
- Tangential infeed of a wire in five-axis welding



Sequence

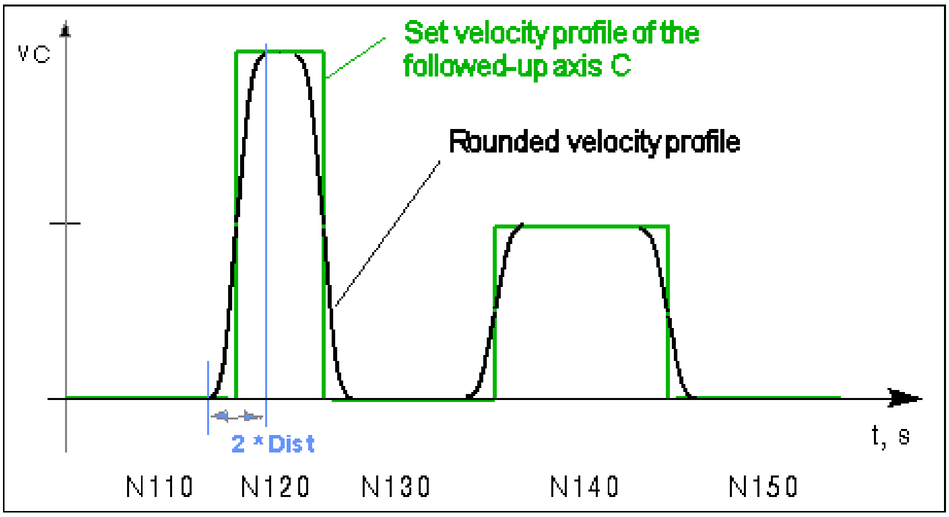
Defining following axis and leading axis

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).
The follow-up can be performed in the basic coordinate system "B" (default) or the workpiece coordinate system "W".
Example:
`TANG(C,X,Y,1,"B")`
Meaning:
Rotary axis C follows geometry axes X and Y.

Optimization options SW Version 7 and higher:

If `Opt="P"` the dynamic response of the following axis is included in the velocity limitation of the leading axes.

The parameters (`Dist` and `Angletol`) limit the error between the following axis and the tangent of the leading axes precisely. 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.
`Opt="P"` is particularly recommended in kinematic transformations.



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(...) instruction must be programmed immediately after the axis assignment with TANG(...). Example:

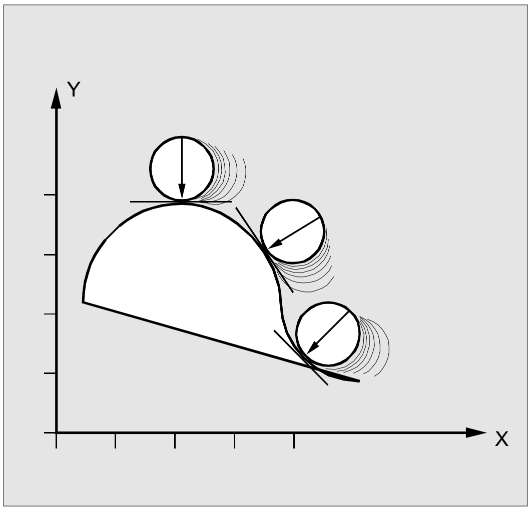
```
TANG(C,X,Y...)
TLIFT(C)
```

Deactivate TLIFT
Repeat axis assignment TANG(...) without following it by TLIFT(...).

Activating/deactivating tangential control: TANGON, TANGOF

Tangential control is called with TANGON specifying the following axis and the desired offset angle of the following axis:

```
TANGON(C,90)
```

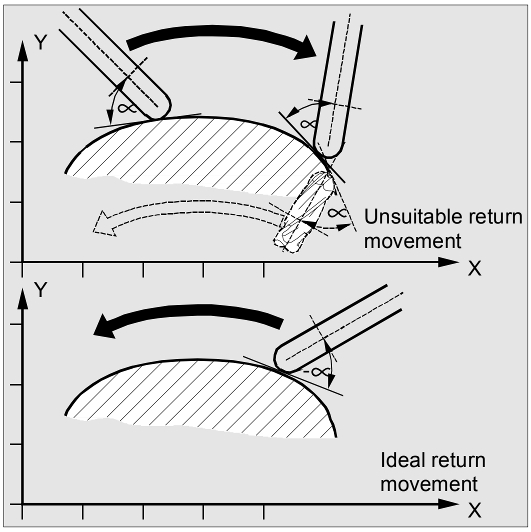


Meaning:
C axis is the following axis. On every movement of the path axes, it is rotated into a position at 90° to the path tangent.

The following axis is specified in order to deactivate the tangential control:

```
TANGOF(C)
```

Angle limit through 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.

Software 7:
This function is not active when `Opt="P"`. Alarm output is stopped when the working area limitation is reached.

Other Information

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 TILIFT 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 FGREF[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.

The angular change limit at which an intermediate block is automatically inserted is defined via machine data
`$MA_EPS_TLIFT_TANG_STEP`

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.

```
TANGDEL (FAxis)                            Delete tangential follow-up
```

Deletion is only possible if the coupling with TANGOF(Faxis) is deactivated.

Programming example

Example of 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
...
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
```

Programming example

No alarm is produced.

With geometry axis switchover and TANGDEL

```
N10 GEOAX(2,Y1)                            Y1 is geo 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 geo 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.
...
```

Programming example

Tangential follow-up with **automatic optimization** by means of Dist and Angle Tolerance

```
N80 G0 C0
N100 F=50000
N110 G1 X1000 Y500
N120 TRAORI                               ; Rounding with axial tolerance
N130 G642
N171 TRANS X-1200 Y-550                   ; Autom. 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
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
```

Other information

Influence 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

Programming

```
TRAILON(Faxis,Laxis,Coupling)
TRAILOF(Faxis,Laxis,Axis2)
```

Explanation of the commands and parameters

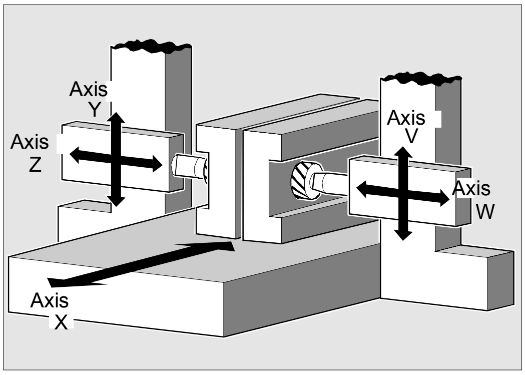
`TRAILON`	Activate and define coupled axes; modal
`TRAILOF`	Deactivate coupled axes
`Faxis`	Axis name of trailing axis
`Laxis`	Axis name of trailing axis
`Coupling`	Coupling factor = Path of coupled-motion axis/path of trailing axis Default = 1

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-sided machining with 2 combined axis pairs:
  1. leading axis Y, trailing axis V
  2. leading axis Z, trailing axis W



Sequence

Defining coupled-axis combinations, TRAILON
The coupled axes are defined and activated simultaneously with the modal language command TRAILON.

```
TRAILON(V,Y)
```

V = trailing axis, Y = leading axis

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.

Coupled axis motion is always executed in the base coordinate system (BCS).

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.

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.

Coupling factor
The coupling factor specifies the desired relationship between the paths of the coupled axis and the leading axis.

$$\text{Coupling factor} = \frac{\text{Path of trailing axis}}{\text{Path of leading axis}}$$

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.

Deactivate coupled axes
The coupling of one leading axis is deactivated with the command:

```
TRAILOF(V,Y)
```

V = trailing axis, Y = leading axis
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)
```

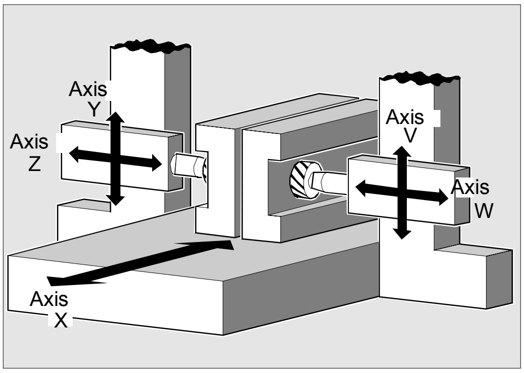
Other information

Acceleration and velocity

The acceleration and velocity limits of the combined axes are determined by the "weakest axis" in the combined axis pair.

Programming example

The workpiece is to be machined on two sides with the axis configuration shown in the diagram. To do this, you create 2 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

9.3 Curve tables CTABDEF, CTABEND, CTABDEL, CTAB, CTABINV/CTABFNO

9.3.1 Language commands with curve tables CTABID, CTABLOCK, CTABUNLOCK

Programming

Modal language commands with curve tables:

A) Main functions:
Curve tables are defined in a parts program. The parameters are explained at the end of the list of language commands.

`CTABDEF (Faxis,Laxis,n,applim, memType)`	Define beginning of curve table.
`CTABEND ()`	Define end of curve table.
`CTABDEL ()`	Delete all curve tables, **irrespective of memory type** (SW 6.3 and higher).

B) General form (SW 6.3 and higher):

`CTABDEL (n, m, memType)`	In memory type SRAM or DRAM: Deletion of the curve tables of the curve table range that are stored in memType.
`CTABNOMEM (memType) (as from SW 6.4)`	Number of **defined** curve tables.
`CTABFNO (memType)   (as from SW 6.4)`	Number of **possible** tables.
`CTABID (n, memType) (as from SW 6.4)`	Outputs table number entered in memory type as the nth curve table.
`CTABLOCK (n, m, memType)`	**Set lock** against deletion and overwriting.
`CTABUNLOCK (n, m, memType)`	**Cancel lock** against deletion and overwriting. 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.

C) Application of further forms (SW 6.3 and higher)	**Optional definitions for selections**
`CTABDEL(n)`	Delete **one** curve table
`CTABDEL(n, m)`	Delete **one** curve table range
`CTABDEL(, , memType)`	Delete **all** curve tables in the specified memory.
`CTABLOCK(n)`	**Lock** 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)`	**Unlock** 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.
D) Application of further forms (SW 6.4 and higher)	**For diagnosing axis coupling:**
`CTABID (n, memType)`	Outputs table number of the nth/pth curve table **with memory type** memType.
`CTABID(p, memType)`	
`CTABID(n)`	Outputs table number of the nth curve table with memory type **defined** in MD 20905:CTAB_DEFAULT_MEMORY_TYPE.
`CTABISLOCK(n)`	Returns the lock status of the curve table **with number n**.
`CTABEXIST (n)`	**Checks** curve table with number n
`CTABMEMTYP (n)`	**Returns** the **memory** in which curve table with No. n is entered.
`CTABPERIOD (n)`	Returns the **table periodicity**.
`CTABSEG (memType)`	Number of curve segments already **used** in the specified memory type.
`CTABSEGID (n)`	Number used by the curve table with number n. Curve segments.
`CTABFSEG(memType)`	Number of **possible** curve segments.
`CTABMSEG(memType)`	**Maximum** possible number of curve seg segments.
`CTABPOLID (n)`	Number used by the curve table **with number n**. Curve polynomials.
`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.

Trailing or leading position derived from curve table with CTAB, CTABINV	**(SW 5.1 and higher)**
`R10=CTAB (LW,n,degrees,Faxis,Laxis)`	Following value for a leading value
`R10=CTABINV (FW,aproxLW,n,degrees,Faxis,Laxis)`	Leading value to a following value
Determining the segments of the curve table by specifying a leading value with CTABSSV, CTABSEV	**(SW 6.3 and higher)**
`R10=CTABSSV(LV,n,degree,Faxis,Laxis)`	Starting value of the following axis in the segment belonging to the LV
`R10=CTABSEV(LV,n,degree,Faxis,Laxis)`	End value of the following axis in the segment belonging to the LV
Values of the trailing and leading axis located at the beginning and end of a curve table CTABTSV, CTABTEV, CTABTSP, CTABTEP	**(SW 6.4 and higher)**
`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)`	Leading value at beginning of curve table
`R10=CTABTEP (n,degrees,Laxis)`	Leading 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 leading value
`R10=CTABTMAX(n, a, b, Faxis, Laxis)`	Maximum following value of curve table in interval a…b of leading value

R parameter assignments in the table definition are reset.

Example:

```
...
R10=5 R11=20
...
CTABDEF
G1 X=10 Y=20 F1000
R10=R11+5      ;R10=25
X=R10
CTABEND
...            ;R10=5
```

Explanation of the parameters

`Faxis`	Following axis Axis that is programmed via the curve table.
`Laxis`	Leading axis Axis that is programmed with the leading value.
`R10`	Parameter name Parameter name in which starting and end value is to be stored.
`LW`	Master value Positional value of the leading axis for which a following value is to be calculated.
`FW`	Calculate Positional value of the following axis for which a leading value is to be calculated.
`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.
`a, b`	Interval of leading value (entry in CTABMIN and CTABMAX optional) Limits a and b are required for the interval. If values lie outside the range of definition of the leading value of the curve table, one of the two limit values of the leading value is used.
`degrees`	Parameter name for gradient parameter.
`p`	Entry location (in memory area memType).
`applim`	Identifier for table periodicity: 0 Table is not periodic 1 Table is periodic with regard to the leading axis 2 Table is periodic with regard to leading axis and following axis
`aproxLW`	Approximation solution for leading value if no specific leading value can be determined for a following value.
`FAxis,LAxis`	Optional specification of the following and/or leading 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.

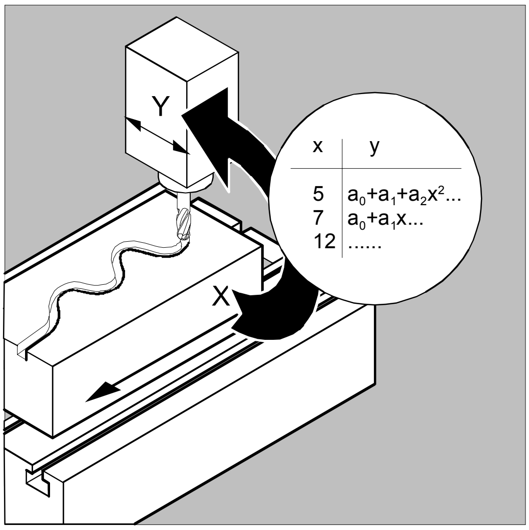
For further information about leading and following values, see Section "Axial leading value coupling" in this section.

Additional functions exist for diagnostics of axis coupling and optimization of resource use. For further information, see
/FB/, M3, Coupled Axes and ESR

Function

You can use curve tables to program position and velocity relationships between 2 axes.

Example of substitution of mechanical cam: The curve table forms the basis for the axial leading 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 plate from the relative positions of the leading and following axes.

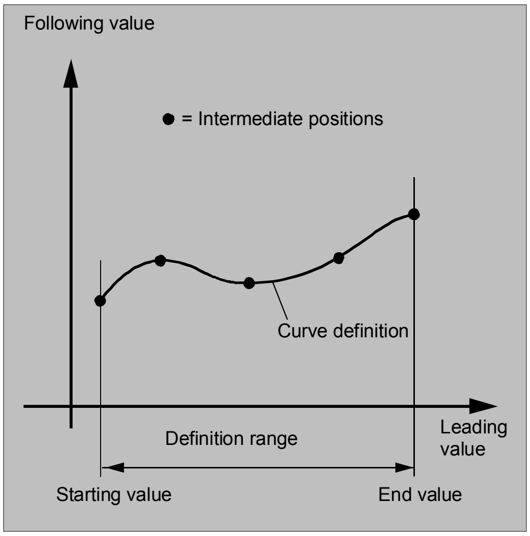


Other Information

To create curve tables the memory space must be reserved by setting the machine data.

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 3rd order.

As from SW 6, intermediate points for curve definitions can be calculated as 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 located in front of and after the table definition in the same state.

Other Information

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 CSPINE is activated in a curve table CTABDEF()... CTABEND, at least one starting point should be programmed before this spline is activated. Immediate activation after CTABDEF must be avoided as otherwise the spline will depend on the current axis position before the curve table definition.
Example:

```
...
CATBDEF(Y, X, 1, 0)
X0 Y0
ASPLINE
X=5 Y=10
X10 Y40
...
CTABEND
```

SW 6.3 and higher

Dependent 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 give in the notice still apply.
When creating and deleting tables you can use the definitions of the memory type of the NC.

Repeated use of curve tables
The function relation between the leading axis and the following axis calculated through the curve table is retained under the table number 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.

Deleting curve tables, **CTABDEL**
With CTABDEL you can 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.
As from SW 6.3, curve tables of a certain memory type can be deleted by optional memory type specification.

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 part program processing is canceled with alarm 15050.

Overwriting curve tables
A curve table is overwritten as soon as is number is used in another table definition.
Exception: A curve table is either active in an axis coupling or locked with CTABLOCK().

Other Information

- 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.

Programming example

Using CTABDEF and CTABEND

A program section is to be used unchanged for defining a curve table. The command for preprocess stop STOPRE can remain and is active again immediately as soon as the program section is not used for table definition and CTABDEF and CTABEND have been removed:

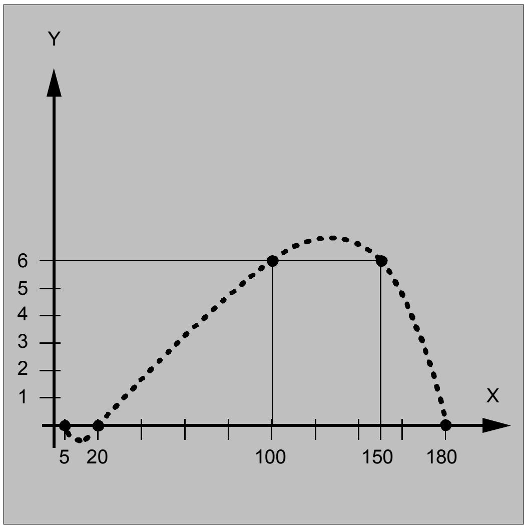
```
CTABDEF(Y,X,1,1)
…
…
IF NOT ($P_CTABDEF)
STOPRE
ENDIF
…
…
CTABEND
```

Curve tables and various operating states

During active block search, calculation of curve tables is not possible. If the target block is within the definition of a curve table, an alarm is output when CTABEND is reached.

Programming example 1

Definition of a curve table



`N100 CTABDEF(Y,X,3,0)`	Beginning of the definition of a non-periodic curve table with number 3
`N110 X0 Y0`	1. Traverse statement defines 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 (circle, linear, spline interpolation); The parts program state before the beginning of the definition is restored.

Programming example 2

Definition of a periodic curve table with number 2, leading 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 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 leading 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:

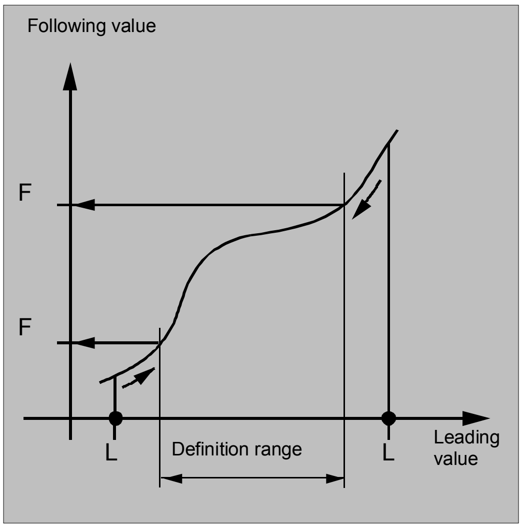
```
N190 LEADON(Y,X,2)
N200 G1 X110 F1000
N210 LEADOF(Y,X)
N220 M30
```

9.3.2 Behavior at curve table edges CTABTSV/CTABTSP, CTABTMIN

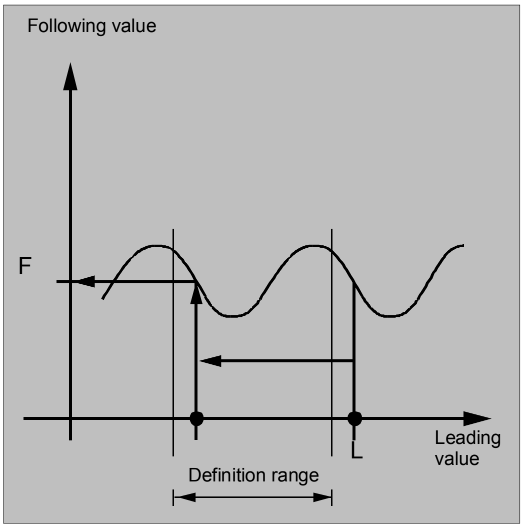
Edge values at start and end values: CTABTSV, CTABTEV, CTABTSP, CTABTEP (SW 6.4 and higher)

Min and max value ranges: CTABTMIN, CTABTMAX (SW 6.4 and higher)

Non-periodic curve table
If the leading value is outside the definition range, the following value output is the upper or lower limit.



Periodic curve table
If the leading value is outside the definition range, the leading value is evaluated modulo of the definition range and the corresponding following value is output.



Reading edge values from curve tables CTABTSV, CTABTEV, CTABTSP, CTABTEP

With CTABTSV a **following axis** can read the value at the **beginning** of the curve table.
With CTABTEV a **following axis** can read the value at the **end** of the curve table.

With CTABTSP a **leading axis** can read the value at the **beginning** of the curve table.
With CTABTEP a **leading axis** can read 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 leading values. The start value is always defined by the lower interval limit, and the end value by the upper interval limit.

CTABTMIN, CTABTMAX

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 leading value.

The language commands CTABTSV, CTABTEV, CTABTSP, CTABTEP, CTABTMIN, CTABTMAX can be used by the

- parts program or
- directly from synchronized actions.

The internal execution time of the function of

- CTABINV( ) is **dependent**
- CTABTSV, CTABTEV, CTABTSP, CTABTEP, (CTABTMIN, CTABTMAX only if no interval is specified for the leading value interval) is **independent**

of the number of table segments.

Reading 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 10.

Programming example

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)
N110 X0 Y10
N120 X30 Y40
N130 X60 Y5
N140 X70 Y30
N150 X80 Y20
N160 CTABEND
...
N200 STARTPOS     = CTABTSV(1, GRADIENT)
N210 ENDPOS       = CTABTEV(1, GRADIENT)
N220 SRARTPARA    = CTABTSP(1, GRADIENT)
N230 ENDPARA      = CTABTEP(1, GRADIENT)
...
N240 MINVAL       = CTABTMIN(1)
N250 MAXVAL       = CTABTMAX(1)
```

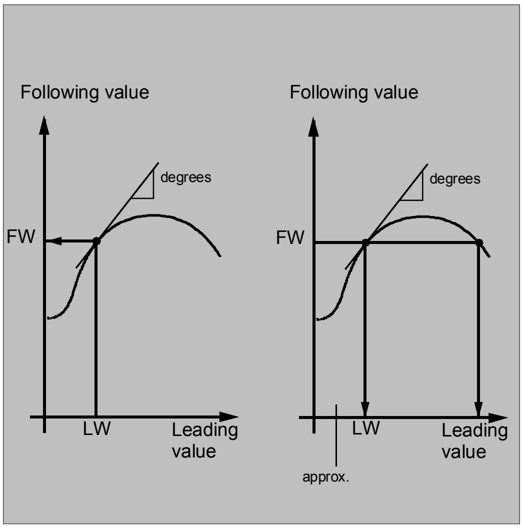
Beginning of table definition
Starting value 1st table segment
End value 1st table segment = start value 2nd table segment ...

End of table definition
Read starting position STARTPOS = 10, end position ENDPOS = 20 of table and STARTPARA = 10, ENDPARA = 80 of value range of following axis.

Minimum value when Y = 5 and maximum value when Y = 40

9.3.3 Access to curve table positions and table segments CTAB, CTABINV

Reading table positions, CTAB, CTABINV
With CTAB you can read the following value for a leading value directly from the parts program or from synchronized actions (Chapter 10).



With CTABINV, you can read the leading value for a following value. This assignment does not always have to be unique. CTABINV therefore requires an approximate value (aproxLW) 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.

Reading curve segment positions, CTABSSV, CTABSEV (SW 6.3 and higher)
With CTABSSV the **starting value** of the curve segment that belongs to the specified leading value can be read.

With CTABSSV the **end value** of the curve segment that belongs to the specified leading value can be read.

Language commands CTAB, CTABINV, and CTABSSV, CTABSEV can be used directly

- from the parts program or
- directly from synchronized actions.

All related information about programming synchronized actions is given in Chapter 10.

Other information

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.

Language commands CTABSSV and CTABSEV are **not suitable for** querying programmed segments in the following cases:

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 active e.g. with COMPON, COMPCURV, COMPCAD.

Programming example

Use of CTABSSV and CTABSEV

Determining the curve segment belonging to leading value X = 30.

```
N10 DEF REAL STARTPOS
N20 DEF REAL ENDPOS
N30 DEF REAL GRADIENT
...
                                         Beginning of table definition
N100 CTABDEF(Y,X,1,0)                    Starting position 1st table segment
N110 X0 Y0                               End position 1st table segment = start
N120 X20 Y10                             position 2nd table segment ...
N130 X40 Y40
N140 X60 Y10
N150 X80 Y0
N160 CTABEND                             End of table definition
...
N200 STARTPOS = CTABSSV(30.0, 1,         Start position Y in segment 2 = 10
                        GRADIENT)
...                                      End position Y in segment 2 = 40
N210 ENDPOS = CTABSEV(30.0, 1,           Segment 2 belongs to LV X = 30.0.
                        GRADIENT)
```

9.4 Axial leading value coupling, LEADON, LEADOF

Programming

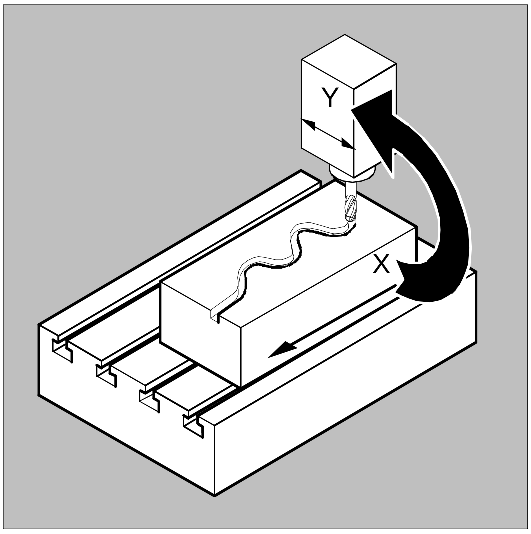
```
LEADON(FAxis,LAxis,n)
LEADOF(FAxis,LAxis,n)
```

Explanation

`LEADON`	Activate leading value coupling
`LEADOF`	Deactivate leading value coupling
`Faxis`	Following axis
`Laxis`	Leading axis
n	Curve table number

Function

With the axial leading 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.



Leading axis is the axis which supplies the input values for the curve table. **Following axis** is the axis which takes the positions calculated by means of the curve table.

The leading value coupling can be activated and deactivated both from the parts program and during the movement from synchronized actions (Chapter 10).

The leading value coupling always applies in the basic coordinate system.

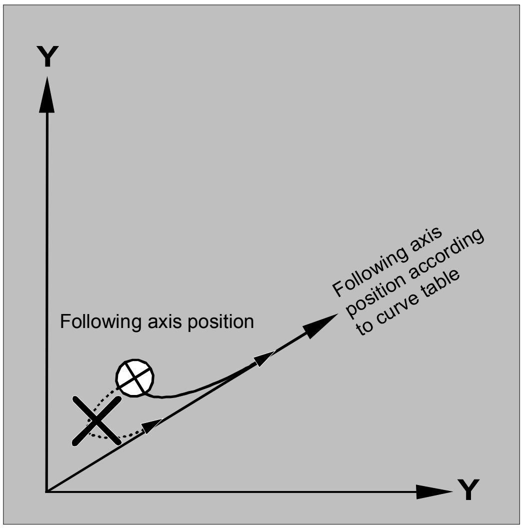
See Section "Curve tables" in this chapter for information about how to create curve tables.
For information about leading value coupling see /FB/, M3, Coupled Axes and Leading Value Coupling

Sequence

Leading 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 leading value coupling is activated.

The tolerance range for the position of the following axis is defined via machine data 37200 COUPLE_POS_TOL_COARSE.

If the following axis is not yet at the correct position when the leading 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 CTAB).



Other Information

If the following axis position calculated moves away from the current following axis position when the leading value coupling is activated, it is not possible to establish synchronization.

Actual value and setpoint coupling

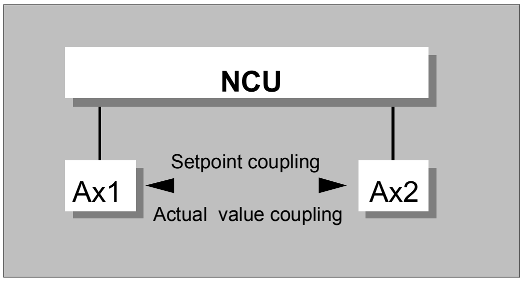
The following can be used as the leading value, i.e. as the output values for position calculation of the following axis:

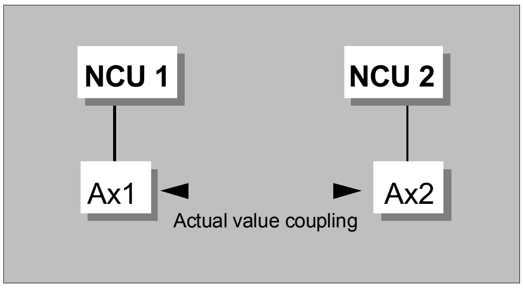
- Actual values of the leading axis position: Actual value coupling
- Setpoints of the leading axis position: Setpoint value coupling

Other Information

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.





Switchover between actual and setpoint coupling

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.

Sample application:

You cannot read the actual values without error during large machine vibrations. If you use leading 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.

Leading value simulation with setpoint simulation

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.

Leading values generated from a setpoint link can be read from the following variables so that they can be used, for example, in synchronized actions:

- `$AA_LEAD_P`	Leading value position
- `$AA_LEAD_V`	Leading value velocity

Other information

As an option, leading values can be generated with other self-programmed methods. The leading values generated in this way are written to and read from variables

- `$AA_LEAD_SP`	Leading value position
- `$AA_LEAD_SV`	Leading value velocity

Before you use these variables, 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	Leading value coupling active

Deactivate leading value coupling, LEADOF

When you deactivate the leading value coupling, the following axis becomes a normal command axis again!

Axial leading value coupling and different operating states

Depending on the setting in the machine data, the leading value couplings are deactivated with RESET.

Programming example

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 LW (stanchion shaft), transfer axes and auxiliary axes are controlled as following axes that are defined via curve tables.

Following axes	X	Feed or longitudinal axis
	YL	Closing or lateral axis
	ZL	Stroke axis
	U	Roller feed, auxiliary axis
	V	Guiding head, auxiliary axis
	W	Greasing, auxiliary axis

Status management

Switching and coupling events are managed via real-time variables:

$AC_MARKER[i]=n	i	Marker number
with:	n	Status value

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 leading 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 controller, 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,LW,1) LEADON(YL,LW,2)
       LEADON(ZL,LW,3) $AC_MARKER[0]=1
                                               ;**** E1 0=>1 coupling roller feed ON
N20    IDS=11       EVERY ($A_IN[1]==1) AND
($A_IN[5]==0) AND ($AC_MARKER[5]==0)
DO     LEADON(U,LW,4) PRESETON(U,0)
       $AC_MARKER[5]=1
                                               ; **** E1 0->1 coupling guide head ON
N21    IDS=12       EVERY ($A_IN[1]==1) AND
($A_IN[5]==0) AND ($AC_MARKER[6]==0)
DO     LEADON(V,LW,4) PRESETON(V,0)
       $AC_MARKER[6]=1
                                               ; **** E1 0->1 coupling greasing ON
N22    IDS=13       EVERY ($A_IN[1]==1) AND
($A_IN[5]==0) AND ($AC_MARKER[7]==0)
DO     LEADON(W,LW,4) PRESETON(W,0)
       $AC_MARKER[7]=1
                                               ; **** E2 0=>1 coupling OFF
N30    IDS=3  EVERY ($A_IN[2]==1)
DO     LEADOF(X,LW) LEADOF(YL,LW)
       LEADOF(ZL,LW) LEADOF(U,LW)
LEADOF(V,LW) LEADOF(W,LW) $AC_MARKER[0]=0
$AC_MARKER[1]=0 $AC_MARKER[3]=0
$AC_MARKER[4]=0 $AC_MARKER[5]=0
$AC_MARKER[6]=0 $AC_MARKER[7]=0
....
N110   G04 F01
N120   M30
```

9.5 Feedrate response, FNORM, FLIN, FCUB, FPO

Programming

```
F… FNORM
F… FLIN
F… FCUB
F=FPO(…,…,…)
```

Explanation

`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 value (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.

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.

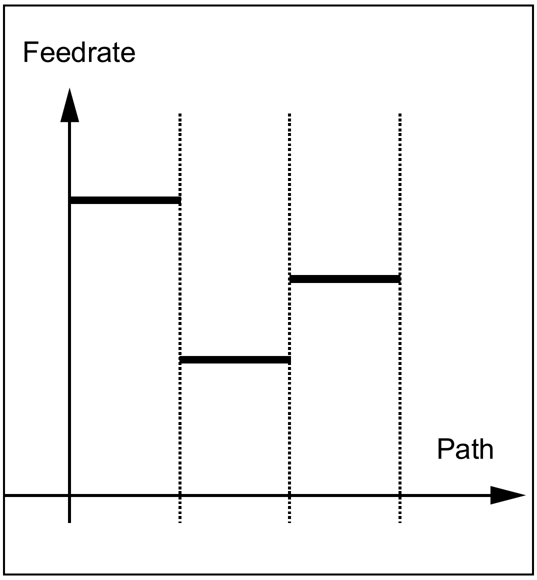
These additional characteristics make it possible to program continuously smooth velocity characteristics depending on the curvature of the workpiece to be machined.

These additional characteristics make it possible to program continuously smooth velocity characteristics depending on the curvature of the workpiece to be machined.

Sequence

FNORM
The feed address F defines the path feed as a constant value according to DIN 66025.

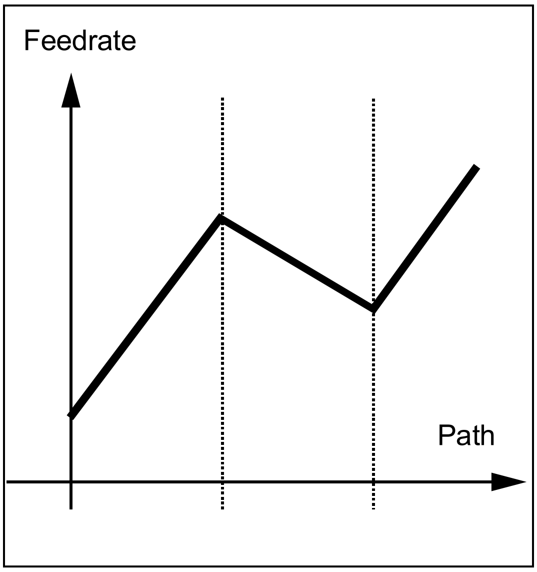
Please refer to Programming Guide "Fundamentals" for more detailed information on this subject.



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
```

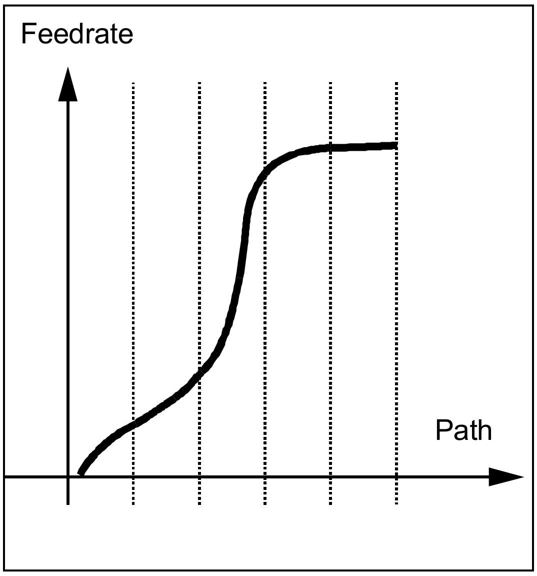


FCUB

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
...
```



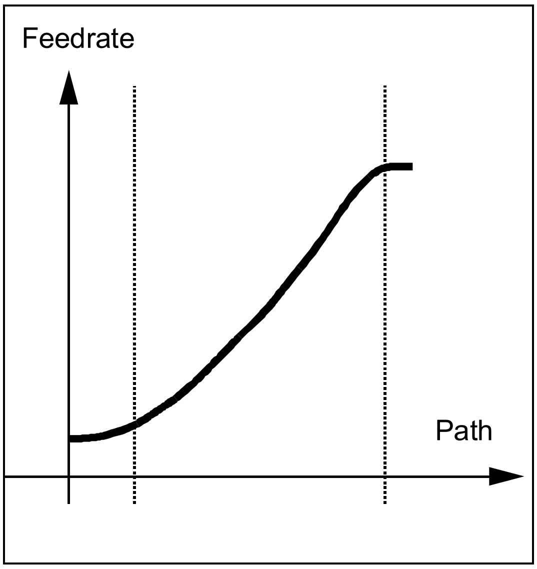
F=FPO(…,…,…)

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.

Other information

Compressor

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`

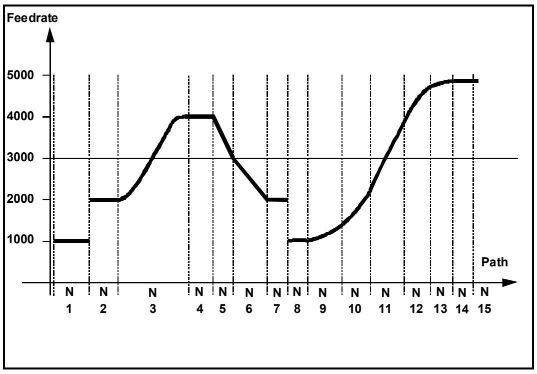
F=FPO(…,…,…)
These blocks are not compressed.

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.

Programming example

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`	

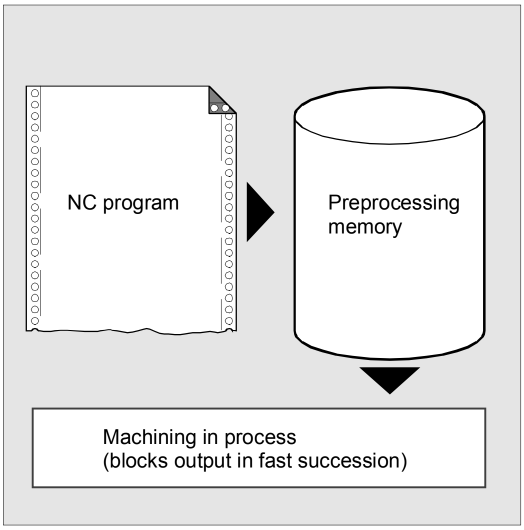
9.6 Program run with preprocessing memory, STARTFIFO, STOPFIFO, STOPRE

Explanation of the commands

`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`	Preprocessor stop

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.
STARTFIFO stops the machining process until the preprocessing memory is full or until STOPFIFO or STOPRE is detected.



Sequence

Mark processing section

The high-speed processing section to be buffered in the preprocessing memory is marked at the beginning and end with STARTFIFO and STOPFIFO respectively.

Example:

```
N10 STOPFIFO
N20…
N100
N110 STARTFIFO
```

Execution of these blocks does not begin until the preprocessing memory is full or command STARTFIFO is detected.

Restrictions

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, ...).

Stop preprocessing

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 on access to machine status data ($A...).

Example:

```
R10 = $AA_IM[X]       ;Read actual value of X axis
```

⚠ **Note**

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

Conditionally interruptible part program sections are called **stop delay sections**.

Programming

```
N... DELAYFSTON
N... DELAYFSTOF
```

The commands are programmed separately in a part program line. **DELAY F**eed **ST**op **ON** /**OF**

Explanation

`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
	Both commands are only permitted in part programs but not in synchronous actions.
	In the case of machine data 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.

Function

Some program sections should avoid **stops** and changes in **feedrate**. 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.

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

Event name	Response	Remarks
RESET	immediate	VDI: DB21.DBB7.Bit7 and DB11.DBB20.Bit7
PROG_END	Alarm 16954	NC prog.: M30
INTERRUPT	delayed	VDI: FC-9 and DB10.DBB1 (Asup)
DELDISTOGO_SYNC	immediate	VDI: Delete distance-to-go in channel and axially
PROGRESETREPEAT	delayed	VDI: DB21.DBB6.Bit3
PROGCANCELSUB	delayed	VDI: DB21.DBB6.Bit4
SINGLEBLOCKSTOP	delayed	VDI: Stop after each block Note: If single block mode is activated in a stop delay section the NCK stops at the end of the 1st block after the stop delay section. If single block is selected before the stop delay section the NCK stops at every block boundary (even in the stop delay section).
SINGLEBLOCK_IPO	delayed	VDI: Switch on single block type 1
SINGLEBLOCK_DECODIER	delayed	VDI: Switch on single block type 2
STOPALL	immediate	VDI: DB21.DBB7.Bit4 and DB11.DBB20.Bit6
STOPPROG	delayed	VDI: DB21.DBB7.Bit3 and DB11.DBB20.Bit5
OVERSTORE_BUFFER_END_REACHED	Alarm 16954	NC prog.: Stop because of empty overstore buffer
PREP_STOP	Alarm 16954	NC prog.: STOPRE and all implicit Stopres
PROG_STOP	Alarm 16954	NC prog.: M0 and M1
STOPPROGATBLOCKEND	delayed	VDI: DB21.DBB7.Bit2
STOPPROGATASUPEND	System error	SR end should always deselect the stop delay section.
WAITM	Alarm 16954	NC prog.: WAITM
WAITE	Alarm 16954	NC prog.: WAITE
INIT_SYNC	Alarm 16954	NC prog.: INIT with parameter "S"
MMCCMD	Alarm 16954	NC prog.: MMC(STRING, CHAR)
PROGMODESLASHON	delayed	VDI: DB21.DBB26 Activate or switch over skip block
PROGMODESLASHOFF	delayed	VDI: DB21.DBB26 Deactivate skip block
PROGMODEDRYRUNON	delayed	VDI: Switch on DB21.DBB0.Bit6 dryrun
PROGMODEDRYRUNOFF	delayed	VDI: Switch off DB21.DBB0.Bit6 dryrun
BLOCKREADINHIBIT_ON	delayed	VDI: Switch on DB21.DBB6.Bit1 read-in disable
STOPATEND_ALARM	immediate	Alarm: Alarm configuration STOPATENDBYALARM
STOP_ALARM	immediate	Alarm: Alarm configuration STOPBYALARM
STOPATIPOBUFFER_ISEMPTY_ALARM	immediate	Internal: Stop after alarm if interpolation buffer is empty
STOPATIPOBUF_EMPTY_ALARM_REORG	immediate	Internal: Stop after alarm if interpolation buffer is empty
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
BLOCKSEARCHRUN_NEWCONF	Alarm 16954	NC prog.: NEWCONF
SET_USER_DATA	delayed	OPI: PI "_N_SETUDT"
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 **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 secton.
Alarm 16955		Program is continued, an illegal action has taken place in the stop delay section.

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 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

Part program command G4 is permitted in a stop delay section whereas other part 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 11550, the largest possible stop delay section is 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.

Remarks: 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! That means if any other type of stop occurs in this program section, it stops immediately. Only by reprogramming (repeat DELAYFSTON) can a new stop delay section start.
If the Stop button is pressed before the stop delay section and the NCK has to enter the stop delay section to decelerate, the NCK stops in the stop delay section and the stop delay section remains deselected!
This applies to all "soft" stop events.

STOPALL can be used to decelerate in the stop delay section. However, with STOPALL all other stop events that had been delayed are immediately activated.

Programming example

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  subprogram2
...
      ...                                 ; interpretation of subprogram 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
```

System variables:
A stop delay section can be detected in the part program with **$P_DELAYFST**. If bit 0 of the system variables is set to 1, part program processing is now in a stop delay section.
A stop delay section can be detected in synchronized actions with **$AC_DELAYFST**. If bit 0 of the system variables is set to 1, part program processing is now in a stop delay section.

Other Information

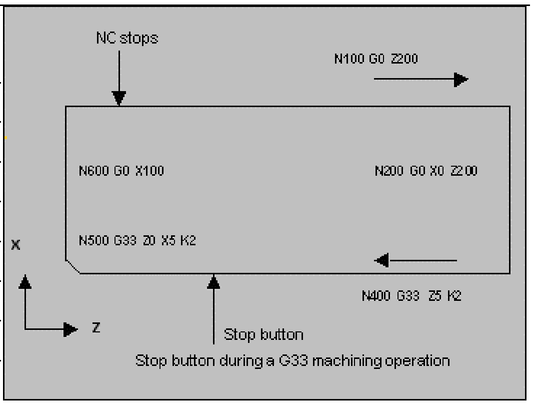
Compatibility:
Default of machine data 11550:
STOP_MODE_MASK Bit 0 = 0 triggers implicit stop delay section during a G code group G331/G332 and when a path movement or G4 is programmed.

Bit 0 = 1 enables a stop during G code group G331/G332 and when a path movement or G4 is programmed (response as for SW 6). Commands DELAYFSTON/DELAYFSTOF must be used to define a stop delay section.

Example program extract

; the following program block is repeated in a loop

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



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.

For details about SERUPRO type *block searches* see /FB/, K1, Channels, Program Operation, Reset Response.
/FB/, V1, *Feedrates* in connection with G331/G332 Feedrate for rigid tapping.

9.8 Preventing program position for SERUPRO, IPTRLOCK, IPTRUNLOCK

Programming

```
N... IPTRLOCK
N... IPTRUNLOCK
```

The commands are programmed separately in a part program line and permit a programmable interruption pointer

Explanation

IPTRLOCK	Start of untraceable program section
IPTRUNLOCK	End of untraceable program section
	Both commands are only permitted in part programs but not in synchronous actions.

Function

For some complicated mechanical situations on the machine it is necessary to the stop block search SERUPRO. By using a programmable interruption pointer it is possible to intervene before an untraceable point with "Search at point of interruption". It is also possible to define untraceable sections in part program sections that the NCK cannot yet re-enter. When a program is aborted the NCK remembers the last processed block that can be traced from the HMI user interface.

Sequence

Acquiring and finding untraceable sections
Untraceable program sections are identified with language commands IPTRLOCK and IPRTUNLOCK. 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. 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.

Other Information

Rules for nesting:
The following features regulate the interaction between language commands IPTRLOCK and IPTRUNLOCK with nesting and end of subroutine:

1. IPTRUNLOCK is activated implicitly at the end of the subroutine in which IPTRLOCK 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.

System variable:
An untraceable section can be detected in the part program with **$P_IPRTLOCK**.

Programming 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  subprogram2
...                             ; interpretation of subprogram 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.

Automatic interruption pointer
The automatic interruption pointer automatically defines a previously defined coupling type as untraceable. With machine data MD 22680: AUTO_IPR_LOCK the automatic interruption pointer is activated for

- electronic gearbox with EGON
- axial leading value coupling with LEADON.

If the programmed and automatic interruption pointers overlap (one from the language commands and the other from MD 22680: AUTO_IPR_LOCK), the largest possible untraceable section is generated.

For further information, see
/FB/, K1, Channels, Program Operation, Reset Response.

9.9 Repositioning at contour, REPOSA/L REPOSQ/H, RMI/N RMB/E

Programming

`REPOSA RMI DISPR=…` or `REPOSA RMB` or `REPOSA RME` or `REPOSA RMN`

`REPOSL RMI DISPR=…` or `REPOSL RMB` or `REPOSL RME` or `REPOSL RMN`

`REPOSQ RMI DISPR=… DISR=…` or `REPOSQ RMB DISR=…` or `REPOSQ RME DISR=…` or `REPOSQA DISR=…`

`REPOSH RMI DISPR=… DISR=…` or `REPOSH RMB DISR=…` or `REPOSH RME DISR=…` or `REPOSHA DISR=…`

Explanation of the commands

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

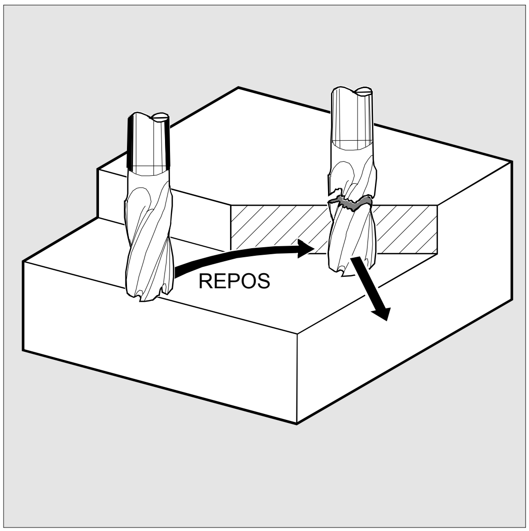
Reapproach point

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

Function

If you interrupt the program run and retract the tool during the machining operation because, for example, the tool has broken or you wish to check a measurement, you can reposition at any selected point on the contour under control by the program.

The REPOS command acts in the same way as a subprogram return jump (e.g. via M17). Blocks programmed after the command in the interrupt routine are not executed.

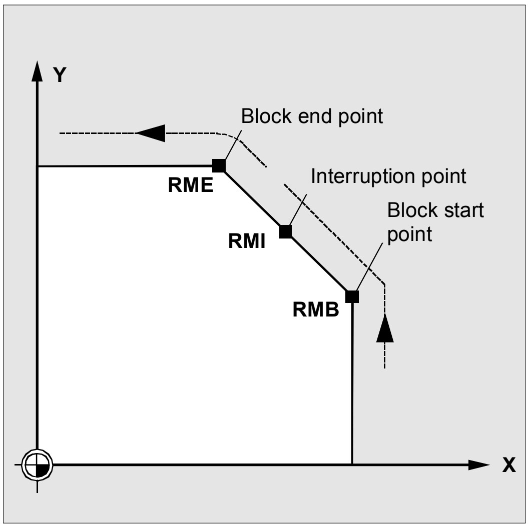


For information about interrupting program runs, see also Section "Interrupt routine" in Programming Guide "Advanced".

Sequence

Defining the repositioning point (up to SW 6.2)
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`).

As of SW 5.2:
The sign before `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.

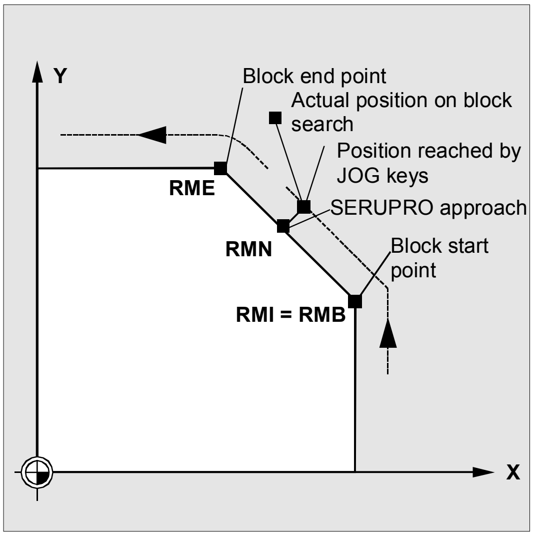
The distance between interruption point and approach point depends on the value of DISPR. Even for higher values, this point can lie in the block end point at the maximum.

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 (SW 6.3 and higher)

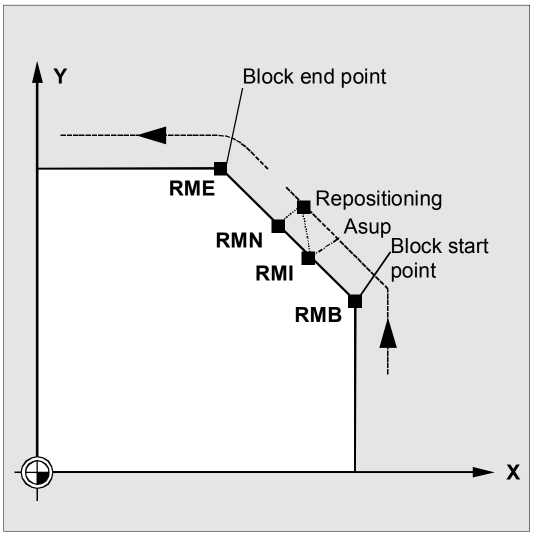
If abort is forced during machining at any position, the shortest path from the abort point is approached with SERUPRO approach and RMN so that afterward only the distance-to-go is processed. The user starts a SERUPRO process at the interruption block and uses the JOG keys to move in front of the problem component of the target block.



RMI and RMB are identical with respect to SERUPRO. RMN is not limited to SERUPRO but is generally applicable.

Approach from the nearest path point RMN

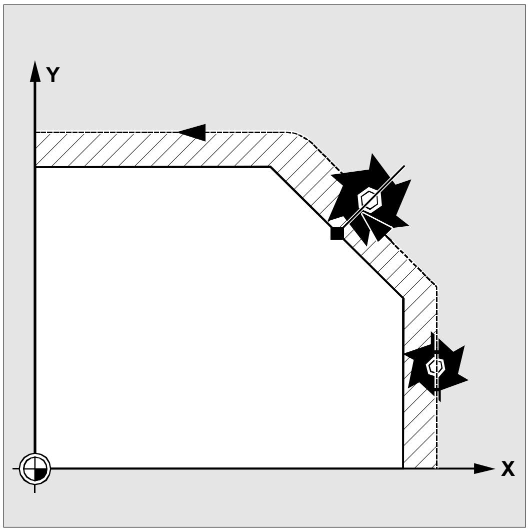
When REPOSA is interpreted, the repositioning block with RMN is not started again in full after an interruption, but only the distance-to-go processed. The nearest path point of the interrupted block is approached.



System variable SW 6.4 and higher:
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:	Approach to next path point of interrupted block.

Approach with 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.

Commands REPOSA, REPOSQA and REPOSHA automatically reposition all axes. Individual axis names need not be specified.

When 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 to be repositioned must be specified in the commands.

Approach along a straight line, REPOSA, REPOSL
The tool approaches the repositioning point along a straight line.

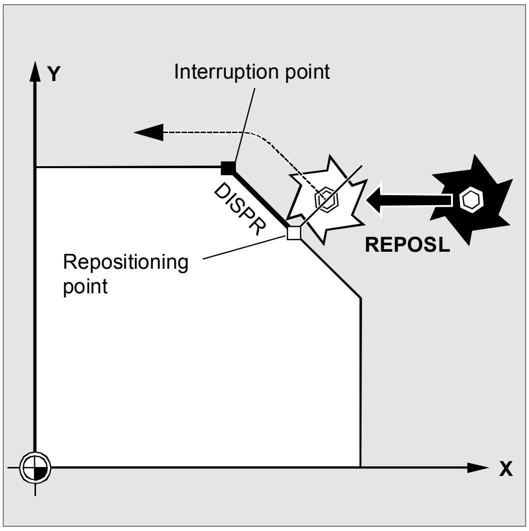
All axes are automatically traversed with command REPOSA. With REPOSL you can specify which axes are to be moved.

Example:

```
REPOSL RMI DISPR=6 F400
```

or

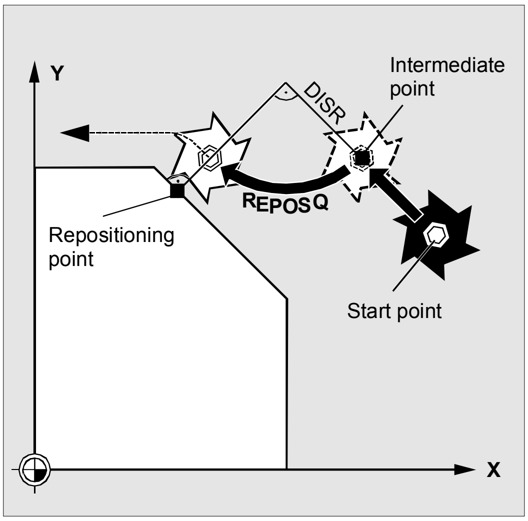
```
REPOSA RMI DISPR=6 F400
```



Approach along 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:

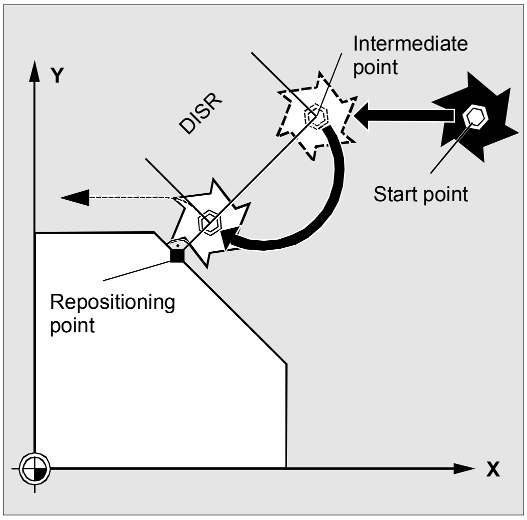
```
REPOSQ RMI DISR=10 F400
```



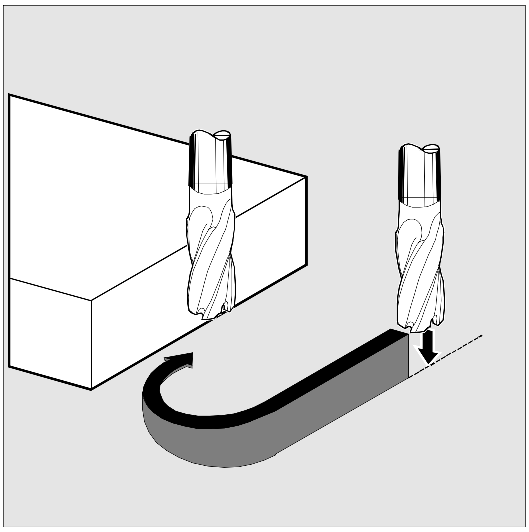
Approach along semi-circle, REPOSH, REPOSHA
The tool approaches the repositioning point along a semi-circle with a diameter of DISR=... The control system automatically calculates the intermediate point between the start and repositioning points.

Example:

```
REPOSH RMI DISR=20 F400
```



The following applies to circular motions REPOSH and REPOSQ:
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.

■

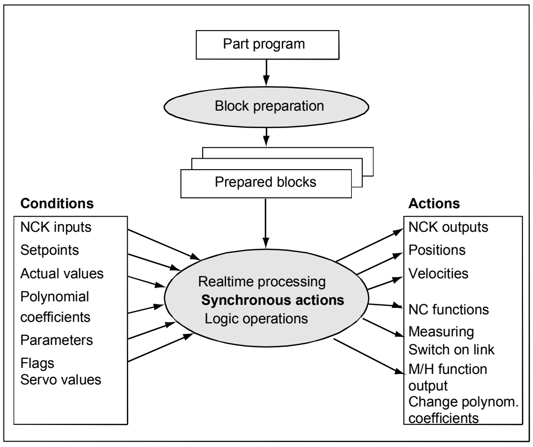
Notes

Motion Synchronous Actions

10.1 Structure, basic information

Function

Synchronized actions allow you to start different actions from the current part program and to execute them synchronously.
The starting point of these actions can be defined with conditions evaluated in real time (in interpolation cycles). The actions are therefore responses to real-time events, execution of them is not limited by block boundaries.
A synchronized action also contains information about the effectiveness of the actions and about the frequency with which the programmed real-time 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.



Programming

```
DO action1 action2 …
VOCABULARY_WORD condition DO action1 action2 …
ID=n VOCABULARY_WORD condition DO action1 action2 …
IDS=n VOCABULARY_WORD condition DO action1 action2 …
```

Explanation

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
Vocabulary word	
`No vocabulary word`	Execution of the action is not subject to any condition. The action is executed cyclically in any interpolation cycles.

`WHEN`	The condition is tested until it is fulfilled once, the associated action is executed once.
`WHENEVER`	The condition is tested cyclically. The associated action is executed cyclically while the condition is fulfilled.
`FROM`	After the condition has been fulfilled once, the action is executed cyclically while the synchronized action is active.
`EVERY`	The action is initiated once when the condition is fulfilled and is executed again when the condition changes from the FALSE state to the TRUE state. The condition is tested cyclically. Every time the condition is fulfilled, the associated action is executed.
`Condition`	Gating logic for real-time variables, the conditions are checked in the interpolation cycle. In **SW 5** and higher, the G codes can be programmed in synchronized actions for condition evaluation.
`DO`	Triggers the action if the condition is fulfilled.
`Action`	Action started if the condition is fulfilled, e.g. assign variable, activate axis coupling, set NCK outputs, output M and H functions, ... In **SW 5** and higher, the G codes can be programmed in synchronized actions for actions/technology cycles.
Coordination of synchronized actions/technology cycles	
`CANCEL[n]`	Cancel synchronized action
`LOCK[n]`	Inhibit technology cycle
`UNLOCK[n]`	Unlock technology cycle
`RESET`	Reset technology cycle

Programming 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.
```

If real-time 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 real-time variables obtained.
The real-time variables used are evaluated in interpolation cycles.

Advantages with synchronized actions:

Preprocessing is not stopped.

Possible applications:

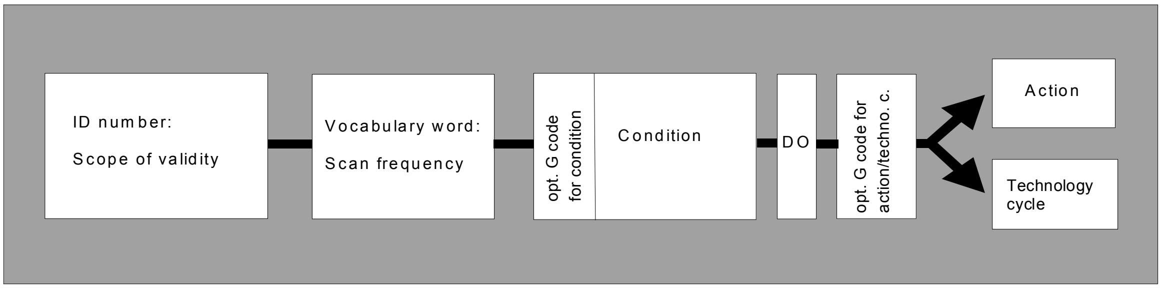
- Optimization of runtime-critical applications (e.g. tool changing)

- Fast response to an external event
- Programming AC controls
- Setting up safety functions
- ....

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 in the next executable block (e.g. traversing movement with G0, G1, G2, G3; block with auxiliary function output).
Synchronized actions consist of up to five command elements each with a different task:



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 (modal ID):

- **No** modal ID
  The synchronized action is active in automatic mode only. It applies only to the next executable block (block with motion instructions 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 is deactivated by CANCEL(n) or by programming a new synchronized action with the same ID.
  The synchronized actions that apply in the M30 block are also still active (if necessary deactivate with the CANCEL command).
  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
  These **static** synchronized actions apply **modally** in **all operating modes**.
  They can be defined not only for starting from a part program but also directly after power-on from an asynchronous subprogram (ASUB) started by the PLC. 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
```

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.

10.1.3 Vocabulary word

Function

The vocabulary word determines how many times the following condition is to be scanned and the associated action executed.

- No vocabulary word:
  If no vocabulary word is programmed, the condition is considered to be always fulfilled. The synchronous commands are executed cyclically.

Example:

```
DO $A_OUTA[1]=$AA_IN[X]
```

;Output of act. val. on analog output

- WHEN
  The condition is scanned in each interpolation cycle until it is fulfilled once, whereupon the action is executed once.
- WHENEVER
  The condition is scanned in each interpolation cycle. The action is executed in each interpolation cycle while the condition is fulfilled.
- FROM
  The condition is tested 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 whenever the condition is fulfilled.

Example:

```
ID=1 EVERY $AA_IM[B]>75 DO
POS[U]=IC(10) FA[U]=900;
```

Pulse edge control:
The action is initiated again when the condition changes from FALSE to TRUE.

When the actual value of axis B overshoots the value 75 in machine coordinates, the U axis should move forwards by 10 with an axial feed.

Condition

Defines whether an action is to be executed by comparing two real-time variables or one real-time variable with an expression calculated during preprocessing.
As of SW 4:
Results of comparisons can also be gated by Boolean operators in the condition ().
The condition is checked in the interpolation cycle.
If it is fulfilled, the associated action is executed.
as of SW 5:
Conditions can be specified with a G code. This allows defined settings to exist for the evaluation of the condition and the action/technology cycle to be executed, independent of the current part program status. It is necessary to separate the synchronized actions from the program environment, because synchronized actions are required to execute their actions at any time from a defined initial state as a result of fulfilled trigger conditions.
Applications:
Definition of the systems of measurement for condition evaluation and action through G codes G70, G71, G700, G710.
In SW 5 only these G codes are allowed.
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.

Programming example

`WHENEVER $AA_IM[X] > 10.5*SIN(45) DO …`	Comparison with an expression calculated during preprocessing
`WHENEVER $AA_IM[X] > $AA_IM[X1] DO …`	Comparison with other real-time variable
`WHENEVER ($A_IN[1]==1) OR ($A_IN[3]==0) DO ...`	Two logic-gated comparisons

Possible conditions:

- Comparison of real-time 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, measured results
- Servo values
- Velocities, axis status

10.1.4 Actions

Function

In each synchronized action, you can program one or more actions. All actions programmed in a block are started in the same interpolation cycle.
In **SW 5** and higher, actions can be used with a G code for the action/technology cycle. 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 after the end of the technology cycle until the next G code.
Only a G code from the G code group (G70, G71, G700, G710) may be programmed.

Possible actions:

- Assign variables
- Write setting data
- Set control parameters
- DELDTG: Delete fast distance-to-go
- RDISABLE: Set read-in disable
- Output of M, S and H auxiliary functions
- STOPREOF: Cancel preprocessing stop
- FTOC: Online tool offset
- Definition of evaluation functions (polynomials)
- SYNFCT: Activate evaluation functions: adaptive control
- Switchover between several feedrates in a programmed block depending on binary and analog signals
- Feedrate overrides
- Start/position/stop positioning axes (POS) and spindles (SPOS)
- PRESETON: Preset actual value memory
- Activate or deactivate coupled-axis motion/leading value coupling
- Measuring
- Set up additional safety functions
- Output of digital and analog signals
- ...

Programming example

Synchronized action with two actions

```
WHEN $AA_IM[Y] >= 35.7 DO M135 $AC_PARAM=50
```

;If the condition is fulfilled, M135 is output to the PLC and the override is set to 50%.

As the action, you can also specify a program (single-axis program, technology cycle). This must only comprise those actions that can also be programmed individually in synchronized actions. The individual actions of such a program are executed sequentially in interpolation cycles.

Note

Actions can be executed whatever mode is selected. The following actions are only active in automatic mode when the program is active

- STOPREOF,
- DELDTG.

10.1.5 Overview of synchronized actions

Up to SW 3.x

- Programming of sequences in the interpolation cycle at the user level (part program)
- Response to events/statuses in the interpolation cycle
- Gating logic in real time
- Access to I/Os, control status and machine status
- Programming of cyclic sequences that are executed in the interpolation cycle
- Triggering of specific NC functions (read-in disable, axially overlaid motion, ...)
- Execution of technology functions in parallel with path motion
- Triggering of technology functions regardless of block boundaries

SW 4 and higher

- Diagnosis possible for synchronized actions
- Expansion of the main run variable used in synchronized actions
- Complex conditions in synchronized actions
- Expansion of expressions in synch. actions: Combination of real-time variables with basic arithmetic operations and functions in the interpolation cycle, indirect addressing of main run variables via index can be changed online
  Setting data from synchronized actions can be modified and evaluated online
- Configuration possibilities: Number of simultaneously active synchronized actions can be set via machine data.
- Start positioning axis motion and spindles from synchronized actions (command axes)
- Preset from synchronized actions
- Activation, deactivation, parameterization of axis coupling: Leading value coupling, coupled-axis motion
- Activation/deactivation of axial measuring function
- Software cam
- Deletion of distance-to-go without preprocessing stop
- Single-axis programs, technology cycles
- Synchronized actions active in JOG mode beyond the boundaries of the program
- Synchronized actions that can be influenced from the PLC
- Protected synchronized actions
- Expansion for overlaid motion / clearance control
- **SW 5.x and higher**
- Travel to fixed stop FXS:
  Synchronized actions, triggered with FXS, FXST and FXSW
- Travel with limited moment/force FOC:
  Synchronized action is activated either modally or non-modally with FOCON and deactivated with FOCOF.

10.2 Basic modules for conditions and actions

Real-time variables
Real-time variables are evaluated and written in the interpolation cycle.
The real-time variables are

- $A... , main run variable,
- $V... , servo variable.

To identify them specially, these variables can be programmed with **$$:**
$AA_IM[X] is equivalent to $$AA_IM[X].
Setting and machine data must be identified with $$ when evaluation/assignment takes place in the interpolation cycle.

A list of variables is given in the Appendix.

Real-time calculations
Calculations in real time are restricted to the data types INT, REAL and BOOL.
Real-time expressions are calculations that can be executed in interpolation cycles that can be used in the condition and the action for assignment to NC addresses and variables.

- **Comparisons**
  In conditions, variables or partial expressions of the same data type can be compared. The result is always of data type BOOL.
  All the usual comparison operators are permissible (==, <>, <, >, <=, >=).
- **Boolean operators**
  Variables, constants and comparisons can be gated using the usual Boolean operators (NOT, AND, OR, XOR)

- **Bit operators**
  The bit operators B_NOT, B_AND, B_OR, B_XOR can be used.
  Operands are variables or constants of the INTEGER type.

- **Basic arithmetic operations**
  Real-time variables of types INTEGER and REAL can be subjected to the basic arithmetic operations, with each other or with a constant (+, -, *, /, DIV, MOD).

- **Mathematical functions**
  Mathematical functions can be applied to real-time variables of data type REAL (SIN, COS, TAN, ASIN, ACOS, ABS, TRUNC, ROUND, LN, EXP, ATAN2, ATAN, POT, SQRT, CTAB, CTABINV).
  Example:

```
    DO $AC_PARAM[3] = COS($AC_PARAM[1])
```

Notes

Only variables of the same data type can be gated.

Correct:	$R10=$AC_PARAM[1]
Incorrect:	$R10=$AC_MARKER[1]

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 (SW 4 and higher).

Example:

```
      DO $AC_PARAM[3] = $A_INA[1]-$AA_IM[Z1]        ;Subtraction of two real-time variables
      WHENEVER $AA_IM[x2] < $AA_IM[x1]-1.9 DO $A_OUT[5] = 1
                                                    ;Subtraction of a constant from real-time variable
      DO $AC_PARAM[3] = $INA[1]-4*SIN(45.7 $P_EP[Y])*R4
                                                    ;Constant expression, calculated during preprocessing
```

- **Indexation**
  Real-time variables can be indexed with real-time variables.

Notes

Variables that are not formed in real time must not be indexed with real-time variables.
Example:

```
      WHEN…DO $AC_PARAM[$AC_MARKER[1]] = 3
```

Illegal:

```
      $AC_PARAM[1] = $P_EP[$AC_MARKER]
```

Programming example

Example of 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
```

10.3 Special real-time variables for synchronized actions

The real-time variables listed below can be used in synchronized actions:

10.3.1 Flags/counters $AC_MARKER[n]

Function

Flag variables can be read and written in synchronized actions.

Channel-specific flags/counters

$AC_MARKER[n]

Data type: INTEGER

A channel-specific flag variable exists under the same name once in each channel.

Example:

```
WHEN ... DO $AC_MARKER[0] = 2
WHEN ... DO $AC_MARKER[0] = 3
WHEN $AC_MARKER == 3 DO $AC_OVR=50
```

10.3.2 Timer variable $AC_TIMER[n], SW 4 and higher

Function

(not 840D NCU 571, FM-NC)

System variable $AC_TIMER[n] permits actions to be started after defined periods of delay.
Data type: REAL
Units: s
n: Number of time variable

- **Set timer**
  A timer variable is incremented via value assignment $AC_TIMER[n]=value
  n: Number of the timer variable
  value: Starting value (usually 0)

- **Halt timer**
  Incrementation of a timer variable is halted by assigning a negative value $AC_TIMER[n]=-1
- **Read timer**
  The current time value can be read when the timer is running or when it has 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
```

10.3.3 Synchronized action parameters $AC_PARAM[n]

Function

Data type: REAL
n: Number of parameter 0-n
Synchronized action parameters $AC_PARAM[n] are used for calculations and as a buffer in the synchronized actions.
The number of available AC parameter variables in each channel is programmed via machine data MD 28254: MM_NUM_AC_PARAM.
These parameters exist once in each channel under the same name. The $AC_PARAM flags are stored in the dynamic memory.

10.3.4 Access to R parameters $Rxx

Function

Data type: REAL
These static variables are used for calculations in the part program etc. They can be addressed in the interpolation cycle by appending **$**.
Examples:

`WHEN $AA_IM[X]>=40.5 DO $R10=$AA_MM[Y]`	Write access to the R parameter 10.
`WHEN $AA_IM[X]>=6.7 DO $R[$AC_MARKER[1]]=30.6`	; Read access to the R parameter whose number is given in flag 1

Notes

Application
The use of R parameters in synchronized actions permits

- 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
- archiving of values determined for synchronized actions

The R parameters must be used either as "normal" arithmetic variables Rxx **or** as real-time variables $Rxx.
If you want the R parameter to be used as a "normal" arithmetic variable again after it has been used in a synchronized action, make sure that the preprocessing stop is programmed explicitly with STOPRE for synchronization of preprocessing and the main run:

Example:

`WHEN $AA_IM[X]>=40.5 DO $R10=$AA_MM[Y]`	Use of R10 in synchronized actions
`G01 X500 Y70 F1000`	
`STOPRE50`	Preprocessor stop
`IF R10>20`	Evaluation of the arithmetic variable

10.3.5 Machine and setting data read/write, SW 4 and higher

Function

From SW 4 and higher, it is possible to read and write the machine and setting data (MD, SD) of synchronized actions.

- **Read fixed MD, SD**
  They are addressed from within the synchronized action in the same manner as in normal part program commands and are preceded by a $ character.
  Example:

```
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 modifiable MD, SD**
  They are addressed from within the synchronized action, preceded by **$$** characters and evaluated in the interpolation cycle.
  Example:

```
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
                machining.
```

- **Write MD, SD**
  Precondition:
  The current setting for access authorization must permit write access. It is not meaningful to change MD and SD from synchronized actions unless the change takes **immediate** effect. The active states are listed for all MD and SD in
  References: /LIS/, Lists
  Addressing:
  The MD and SD to be modified 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.
```

10.3.6 FIFO variable $AC_FIFO1[n] … $AC_FIFO10[n], SW 4 and higher

Function

Data type: REAL
10 FIFO variables (circulating buffer store) are available to store associated data sequences.
Application:

- Cyclical measurement
- Pass execution

Each element can be accessed in read or write mode.
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: Oldest element is read and removed from 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

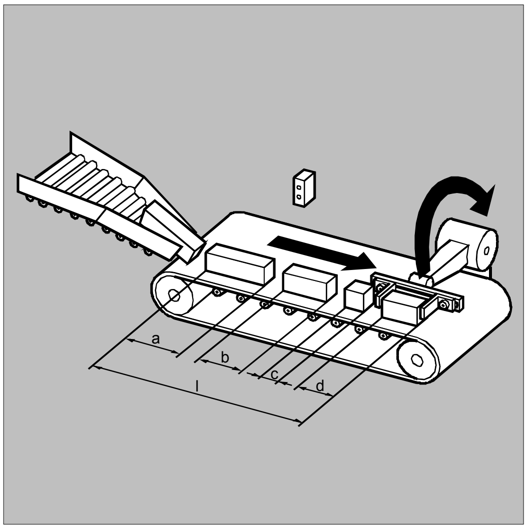
n=5: Current write index relative to start of FIFO

n=6 to 6+n_{max}:
Access to nth FIFO element

Programming example

Circulating memory

During a production run, a conveyor belt is used to transport products of different lengths (a, b, c, d). The conveyor belt of transport length "l" 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]*BETW>=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.

10.3.7 Information about block types in the interpolator (SW 6.4 and 7.1)

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

$AC_BLOCKTYPE **Value:**		**$AC_BLOCKTYPEINFO** **Value:**				
0	Not equal to 0	T	H	Z	E	Meaning:
Original block	Intermediate block					Trigger for intermediate block:
	1 (SW 6.4 and higher)	1	0	0	0	Internally generated block, no further information (as from SW 7.1)
	(SW 7.1 and higher)	(SW 7.1 and higher)				
	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)
						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

$AC_BLOCKTYPE **Value:**		**$AC_BLOCKTYPEINFO** **Value:**				
0	Not equal to 0	T	H	Z	E	Meaning:
Original block	Intermediate block					Trigger for intermediate block:
						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

Notes

$AC_BLOCKTYPEINFO always contains the value for the block type in the thousands digit (T) in case there is an intermediate block. ($AC_BLOCKTYPE is not equal to 0)

T	Thousands digit
H	Hundreds digit
Z	Tens digit
E	Units digit

in $AC_BLOCKTYPEINFO

$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

Programming example

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 Auxiliary functions output

Function

If the conditions are fulfilled, up to 10 M, H and S functions can be output per machining block.
Auxiliary function output is activated using the action codeword "DO".
The auxiliary functions are output **immediately** in the interpolation cycle. The output timing defined in the machine data for auxiliary functions is not active. The output timing is determined 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
```

Sequence

Auxiliary functions must only be programmed with the vocabulary words WHEN or EVERY in non modal synchronized actions (without model ID).
Whether an auxiliary function is active or not is determined by the PLC, e.g. via NC start.

Notes

Not possible from a motion synchronized action:

- 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

Programming 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.

10.4.2 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.

At the beginning of the block with RDISABLE, exact positioning is always triggered whether RDISABLE is active or not.

Programming example

Start the program in interpolation cycles dependent on external inputs.

`...`	
`WHENEVER $A_INA[2]<7000 DO RDISABLE`	;If the voltage 7V is exceeded 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`	
`...`	

10.4.3 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.

Notes

STOPREOF must be programmed with the vocabulary word WHEN and non modally (without ID number).

Programming 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.

10.4.4 Deletion of distance-to-go

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
- Delete distance-to-go without preparation (SW 4.3 and higher)

10.4.5 Delete distance-to-go with preparation, DELDTG, DELDTG ("Axis 1 to x")

Notes

The axis designation contained in brackets behind DELDTG is only valid for one positioning axis.

Function

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.

Sequence

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.

Programming example

Rapid delete 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…
```

Programming example

Rapid axial delete distance-to-go

Stopping a programmed 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 voltage on input 5 exceeds 8V, delete distance-to-go for axis X1.
                  Path motion continues.
POS[X1]=100 FA[X1]=10 G1 Z100 F1000
```

Restrictions

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, block-synchronized

Programming

```
FCTDEF(Polynomial_No.,LLIMIT,ULIMIT,a0,a1,a2,a3)
```

Explanation

`Polynomial_No.`	Number of the 3rd degree polynomial
`LLIMIT`	Lower limit for function value
`ULIMIT`	Upper limit for function value
a_0, a_1, a_2, a_3	Polynomial coefficient

Function

FCTDEF allows 3rd degree polynomials to be defined as $y=a_0+a_1 \cdot x+a_2 \cdot x^2+a_3 \cdot x^3$. These polynomials are used by the online tool offset FTOC and the evaluation function SYNFCT to calculate function values from the main run variables (real-time variables).

The polynomials are defined either block-synchronized with the function FCTDEF or 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
`n`	Number of the polynomial

Notes

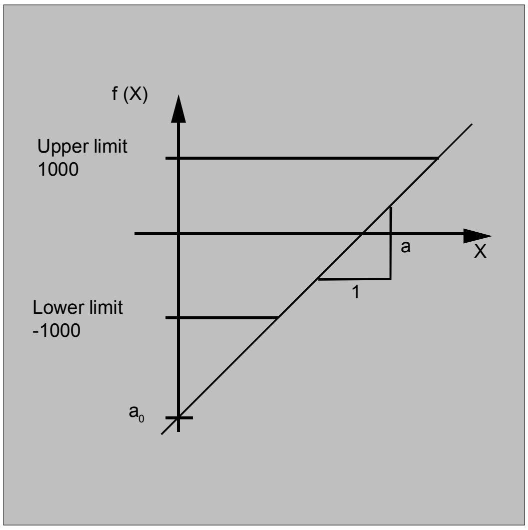
- The system variables can be written from the parts program or from a synchronized action. When writing from parts programs, program STOPRE to ensure that writing is block synchronized.
- As of SW 4:
  The system variables $AC_FCTLL[n], $AC_FCTUL[n], $AC_FCT0[n] to $AC_FCTn[n] can be modified from within synchronized actions
  (not SINUMERIK FM-NC,
  not SINUMERIK 840D with NCU 571).

When writing form synchronized actions, the polynomial coefficients and function value limits are active immediately.

Programming example

Polynomial for straight section:

With upper limit 1000, lower limit -1000, ordinate section a_0=$AA_IM[X] and linear gradient 1 the polynomial is:



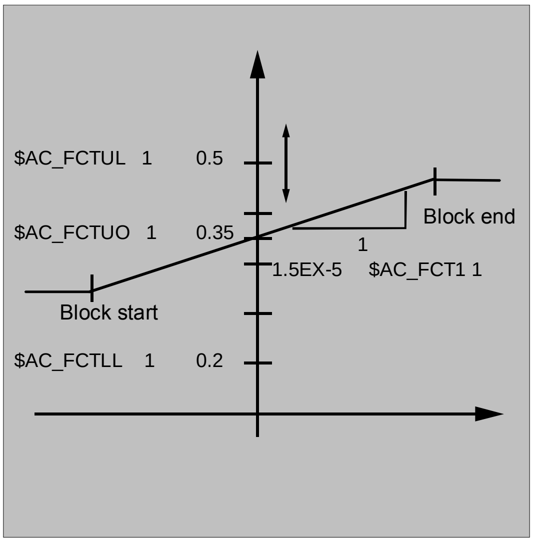
```
FCTDEF(1, -1000,1000,$AA_IM[X],1)
```

10.4.7 Laser power control

Programming example

Polynomial definition using variables

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.

10.4.8 Evaluation function SYNFCT

Programming

```
SYNFCT (Polynomial_No., Realtime variable output, Realtime variable
input)
```

Explanation

`Polynomial_No.`	With polynomial defined with FCTDEF (see Subsection "Polynomial definition").
`Real-time variable output`	Write real-time variables
`Real-time variable input`	Read real-time variable

Function

SYNFCT reads real-time variables in synchronism with execution (e.g. analog input, actual value, ...) and uses them to calculate function values up to the 3rd degree (e.g. override, velocity, axis position, ...) using an evaluation polynomial (FCTDEF). The result is output in to real-time variables and subjected to upper and lower limits with FCTDEF (see Subsection 10.4.7).

For the real-time variable output, it is possible to select variables that

- with additive influencing
- with multiplicative influencing
- as a position offset or
- directly

into the machining process.

Application

The evaluation function is used

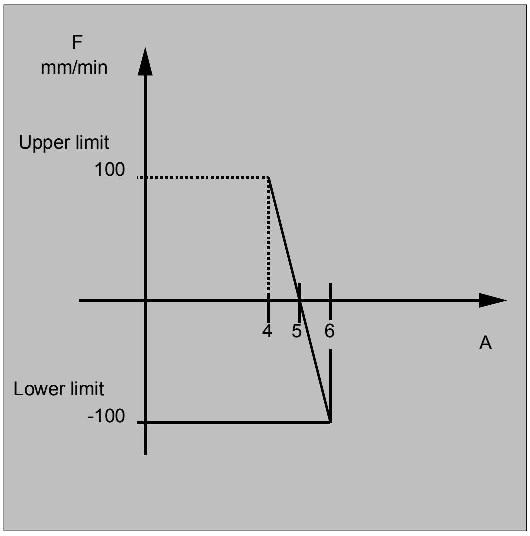
- in AC control (adaptive control)
- in laser output control
- with position feedforward

10.4.9 Adaptive control (additive)

Programming example

Additive influence on the programmed feedrate

A programmed feedrate is to be controlled by adding 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

$a_2 = a_3 = 0$ (not quadratic or cubic element)

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.

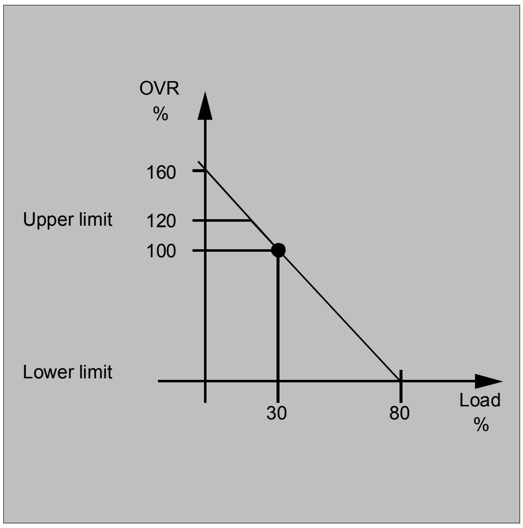
10.4.10 Adaptive control (multiplicative)

Programming example

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 (not quadratic or cubic element)

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])
```

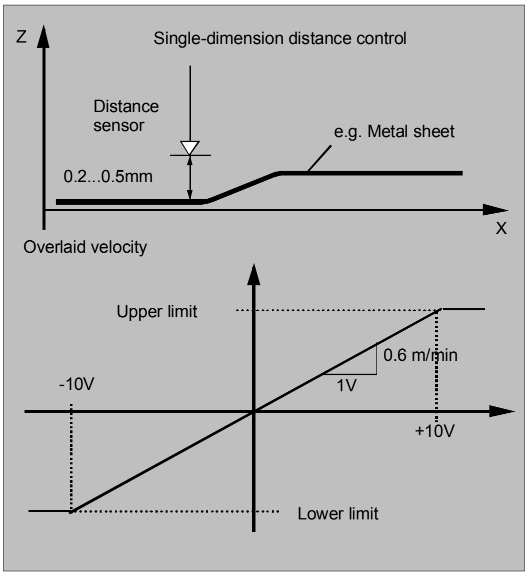
;Read the current axis load (% of the max. drive current) via $AA_LOAD[x], calculate the feedrate override with the polynomial defined above.

10.4.11 Clearance control with limited compensation

Programming example

Integrating calculation of the distance values with boundary check
$AA_OFF_MODE = 1
Important:
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:
Restriction of the velocity of the overlying interpolator with MD 32020: JOG_VELO with an interpolation cycle of 12 ms:
Velocity:

$$\frac{0.120mm}{0.6ms}/mV \quad 0.6\frac{m}{\min}/V$$



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 DO $AA_OVR[X] = 0	Disable axis X when limit value is 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	

Notes

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.
In SW 4 and later, it is possible to limit the value to be compensated absolutely (real-time variable output) to the value stored in setting data SD 43350: AA_OFF_LIMIT.
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

10.4.12 FTOC (online tool offsets)

Programming

```
FTOC(Polynomial_No., RV, Length1_2_3 or Radius4,
channel, spindle)
```

Explanation

Polynomial_No.	For polynomial defined with FCTDEF, see Subsection "Polynomial definition" in this Section.
RV	Real-time variable for which a function value for the specified polynomial is to be calculated.
Length1_2_3 Radius4	Length compensation ($TC_DP1 to 3) or radius compensation 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.

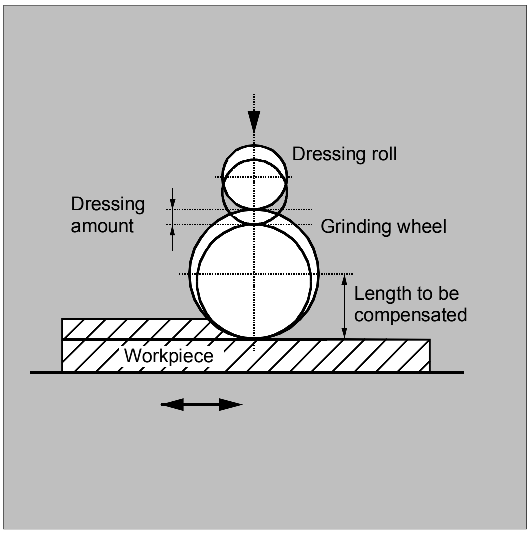
Function

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.
This means that you can also program modal, Online tool compensations or clearance controls as synchronized actions.

Application
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 Chapter 5 "Tool Offsets".

Programming 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
...	

10.4.13 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**.

In SW 5 and higher, G codes G70/G71/G700/G710 can be programmed in synchronized actions. They can be used for defining the measuring system for positioning tasks in synchronized actions.

References: /PG/ Chapter 3 "Specifying paths"
/FBSY/ "Starting Command Axes"

The measuring system is defined using 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.

Example 1

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=10mm	X=10mm
`N160 G70 Z10 F10`	Z=254mm	X=254mm
`N170 G71 Z10 F10`	Z=10mm	X=10mm
`N180 M30`		

Example 2

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=10mm	X=10mm
`N160 G70 Z10 F10`	Z=254mm	X=10mm (X is always to 10mm)
`N170 G71 Z10 F10`	Z=10mm	X=10mm
`N180 M30`		

Programming example

Disabling of a programmed axis motion

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
```

10.4.14 Position axis POS

Function

```
POS[axis]=value
```

Unlike programming from the parts program, the positioning axis movement has no effect on execution of the parts program.

Explanation

`Axis:`	Name of the axis to be traversed
`Value:`	The value to traverse by (depending on traverse mode)

Programming 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 real-time variables.

10.4.15 Start/stop axis MOV

Programming

```
MOV [axis]=Value
```

Explanation

`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

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 example

`... DO MOV[U]=0`	Axis U is stopped

Note

If an indexing axis is stopped with MOV[Axis]=0, the axis is halted at the next indexing position.

10.4.16 Axial feed: FA

Programming example

FA[axis]=feedrate

```
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 real-time 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 in the motion block 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.

Programming example

Move the X axis from either the parts program or the synchronized actions:

`N10 G01 **X100** Y200 F1000`	X axis programmed in part program
…	

`N20 ID=1 WHEN $A_IN[1]==1 DO`
`**POS[X]=150** FA[X]=200`
	Starting positioning from the synchronized action if a digital input is set

…	
`CANCEL(1)`	Select 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.

Programming example

Change 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 Preset actual value memory

Function

When PRESETON (axis, value) is executed, the current axis position is not changed but a new value is assigned to it.

Notes

PRESETON from synchronized actions can be programmed for

- modulo rotary axes that have been started from the part program and
- all command axes that have been started from a synchronized action.

Restriction:
PRESETON cannot be programmed for axes which are involved in a transformation.

Programming example

```
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.

Restrictions

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.

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.

Programming example

Start/stop/position 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

Sequence of execution

If conflicting commands are issued for a spindle via simultaneously active synchronized actions, the most recent spindle command takes priority.

Programming example

Set direction and speed of rotation/
position 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) DO SPOS=0`	Position spindle

10.4.21 Coupled motion: TRAILON, TRAILOF

Function

`DO TRAILON(following axis, leading axis, coupling factor)`	Activate coupled-axis motion
`DO TRAILOF(following axis, leading axis, leading axis 2)`	Deactivate coupled-axis motion

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".

Activate unsynchronized coupled motion:

`... DO` **`TRAILON`**`(FA, LA, Kf)`

where:		
	FA:	Following axis
	LA:	Leading axis
	CF:	Coupling factor

Deactivate unsynchronized coupled motion:

`... DO` **`TRAILOF`**`(FA, LA, LA2)`

where:		
	FA:	Following axis
	LA:	Leading axis
	LA2:	Leading axis 2, option

Programming example

`$A_IN[1]==0 DO TRAILON(Y,V,1)`	Activate 1st combined axis pair when digital input is 1
`$A_IN[2]==0 DO TRAILON(Z,W,-1)`	Activate 2nd coupled axis grouping
`G0 Z10`	Infeed Z and W axes in oppositeaxial directions
`G0 Y20`	Infeed of Y and V axes in same axis directions
`...`	
`G1 Y22 V25`	Overlay a dependent and an independent motion of coupled axis "V"
`...`	
`TRAILOF (Y,V)`	Deactivate 1st coupled axis grouping
`TRAILOF (Z,W)`	Deactivate 2nd coupled axis grouping

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 10.1).
To avoid conflict situations the following procedure should be followed.

```
...
N50 WHEN TRUE DO TRAILOF(X, Y)
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.

Activate axial leading value coupling:

...DO **LEADON**(FA,LA,NR)	where:	FA:	Following axis
		LA:	Leading axis
		NR:	Number of the stored curve table

Deactivate axial leading value coupling:

...DO **LEADOF**(FA,LA)	where:	FA:	Following axis
		LA:	Leading axis

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)
```

Programming example

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

For the purpose of this example, it is assumed that the cutting tool infeed is 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_SHEARS1_MPF
;$PATH=/_N_WCS_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 pos. 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 $R1==1 DO MOV[X]=1        ; Set extruded material axis continuously in
FA[X]=$R4                                 motion
N2200 M30
```

10.4.23 Measuring

Compared with use in traverse blocks of the parts program, the measuring function can be activated and deactivated as required.

- Axial measurement without deletion of distance-to-go:

```
MEAWA[axis]=(mode, trigger_event_1, ..._4
```

- Continuous measurement without deletion of distance-to-go:

```
MEAC[axis]=(mode, trigger_event_1, ..._4
```

For further information on measuring: See Chapter 5, "Extended Measuring Function"

10.4.24 Set/clear wait marks: SETM, CLEARM (SW 5.2 and higher)

Function

`SETM(MarkerNumber)`	Set wait marker for channel
`CLEARM(MarkerNumber)`	Clear wait marker for channel

In synchronized actions, wait markers can be set or deleted for the purpose of coordinating channels, for example.

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

10.4.25 Fault responses

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

Programming 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.
```

10.4.26 Travel to fixed stop FXS and FOCON/FOCOF

Explanation

FXS and FOC in synchronized actions

`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. The following is permissible: – Geometry axis identifier – Channel axis identifier – Machine axis identifier

Function

The commands for **travel to fixed stop** are programmed in synchronized actions/technology cycles with the parts program commands FXS, FXST and FXSW.
Activation can take place without movement; the torque is immediately limited. As soon as the axis is moved via a setpoint, the limit stop monitor is activated.
Travel with limited torque/force (FOC):
This function allows torque/force to be changed at any time via synchronized actions and can be activated modally or non-modally.

Notes

Multiple selection
A selection may only be carried out once. If incorrect programming activates the function again although it has already been activated (FXS[axis]=1), alarm 20092 "Travel to fixed stop still active" is output.
Programming code that scans $AA_FXS[] or a separate flag (here R1) in the condition will ensure that the function is not activated more than once.
Parts program extract:

```
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.
Programming example:

```
N10 G0 G90 X0 Y0
N20 WHEN $AA_IW[X] > 17 DO FXS[X]=1      ; If X reaches a position greater than 17mm
N30 G1 F200 X100 Y110                    ; FXS is activated
```

Static and block-related synchronized actions:
The same commands FXS, FXST and FXSW can be used in static and block-related synchronized actions as in normal parts program execution. The values assigned can be resulted from a calculation.

Programming example

Travel to fixed stop (FXS)

Triggered by a synchronized action

```
Y axis:                                           ; Activate static synchronized actions:
N10 IDS=1 WHENEVER (($R1==1) AND                  ; By setting $R1=1, FXS is activated for
            ($AA_FXS[y]==0)) DO                   ; axis Y, the effective torque is reduced to
            $R1=0 FXS[Y]=1 FXST[Y]=10             ; 10% and a traverse motion is initiated
            FA[Y]=200 POS[Y]=150                  ; in the direction of the fixed stop.
N11 IDS=2 WHENEVER ($AA_FXS[Y]==4) DO             ; As soon as the fixed stop is detected
            FXST[Y]=30                            ; ($AA_FXS[Y]==4), torque is increased
                                                  ; to 30%
N12 IDS=3 WHENEVER ($AA_FXS[Y]==1) DO             ; After the fixed stop is reached, torque
            FXST[Y]=$R0                           ; is controlled by R0
N13 IDS=4 WHENEVER (($R3==1) AND                  ; Deselection according to
            ($AA_FXS[Y]==1)) DO                   ; 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 path group
```

Programming example

Activate 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
```

10.4.27 Determining s in synchronous actions

Function

The system variable $AC_TANEB (**T**angent **AN**gel at **E**nd of **B**lock) readable in synchronized actions defines the angle between the path tangent in the end point of the current block and the path tangent in the start point of the programmed following block.

The tangent angle is always output positive in the range 0.0 to 180.0 degrees. If there is not following block in the main run, the angle -180.0 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).

Programming example

```
ID=2 EVERY $AC_BLOCKTYPE==0 DO $R1 = $AC_TANEB;
```

10.4.28 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

Resulting override can be calculated as:
$AA_OVR * $AA_PLC_OVR or
$AC_OVR * $AC_PLC_OVR

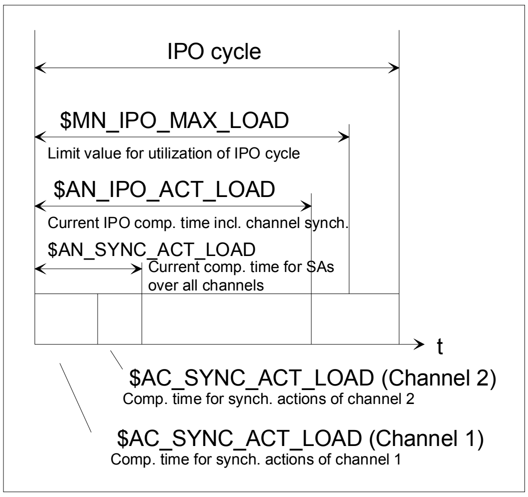
10.4.29 Time use evaluation of synchronized actions

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.
The variables only have valid values if the machine data $MN_IPO_MAX_LOAD are greater than 0. Otherwise, the variables always return the gross computing time. This results from:

- synchronized action time,
- position control time and
- remaining IPO computing time

The variables always contain the values of the **previous** IPO cycle.


IPO cycle
$MN_IPO_MAX_LOAD
Limit value for utilization of IPO cycle
$AN_IPO_ACT_LOAD
Current IPO comp. time incl. channel synch.
$AN_SYNC_ACT_LOAD
Current comp. time for SAs over all channels
t
$AC_SYNC_ACT_LOAD (Channel 2)
Comp. time for synch. actions of channel 2
$AC_SYNC_ACT_LOAD (Channel 1)
Comp. time for synch. actions of channel 1

$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

Overload notification:

MD $MN_IPO_MAX_LOAD is used to set the gross IPO computing time (in % 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 MD is 0, the entire diagnostic function is deactivated. Evaluation of $AN_IPO_LOAD_LIMIT allows the user to define a strategy for avoiding level overflow.

System variables:

$AN_SERVO_MAX_LOAD	longest computing time of the position controller
$AN_SERVO_MIN_LOAD	shortest computing time of the position controller
$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_SYNC_MAX_LOAD	longest computing time for synchronized actions over all channels
$AC_SYNC_MAX_LOAD	longest computing time for synchronized actions in the channel

can be written from synchronized actions. These variables are reset to the current load on each write access, irrespective of the value written to them.

Programming example

$MN_IPO_MAX_LOAD = 80	MD: Time use limit for IPO cycle As soon as $AN_IPO_LOAD_PERCENT > 80 %, $AN_IPO_LOAD_LIMIT is set to TRUE.

```
N01   $AN_SERVO_MAX_LOAD=0
N02   $AN_SERVO_MIN_LOAD=0
N03   $AN_IPO_MAX_LOAD=0
N04   $AN_IPO_MIN_LOAD=0
N05   $AN_SYNC_MAX_LOAD=0
N06   $AC_SYNC_MAX_LOAD=0
N10   IDS=1 WHENEVER $AN_IPO_LOAD_LIMIT == TRUE DO M4711 SETAL(63111)
N20   IDS=2 WHENEVER $AN_SYNC_TO_IPO > 30 DO SETAL(63222)
N30   G0 X0 Y0 Z0
...
N999  M30
```

The first synchronized action generates an auxiliary function output and an alarm, if the entire time use limit is exceeded.
The second synchronized action generates an alarm if the share that the synchronized action has of the IPO computing time (over all channels) exceeds 30 %.

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. As far as the user is concerned, they are called up like subroutines. Parameter transfer is not possible.

It is possible to process several technology cycles or actions in parallel in one channel.

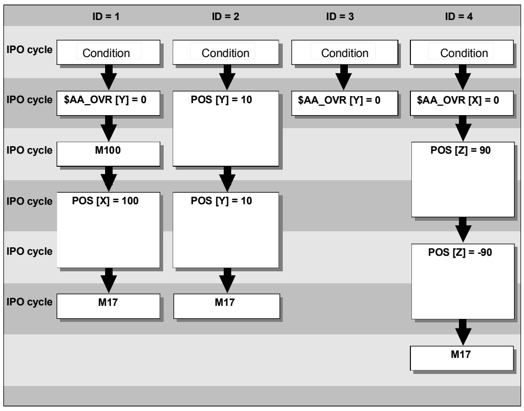
End of program is programmed with M02 / M17 / M30 / RET. A maximum of one axis movement per block can be programmed.

Application

Technology cycles as axis programs: Each technology cycle controls only one axis. In this way, different axis motions can be started in the same interpolation cycle under event control. The parts program is now only used for the management of synchronized actions in extreme cases.

Programming example

Axis programs are started by setting digital inputs.



Main program:

`ID=1 EVERY $A_IN[1]==1 DO AXIS_X`	If input 1 is at 1, axis program X starts
`ID=2 EVERY $A_IN[2]==1 DO AXIS_Y`	If input 2 is at 1, axis program Y starts
`ID=3 EVERY $A_IN[3]==1 DO $AA_OVR[Y]=0`	If input 3 is at 1, the override for axis Y is at 0
`ID=4 EVERY $A_IN[4]==1 DO AXIS_Z`	If input 4 is at 1, axis program Z starts
`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
```

Technology cycles are started as soon as their conditions have been fulfilled. Several IPO cycles are required to execute positioning axes. Other functions (OVR) are executed in one cycle.
In the technology cycle, blocks are executed in sequence.

Notes

If actions are called in the same interpolation cycle that are mutually exclusive, the action is started that is called from the synchronized action with the higher ID number.

10.5.1 Lock, unlock, reset: LOCK, UNLOCK, RESET

Programming

`LOCK (n, n, ...)`	Lock technology cycle, the active action is interrupted
`UNLOCK (n, n, ...)`	Unlock technology cycle
`RESET (n, n, ...)`	Reset technology cycle, the active action is interrupted
`n`	Identification number of the synchronized action

Function

Execution of a technology cycle can be locked, unlocked or reset from within a synchronized action or from a technology cycle.

Lock technology cycle, LOCK

Technology cycles can be locked using LOCK from another synchronized action or from a technology cycle.

Example:

```
N100 ID=1 WHENEVER $A_IN[1]==1 DO M130
...
N200 ID=2 WHENEVER $A_IN[2]==1 DO LOCK(1)
```

Unlock technology cycle, UNLOCK

Locked technology cycles can be unlocked again from another synchronized action/technology cycle with UNLOCK. With UNLOCK, this is continued at the current position, this also applies to an interrupted positioning procedure.

Example:

```
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)
```

Reset technology cycle, RESET

Technology cycles can be reset using RESET from another synchronized action or from a technology cycle.

Example:

```
N100 ID=1 WHENEVER $A_IN[1]==1 DO M130
...
N200 ID=2 WHENEVER $A_IN[2]==1 DO RESET(1)
```

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.

Notes

A programmed synchronized action is active as standard and can be protected against overwriting/locking by a machine data setting.

Application:
It should not be possible for end customers to modify synchronized actions defined by the machine manufacturer.

10.6 Cancel synchronized action: CANCEL

Programming

`CANCEL(n,n,...)`	Cancel synchronized action
`n`	Identification number of the synchronized action

Explanation

Modal synchronized actions with the identifier ID(S)=n can only be canceled directly from the parts program with CANCEL.

Example:

`N100 ID=2 WHENEVER $A_IN[1]==1 DO M130`	
`...`	
`N200` **`CANCEL(2)`**	Cancel synchronized action No. 2

Notes

Incomplete movements originating from a canceled synchronized action are completed as programmed.

10.7 Restrictions

- **Power ON**
  With Power ON no synchronized actions are active.
  However, static synchronized actions can be activated on Power On with an asynchronized subroutine (ASUB) started by the PLC.

- **Mode change**
  Synchronized actions activated with the vocabulary word IDS remain active following a changeover 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**
  With NC reset, all actions started by synchronized actions are stopped. Static synchronized actions remain active. They can start new actions.
  The **RESET** command can be used from the synchronized action or from a technology cycle to reset a modally active synchronized action. If a synchronized action is reset while the positioning axis movement that was activated from it is still active, the positioning axis movement is interrupted.
  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	$MA_SPIND_ACTIVE_AFTER_RESET==1: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==0: Spindle stops
Leading value coupling	$MC_RESET_MODE_MASK, bit13 == 1: Leading value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: Guide value coupling is separated	$MC_RESET_MODE_MASK, bit13 == 1: Leading value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: Guide 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 on 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** are reset	**Static (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 Spindle remains active on mode change	Spindle remains active
Leading value coupling	$MC_RESET_MODE_MASK, bit13 == 1: Leading value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: Guide 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 found during a block search are collected 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 asynchronized subroutine**
  ASUB start:
  Modal and static motion-synchronized actions remain active and are also active in the asynchronized subroutine.

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**
  On repositioning REPOS, the synchronized actions that were active in the interrupted block are reactivated.
  Modal synchronized actions changed from the asynchronized subroutine are not active after REPOS when the rest of the block is executed.
  Polynomial coefficients programmed with FCTDEF are not affected by asynchronized 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 movements are terminated in accordance with programming.
  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:

```
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

11.1 Asynchronous oscillation

Explanation of the commands

`OSP1 [axis]=`
`OSP2 [axis]=`
	Position of reversal point 1
	Position of reversal point 2

`OST1 [axis]=`
`OST2 [axis]=`
	Stopping time at reversal points in seconds

`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

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.

The oscillating axis

For the oscillating axis, the following applies:

- Every axis may be used as an oscillation axis.
- Several oscillation axes can be active at the same time (but no more than the number of positioning axes).
- Linear interpolation G1is always active for the oscillating axis – irrespective of the G command currently valid in the program.

The oscillating axis can

- act as an input axis for 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 for positioning axes).

Oscillation reversal points

The current offsets must be taken into account when oscillation positions are defined:

- Absolute specification

OSP1[Z]=value
Position of reversal point = sum of offsets + programmed value

- Relative specification

OSP1[Z]=IC(value)
Position of reversal point = reversal point 1 + programmed value

Example:

```
N10 OSP1[Z]=100 OSP2[Z]=110
.
.
N40 OSP1[Z]=IC(3)
```

Properties of asynchronized oscillation

- Asynchronized oscillation is active beyond block limits on an axis-specific basis.
- Block-oriented activation of the oscillation movement is ensured by the part program.
- Combined interpolation of several axes and superimposing of oscillation paths are not possible.

Setting data

The setting data necessary for asynchronous oscillation can be set in the part program.

If the setting data are described directly in the program, the change takes effect during preprocessing. A synchronized response can be achieved by means of a STOPRE.

Example:

Oscillation with online change of reversal position

```
$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
```

Notes on individual functions

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.

Activating/deactivating oscillation: OS

`OS[axis] = 1`: Activate

`OS[axis] = 0`: Deactivate

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.

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

Note

Oscillation with motion-synchronous action and stopping times "OST1/OST2".
When the stopping times have elapsed, the internal block change takes place during oscillation (visible at the new residual paths of the axes).
When block change has been completed, the deactivation function is checked. During checking, the deactivation function is defined according to the control setting for the "OSCTRL" sequence of motions.

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.
The impression created is that the deactivation response changes. 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.

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 = 0: Path overlay is active 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:

```
OSCTRL[Z] = (1+4,16+32+64)
```

Programming example

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. At the end of the machining operation, 3 spark-out strokes must be executed and end position 200 approached with the oscillating axis.
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 ->`
`-> OSE[Z]=200 ->`
`-> OST1[Z]=0 OST2[Z]=-1 ->`
 	
`-> FA[Z]=250 FA[X]=1 ->`
`-> OSCTRL[Z]=(4,0) ->`
`-> OSNSC[Z]=3 ->`
`N60 OS[Z]=1`
	Reversal point 1, reversal point 2
	End position
	Stopping time at U1: Exact stop fine;
	Stopping time at U2: Exact stop coarse
	Feed for oscillating axis, infeed axis
	Setting options
	Three spark-out strokes
	Start oscillation

`N70 WHEN $A_IN[3]==TRUE ->`
`-> DO DELDTG(X)`
	Deletion of distance-to-go

`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.

11.2 Control oscillation via synchronized actions

Programming

1. **Define parameters for oscillation**
2. **Define motion-synchronous actions**
3. **Assign axes, define infeed**

Define parameters for oscillation

`OSP1[oscillating axis]=`	Position of reversal point 1
`OSP2[oscillating axis]=`	Position of reversal point 2
`OST1[oscillating axis]=`	Stopping time at reversal point 1 in seconds
`OST2[oscillating axis]=`	Stopping time at reversal point 2 in seconds
`FA[OscillationAxis]=`	Feed for oscillating axis
`OSCTRL[OscillationAxis]=`	Set or reset options
`OSNSC[oscillating axis]=`	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 Chapter 3)
`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 ...

Control oscillation via synchronized actions

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 been finished executing.

Sequence

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 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**.

Assignment of oscillating and infeed axes: OSCILL

```
OSCILL[oscillating axis] = (infeed axis1, infeed axis2, infeed axis3)
```

The axis assignments and the start of the oscillation movement are defined with the OSCILL command.

Up to 3 infeed axes can be assigned to an oscillating axis.

Before oscillation starts, the synchronization conditions must be defined for the behavior of the axes.

Define infeeds: POSP

```
POSP[infeed axis] = (end position, part section, 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 2 equal steps (default setting).
Mode = 1	All partial infeeds are of equal size. They are calculated from the complete infeed.

The 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

In individual cases, the synchronization conditions can be programmed differentially.

Vocabulary words

`WHEN … DO …`	when ... , do ...
`WHENEVER … DO`	whenever ... , do ...

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

You will find more information on synchronized motion actions in Section 11.3.

Infeed in reversal area

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_REVERSE_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_REVERSE_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 1:

```
WHENEVER $AA_IM[Z]<>$SA_OSCILL_REVERSE_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 2:

For reversal point 2:

```
WHENEVER $AA_IM[Z]<>$SA_OSCILL_REVERSE_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%.

Stopping oscillating movement at 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 1:

```
WHENEVER $SA_IM[Z]==$SA_OSCILL_REVERSE_POS1[Z]DO $AA_OVR[Z]=0 ->
-> $AA_OVR[X] = 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 2:

```
WHENEVER $SA_IM[Z] ==$SA_OSCILL_REVERSE_POS2[Z]DO $AA_OVR[Z]= 0 ->
-> $AA_OVR[X]=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.

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 is
equal to	zero,
then	set the axial override of the oscillating axis to 100%.

Next partial infeed

When infeed is complete, a premature start of the next partial infeed must be inhibited. A channel-specific marker (`$AC_MARKER[Index]`) is used for this purpose. It is enabled at the end of the partial infeed (partial distance-to-go ≡ 0) and deleted when the axis leaves the reversal area. The next infeed movement is then prevented by a synchronized action.

On the basis of the given assumptions, the following instructions apply for reversal point 1:

1. Set marker

```
WHENEVER $AA_DTEPW[X] == 0 DO $AC_MARKER[1]=1
```

Whenever	the distance-to-go for the partial infeed on infeed axis X in the WCS is
equal to	zero,
then	set the marker with index 1 to 1.

2. Clear marker

```
WHENEVER $AA_IM[Z]<>$SA_OSCILL_REVERSE_POS1[Z] D0 $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.

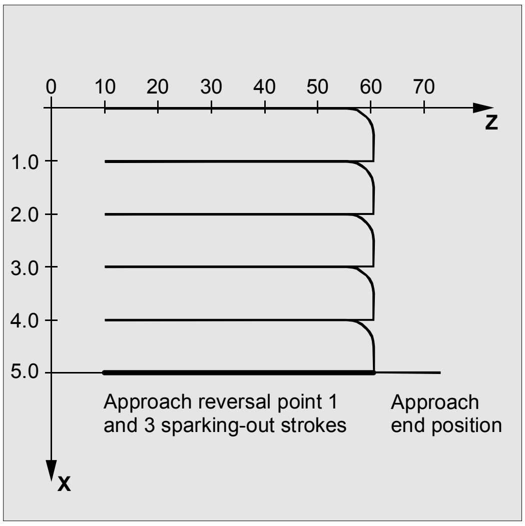
3. Inhibit infeed

```
WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0
```

Whenever	marker 1 is
equal to	1,
then	set the axial override of the infeed axis to 0%.

Programming 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..



Program section

1. Define 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`	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 actions

```
WHENEVER $AA_IM[Z]<$SA_OSCILL_REVERSE_POS2[Z]-ii2 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	0,
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	1,
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 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

■

Notes

Punching and Nibbling

12.1 Activation, deactivation

12.1.1 Language commands, SPOF, SON, PON, SONS, PONS, PDELAYON/OF

Programming

```
PDELAYON
PON G... X... Y... Z...
PONS G... X... Y... Z...
PDELAYOF
SON G... X... Y... Z...
SONS G... X... Y... Z...
SPOF
PUNCHACC(Smin, Amin, Smax, Amax)
```

Explanation of the 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 greater than A_{max}
• "A_{max}"	The end acceleration A_{max} can be less than A_{min}

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 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.

As from **SW 7.1**, 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 NC command PUNCHACC(S_{min}, A_{min}, S_{max}, A_{max}) specifies an acceleration characteristic that defines different accelerations (A), depending on the hole spacing (S). Example for PUNCHACC(2, 50, 10, 100)

Distance between holes less than 2mm:
Traversal acceleration is 50% of maximum acceleration.

Distance between holes from 2mm to 10mm:
Acceleration is increased to 100%, proportional to the spacing.

Distance between holes more than 10mm:
Traverse at an acceleration of 100%.

Punching with delay ON
PDELAYON effects 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.

Stroke initiation

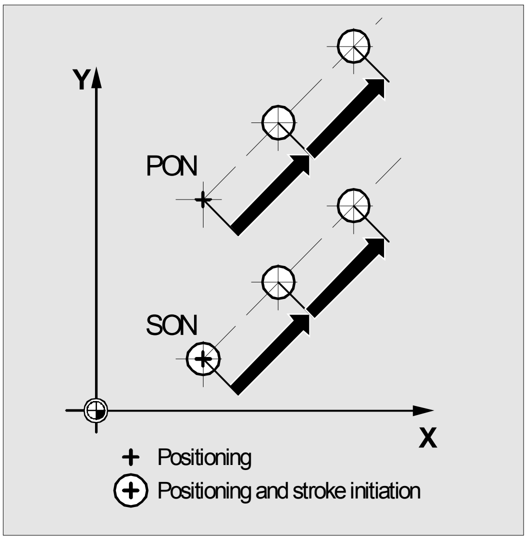
Initiation of the first stroke
The instant at which the first stroke is initiated after activation of the function differs depending on whether nibbling or punching is selected:

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.

Other Information

Machining with rotatable tools
Use the tangential control function if you wish to position rotatable tools at a tangent to the programmed path.

12.1.2 Use of M commands

By using macro technology, you can also use M commands instead of language commands:

DEFINE M22 AS SON	Nibbling ON
DEFINE M122 AS SONS	Nibbling with leader on
DEFINE M25 AS PON	Punching ON
DEFINE M125 AS PONS	Punching 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

12.2 Automatic path segmentation

Programming

```
SPP=
SPN=
```

Explanation

SPP	Size of path section (maximum distance between strokes); modal
SPN	Number of path sections per block; non-modal

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.

12.2.1 Path segmentation for path axes

Sequence

Length of SPP path segment
With the SPP command, you specify the maximum distance between strokes and thus the maximum length of the path segments into which the total traversing distance is to be divided.

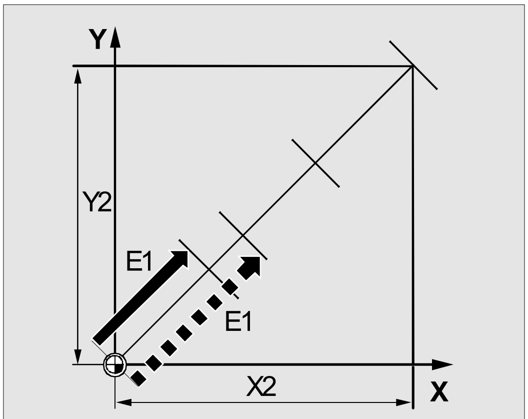
The command is deactivated with SPOF or SPP=0.

Example:
N10 G1 SON X0 Y0
N20 **SPP=2** X10

In this example, the total traversing distance of 10mm is divided into 5 path segments of 2mm (SPP=2) each.

The path segments effected by SPP are always equidistant, i.e. all segments are equal in length.
In other words, the programmed path segment size (SPP setting) is valid only if the quotient of the total traversing distance and the SPP value is an integer. If this is not the case, the size of the path segment is reduced internally such as to produce an integer quotient.

Example:
N10 G1 G91 SON X10 Y10
N20 SPP=3.5 X15 Y15

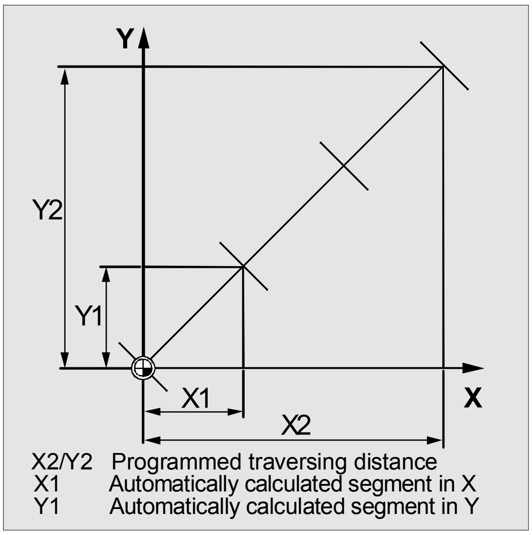


X2/Y2 Programmed path
(nibbling or punching block)
E1 Programmed path segment
E1' Automatically rounded path segment length

When the total traversing distance is 15mm and the path segment length 3.5mm, 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.


Y
Y2
Y1
X1
X2
X
X2/Y2 Programmed traversing distance
X1 Automatically calculated segment in X
Y1 Automatically calculated segment in Y

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.

Other Information

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 when the punching/nibbling functions are not in use.

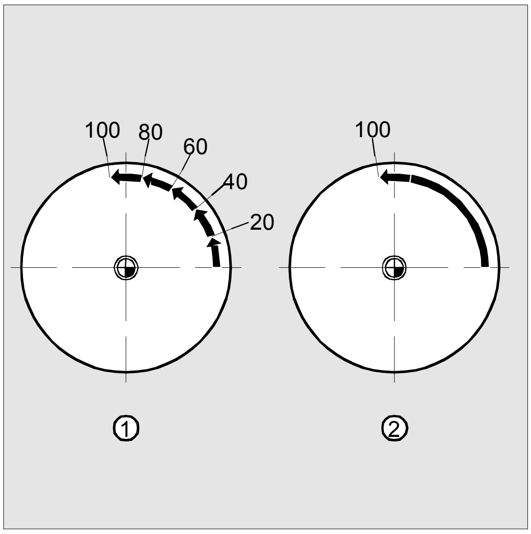
12.2.2 Path segmentation for single axes

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 3mm, 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:

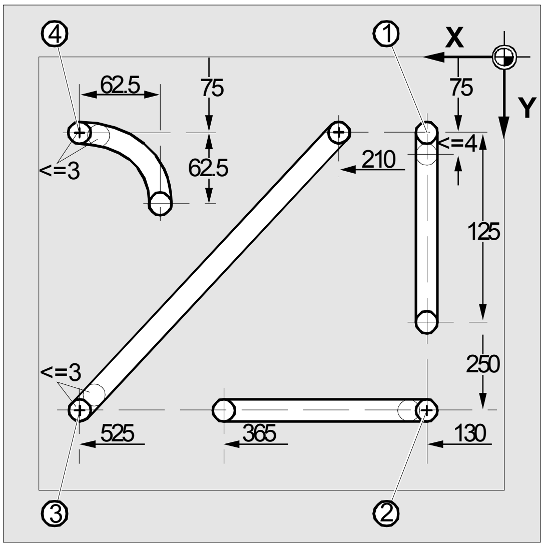
- Circular interpolation: Path segmentation
- Linear interpolation: No path segmentation

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.

12.2.3 Programming examples

Programming example 1

The programmed nibbling paths must be divided automatically into equidistant path segments.

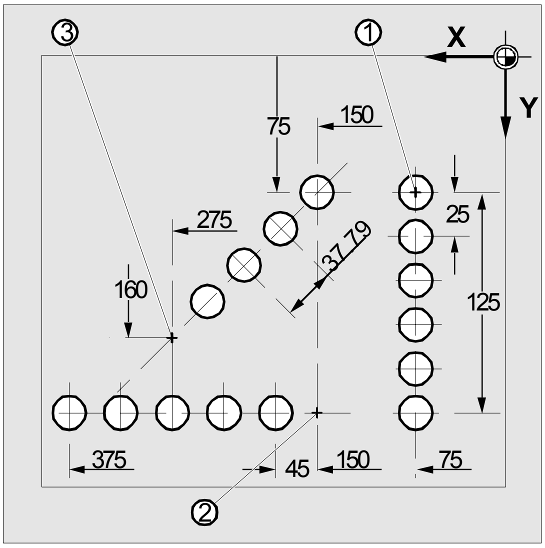


Program section

`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

Programming 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.



Program section

`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 length.
`N160 G00 Y300 SPOF`	Punching off; position

■

Notes

Additional Functions

13.1 Axis functions AXNAME, SPI, ISAXIS, AXSTRING (SW 6 and higher)

Programming

```
AXNAME("TRANSVERSE AXIS")
AX[AXNAME("String")]
AXSTRING ( SPI(n) )
SPI(n)
ISAXIS(geometry axis number)
```

Explanation of the commands

AXNAME	Converts an input string to an axis identifier. The input string must contain valid axis names.
SPI	Converts the spindle number into an axis identifier; the transfer parameter must contain a valid spindle number.
n	Spindle number
AXSTRING	Up until SW 5, the axis index of the axis which was assigned to the spindle was output as spindle number. From SW 6 the string is output with the associated spindle number.
AX	Integer without sign
ISAXIS	Checks whether the specified geometry axis exists.

Function

AXNAME is used, for example, to create generally applicable cycles when the name of the axes are not known (see also Section 13.10. "String functions").
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.
(SW 6 and higher)
Extensions SPI(n):
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.
Additional programming of the axis position via address AX[SPI(1)] = <axis_position> allows an axis to be traversed.

Troubleshooting for `AXSTRING( SPI(n) )`

When programming
`AXSTRING( SPI(n) )` up to SW 5
the axis index of the axis which was assigned to the spindle was output as spindle number.
Example:
Spindle 1 is assigned to the 5th axis.
`($MA_SPIND_ASSIGN_TO_MACHAX[AX5]=1),`
`AXSTRING( SPI(1) )` returns the incorrect string "`S4`"

With SW 6 and higher,
`AXSTRING[ SPI(n) ]` will output the string "`Sn`".
Example:
`AXSTRING( SPI(2) )` returns string "`S2`"

Programming 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 Function call ISVAR ( ) (SW 6.3 and higher)

Programming

```
ISVAR ("variable_identifier")
ISVAR (identifier, [value, value])
```

Explanation of the commands

`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
`Value`	Function value of type BOOL

Structure

The transfer parameter can have the following structure:

1. Undimensioned variable:
   Name of identifier
2. 1-dimensional variable without array index:
   identifier[ ]
3. 1-dimensional variable with array index:
   identifier[value]
4. 2-dimensional variable without array index:
   identifier[,]
5. 2-dimensional variable with array index:
   identifier[value, value]

Function

The ISVAR command is a function as defined in the NC language with 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 GUD's).

Checks

The following checks are make 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 this 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.

Examples:

```
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 is TRUE with a 2-dimensional array
IS_VAR=ISVAR(‟VARARRAY[,]‟)
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 is FALSE, syntax error for missing "]"
IS_VAR=ISVAR(‟VARARRAY[8,8]‟)
N60                               ; IS_VAR is TRUE, array index is allowed
IS_VAR=ISVAR(‟VARARRAY[,8]‟)
N70                               ; IS_VAR is TRUE
IS_VAR=ISVAR(‟VARARRAY[8,]‟)

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
```

13.3 Learn compensation characteristics: QECLRNON, QECLRNOF

Explanation of the commands

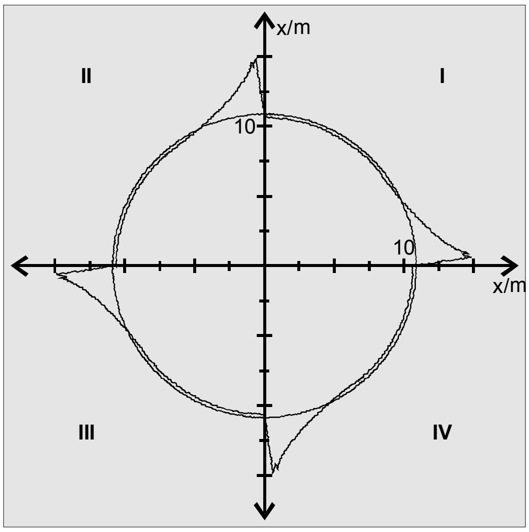
`QECLRNON (axis.1,…4)`	Activate "Learn quadrant error compensation" function
`QECLRNOF`	Deactivate "Learn quadrant error compensation" function

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.



Sequence

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:

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

Subsequent learning

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 of the learning cycle (e.g. QECLRN.SPF) can also be changed for optimization

- Set "Learn mode" = 1
- Reduce "Number of learn passes" if required
- Activate "Modular learning" if required and define area limits.

Activate 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 learning process: QECLRNOF

When the learning movements for the desired axes are complete, the learning process is deactivated simultaneously for all axes with QECLRNOF.

13.4 Synchronous spindle

Programming

```
COUPDEF (FS,LS,SR_FS,SR_LS, block change
beh., coupling)
COUPDEL (FS,LS)
COUPRES (FS,LS)
COUPON (FS,LS,PS_FS)
COUPOF (FS,LS,POS_FS,POS_LS)
WAITC (FS,block ratio,LS,block change
beh.)
```

Explanation of the commands

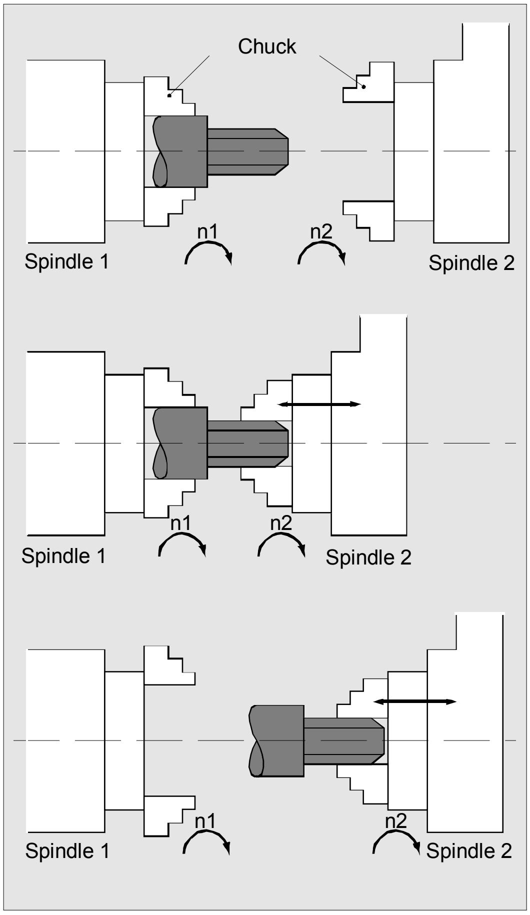
COUPDEF	Define/change user coupling
COUPON	Activate coupling
COUPOF	Deactivate coupling
COUPRES	Reset coupling parameters
COUPDEL	Delete user-defined coupling
WAITC	Wait for synchronism condition

Explanation of the parameters

FS, LS	Name of following and leading spindle; specified with spindle number: e.g. S2
SR_{FS}, SR_{LS}	Speed ratio parameter for following spindle and leading spindle Default setting = 1.0; specification of denominator optional
Block change behavior:	Block change method; Block change is implemented by:
• "NOC"	Immediately (default)
• "FINE"	in response to "Synchronization run fine"
• "COARSE"	in response to "Synchronization run coarse"
• "IPOSTOP"	in response to IPOSTOP (i.e. after setpoint synchronization run)
Coupling	Coupling type: Coupling between FS and LS
• "DV"	Setpoint coupling (default)
• "AV"	Actual-value coupling
PS_{FS}	Angle offset between leading and following spindles
POS_{FS}, POS_{LS}	Deactivation positions of following and leading spindles

Function

In synchronized mode, there is a leading spindle (LS) and a following spindle (FS). They are referred to as the **synchronous spindle pair**. The following spindle follows the movements of the leading spindle when the coupling is active (synchronized mode) in accordance with the functional relationship specified in the parameters.

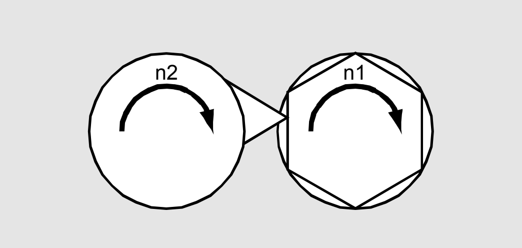


This function enables turning machines to perform workpiece transfer from spindle 1 to spindle 2 on-the-fly, e.g. for final machining. This avoids downtime caused, for example, by rechucking.

The transfer of the workpiece can be performed with:

- Speed synchronism ($n_{FS} = n_{LS}$)
- Position synchronism (($\varphi_{FS} = \varphi_{LS}$)
- Position synchronism with angular offset ($\varphi_{FS} = \varphi_{LS} + \Delta\varphi$)

A speed ratio SR_T can also be specified between the main spindle and a "tool spindle" for multi-edge machining (polygon turning).



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.

Sequence

Define synchronized spindle pair: Options

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 language instruction 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,SR_{FS},SR_{LS},block change beh., 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,SRFS,SRLS)
```

Meaning:
S2 = following spindle, S1 = leading spindle

Position the following spindle: Options
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 Guide "Fundamentals" for more information about SPOS.

Example:

```
N30 SPOS[2]=IC(-90)
```

FA, ACC, OVRA:
Velocity, acceleration
The position speeds and acceleration rates for following spindles can be programmed with FA [SPI (Sn)] or FA[Sn], ACC[SPI(Sn)] or ACC [Sn] and OVRA [SPI(n)] bzw. OVRA[Sn] (see Programming Guide, Fundamentals). "n" stands for spindle number 1...n.

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.

Speed ratio SR_T

The speed ratio is defined with parameters for FS (numerator) and LS (denominator).

Options:

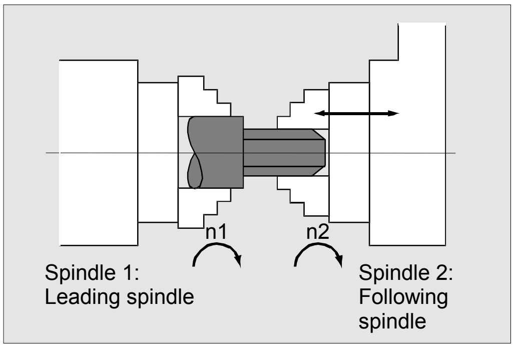
- The following and leading spindles rotate at the same speed ($n_{FS} = n_{LS}$; SR_T positive)
- Rotation in the same or opposite direction (SR_T negative) between LS and FS
- Following and leading spindles rotate at different speeds
  ($n_{FS} = SR_T \cdot n_{LS}$; $SR_T \neq 1$)
  Application: Polygonal turning

Example:

```
N… COUPDEF(S2, S1, 1.0, 4.0)
```

Meaning:

Following spindle S2 and leading spindle S1 rotate at a speed ratio of 0.25.



- 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

The following options can be selected during definition of the coupling to determine when the block change takes place:

"**NO**C"	Immediately (default)
"**FI**NE"	At "Synchronization fine"
"**CO**ARSE"	At "Synchronization coarse"
"**IP**OSTOP"	At IPOSTOP (i.e. after synchronization on the setpoint side)

The block change response can be specified simply by writing the letters in bold print.

The block change method is modal!

Coupling type

"DV"	Setpoint coupling between FS and LS (default)
"AV"	Act.-val. coupl. between FS and LS

The coupling type is modal.

Notice

The coupling type may be changed only when the coupling is deactivated!

Activating synchronized mode

- Fastest possible activation of coupling with any angle reference between LS and FS:

```
N … COUPON (S2, S1)
```

- Activation with angular offset POS_{FS}
  Position-synchronized coupling for profiled workpieces.
  POS_{FS} refers to the 0° position of the lead spindle in the positive direction of rotation.

  Value range POS_{FS}: 0°… 359,999°:

```
COUPON (S2,S1,30)
```

You can use this method to change the angle offset even when the coupling is already active.

Deactivating synchronized mode COUPOF

Three variants are possible:

- For the fast possible activation of the coupling and immediate enabling of the block change:

```
COUPOF (S2,S1)
```

- 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 following spindle stop COUPOFS (SW 6.4 and higher)
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 deactivation position that refers to the machine coordinate system has been crossed, the block change is not enabled until the deactivation positions POS_{FS} have been crossed.

COUPOFS (S2,S1,POS_{FS})

Value range 0° ... 359.999°:

Deleting couplings COUPDEL (up to SW 6.3)
If a new synchronous spindle coupling relationship needs to be defined and all available, freely configurable couplings (1 or 2) are already configured, then one of the couplings will have to be deleted first.

```
N … COUPON (S2,S1)
```

SPI(2) = following spindle, SPI(1) = leading spindle

⚠ *A coupling can only be deleted if it has been deactivated first (*COUPOF*).*
A permanently configured coupling cannot be deleted by means of COUPDEL up to SW 6.3.

⚠ **Deleting couplings COUPDEL (SW 6.4 and higher)**
The coupling is now also deactivated on an active synchronized spindle coupling and the coupling data are deleted. The following spindle takes over the last speed and its behavior is the same as that of the COUPOF(FS;LS) so far.

Reset coupling parameters, COUPRES

Language instruction "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)
```

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.

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.

Programming example

Working with master and slave spindles.

	; Leading spindle = master spindle =
	; Following spindle = spindle 2
`N05 M3 S3000 M2=4 S2=500`	; Master spindle rotates at 3000rpm, slave spindle at 500rpm
`N10 COUPDEF (S2, S1, 1, 1, "NOC", "Dv")`	; Def. of coupling, can also be configured
…	
`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`	

13.5 EG: Electronic gear (SW 5 and higher)

Introduction

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

from the part program.

The following axis movement can be optionally derived from

- Setpoints of the leading axes, as well as
- Actual values of leading axes

As an expansion, with **SW 6** and higher nonlinear relations between the leading axes and the following axis can also be achieved via **curve tables** (see Chapter 9). Electronic gears can be cascaded, i.e. the following axis of an electronic gear can be the leading axis for another electronic gear.

13.5.1 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,...)

Explanation

`Following axis`	Axis that is influenced by the leading axes
`Leading axis 1, ... leading axis 5`	Axes that influence the following axis
`Coupling type 1, ... coupling type 5`	Following axis is influenced by: 0: Actual point 1: Setpoint of the respective leading axis

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.
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).

Note

EGDEF triggers preprocessing stop. Gear definition with EGDEF must also be used unchanged, if with systems using **SW 6** and higher, one or more leading axes influence the following axis via the **curve table**.

13.5.2 Activating an electronic gear

There are 3 variants for the activation command:

- **Variant 1:**

The EG axis grouping is activated selectively **without** synchronization with:

```
EGON(FA, "Block change mode", LA1, Z1,
N1, LA2 , Z2, N2,..LA5, Z5, N5.)
```

Explanation

`FA`	Following axis
`Block change mode`	The following modes can be used: "NOC" Immediate block change "FINE" Block change occurs at "Synchronization fine" "COARSE" Block change occurs at "Synchronization coarse" "IPOSTOP" Block change occurs at setpoint synchronization run
`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 following axis must be programmed.
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:**

The EG axis grouping is activated selectively **with** synchronization with:

```
EGONSYN(FA, "Block change mode", SynPosFA,[, LAi, SynPosLAi, Zi,
Ni])
```

Explanation

`FA`	Following axis
`Block change mode`	The following modes can be used: "NOC" Immediate block change "FINE" Block change occurs at "Synchronization fine" "COARSE" Block change occurs at "Synchronization coarse" "IPOSTOP" Block change occurs at setpoint synchronization run
`[, 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

- **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])
```

Explanation

	The parameters are the same as for variation 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 to programmed synchronized position

	"DCP"	Path-optimized to programmed synchronized position

Variation 3 only affects modulo following axes that are coupled to modulo leading axes. Time optimization takes account of velocity limits of the following axis. 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.

SW 6

If a **curve table** is used for one of the leading axes, then:

`Ni`	The denominator of the coupling factor for linear coupling must be set to 0. Denominator 0 would be illegal for linear couplings.) Denominator zero tells the control that
`Zi`	is the number of the curve table to use. The curve table with the specified number must already be defined when the control is switched on.
`LAi`	The leading axis specified corresponds to the one specified in coupling via coupling factor (linear coupling).

For more information about using curve tables and cascading and synchronizing electronic gears, please refer to:
/FB/ M 3, Coupled Motion and Leading Value Coupling

Only leading axes previously specified with the EGDEF command may be programmed in the activation line.

Via the programmed "synchronized positions" for the following axis (SynPosFA) and for the leading axes (SynPosLA), positions are defined in which the coupling group is valid as *synchronized*. 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.
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 (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.

13.5.3 Deactivating an electronic gear

There are three different ways to deactivate an active EG axis grouping.
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 axis 1,
... leading axis 5)
```

This command parameter setting make it possible to **selectively** remove the control the individual leading axes have over the following axis' motion.

At least one following 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 spindle)`

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 functions is only allowed for spindles.

13.5.4 Deleting the definition of an electronic gear

An EG axis grouping must be deactivated as described in the preceding section before you can delete its definition.

`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.

13.5.5 Revolutional feedrate (G95)/electronic gear (SW 5.2)

The FPR() command can be used in SW 5 and higher 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.

13.5.6 Response of EG 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.

13.5.7 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.

Other information

The system variables for the electronic gear are listed in the Annex. They are characterized by names beginning with:

```
$AA_EG_ ...
```

or

```
$VA_EG_ ...
```

13.6 Extended stopping and retract (SW 5 and higher)

Function

The "Extended stopping and retract" function ESR provides a means to react flexibly to selective error sources while preventing damage to the workpiece. "Extended stopping and retract" provides the following part reactions:

- "**Extended stopping**" (independent drive, SW 5) is a time-delayed stop.
- "**Retract**" (independent of drive) means "escaping" from the machining plane to a safe retraction position. This means any risk of collision between the tool and the workpiece is avoided.
- **"Generator operation"** (independent of drive) For the cases in which the energy of the DC link is not sufficient for a safe retraction, generator operation is possible. 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 or similar failure.

From SW 6 also:

- **Extended shut down** (NC-controlled) is a defined, time-delayed, contour-friendly shut down controlled by the NC.
- **Retract** (NC-controlled) means "escaping" from the machining level to a safe retraction 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, see
/FB/ M3, Axis Couplings and ESR

13.6.1 Independent drive reactions

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" draws attention to this.

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)
Independent drive 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.

13.6.2 NC-prompted reactions

Function

Retraction

Preconditions:

- the axes selected with POLFMASK or POLFMILIN
- the axis-specific positions defined with POLF
- the retraction positions of a single axis defined with POLFA (SW 6.4 and higher)
- 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 defined ESR response
  MD 37500: ESR_REACTION = 21
- LFPOS from the modal 46. G code group

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** is activated in this channel.

The retraction position POLF must be programmed in the part 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.

Other Information

The lift configured with **LFPOS**, **POLF** for the axis/axes selected with **POLFMASK** or **POLFMLIN** replaces the continuous-path motion defined for this axis/these axes in the part program.

The extended retraction (i.e. LIFTFAST/LFPOS initiated through $AC_ESR_TRIGGER) is **cannot be interrupted** and can only be terminated prematurely per EMERGENCY STOP. The maximum time available for retraction is the sum of the times MD 21380: ESR_DELAY_TIME2 and MD 21381: ESR_DELAY_TIME2. When this time has expired, rapid deceleration with follow-up is also initiated for the retraction axis.
The frame valid at the time when lift fast was activated is considered.
Important:
frames with rotation also affect the direction of lift via **POLF**.
The NC-controlled retraction is

- configured: via MD 37500: **21** and 2 time specified via MD see above;
- enabled ($AA_ESR_ENABLE) and
- started: System variable $AC_ESR_TRIGGER with $AA_ESR_TRIGGER for single axes

Example of 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.

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)	Abbreviated form for single axis
POLFA(axis, 0/1/2)	retraction high-speed deactivation / activation
POLFA(axis, 0, $AA_POLFA[axis])	causes **a** preprocessing stop does **not**
POLFA(axis, 0)	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

Warning

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.

Explanation of the commands

geo | mach	Geometry axis or channel/machine axis that retracts.
Axis	Axis identifiers of the valid rotary axes (SW 6.4 and higher)
Type	Position values of the single axes (SW 6.4 and higher) of type: 0 Invalidate the position value 1 Position value is absolute 2 Position value is incremental (distance)
Value	Retract position, WCS is valid for geometry axis, otherwise MCS. When using the **same** identifiers for geometry axis and channel/machine axis, the workpiece coordinate system is used for retraction. Incremental programming is permissible. Retract position where Type=1 for single axes (from SW 6.4) Retract path where Type=2 for single axes (from SW 6.4) 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 fast list to a fixed position can be activated via **POLFMASK** or **POLFMLIN**, a position must be programmed with **POLF** for the selected axis. 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.

If axes are enabled in succession with POLFMASK, POLFMLIN or POLFMLIN, POLFMASK, the last definition always applies to each axis.

Notice

*The positions programmed with **POLF** and the activation by **POLFMASK** or **POLFMLIN** are deleted when the part program is started. This means that the user must program the values for **POLF** and the selected axes again in **POLFMASK** or **POLFMLIN** in the part program.*

For more information on changing the coordinate system, the effect on modulo rotary axes, etc. see Function Description M3.

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 for the time duration set in MD 21380 axis, as programmed.

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.
The NC-controlled stop is

- configured: via MD 37500: **22** and
  2 time specified via MD see above;
- enabled ($AA_ESR_ENABLE) and
- started: System variable $AC_ESR_TRIGGER with $AA_ESR_TRIGGER for single axes

Example of stopping of 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.
```

13.6.3 Possible trigger sources

Function

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, $VA_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].

13.6.4 Logic gating functions: Source/reaction logic operation

Function

The static synchronized actions' flexible gating possibilities are used to trigger specific reactions 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 more detailed description on how to use synchronized actions, please refer to
References: /FBSY/ Description of Functions Synchronized Actions

13.6.5 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.

Triggering functions (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).

13.6.6 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
/FB/ M 3, Coupled Motion and Leading Value Coupling

13.6.7 Independent drive stop

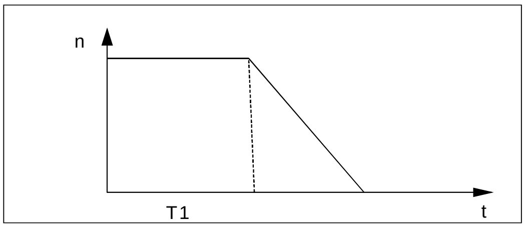
Function

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).

13.6.8 Independent drive retract

Function

Axes with digital 611D 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 drive 611D.

After the beginning of the retraction phase the drive independently maintains its enables at the previously valid values.

For more information see

/FB/ M 3, Coupled Motion and Leading Value Coupling

13.6.9 Example: Use of independent drive reaction

Example configuration

- Axis A is to operate as generator drive,
- in the event of an error, axis X must retract by 10mm at maximum speed, and
- axes Y and Z must stop after a 100ms delay to give the retraction axis time to cancel the mechanical coupling.

Sequence

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	(retract 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 part 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. Get the generator operation 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 $AN_ESR_TRIGGER=1
  - and also dependent on EU synchronized operation (if, for example, Y is defined as EU following axis and if the max. allowed deviation of synchronized operation shall be 100 µml):
    IDS=03 WHENEVER ABS($VA_EG_SYNCDIFF[Y])>0.1
    DO $AN_ESR_TRIGGER=1

13.7 Link communication (SW 5.2 and higher)

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 networked NCUs connected by means of an NCU link module represent an open, scalable solution that meets all the requirements of this type of machine tool. The NCU link module (hardware) provides high-speed NCU-to-NCU communication.

Options providing this functionality can be ordered separately.

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 as system variables.
  As a rule, the machine manufacturer defines and documents the meaning of these variables.
- Applications for link variables:
  - Global machine states
  - Workpiece clamping open/closed
  - Etc.
- 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.

Link variables are **global system data** that can be addressed by the connected NCUs as **system variables**. The
- **contents** of these variables,
- their **data type**,
- **use**, and
- position (**access index**) in the link memory
are defined by the user (in this case generally the machine manufacturer).

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

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, please refer to the Description of Functions B3 (SW 5)

Programming example

```
$A_DLB[5]=21
```

The 5th byte in the shared link memory is assigned value 21.

13.8 Axis container (SW 5.2 and higher)

Function

On rotary indexing machines/multi-spindle machines, the work-holding axes move from one machining unit to the next.
As the machining stations are controlled by different NCU channels, at station/position change the axes holding the workpiece must be dynamically reassigned to the appropriate NCU channel. **Axis containers** are provided for this purpose.
Only one workpiece clamping axis/spindle can be active at any one time at the local machining station.
The axis container provides the possible connections to all clamping axes/spindles, of which exactly **one** is **activated** for the machining unit.
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.
This switching function can be triggered from 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.

Detailed information on configuring axis containers can be found in /FB/, B3 (SW 5.2)

The entries in the axis container can be switched by increment n via the commands:

Programming

AXCTSWE (CT_i)	AXIS CONTAINER SWITCH ENABLE
AXCTSWED(CT_i)	AXIS CONTAINER SWITCH ENABLE DIRECT

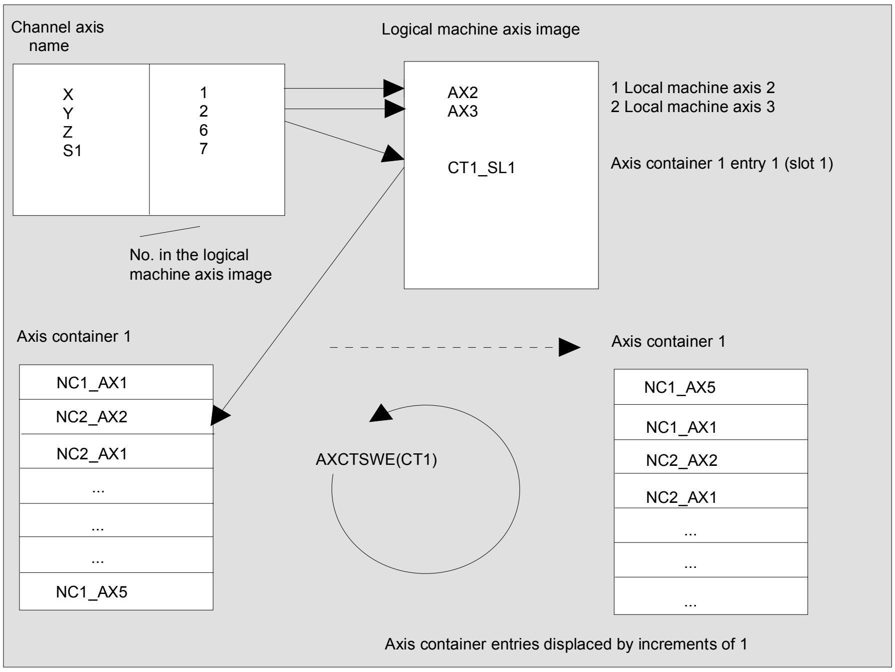
Explanation

CT_i or e.g. A_CONT1	Number of the axis container whose contents are to be switched or individual name of axis container set via MD.

Function

`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.

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.

13.9 Program runtime/workpiece counter (SW 5.2 and higher)

Function

Information on the program execution time and on the workpiece count are provided to support the person working at 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 MMC at the operator panel front interface.

13.9.1 Program runtime

Function

Under this function, timers are provided as system variables, which can be used to monitor technological processes.
These timers can only be read. They can be accessed at any time by the MMC in read mode.

Explanation

The following two timers are defined as NCK-specific system variables and always active.

`$AN_SETUP_TIME`	Time in minutes since the last setup; is reset with SETUP
`$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

All timers are reset with default values when the control is powered up, and can be read independent of their activation.

Programming 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'
```

13.9.2 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.

Explanation

The following counters are available:

`$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 actually produced (total actual) The counter indicates the total number of workpieces produced since the starting time. The counter is automatically reset with default values only when the control is powered up.
`$AC_ACTUAL_PARTS`	Number of actual workpieces. This counter records the number of all workpieces produced since the starting 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 user-defined workpiece counting. Alarm output can be defined for the case of identity with $AC_REQUIRED_PARTS (workpiece target). The user must reset the counter

The "Workpiece counter" function operates independently of the tool management functions.
All counters can be read and written from the MMC.
All counters are reset with default values when the control is powered up, and can be read/written independent of their activation.

Programming example

1. Activate workpiece counter $AC_REQUIRED_PARTS:	
$MC_PART_COUNTER=’H3’	$AC_REQUIRED_PARTS is active, display alarm on $AC_REQUIRED_PARTS == $AC_SPECIAL_PARTS
2. Activate workpiece counter $AC_TOTAL_PARTS:	
$MC_PART_COUNTER=’H10’ $MC_PART_COUNTER_MCODE[0]=80	$AC_TOTAL_PARTS is active, the counter is incremented by 1 on each M02, $MC_PART_COUNTER_MCODE[0] is irrelevant
3. Activate workpiece counter $AC_ACTUAL_PARTS:	
$MC_PART_COUNTER=’H300’ $MC_PART_COUNTER_MCODE[1]=17	$AC_TOTAL_PARTS is active, the counter is incremented by 1 on each M17
4. Activate workpiece counter $AC_SPECIAL_PARTS:	
$MC_PART_COUNTER=’H3000’ $MC_PART_COUNTER_MCODE[2]=77	$AC_SPECIAL_PARTS is active, the counter is incremented by 1 on each M77
5. Deactivate workpiece counter $AC_ACTUAL_PARTS:	
$MC_PART_COUNTER=’H200’ $MC_PART_COUNTER_MCODE[1]=50	$AC_TOTAL_PARTS is not active, rest irrelevant
6. Activating all counters in examples 1-4:	
$MC_PART_COUNTER =’H3313’ $MC_PART_COUNTER_MCODE[0] =80 $MC_PART_COUNTER_MCODE[1] =17 $MC_PART_COUNTER_MCODE[2] =77	$AC_REQUIRED_PARTS is active Display alarm on $AC_REQUIRED_PARTS == $AC_SPECIAL_PARTS $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

13.10 Interactive window call from parts program, command: MMC (SW 4.4 and higher)

Programming

```
MMC ("CYCLES, PICTURE_ON, T_SK.COM, DISPLAY, MGUD.DEF, PICTURE_3.AWB,
TEST_1, A1","S")
```

Explanation

`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 displays can show user variables and/or comments.
`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

Function

With the MMC command, user-defined dialog windows (dialog displays) can be displayed on the MMC/HMI from the parts program.
The dialog window design is defined in pure text configuration (COM file in cycles directory), while the MMC/HMI system software remains unchanged.
User-defined dialog windows cannot be called simultaneously in different channels.
Please see the detailed notes on how to program the MMC command (incl. programming examples) in /IAM/ in the manuals IM1 through IM4 depending on the MMC/HMI software used.

13.11 Influencing the motion control

13.11.1 Percentage jerk correction: JERKLIM

Programming

```
JERKLIM[axis]= ...
```

Explanation of the command

`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

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.

Sequence

In the AUTOMATIC modes, the jerk limit is limited to the percentage of the jerk limit stored in the machine data.

Example: `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.

Value range: 1 ... 200
100 corresponds to: no effect on jerk.
100 is applied after RESET and parts program start.

Other Information

A further example will follow at the end of the next subsection.

13.11.2 Percentage velocity correction: VELOLIM

Programming

```
VELOLIM[axis]= ...
```

Explanation of the command

`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

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.

Sequence

In the AUTOMATIC modes, the velocity limit is limited to the percentage of the velocity limit stored in the machine data.

Example: `N70 VELOLIM[X]=80`
Meaning: The axis carriage in the X direction must travel at only 80% of the velocity permissible for the axis.
Value range: 1 ... 100
100 corresponds to: no effect on velocity.
100 is applied after RESET and parts program start.

Programming 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
```

13.12 Master/slave grouping

Programming

`MASLDEF(Slv1, Slv2, ..., master axis)`	For dynamic configuration (SW 6.4 and higher)
`MASLDEL(Slv1, Slv2, ..., )`	For dynamic configuration (SW 6.4 and higher)
`MASLON(Slv1, Slv2, ..., )`	
`MASLOF(Slv1, Slv2, ..., )`	
`MASLOFS(Slv1, Slv2, ..., )`	(SW 6.4 and higher)

Explanation of parameters

`Slv1, Slv2, ...`	Slave axes led by a master axis
`Master axis`	Axis leading slave axes defined in a master/slave grouping

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 6.4 permits coupling and uncoupling of **rotating**, speed-controlled spindles and dynamic configuration.

Dynamic configuration

MASLDEF

(SW 6.4 and higher)
Definition of a master/slave grouping from the part program: Before SW 6.4, this was defined exclusively via machine data.

MASLDEL

(SW 6.4 and higher)
The instruction cancels assignment of the slave axes to the master axis and simultaneously uncouples the current coupling, like MASLOF.
The master/slave definitions declared in the machine data are retained.

General

MASLON

Enable a temporary coupling

MASLOF

This instruction uncouples an active coupling.
(SW 6.4 and higher)
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.

MASLOFS

(SW 6.4 and higher)
The MASLOFS instruction can be used to decelerate slave spindles automatically on uncoupling. For axes and spindles in positioning mode, uncoupling is only possible while stopped.

More information (SW 6.4 and higher)

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.

Programming example

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 ; uncoupling
`AXCTSWE(CT1)`	; Container rotation

More information (SW 6.4 and higher)

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 has no effect on the language commands of the temporary coupling.

Programming example

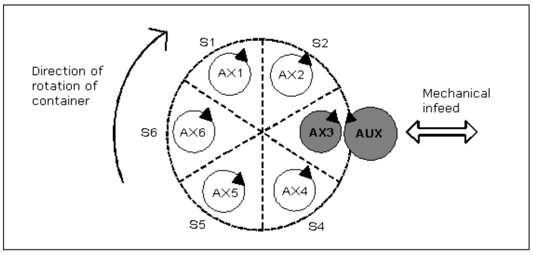
Actual-value coupling of a slave axis set to the same value as the master axis with PRESETON.

In a permanent master/slave coupling, the actual value on the SLAVE axis is to be changed by PRESETON.

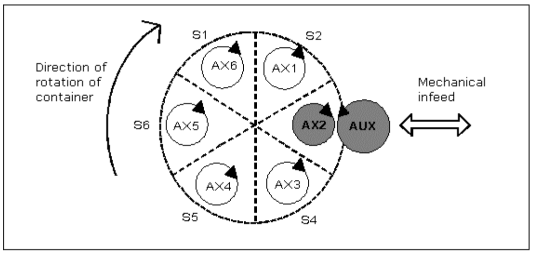
```
N37262                                      ; Activate permanent coupling
$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=0
N37263 NEWCONF                              ;
N37264 STOPRE                               ;
MASLOF(Y1)                                  ; Temporary coupling off
N5 PRESETON(Y1, 0, Z1, 0, B1, 0, C1,        ; Set actual value of the unreferenced
             0, U1, 0)                      ; slaveaxes because they are activated
                                            ; on Power on
N37262                                      ; Activate permanent coupling
$MA_MS_COUPLING_ALWAYS_ACTIVE[AX2]=1
N37263 NEWCONF
```

To enable coupling with another spindle after container rotation, the previous coupling must be uncoupled, the configuration cleared, and a new coupling configured.
Example of a coupling sequence Position 3 / Container CT1

See /FB/, B3 Sec. 2.6 Axis container



Initial situation



After rotation by one slot

■

Notes

User Stock Removal Programs

14.1 Supporting functions for stock removal

User stock removal programs

Preprogrammed stock removal programs are provided for stock removal. You can also use the following functions to develop your own stock removal programs.

`CONTPRON`	Activate tabular contour preparation (11 columns)
`CONTDCON`	Activate tabular contour decoding (6 columns)
`INTERSEC`	Calculate intersection of two contour elements. (Only for tables created by CONTPRON).
`EXECTAB`	Block-by-block execution of contour elements of a table (Only for tables created by CONTPRON).
`CALCDAT`	Calculate radii and center points

You can use these functions universally, not just for stock removal.

14.2 Contour preparation - CONTPRON

Programming

```
CONTPRON (TABNAME, MACH, NN, MODE)
EXECUTE (ERROR)
```

Explanation of the parameters

CONTPRON	Activate contour preparation
TABNAME	Name of 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 (SW 4.4 and higher)	Direction of machining, type INT 0 = Contour preparation forward (SW 4.3 and lower, default value) 1 = Contour preparation in both directions
EXECUTE	Terminate contour preparation
ERROR	Variable for error checkback, type INT 1 = error; 0 = no error

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.
EXECUTE deactivates the contour preparation and switches back to the normal execution mode.
Example:

```
N30 CONTPRON(…)
N40 G1 X… Z…
N50
N100 EXECUTE(…)
```

Other Information

Preconditions for the call

Before CONTPRON is called

- a starting point must be approached which permits collision-free machining,
- tool edge radius compensation with G40 must be deactivated.

Permitted traversing commands, coordinate system

Only G commands G0 to G3 are permitted for contour programming in addition to rounding and chamfer.

SW 4.4 and higher supports circular-path programming via CIP and CT.

The functions Spline, Polynomial, thread 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 is error free).

Relief cut elements

The contour description for the individual relief cut elements can be performed either in a subroutine or in individual blocks.

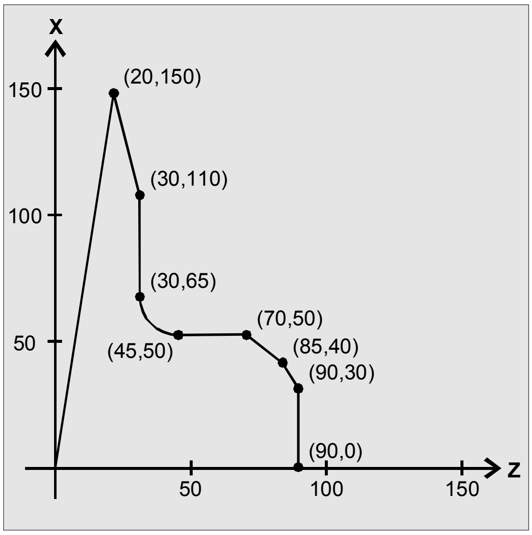
Stock removal irrespective of the programmed contour direction (SW 4.4 and higher)

In SW 4.4 and higher, contour preparation has been expanded. Now when CONTPRON is called, the contour table is available irrespective of the programmed direction.

Programming example 1

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 size
`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`
`N70 X110 Z30`
`N80 X50 RND=15`
`N90 Z70`
`N100 X40 Z85`
`N110 X30 Z90`
`N120 X0`
	N60 to N120 contour description

`N130 EXECUTE(ERROR)`	Terminate filling of contour table, switch to normal program execution
`N140 …`	Continue processing table

Table KTAB

(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

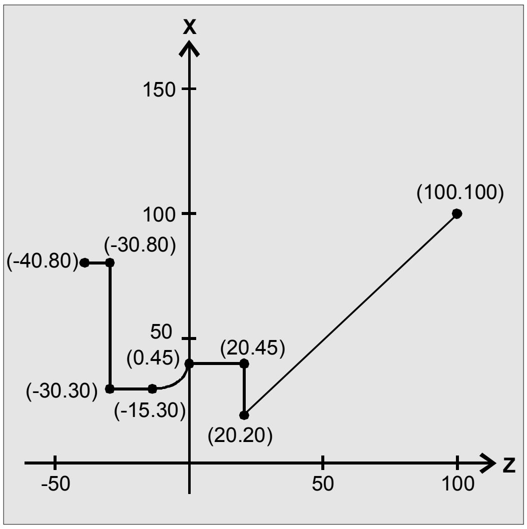
(8) Maximum value between contour element and abscissa (for longitudinal machining) or ordinate (for transverse machining)
The angle depends on the type of machining programmed.

(9), (10) Center point coordinates of contour element, if it is a circle block.
(9) = abscissa, (10) = ordinate

Programming example 2

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 size
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.

Line	Column									
	(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

(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 transverse machining)
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

Explanation of comment in columns

Always in table line 0: 1) Previous: Line n contains the contour end forwards
2) Following: Line n is the contour table end forwards

Once each within the contour elements forwards:
3) Previous: Contour start (forwards)

4) Following: Contour end (forwards)

Always in line contour table end (forwards) +1:

5) Previous: Number of relief cuts forwards
6) Following: Number of relief cuts backwards

Once each within the contour elements backwards:

7) Following: Contour end (backwards)
8) Previous: Contour start (backwards)

Always in last line of table:

9) Previous: Line n is the contour table start (backwards)
10) Following: Line n contains the contour start (backwards)

14.3 Contour decoding - CONTDCON (SW 5.2 and higher)

Programming

```
CONTDCON (TABNAME,MODE)
EXECUTE (ERROR)
```

Explanation of the parameters

`CONTDCON`	Activate contour preparation
`TABNAME`	Name of contour table
`MODE`	Direction of machining, type INT 0 = Contour preparation (default) according to the contour block sequence
`EXECUTE`	Terminate contour preparation
`ERROR`	Variable for error checkback, type INT 1 = error; 0 = no error

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. The G codes permitted for CONTDCON in the program section to be included in the table are more comprehensive than for the CONTPRON function. In addition, feedrates and feed type are also stored for each contour section.
EXECUTE deactivates the contour preparation and switches back to the normal execution mode.

```
Example:
N30 CONTDCON(…)
N40 G1 X… Z…
N50
N100 EXECUTE(…)
```

Other Information

Preconditions for the call
Before CONTDCON is called

- a starting point must be approached which permits collision-free machining,
- tool edge radius compensation with G40 must be deactivated.

Permitted traversing commands, coordinate system
The following G groups and specified commands are permissible for contour programming:

G group 1:	G0, G1, G2, G3
G group 10:	G9
G group 11:	G60, G44, G641, G642
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 functions Spline, Polynomial, thread 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 produced 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 error (contour produced no errors)

1 = error

Impermissible commands, incorrect initial conditions, CONTDCON call repeated without EXECUTE( ), too few contour blocks or table definitions too small produce additional alarms.

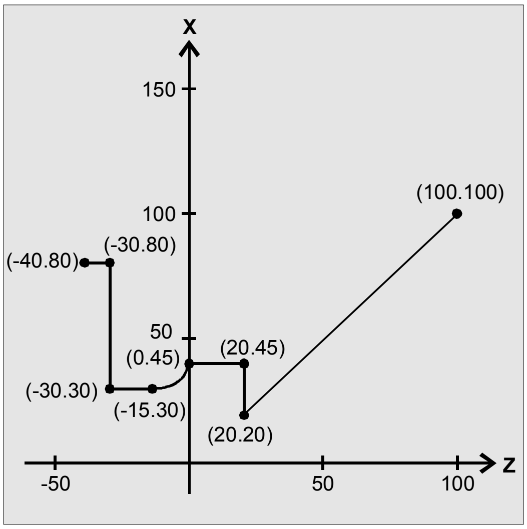
Stock removal in the programmed contour direction

The contour table produced using CONTDCON is used for stock removal in the programmed direction of the contour.

Programming example

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 fixed.
`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`
`N120 G9 X45 F300`
`N130 Z0 F400`
	Contour description

`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
`...`	

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 **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

14.4 Intersection of two contour elements - INTERSEC

Programming

```
VARIB=INTERSEC (TABNAME1[n1], TABNAME2[n2], TABNAME3)
```

Explanation of the parameters

VARIB	Variable for status	TRUE: Intersection found FALSE: No intersection found
TABNAME1[n1]	Table name and n1st contour element of the first table	
TABNAME2[n2]	Table name and n2nd contour element of the second table	
TABNAME3	Table name for the intersection coordinates in the active plane G17-G19	

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. The indicated status specifies whether or not an intersection exists (TRUE = intersection, FALSE = no intersection).

Other Information

Please note that variables must be defined before they are used.

Programming example

Calculate the intersection of contour element 3 in table KTAB1 and contour element 7 in table KTAB2. 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 KTAB1 [12, 11]`	Contour table 1
`DEF REAL KTAB2 [10, 11]`	Contour table 2
`DEF REAL CUT [2]`	Intersection table
`DEF BOOL ISPOINT`	Variable for status
…	
`N10 ISPOINT=INTERSEC (KTAB1[3],KTAB2[7],CUT)`	
	Call intersection of contour elements
`N20 IF ISPOINT==FALSE GOTOF NOCUT`	Jump to NOCUT
…	

14.5 Traversing a contour element from the table - EXECTAB

Programming

```
EXECTAB (TABNAME[n])
```

Explanation of the parameter

`TABNAME[n]`	Name of table with number n of the element

Function

You can use command EXECTAB to traverse contour elements block by block in a table generated, for example, with the CONTPRON command.

Programming example

The contour elements stored in Table KTAB are traversed non-modally by means of subroutine EXECTAB. Elements 0 to 2 are passed 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

Programming

```
VARIB = CALCDAT(PT[n,2],NO,RES)
```

Explanation of the parameters

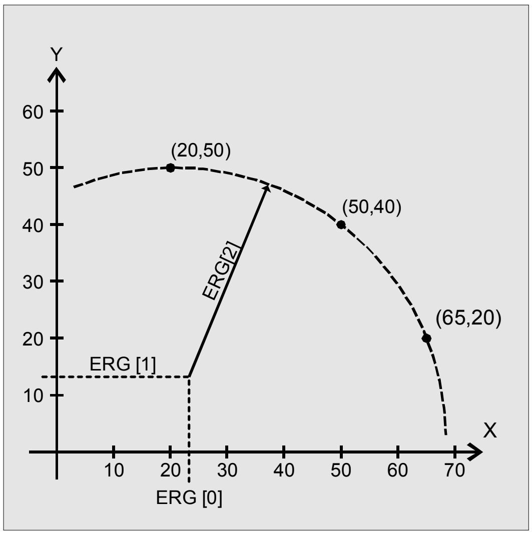
`VARIB`	Variable for status TRUE = circle, FALSE = no circle
`PKT [n,2]`	Points for calculation n = number of points (3 or 4); 2 = point coordinates
`ANZ`	Number of points used for calculation: 3 or 4
`ERG [3]`	Variable for result: specification of circle center point coordinates and radius; 0 = abscissa, 1 = ordinate of circle center point; 2 = radius

Function

Calculation of radius and circle center point coordinates from three or four known circle points.
The specified points must be different.
Where 4 points do not lie directly on the circle an average value is taken for the circle center point and the radius.

Programming 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)`	Point definition
`N20 DEF REAL RES[3]`	Result
`N30 DEF BOOL STATUS`	Variable for status
`N40 STATUS = CALCDAT(PT,3,RES)`	Call calculated circle data
`N50 IF STATUS == FALSE GOTOF ERROR`	Jump to error

■

Tables

15.1 List of instructions

Legend:

[1] Default setting at beginning of program (factory settings of the control, if nothing else programmed).

[2] The group numbers correspond to the table "List of G functions/Preparatory functions" in /PG/, Programming Guide Fundamentals, Section 12.3

[3] Absolute end points: modal; incremental end points: non-modal; otherwise modal/non-modal (m, s) depending on syntax of G function.

[4] As arc centers, IPO parameters act incrementally. They can be programmed in absolute mode with AC. With other meanings (e.g. pitch), the address modification is ignored.

[5] The vocabulary word is not valid for SINUMERIK FM-NC/810D

[6] The vocabulary word is not valid for SINUMERIK FM-NC/810D/NCU571

[7] The vocabulary word is not valid for SINUMERIK 810D

[8] The OEM can add two extra interpolation types. The names can be changed by the OEM.

[9] The vocabulary word is only valid for SINUMERIK FM-NC

[10] Extended address notation cannot be used for these functions.

Name	Meaning	Value assignment	Description, comment	Syntax	Modal/ non-modal (s)	Group[2]
:	Block number - main block (see N)	0 ... 9999 9999 integers only, without signs	Special code for blocks – instead of N... ; this block should contain all instructions for a following complete machining section	e.g. **:**20		
A	Axis	Real			m,s [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 Z0 ; separate block	s	3
AND	Logical AND					
ANG	Contour angle	Real				
AP	Angle polar	0, ..., ± 360°			m,s [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,s [3]	
AROT	Programmable rotation (additive rotation)	Rotation about 1st geom. axis: -180° .. 180° 2. g. axis: -89.999° .. 90° 3. g. 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... ;separate AROT RPL= 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,s[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	Up to SW 5, axis identifier is converted to string (get axis as string) With SW 6 and higher, the spindle number converts the string (get string)	Up to SW 5 AXIS from SW 6 string	Name of file can be added	AXSTRING(SPI(n)) From SW 6 AXSTRING[SPI(n)]		
B	Axis	Real			m,s[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,s[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 programming 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 subprogram call			CALL PROGVAR		
CALLPATH	Programmable search path for subprogram calls		A path can be programmed to the existing NCK file system with CALLPATH.	CALLPATH(/_N_WKS_DIR/_N_MYWPD/subprogram_ID_SPF)		
CANCEL	Cancel modal synchronized action	INT	Cancel with specified ID. Without parameter: 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 at tool center point (center-point path) (constant feed in tool-center-point)				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 SW 3.5 and higher 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			
CLGOF	Const. workpiece speed for centerless grinding OFF					
CLGON	Const. workpiece speed for centerless grinding ON					
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
CONTPRON	Activate contour preparation (contour preparation ON)				m	49
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: software on "Synchronization fine / coarse", IPOSTOP: software on setpoint-dependent termination of overlaid movement			
COUPDEL	Delete ELG group (couple delete)					
COUPOF	ELG group / synchronous spindle pair OFF (couple OFF)					
COUPON	ELG group / synchronous spindle pair ON (couple ON)					
COUPRES	Reset ELG group (couple reset)		Programmed values invalid; machine data values valid			
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	Maximum number of parameters: 6			
CROTS	Programmable frame rotations with solid angles (rotations in the indicated axes)			CROT X... Y... CROT Z... X... CROT Y... Z... ;own CROT RPL= block	S	
CSCALE	Scale factor for multiple axes.	FRAME	Maximum number of parameters: 2 * axis number$_{max}$			
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					
CTABEXIST	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		2 parameters 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		3 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		Leading 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		Leading value at start 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		3 parameters n, m, and memType			
CTRANS	Zero offset for multiple axes	FRAME	Max. of 8 axes			
CUT2D [1]	2 ½D tool offset (cutter compensation type 2-dimensional)				m	22
CUT2DF	2 ½D 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, ..., 9 in SW 3.5 and higher 1, ... 32 000	Includes compensation data for a certain tool T... ; D0 → compensation 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)		Implicit, 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 progr. 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					
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)		Triggers preprocessing stop			
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 synchronization			
EGONSYN	Switch on electronic gear (electronic gear ON synchronized)		With synchronization			
EGONSYNE	Switch on electronic gearing, stating approach mode (electronic gear ON synchronized)		With synchronization			
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		Switch back to normal program execution from reference point edit mode or after creating a protection zone			
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					

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					
FORI1	Feedrate for swiveling the orientation vector on the large arc				m	
FORI2	Feedrate for overlaid rotation about the swiveled orientation vector				m	
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 degree 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 predefined, integer 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... ; arc angle and center point G2 AR=... X... Z... F... ; arc 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=... ; cylinder thread G33 X... I... SF=... ; face thread G33 Z... X... K... SF=... ; taper thread (path longer in Z axis than in X axis) G33 Z... X... I... SF=... ; taper thread (path longer in X axis than in Z axis)	m	1
G34	Increase in thread pitch (progressive change)		Motion command	G34 Z... K... F_{ZU}=...	m	1
G35	Decrease in thread pitch (degressive change)		Motion command	G35 Z... K... F_{AB}=...	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 with active tool radius compensation (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		Machine axes	G74 X... Z...; separate block	s	2
G75	Fixed point approach			G75 FP=.. X1=... Z1=... ; separate block	s	2
G90 [1]	Absolute dimensions			G90 X... Y... Z...(...) Y=AC(...) or X=AC Z=AC(...)	m s	14
G91	Incremental dimension input			G91 X... Y... Z... or X=IC(...) Y=IC(...) Z=IC(...)	m s	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
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		Tool radius compensation		m	18
G451	Intersection of equidistances		Tool radius compensation		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					
GOTOF	Jump forwards (towards the end of the program)					
GOTOB	Jump backwards (towards the start of the program)					
GOTO	Jump instruction first forward then backward (direction initially to end of program and then to start of program Start					
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	Can be set by MD (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	
IDS	Identification of static synchronized actions					
IF	Introduce conditional jump		Structure: IF-ELSE-ENDIF			
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					
INT	Data type: Integer with leading sign	- (2^{31}-1), ..., 2^{31}-1				
INTERSEC	Calculate intersection between two contour elements	VAR REAL [2]	Error status BOOL			
IP	Variable interpolation parameter	Real				
IPOBRKA	Motion criterion from braking ramp activation		Braking ramp with 100% to 0%	IPOBRKA=.. or IPOBRKA(<axis>[,REAL])	m	
IPOENDA	Motion ends when "IPO Stop" is reached			IPOENDA=.. or IPOENDA[n]..	m	
IPTRLOCK	Freeze start of the untraceable program section at next machine function block.		Freeze interruption pointer		m	
IPTRUNLOCK	Set end of untraceable program section at current block at time of interruption		Set interruption pointer		m	
ISAXIS	Check if geometry axis 1 – 3 specified as parameter exist	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			
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	Subprogram number	Integer, up to 7 places		L10	s	
LEAD [5]	Lead angle	Real			m	

LEADOF	Leading value coupling OFF (lead off)				
LEADON	Leading 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 manufact.		
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 subprogram call		Without subprogram 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 1, 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 MMC/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, w/o signs	Can be used to identify blocks by means of a number; written 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	Vocabulary word 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 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 2. contact point of tool: XH, YH, ZH	Parameter settings as follows: Direction vectors normalized A6=0, B6=0, C6=0 Arc 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 referred to path		Transformation package handling, see /FB/, TE4		m	51
ORIRPY	Orientation angle via RPY angle				m	50
ORIS [5]	Orientation modification (orientation smoothing factor)	Real	Referring 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 machine coordinate system				m	25
ORIWKS [1,6]	Tool orientation in the workpiece coordinate system				m	25
OS	Oscillation ON / OFF	Integer, w/o signs				
OSC [6]	Continuous tool orientation smoothing				m	34
OSCILL	Axis assignment for oscillation - activate oscillation		Axis: 1-3 infeed axes		m	

OSCTRL	Oscillation control options	Integer, w/o signs			M	
OSE	Oscillating: End point				m	
OSNSC	Oscillating: Number of spark-out cycles 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
OST1	Oscillating: Stop at left reversal point	Real			m	
OST2	Oscillating: Stop at right reversal point	Real			m	
OVR	Speed override	1, ..., 200%			m	
OVRA	Axial speed override	1, ..., 200%			m	

P	Number of subprogram passes	1 ... 9999, integer w/o sign		e.g. L781 P... ; separate block		
PCALL	Call subprograms 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	
POLFMIN	Enable axes for retraction **with** a linear connection between the axes		Selected axes	POLFMIN(AX1, AX2, ...)	m	
POLY[5]	Polynominal 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	Vocabulary word for setting the priority for interrupt processing					
PROC	First instruction in a program			Block number - PROC - identifier		
PTP	**P**oint **t**o **P**oint (point-to-point motion)				m	49
PUTFTOC	Tool offset fine for parallel dressing (continuous dressing)					
PUTFTOCF	PutFineToolCorrectionFunctionDependent: Fine tool offset dependent on a function for continuous dressing defined with FCtDEF					
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 parameters SW 5 and higher: also as settable address identifier and with numerical extension	±0.0000001, ..., 9999 9999	Number of R parameters can be set by MD	R10=3 ;R parameter assignment X=R10 ;axis value R[R10]=6 ;indirect prog.		
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	Vocabulary word for initialization of all elements of an array with the same value					
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: 1. Parameter 1, 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 around 1st geometry axis: -180° .. 180° 2. g. axis: -89.999°, ..., 90° 3. g. 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... ;own ROT RPL= block	s	3
ROUND	Round decimal places	Real				
RP	Polar radius	Real			m,s ^{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 velocity 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	Vocabulary word for initialization of all elements of an array with listed values					
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			
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				
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. programm. 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 via T No.: or via tool name:	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 current tool orientation		Necessary after reset, e.g. through manual setting		m	42
TCOFR	Determine tool length components from the orientation of the active frame				m	42
TCOFRX	Determine tool orientation of an active frame 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			m	
TMOF	Deselect tool monitoring		T no. is only necessary if the tool with this number is not active.	TMOF (T no.)		
TMON	Activate tool monitoring		T No. = 0: Deactivate monitoring function 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 (**T**ool **Off**set **ON**)		3-dimensional offset direction	TOFFON (Z, 25) with offset direction Z offset value 25		
TOFRAME	Set current programmable frame to tool coordinate system		Frame rotations in tool direction		m	53
TOFRAMEX	X axis parallel to tool direction, secondary axis Y, Z				m	53
TOFRAMEY	Y axis parallel to tool direction, secondary axis Z, X				m	53
TOFRAMEZ	Z axis parallel to tool direction, secondary axis X, Y				m	53
TOFROF	Frame rotations in tool direction OFF				m	53
TOFROT	Z axis parallel to tool orientation		Frame rotations ON Rotation component of programmed 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 tool holder)				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 settable per 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 defined 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	Vocabulary word: 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)		
WAITM	Wait for marker in specified channel; end previous block with exact stop			WAITM(1,1,2)		
WAITMC	Wait for marker in specified channel; exact stop only if the other channels have not yet reached the marker			WAITMC(1,1,2)		

WAITP	Wait for end of traversing			WAITP(X) ; separate block		
WAITS	Waiting to reach spindle position			WAITS (main spindle) WAITS (n,n,n)		
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,s [3]	
XOR	Logical exclusive OR					
Y	Axis	Real			m,s [3]	
Z	Axis	Real			m,s [3]	

Legend:

[1] Default setting at beginning of program (factory settings of the control, if nothing else programmed).

[2] The group numbering corresponds to the numbering in table "Overview of instructions" in Section 11.3

[3] Absolute end points: modal; incremental end points: non-modal; otherwise modal/non-modal (m, s) depending on syntax of G function.

[4] As arc centers, IPO parameters act incrementally. They can be programmed in absolute mode with AC. With other meanings (e.g. pitch), the address modification is ignored.

[5] The vocabulary word is not valid for SINUMERIK FM-NC/810D

[6] The vocabulary word is not valid for SINUMERIK FM-NC/810D/NCU571

[7] The vocabulary word is not valid for SINUMERIK 810D

[8] The OEM can add two extra interpolation types. The names can be changed by the OEM.

[9] The vocabulary word is only valid for SINUMERIK FM-NC

[10] Extended address notation cannot be used for these functions.

15.2 List of system variables

Reference notes:

In SW 7.1 and higher the system variables are listed in a separate publication: SINUMERIK 840D/840Di/810D "Lists of System Variables".

■

Appendix

A Abbreviations

AS	Automation System
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
ASUB	Asynchronous Subroutine
BA	Mode of operation
BAG	Mode Group
BCD	Binary Coded Decimals
BCS	Basic Coordinate System
BIN	Binary Files
BIOS	Basic Input Output System
BOT	Boot Files for SIMODRIVE 611 D
BP	Basic Program
C Bus	Communication Bus
C1 .. C4	Channel 1 to channel 4
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CNC	Computerized Numerical Control
COM	Communication
COR	Coordinate Rotation

CP	Communication Processor
CPU	Central Processing Unit
CR	Carriage Return
CRT	Cathode Ray Tube
CSB	Central Service Board
CSF	Control System Flowchart (PLC programming method)
CTS	Clear To Send (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: The rotary axis is moved along the shortest path to the absolute position within one revolution.
DCD	Carrier Detect
DCE	Data Communications Equipment
DDE	Dynamic Data Exchange
DIN	German Industry Standard
DIO	Data Input/Output: Data transfer display
DIR	Directory

DLL	Dynamic Link Library: Module which can be accessed by a running program. Often contains program sections that are required by different programs.
DOS	Disk Operating System: Operating system
DPM	Dual-Port Memory
DPR	Dual-Port RAM
DRAM	Dynamic Random Access Memory
DRF	Differential Resolver Function
DRY	Dry Run
DSB	Decoding Single Block
DTE	Data Terminal Equipment
DW	Data Word
EIA Code	Special punchtape code, number of punched holes per character always odd
ENC	Encoder
EPROM	Erasable Programmable Read Only Memory
ERROR	Error from printer
FB	Function Block
FC	Function Call: Function block in the PLC
FDD	Feed Drive
FDD	Floppy Disk Drive
FEPROM	Flash EPROM

FIFO	First-In-First-Out: Memory which operates without address specification from which data are read in the same order as they are stored.
FIPO	Fine interpolator
FM	Function Module
FM-NC	Function Module – Numerical Control
FPU	Floating Point Unit
FRA	Frame Block
FRAME	Data Record (frame)
FST	Feed Stop
GUD	Global User Data
HD	Hard Disk
HEX	Abbreviation for hexadecimal
HHU	Handheld Unit
HMI	Human Machine Interace
HMS	High-resolution Measuring System
HW	Hardware
I	Input
I/O	Input/Output
I/RF	Infeed/Regenerative Feedback Unit (power supply) of SIMODRIVE 611(D)
IK (GD)	Implicit Communication (Global Data)
IKA	Interpolative Compensation Interpolative compensation

IM	Interface Module
IMR	Interface Module Receive
IMS	Interface Module Send
INC	Increment Increment
INI	Initializing Data
IPO	Interpolator
IS	Interface Signal
ISO	International Standard Organization
ISO Code	Special punchtape code, number of punched holes per character always even
JOG	Jog mode
$K_{Ü}$	Transmission Ratio
K_v	Servo Gain Factor
LAD	Ladder Diagram (PLC programming method)
LCD	Liquid Crystal Display
LEC	Leadscrew Error Compensation
LED	Light Emitting Diode
LF	Line Feed
LUD	Local User Data
MB	Megabyte
MC	Measuring Circuit
MCP	Machine Control Panel

MCS	Machine Coordinate System (Machine)
MD	Machine Data
MDA	Manual Data Automatic (MDI)
MMC	Human Machine Communication: User interface on numerical control systems for operator control, programming and simulation. MMC and HMI are identical in meaning.
MPF	Main Program File: NC parts program (main program)
MPI	Multi Port Interface
MS-	Microsoft
MSD	Main Spindle Drive
NC	Numerical Control
NCK	Numerical Control Kernel (with block preparation, traversing range, etc.)
NCU	Numerical Control Unit: Hardware unit of the NCK
NRK	Designation of the operating system of the NCK
NURBS	Non Uniform Rational B-Spline
O	Output
OB	Organization Block in the PLC
OEM	Original Equipment Manufacturer: The manufacturer of equipment that is marketed by another vendor, typically under a different name.
OI	Operator Interface
OP	Operator Panel

OPI	Operator Panel Interface
OPT	Options
OSI	Open Systems Interconnection
P Bus	Peripheral Bus
PC	Personal Computer
PCIN	Name of SW for exchanging data with the control system
PCMCIA	Personal Computer Memory Card International Association
PDB	Product Database
PG	Programming Device
PLC	Programmable Logic Control
POS	Positioning
PP	Production Planning
RAM	Random Access Memory (read-write memory)
REF	Reference Point Approach Function
REPOS	Reposition Function
RISC	Reduced Instruction Set Computer: Processor type with a small instruction set and high instruction throughput
ROV	Rapid Override
RPA	R Parameter Active: Memory area in the NCK for R-NCK for R parameter numbers
RPY	Roll Pitch Yaw: Type of coordinate system rotation
RS-232	Serial Interface (definition of interchange lines between DTE and DCE)

RTS	Request To Send (serial data interfaces)
SBL	Single Block
SD	Setting Date
SDB	System Data Block
SEA	Setting Data Active: Identification (file type) for setting data
SFB	System Function Block
SFC	System Function Call
SK	Softkey
SKP	Skip Block
SM	Stepper Motor
SOP	Shopfloor-Oriented Programming
SPF	Sub-Program File (subroutine file)
SR	Subroutine
SRAM	Static RAM (battery-backed)
SRK	Cutter Radius Compensation
SSI	Serial Synchronous Interface
STL	Statement List
SW	Software
SYF	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
TOA	Tool Offset Active: Identification (file type) for tool offsets
TRANSMIT	Transform Milling into Turning: Coordinate conversion on turning machines for milling operations
TRC	Tool Radius Compensation
UFR	User Frame: Zero offset
WCS	Workpiece Coordinate System (Work)
WO	Work Offset
WOA	Work Offset Active
WPD	Workpiece Directory
ZO	Zero Offset (WO)
ZOA	Zero Offset Active (WOA): Identification (file type) for zero offset data
µC	Microcontroller

A Abbreviations

B Terms

Important terms are listed below in alphabetical order, accompanied by explanations. Cross-references to other entries in this glossary are indicated by the symbol "->".

A

A spline

The A spline runs tangentially through the programmed interpolation points (3rd degree polynomial).

Absolute dimension

A destination for an axis movement is defined by a dimension that refers to the origin of the currently active coordinate system. See also -> incremental dimension.

Acceleration with jerk limitation

In order to obtain the optimum acceleration gradient for the machine while providing effective protection for the mechanical components, the machining program offers a choice between instantaneous acceleration and continuous (smooth) acceleration.

Access rights

The CNC program blocks and data are protected by a 7-level system of access restrictions:

- Three password levels for system manufacturers, machine manufacturers and users and
- Four keyswitch settings which can be evaluated via the PLC.

Activate/deactivate

Working area limitation is a means of restricting the axis movement over and above the restrictions imposed by the limit switches. A pair of values delimiting the protected zone area can be specified for each axis.

Address

An address is the designation for a specific operand or operand area, e.g. input, output, etc.

Alarms

All -> messages and alarms are displayed in plain text on the operator panel. Alarm text also includes the date, time and corresponding symbol for the reset criterion.
Alarms and messages are displayed separately.

1. Alarms and messages in the parts program
   Alarms and messages can be displayed directly from the parts program in plaintext.
2. Alarms and messages from PLC
   Alarms and messages relating to the machine can be displayed from the PLC program in plaintext. No additional function block packages are required for this purpose.

Analog input/output module

Analog input/output modules are signal transducers for analog process signals.
Analog input modules convert analog measured values into digital values that can be processed in the CPU.
Analog output modules convert digital values into manipulated variables.

Approach fixed machine point

Approach motion towards one of the predefined -> fixed machine points.

Archiving

Exporting files and/or directories to an **external** storage device.

Asynchronous subroutine

- A parts program that can be started asynchronously (or independently) of the current program status by means of an interrupt signal (e.g. "High-speed NC input" signal).

Automatic

Control system operating mode (block-sequential to DIN): Mode in NC systems in which a -> parts program is selected and continuously executed.

Auxiliary functions

Auxiliary functions can be used to pass -> parameters to the -> PLC in -> parts programs, triggering reactions there which are defined by the machine manufacturer.

Axes

CNC axes are classified according to their functional scope as:

- Axes: Interpolative path axes
- Positioning axes: Non-interpolative infeed and positioning axes with axis-specific feedrates; axes can move across block limits. Positioning axes need not be involved in workpiece machining as such and include tool feeders, tool magazines, etc.

Axis address

See -> axis identifier

Axis identifier

In compliance with DIN 66217, axes are identified as X, Y and Z for a right-handed rectangular -> coordinate system.
-> Rotary axes rotating around X, Y, Z are assigned the identifiers A, B, C. Additional axes, which are parallel to those specified, can be identified with other letters.

Axis name

See -> axis identifier

B

B spline The programmed positions for the B spline are not interpolation points, but merely "check points". The curve generated does not pass directly through these check points, but only in their vicinity (1st, 2nd or 3rd degree polynomial).

Back up A copy of the memory contents stored on an external device.

Backlash compensation Compensation of a mechanical machine backlash, e.g. backlash due to reversal of leadscrews. The backlash compensation can be entered separately for each axis.

Backup battery The backup battery provides non-volatile storage for the -> user program in the -> CPU and ensures that defined data areas and flags, timers and counters are retentive.

Backup memory The backup memory guarantees the backup of memory areas of the -> CPU without backup battery. A programmable number of timers, counters, flags and data bytes can be backed up.

Base axis Axis whose setpoint or actual value is employed in calculating a compensatory value.

Basic coordinate system Cartesian coordinate system, is mapped onto machine coordinate system by means of transformation.
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 only to the axis identifiers.

Baud rate Rate at which data transmission takes place (bit/s).

Blank The unmachined workpiece.

Block All files required for programming and program execution are known as blocks.

Block A section of a -> parts program terminated with a line feed.
A distinction is made between -> main blocks and -> subblocks.

Block search The block search function allows selection of any point in the parts program at which machining must start or be continued. The function is provided for the purpose of testing parts programs or continuing machining after an interruption.

Booting Loading the system program after Power ON.

Bus connector A bus connector is an S7-300 accessory that is supplied with the -> I/O modules. The bus connector extends the -> S7-300 bus from the -> CPU or an I/O module to the next adjacent I/O module.

C

C axis Axis about which the tool spindle describes a controlled rotational and positioning movement.

C spline The C spline is the best known and the most widely used spline. The spline passes through each of the interpolation points at a tangent and along the axis of curvature. 3rd-degree polynomials are used.

Channel A channel is characterized by the fact that it can process a -> parts program independent of other channels. A channel exclusively controls the axes and spindles assigned to it. Parts program sequences of different channels can be coordinated through -> synchronization.

Channel structure The channel structure makes it possible to process the -> programs of individual channels simultaneously and asynchronously.

Circular interpolation The -> tool is required to travel in a circle between defined points on the contour at a specified feed while machining the workpiece.

CNC -> NC

CNC high-level language The high-level language offers: -> user variables, -> predefined user variables, -> system variables, -> indirect programming, -> arithmetic and angular functions, -> relational and logic operations, -> program jumps and branches, -> program coordination (SINUMERIK 840D), -> macros.

CNC programming language The CNC programming language is based on DIN 66025 with high-level language expansions. The -> CNC programming language and -> high-level language expansions support the definition of macros (sequenced statements).

COM Numerical control component for the implementation and coordination of communication.

Compensation axis Axis having a setpoint or actual value modified by the compensation value.

Compensation table Table of interpolation points. It supplies the compensation values of the compensation axis for selected positions of the base axis.

Compensation value Difference between the axis position measured by the position sensor and the desired, programmed axis position.

Connecting cables Connecting cables are pre-assembled or user-assembled 2-wire cables with a connector at each end. They are used to connect the -> CPU via the -> multipoint interface (MPI) to a -> programming device or to other CPUs.

Continuous-path mode The purpose of continuous-path mode is to avoid rapid deceleration -> of the path axes at parts program block boundaries and to make the transition to the next block at as constant a velocity as possible.

Contour Outline of a -> workpiece.

Contour monitoring The following error is monitored within a definable tolerance band as a measure of contour accuracy. Overloading of the drive, for example, may result in an unacceptably large following error. In such cases, an alarm is output and the axes stopped.

Coordinate system See -> machine coordinate system, -> workpiece coordinate system

CPU Central Processor Unit -> programmable controller

Cycle Protected subroutine for executing a recurring machining operation on the -> workpiece.

Cycles support The available cycles are listed in menu "Cycle support" in the "Program" operating area. Once the desired machining cycle has been selected, the parameters required for assigning values are displayed in plaintext.

D

Data block

1. Data unit of the -> PLC which can be accessed by -> HIGHSTEP programs.
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 transfer program PCIN PCIN is a routine for transmitting and receiving CNC user data, e.g. parts programs, tool offsets, etc. via the serial interface. The PCIN program can run under MS-DOS on standard industrial PCs.

Data word

A data unit, two bytes in size, within a -> data block.

Deletion of distance-to-go

Command in parts program which stops machining and clears the remaining path distance to go.

Design

- The SINUMERIK FM-NC is installed in the CPU tier of the SIMATIC S7-300. The 200mm wide, fully encapsulated module has the same external design as the SIMATIC S7-300 modules.
- The SINUMERIK 840D is installed as a compact module in the SIMODRIVE 611D converter system. It has the same dimensions as a 50mm wide SIMODRIVE 611D module. The SINUMERIK 840D comprises the NCU module and the NCU box.
- The SINUMERIK 810D has the same design as the SIMODRIVE 611D with a width of 150mm. The following components are integrated: SIMATIC S7-CPU, 5 digital servo drive controls and 3 SIMODRIVE 611D power modules.

Diagnosis

1. Control operating area
2. The control incorporates a self-diagnosis program and test routines for servicing: Status, alarm and service displays.

Digital input/output module

Digital modules are signal transducers for binary process signals.

Dimensions in metric and inch systems

Position and lead/pitch values can be programmed in inches in the machining program. The control is set to a basic system regardless of the programmable unit of measure (G70/G71).

DRF

Differential Resolver Function NC function which generates an incremental zero offset in AUTOMATIC mode in conjunction with an electronic handwheel.

Drift compensation

When the CNC axes are in the constant motion phase, automatic drift compensation is implemented in the analog speed control.

Drive

- SINUMERIK FM-NC has an analog ±10V interface to the SIMODRIVE 611A converter system.
- The SINUMERIK 840D control system is linked to the SIMODRIVE 611D converter system via a high-speed digital parallel bus.

E

Editor

The editor makes it possible to create, modify, extend, join and insert programs/texts/program blocks.

Electronic handwheel
Electronic handwheels can be used to traverse the selected axes simultaneously in manual mode. The handwheel clicks are analyzed by the increment analyzer.

Exact stop
When an exact stop is programmed, a position specified in the block is approached accurately and, where appropriate, very slowly. 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 -> parts program continues execution at the next block.

External zero offset
A zero offset specified by the -> PLC.

F

Fast retraction from contour
When an interrupt is received, it is possible to initiate a motion via the CNC machining program which allows the tool to be retracted quickly from the workpiece contour currently being machined. The retraction angle and the distance retracted can also be parameterized. An interrupt routine can be executed after the rapid retraction.
(SINUMERIK FM-NC, 840D).

Feedforward control, dynamic
Contour inaccuracies resulting from following errors can be almost completely eliminated by the dynamic, acceleration-dependent feedforward control function. Feedforward control ensures an excellent degree of machining accuracy even at high tool path velocities.
Feedforward control can only be selected or deselected for all axes together via the parts program.

Feedrate override
The current feedrate setting entered via the control panel or by the PLC is overlaid on the programmed feedrate (0–200%). The feedrate can also be corrected by a programmable percentage factor (1–200%) in the machining program.

Finished-part contour
Contour of the finished workpiece. See also -> blank.

Fixed machine point
A point defined uniquely by the machine tool, such as the reference point.

Fixed-point approach
Machine tools can execute defined approaches to fixed points such as tool-change points, loading points, pallet-change points, etc. The coordinates of these points are stored on the control. Where possible, the control moves these axes in -> rapid traverse.

Frame

A frame is a calculation rule that translates one Cartesian coordinate system into another Cartesian coordinate system. A frame contains the components -> zero offset, -> rotation, -> scaling and -> mirroring.

G

General reset

The following memories of the -> CPU are erased by a general reset operation:

- -> Working memory
- Read/write area of the -> load memory
- -> System memory
- -> Backup memory

Geometry

Description of a -> workpiece in the -> workpiece coordinate system.

Geometry axis

Geometry axes are used to describe a 2- or 3-dimensional area in the workpiece coordinate system.

Global main run/subroutine

Each global main run/subroutine can be stored only once under its name in the directory; the same program name in different directories with different contents is not possible as a global program.

Ground

"Ground" is the term applied to all the electrically inactive, interconnected parts of a piece of equipment which cannot carry any hazardous contact voltage even in the event of a fault.

H

Helical interpolation

The helical interpolation function is ideal for machining internal and external threads using form milling cutters and for milling lubrication grooves. The helix comprises two movements:

1. Circular movement in one plane
2. Linear movement perpendicular to this plane.

High-speed digital inputs/outputs

As an example, high-speed CNC program routines (interrupt routines) can be started via the digital inputs. High-speed, program-driven switching functions can be initiated via the digital CNC outputs (SINUMERIK 840D).

HIGHSTEP

Combination of the programming features for the -> PLC in the S7-300/400 range.

I

I/O module

I/O modules create the link between the CPU and the process. I/O modules are:

- ->Digital input/output modules
- ->Analog input/output modules
- ->Simulator modules

Identifier

In accordance with DIN 66025, identifiers (names) for variables (arithmetic variables, system variables, user variables), for subroutines, for vocabulary words and for words can contain several address letters. These letters have the same meaning as the words in the block syntax. Identifiers must be unique. Identical identifiers must not be used for different objects.

Inch system of measurement

System of measurement that defines distances in "inches" and fractions thereof.

Increment

Travel path length specification based on number of increments. The number of increments can be stored as a -> setting data or selected with keys labeled with 10, 100, 1000, 10 000.

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 also -> absolute dimension.

Initialization block

Initialization blocks are special -> program blocks. They contain values which must be assigned before the program is executed. Initialization blocks are used primarily for initializing predefined data or global user data.

Initialization file

An initialization file can be created for each -> workpiece. In it, the various variable value instructions which apply exclusively to one workpiece can be stored.

Intermediate blocks

Movements with selected tool offset (G41/G42) can be interrupted by a limited number of intermediate blocks (blocks without axis motions in the offset plane). When such blocks are used, the tool offset can still be calculated correctly. The permissible number of intermediate blocks read in advance by the control can be set via system parameters.

Interpolative compensation

Interpolative compensation provides a means of compensating for leadscrew errors (LEC) and measuring-system errors (MSEC) resulting from the production process.

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.

Interrupt routine Interrupt routines are special -> subroutines which can be started by events (external signals) in the machining process. The parts program block being processed is aborted and the axis position at the instant of interruption is stored automatically.

Inverse-time feedrate On SINUMERIK FM-NC and 840D controls, it is possible to program the time required to traverse the path of a block instead of the feedrate speed for the axis movement (G93).

J

Jog Control system operating mode (setup): The machine can be set up in Jog mode. Individual axes and spindles can be jogged by means of direction keys. Other functions in Jog mode are -> reference point approach, -> Repos and -> Preset -> (set actual value).

K

Keyswitch

1. **S7-300**: The keyswitch is the mode selector switch on the -> CPU. The keyswitch is operated by means of a removable key.
2. **840D**: The keyswitch on the -> machine control panel has 4 positions which are assigned functions by the operating system of the control. There are also three keys of different colors belonging to the keyswitch that can be removed in the specified positions.

$K_Ü$ Transmission Ratio

K_v Servo gain factor, control variable of a control loop

L

Languages The user interface texts, system messages and alarms are available in five system languages (floppy disk):
English, French, German, Italian and **Spanish**.
The user can select **two** of the listed languages at a time in the control.

Leadscrew error compensation Compensation of mechanical inaccuracies in a leadscrew involved in the feed motion. Errors are compensated by the control based on stored deviation measurements.

Limit speed

Minimum/maximum (spindle) speed: The maximum speed of a spindle can 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

On the CPU 314 of the -> PLC, the load memory is identical with the -> working memory.

Look Ahead

The **Look Ahead** function is a means of optimizing the machining velocity by looking ahead over a parameterizable number of traversing blocks.

Look Ahead for contour violations

The control detects and reports the following types of collision:

1. Path is shorter than tool radius.
2. Width of inside corner is less than the tool diameter.

M

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 for direct control of the machine tool 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) measurement systems.

Machining channel

A channel structure makes it possible to reduce downtimes by allowing sequences of motions to be executed in parallel. For example, a loading gantry can execute its movements during a machining operation. In this case, a CNC channel ranks as an autonomous CNC control complete with decoding, block preparation and interpolation.

Macros A set of instructions grouped under one identifier. In the program, the identifier represents the set of grouped 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 name in which other main programs, subroutines or -> cycles may be called.

MDA Control system operating mode: Manual Data Automatic. In the MDA mode, individual program blocks or block sequences with no reference to a main program or subroutine can be input and executed immediately afterwards through actuation of the NC Start key.

Measuring circuits

- SINUMERIK FM-NC: The requisite control circuits for axes and spindles are integrated in the control module as standard. A maximum total of 4 axes and spindles can be implemented, with no more than 2 spindles.
- SINUMERIK 840D: The signals from the sensors are analyzed in the SIMODRIVE 611D drive modules. The maximum configuration is a total of 8 axes and spindles, with up to 5 spindles being permissible.

Messages All messages programmed in the parts program and -> alarms detected by the system are displayed in plain text with date and time and the corresponding symbol for the delete criterion on the operator panel. Alarms and messages are displayed separately.

Metric system Standardized system of units for lengths in millimeters (mm), meters (m), etc.

Mirroring Mirroring exchanges the leading signs of the coordinate values of a contour in relation to an axis. Analogously, several axes can be mirrored simultaneously.

Mode An operating concept on a SINUMERIK control. The modes -> Jog, -> MDA, -> Automatic are defined.

Mode group All axes/spindles are assigned to one and only one channel at any given time. Each channel is assigned to a mode group
The same -> mode is always assigned to the channels of a mode group.

Multipoint interface

The multipoint interface (MPI) is a 9-pin sub-D port. A parameterizable number of devices can be connected to an MPI for the purpose of communicating with one another:

- Programming devices
- MMI (HMI) systems
- Other automation systems

The "Multipoint Interface MPI" parameter block of the CPU contains the -> parameters which define the properties of the multipoint interface.

N

NC

Numerical Control It incorporates all the components of the machine tool control system: -> NCK, -> PLC, -> MMC, -> COM.
Note: CNC (computerized numerical control) would be a more appropriate description for the SINUMERIK 840D control or FM-NC : computerized numerical control.

NCK

Numerical Control Kernel: Component of the NC control which executes -> parts programs and essentially coordinates the movements on the machine tool.

Network

A network is the interconnection of several S7-300s and other terminal devices such as a programming device, for example, interlinked by means of -> connecting cables. The networked devices interchange data via the network.

Node number

The node number is the "contact address" of a -> CPU or the -> programming device or another intelligent I/O module if these devices are exchanging data with one another via a -> network. The node number is assigned to the CPU or the programming device
by the S7 tool -> "S7 Configuration".

NRK

Numeric Robotic Kernel (operating system of the -> NCK)

NURBS

Motion control and path interpolation are implemented internally in the control on the basis of NURBS (Non-Uniform Rational B Splines). A standard procedure is thus available (SINUMERIK 840D) as an internal control function for all modes of interpolation.

O

Oblique-plane machining

Drilling and milling operations on workpiece surfaces which are oblique to the coordinate planes of the machine are supported by the "Oblique surface machining" function.

OEM
The scope for implementing individual solutions (OEM applications) for the SINUMERIK 840D has been provided for machine manufacturers who wish to create their own operator interface or integrate process-oriented functions in the control.

Offset memory
Data area in the control in which tool offset data are stored.

Operator interface
The operator interface (OI) is the human-machine interface of a CNC. It takes the form of a screen and has eight horizontal and eight vertical softkeys.

Oriented spindle stop
Stops the workpiece spindle at a specified orientation angle, e.g. to perform an additional machining operation at a specific position.

Oriented tool retraction
RETTOOL: If machining is interrupted (e.g. when a tool breaks), a program command can be used to retract the tool in a user-specified orientation by a defined distance.

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.

P

Parameters

1. **S7-300**: The S7-300 uses two types of parameter:
   - Parameter of a STEP7 statement
     A parameter of a STEP7 statement is the address of the operand to be processed or a constant.
   - Parameter of a -> parameter block
     A parameter of a parameter block determines the behavior of a module.
2. **840D/FM-NC**:
   - Control operating area
   - Computation parameter, can be set any number of times or queried by the programmer for any purpose in the parts program.

Parts program
A sequence of instructions to the NC control which combine to produce a specific -> workpiece by performing certain machining operations on a given -> blank.

Parts program management
The parts program management function can be organized according to -> workpieces. The size of the user memory determines the number of programs and data to be managed. Each file (programs and data) can be given a name consisting of a maximum of 16 alphanumeric characters.

Path axis

Path axes are all the machining axes in the -> channel which are controlled by the -> interpolator such that they start, accelerate, stop and reach their end positions simultaneously.

Path feed

The 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. With a resolution of 0.1mm, for example, the maximum programmable path velocity is 1000 m/min.

PG

Programming Device

PLC

Programmable Logic Control Component of the -> NC: Programmable controller for processing the control logic on the machine tool.

PLC program memory

- SINUMERIK FM-NC:
  The PLC user program of CPU 314, the user data and the basic PLC program are stored together in the PLC user memory. A user memory of 24 KB is available on the S7-CPU314.

- SINUMERIK 840D:
  The PLC user program, the user data and the basic PLC program are stored together in the PLC user memory. The PLC user memory can be expanded to up to 96 KB.

PLC programming

The PLC is programmed with the **STEP7** software. The STEP 7 programming software is based on the standard **WINDOWS** operating system and incorporates the functionality of STEP5 programming with innovative expansions and developments.

Polar coordinates

A coordinate system which defines the position of a point on a plane in terms of its distance from the origin and the angle formed by the radius vector with a defined axis.

Polynomial interpolation

Polynomial interpolation provides a means of generating a very wide range of curves, including **straight-line, parabolic and exponential functions** (SINUMERIK 840D).

Positioning axis

An axis which performs an auxiliary movement on a machine tool (e.g. tool magazine, pallet transport). Positioning axes are axes that do not interpolate with the -> path axes.

Power ON

The action of switching the control off and then on again.

Preset

The control zero point can be redefined in the machine coordinate system by means of the Preset function. Preset does not cause the axes to move; instead, a new position value is entered for the current axis positions.

Program

1. Control operating area
2. Sequence of instructions to the control system.

Program block

Program blocks contain the main programs and subroutines of the -> parts programs.

Programmable frames

Programmable -> frames can be used to define new coordinate system starting points dynamically while the parts program is running. 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 controllers whose functions are stored as a program in the control unit. The design and wiring of the unit are not, therefore, dependent on the control functions. Programmable logic controllers have the same structure as a computer, i.e. they consist of a CPU with memory, input/output modules and an internal bus system. The I/Os and programming language are selected according to the requirements of the control technology involved.

Programmable working area limitation

Limitation of the movement area of the tool to within defined, programmable limits.

Programming key

Characters and character sequences which have a defined meaning in the programming language for -> parts programs (see Programming Guide).

Protection zone

Three-dimensional area within a -> working area which the tool tip is not permitted to enter.

Q

Quadrant error compensation

Contour errors on quadrant transitions caused by frictional fluctuations on guideways can be largely eliminated by means of quadrant error compensation. A circularity test is performed to parameterize the quadrant error compensation function.

R

R parameter
Calculation parameter. The programmer can assign or request the values of the R parameter in the -> parts program as required (R variable).

Rail
This rail is used to mount the modules of the S7-300 system.

Rapid traverse
The highest traversing speed of an axis used, for example, to bring the tool from an idle position to the -> workpiece contour or retract it from the workpiece contour.

Reference point
Point on the machine tool with which the measuring system of the -> machine axes is referenced.

Reference point approach
If the position measuring system used is not an absolute-value encoder, then a reference point approach operation is required to ensure that the actual values supplied by the measuring system are in accordance with the machine coordinate values.

Remanence
Data areas in data blocks, such as timers, counters and flags are remanent if their contents are not lost in the case of a restart or Power Off.

REPOS

1. Reapproach contour, triggered by operator
   REPOS allows the tool to be returned to the interrupt position by means of the direction keys.
2. Programmed contour reapproach
   A selection of approach strategies are available in the form of program commands: Approach point of interruption, approach start of block, approach end of block, approach a point on the path between start of block and interruption.

Rigid tapping
This function is used to tap holes without the use of a compensating chuck. The spindle is controlled as an interpolative rotary axis and drill axis, with the result that threads are tapped precisely to the final drilling depth, for example, in blind tapped holes (precondition: Spindle axis mode).

Rotary axis
Rotary axes cause the tool or workpiece to rotate to a specified angle position.

Rotary axis, continuously turning
The range of motion of a rotary axis can be set to smaller than 360 degrees or defined as continuous in both directions, depending on the application. Continuously turning rotary axes are used, for example, for eccentric machining, grinding and winding.

Rotation

Component of a -> frame which defines a rotation of the coordinate system through a specific angle.

Rounding axis

Rounding axes cause the workpiece or tool to rotate to an angle position described on a graduated grid. When the grid position has been reached, the axis is "in position".

S

S7 Configuration

S7 Configuration is a tool for parameterizing modules. S7 Configuration is used to set a variety of -> parameter blocks of the -> CPU and the I/O modules on the -> programming device. These parameters are uploaded to the CPU.

S7-300 bus

The S7-300 bus is a serial data bus which supplies modules with the appropriate voltage and via which they exchange data with one another. The connection between the modules is made by means of -> bus connectors.

Safety functions

The control includes continuously active monitoring functions which detect faults in the -> CNC, the programmable controller (-> PLC) and the machine so early that damage to the workpiece, tool or machine rarely occurs. In the event of a fault, the machining operation is interrupted and the drives stopped. The cause of the malfunction is logged and an alarm issued. At the same time, the PLC is notified that a CNC alarm is pending.

Scaling

Component of a -> frame which causes axis-specific scale alterations.

Serial RS-232 interface

For data input / output, one serial RS-232 interface is available on the MMC module MMC100, and two serial RS-232 interfaces on the MMC modules MMC101 and MMC102. Machining programs and manufacturer and user data can be imported and exported via these interfaces.

Services

Control operating area

Setting data

Data which provide the control with information about properties of the machine tool in a way defined by the system software.

Simulator module

A simulator module is a module,

- on which, via control elements, digital input values can be simulated and
- digital output values are displayed.

Softkey

A key whose name appears on an area of the screen. The choice of softkeys displayed is adapted dynamically to the operating situation. The freely assignable function keys (softkeys) are assigned to functions defined in the software.

Software limit switches

Software limit switches define the limits of the travel range of an axis and prevent the slide contacting the hardware limit switches. Two pairs of values can be assigned per axis and activated separately via the -> PLC.

Spindles

The spindle functionality is a two-level construct:

1. Spindles: Speed or position-controlled spindle drives analog digital (SINUMERIK 840D)
2. Auxiliary spindles: Speed-controlled spindle drives "auxiliary spindle" function package, e.g. for power tools.

Spline interpolation

Using the spline interpolation function, the control is able to generate a smooth curve from just a small number of specified interpolation points along a setpoint contour.

Standard cycles

Standard cycles are used to program machining operations which repeat frequently:

- For drilling/milling
- For turning (SINUMERIK FM-NC)

The available cycles are listed in menu "Cycle support" in the "Program" operating area. Once the desired machining cycle has been selected, the parameters required for assigning values are displayed in plaintext.

Subblock

Block prefixed by "N" containing information for a machining step such as a position parameter.

Subroutine

A sequence of instructions of a -> parts program which can be called repeatedly with different initial parameters.
A subroutine is called from within a main program. Each subroutine can be blocked against unauthorized output. -> Cycles are a type of subroutine.

Synchronization

Instructions in -> parts programs for coordination of the operations in different -> channels at specific machining points.

Synchronized actions

1. Auxiliary function output
   While a workpiece is being machined, technological functions (-> auxiliary functions) can be output from the CNC program to the PLC. These auxiliary functions control, for example, ancillary equipment on the machine tool such as the sleeve, gripper, chuck, etc.
2. High-speed auxiliary function output
   The acknowledgement times for the -> auxiliary functions can be minimized and unnecessary halts in the machining process avoided for time-critical switching functions.

Synchronized axes

Synchronized axes require the same amount of time to traverse their path as -> geometry axes for their path.

System memory

The system memory is a memory in the CPU, where the following data is stored:

- Data required by the operating system
- The folllowing operands: Timers, counters, flags

System variable

A variable which exists although it has not been programmed by the -> parts program programmer. It is defined by the data type and the variable name, which is prefixed with **$**. See also -> User-defined variable.

T

Teach In

Teach In is a means of creating or correcting parts programs. The individual program blocks can be input via the keyboard and executed immediately.
Positions approached via the direction keys or handwheel can also be stored.
Additional information such as G functions, feedrates or M functions can be entered in the same block.

Text editor

-> Editor

Tool

Software tool for the entry and editing of -> parameters of a parameter block. Tools include:

- -> S7 Configuration
- S7 TOP
- S7 Info

Tool

A tool employed to shape the workpiece, for example, a turning tool, milling cutter, drill, laser beam, etc.

Tool nose radius compensation

A contour is programmed on the assumption that a pointed tool will be used. Since this is not always the case in practice, the curvature radius of the tool being used is specified so that the control can make allowance for it. The curvature center point is guided equidistantly to the contour at an offset corresponding to the curvature radius.

Tool offset

A tool is selected by programming a **T function** (5 decades, integer) in the block. Up to nine tool edges (D addresses) can be assigned to each T number.
The number of tools to be managed in the control is set in parameterization.

Tool radius compensation

In order to program a desired -> workpiece contour directly, the control must traverse a path equidistant to the programmed contour, taking into account the radius of the tool used (G41/G42).

Transformation

Programming in a Cartesian coordinate system, execution in a non-Cartesian coordinate system (e.g. with machine axes as rotary axes).

Traversing range

The maximum permissible travel range for linear axes is ± 9 decades. The absolute value depends on the selected input and position control resolution and the unit of measurement (inch or metric).

U

User memory

All programs and data such as parts programs, subroutines, comments, tool offsets, zero offsets/frames and channel and program user data can be stored in the common CNC user memory.

User program

User programs for S7-300 automation systems are created using the STEP 7 programming language. The user program has a modular structure and is comprised of individual blocks.
The basic block types are:
Code blocks: Contain the STEP 7 commands.
Data blocks: Contain constants and variables for the STEP 7 program.

User-defined variable

Users can define variables in the -> parts program or data block (global user data) for their own use. A definition contains a data type specification and the variable name. See also -> system variable.

V

Variable definition A variable is defined through the specification of a data type and a variable name. The variable name can be used to address the value of the variable.

Velocity control In order to achieve an acceptable travel velocity in movements which call for very small adjustments of position in a block, the control can -> look ahead.

Vocabulary words Words with a specific notation which have a defined meaning in the programming language for -> parts programs.

W

Work offset -> Zero offset

Working memory The working storage is a Random Access Memory in the -> CPU which the processor accesses as it executes the application program.

Working space Three-dimensional zone into which the tool tip can be moved on account of the physical design of the machine tool.
See also -> protection zone.

Workpiece Part to be produced/machined by the machine tool.

Workpiece contour Setpoint contour of the -> workpiece to be produced/machined.

Workpiece coordinate system The origin of the workpiece coordinate system is 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 forms the starting point for the -> workpiece coordinate system. It is defined by distances to the machine zero.

X

Y

Z

Zero offset

Specification of a new reference point for a coordinate system through reference to an existing zero and a -> frame.
-> Work offset

1. Settable
   SINUMERIK FM-NC: Four independent zero offsets can be selected for each CNC axis.
   SINUMERIK 840D: A parameterizable number of settable zero offsets is available for each CNC axis. Each of the zero offsets can be selected by G functions and selection is exclusive.
2. External
   All offsets which define the position of the workpiece zero can be overlaid with an external zero offset
   - defined by handwheel (DRF offset) or
   - defined by the PLC.
3. Programmable
   Zero offsets can be programmed for all path and positioning axes by means of the TRANS instruction.

■

D Index

E Commands, Identifiers

To
Siemens AG

A&D MC BMS
P.O. 3180

D-91050 Erlangen, Germany

Phone: +49 (0) 180 / 5050 – 222 [Hotline]

Fax: +49 (0) 9131 / 98 – 2176 [Documentation]

Email: motioncontrol.docu@erlf.siemens.de

Suggestions

Correction

For publication/manual:

SINUMERIK 840D/840Di/810D

Programming Guide Advanced

General Documentation

From

Name:

Company/Dept.

Address: ______________________________________

Zip code: ______________________________________

Phone: __________ / ___________________________

Fax: ____________ / ___________________________

Programming Guide

Order No.: 6FC5298-7AB10-0BP0
03.04 Edition

Should you come across any printing errors when reading this publication, please notify us on this sheet. Suggestions for improvement are also welcome.

Suggestions and/or corrections

Overview of SINUMERIK 840D/840Di/810D Documentation (03.2004)

General Documentation

- *SINUMERIK* 840D/810D — Brochure
- *SINUMERIK* 840D/840Di/810D/ — Catalog Ordering Info. NC 60 ***)**
- *SINUMERIK SIROTEC SIMODRIVE* Accessories — Catalog Accessories NC-Z

User Documentation

- *SINUMERIK* 840D/810D/FM-NC — **AutoTurn**
  - Short Guide
  - Programming/Setup
- *SINUMERIK* 840D/840Di/810D — Operator's Guide **– HT 6**
- *SINUMERIK* 840D/840Di/810D — **Diagnostics Guide *)**
- *SINUMERIK* 840D/840Di/810D — **Operator's Guide *)**
  - Short Guide
  - HMI Embedded
  - HMI Advanced

User Documentation

- *SINUMERIK* 840D/840Di/810D — **Program. Guide**
  - Short Guide
  - Fundamentals ***)**
  - Advanced ***)**
  - Cycles
  - Measuring Cycles
  - ISO Turning/Milling
- *SINUMERIK* 840D/810D — Operator's Guide
  - **ManualTurn**
  - Short Guide ManualTurn
  - **ShopMill**
  - Short Guide ShopMill
  - **ShopTurn**
  - Short Guide ShopTurn
- *SINUMERIK* 840Di — **System Overview**

Manufacturer/Service Documentation

- *SINUMERIK* — **Configuring** (HW) ***)**
  - 810D
  - 840D
- *SINUMERIK* 840D/840Di/810D — **Operator Components** (HW) ***)**
- *SINUMERIK* 840D/810D — Description of Functions
  - **ManualTurn**
  - **ShopMill**
  - **ShopTurn**
- *SINUMERIK* 840D/840Di/810D — Description of Functions **Synchronized Actions**

Manufacturer/Service Documentation

- *SINUMERIK SIMODRIVE* 611D 840D/810D — Description of Functions **Drive Functions *)**
- *SINUMERIK* 840D/840Di/810D — **Description of Functions**
  - Basic Machine ***)**
  - Extended Functions
  - Special Functions
- *SINUMERIK* 840D/840Di/810D — Description of Functions **Tool Management**
- *SINUMERIK* 840D/810D — **Configuring Kit** HMI Embedded
- *SINUMERIK* 840D/810D — Description of Functions Operator Interface **OP 030**
- *SINUMERIK* 840D/810D — **MCIS**
  - Computer Link
  - Tool Data Information System
  - NC Data Management
  - NC Data Transfer
  - Tool Data Communication

Manufacturer/Service Documentation

- *SINUMERIK SIMODRIVE* — Description of Functions SINUMERIK **Safety Integrated**
- *SINUMERIK* 840D — Description of Functions **Digitizing**
- *SINUMERIK SIMODRIVE* — **Installation & Start-Up Guide *)**
  - 810D
  - 840D/611D
  - HMI
- *SINUMERIK SIMODRIVE* 840D/840Di 810D 611D
  - **Lists *)**
  - **List Syst. Var.**
- *SINUMERIK SIMODRIVE* 840D 611D — Description of Functions **Linear Motor**
- *SINUMERIK SIMODRIVE* 840D 611D — Description of Functions
  - **Hydraulics Module**
  - **Analog Module**
- *MOTION CONTROL SYSTEMS* — **EMC** Guidelines

Electronic Documentation

- *SINUMERIK SIMODRIVE* 840D/840Di/810D 611, Motors — **DOC ON CD *)** The SINUMERIK System

Manufacturer/Service Documentation

- *SINUMERIK* 840D/840Di/810D — Description of Functions **ISO Dialects** for SINUMERIK
- *SINUMERIK* 840Di — **Manual** (HW + Installation and Start-Up)
- *SINUMERIK* 840D/810D — Description of Functions **Remote Diagnosis**
- *SINUMERIK* 840D/840Di/810D — **Manual @ Event**

***)** These documents are a minimum requirement

Siemens AG
Automation & Drives
Motion Control Systems
P. O. Box 3180, D – 91050 Erlangen
Germany

www.ad.siemens.de

Order No.: 6FC5 298-7AB10-0BP0

Printed in Germany