SIEMENS

SINUMERIK 840D/840Di/810D/FM-NC

Programming Guide **04.2000 Edition**

Advanced

Manual Number: M-409 Part Number: M -0009500-0409

Overview of SINUMERIK 840D/840Di/810D/FM-NC Documentation (04.00)

General Documentation

- SINUMERIK 840D/810D/FM-NC — Brochure
- SINUMERIK 840D/840Di/810D/FM-NC/611 — Catalog Ordering Info NC 60.1 ***)** Technical Info. NC 60.2
- SINUMERIK SIROTEC SIMODRIVE Accessories — Catalog Accessories NC-Z

User Documentation

- SINUMERIK 840D/810D/FM-NC — **AutoTurn**
  - Short Guide
  - Programming (1)
  - Setup (2)
- SINUMERIK 840D/810D — Operator's Guide
  - **Unit Operator Panel**
  - **HPU**
  - **HT 6**
- SINUMERIK 840D/840Di/810D/FM-NC — **Diagnostics Guide *)**
- SINUMERIK 840D/810D/FM-NC — **Operator's Guide**
  - Short Guide
  - Operator's Guide ***)**

User Documentation

- SINUMERIK 840D/840Di/810D/FM-NC — **Program. Guide**
  - Short Guide
  - Fundamentals ***)**
  - Advanced ***)**
  - Cycles
  - Measuring Cycles
- SINUMERIK 840D/810D — Operator's Guide
  - **ManualTurn**
  - Short Guide ManualTurn
  - **ShopMill**
  - Short Guide ShopMill
- SINUMERIK 840Di — **System Overview**

Manufacturer / Service Documentation

- SINUMERIK 840D/810D — Description of Functions
  - **ManualTurn**
  - **ShopMill**
- SINUMERIK 840D/840Di/810D — Description of Functions **Synchronized Actions** Wood, Glass, Ceramics
- SINUMERIK 840D/810D — Descr. of Functions
  - **Computer Link**
  - **Tool Data Information System**
- SINUMERIK — **Configuring** (HW) ***)**
  - FM-NC
  - 810D
  - 840D

Manufacturer / Service Documentation

- SINUMERIK 840D/840Di/810D/FM-NC — **Operator Components** (HW) ***)**
- SINUMERIK SIMODRIVE 611D 840D/810D — Description of Functions **Drive Functions *)**
- SINUMERIK 840D/840Di/810D/FM-NC — **Description of Functions**
  - Basic Machine ***)**
  - Extended Functions
  - Special Functions
- SINUMERIK 840D/810D/FM-NC — **Configuring Kit** MMC 100/101
  - Configuring Syntax
  - Development Kit
- SINUMERIK 840D/810D/FM-NC — **Screen Kit** MMC 100/101 SW Update and Configuration
- SINUMERIK 840D/810D — Description of Functions **Tool Management**
- SINUMERIK 840D/810D/FM-NC — Description of Functions Operator Interface **OP 030**

Manufacturer / Service Documentation

- SINUMERIK SIMODRIVE — Description of Functions SINUMERIK **Safety Integrated**
- SINUMERIK 840D — Description of Functions **Digitizing**
- SINUMERIK SIMODRIVE — **Installation & Start-up Guide *)**
  - FM-NC
  - 810D
  - 840D/611D
  - MMC
- SINUMERIK SIMODRIVE 840D/840Di/810D FM-NC 611D — **Lists *)**
- SINUMERIK SIMODRIVE 840D 611D — Description of Functions **Linear Motor**
- SINUMERIK SIMODRIVE 840D 611D — Description of Functions
  - **Hydraulics Module**
  - **Analog Module**
- SINUMERIK SIMODRIVE SIROTEC — **EMC** Guidelines

Electronic Documentation

- SINUMERIK SIMODRIVE 840D/810D/FM-NC 611, Motors — **DOC ON CD *)** The SINUMERIK System

Manufacturer / Service Documentation

- SINUMERIK 840D/810D — Descr. of Functions **ISO Dialects** for SINUMERIK
- SINUMERIK — Descr. of Functions **CAM Integration** DNC NT-2000
- SINUMERIK 840Di — **Manual** (HW + Installation and Start-up)

***)** These documents are a minimum requirement for the control

SIEMENS

SINUMERIK 840D/840Di/810D/FM-NC

Advanced

Programming Guide

Valid for

Control	Software Version
SINUMERIK 840D	5
SINUMERIK 840Di	5
SINUMERIK 840DE (export version)	5
SINUMERIK 810D	3
SINUMERIK 810DE (export version)	3
SINUMERIK FM-NC	3

04.2000 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 codes 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 or that page.

Edition	Order No.	Remarks
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

This manual is part of the documentation on CD-ROM (**DOCONCD**)

Edition	Order No.	Comment
04.00	6FC5 298-5CA00-0BG2	C

Trademarks

SIMATIC®, SIMATIC HMI®, SIMATIC NET®, SIROTEC®, SINUMERIK® and SIMODRIVE® are Siemens trademarks. 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 V 8.0 and Designer V 7.0.

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 document correspond to the hardware and software described. Nonetheless, differences might exist and therefore we cannot 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. 6FC5298-5AB10-0BP2
Printed in the Federal Republic of Germany

Siemens Aktiengesellschaft

Contents

Preface

Overview of documentation

The SINUMERIK documentation is organized in 3 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 and SINUMERIK FM-NC.

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.

For more detailed information on SINUMERIK 840D/840Di/810D and SINUMERIK FM-NC publications and other publications covering all SINUMERIK controls (e.g. universal interface, measuring cycles...), please contact your local Siemens office.

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

Validity

This Programming Guide is valid for the following controllers:

SINUMERIK 840D	SW5
SINUMERIK 840Di	SW5
SINUMERIK 840DE (export version)	SW5
SINUMERIK 810D	SW3
SINUMERIK 810DE (export version)	SW3
SINUMERIK FM-NC	SW3

Export version

The following functions are not available in the export version:

Function	FM-NC	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 possible
1) Limited functional scope

Structure of the descriptions

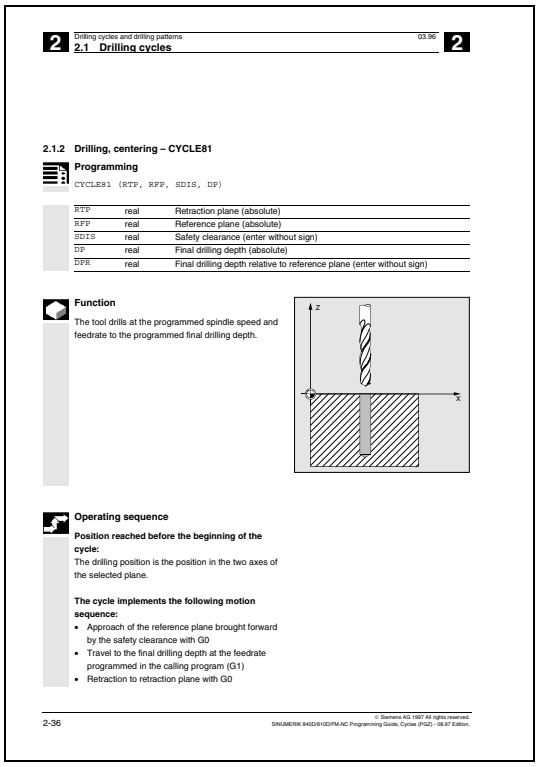
All cycles and programming options have been described – where appropriate and possible – according to the same internal structure. The organization into different information levels allows you to find the information you need quickly.

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.


2 Drilling cycles and drilling patterns 03.96 2
2.1 Drilling cycles

2.1.2 Drilling, centering – CYCLE81

Programming

CYCLE81 (RTP, RFP, SDIS, DP)

RTP	real	Retraction plane (absolute)
RFP	real	Reference plane (absolute)
SDIS	real	Safety clearance (enter without sign)
DP	real	Final drilling depth (absolute)
DPR	real	Final drilling depth relative to reference plane (enter without sign)

Function

The tool drills at the programmed spindle speed and feedrate to the programmed final drilling depth.

Operating sequence

Position reached before the beginning of the cycle:
The drilling position is the position in the two axes of the selected plane.

The cycle implements the following motion sequence:
- Approach of the reference plane brought forward by the safety clearance with G0
- Travel to the final drilling depth at the feedrate programmed in the calling program (G1)
- Retraction to retraction plane with G0

2-36 © Siemens AG 1997 All rights reserved. SINUMERIK 840D/810D/FM-NC Programming Guide, Cycles (PG2) - 08.97 Edition.

2. Detailed explanations

The theory part contains detailed information on the following:

What is the purpose of the command?

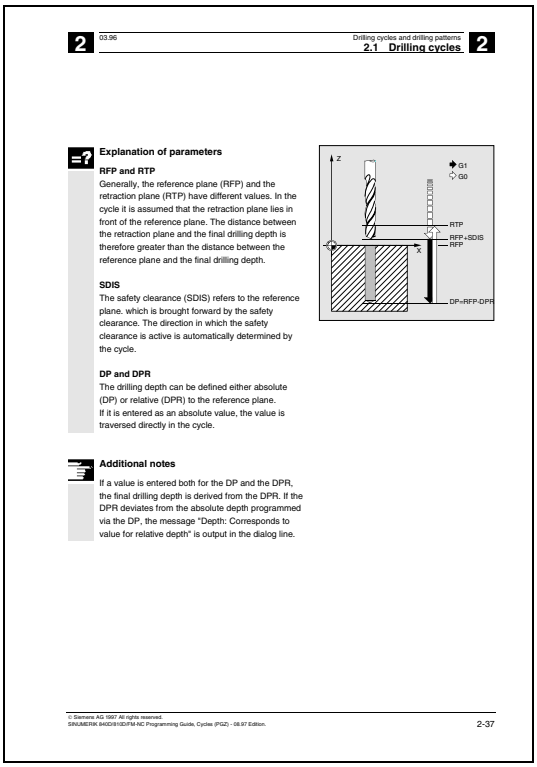
What is the effect of the command?

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.


2 03.96 Drilling cycles and drilling patterns 2.1 Drilling cycles 2

Explanation of parameters

RFP and RTP
Generally, the reference plane (RFP) and the retraction plane (RTP) have different values. In the cycle it is assumed that the retraction plane lies in front of the reference plane. The distance between the retraction plane and the final drilling depth is therefore greater than the distance between the reference plane and the final drilling depth.

SDIS
The safety clearance (SDIS) refers to the reference plane. which is brought forward by the safety clearance. The direction in which the safety clearance is active is automatically determined by the cycle.

DP and DPR
The drilling depth can be defined either absolute (DP) or relative (DPR) to the reference plane.
If it is entered as an absolute value, the value is traversed directly in the cycle.

Additional notes

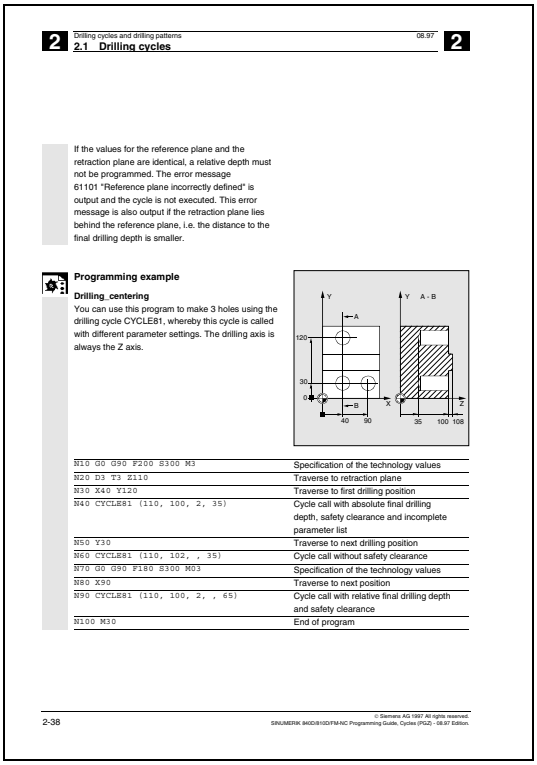
If a value is entered both for the DP and the DPR, the final drilling depth is derived from the DPR. If the DPR deviates from the absolute depth programmed via the DP, the message "Depth: Corresponds to value for relative depth" is output in the dialog line.

© Siemens AG 1997 All rights reserved.
SINUMERIK 840D/810D/FM-NC Programming Guide, Cycles (PGZ) - 08.97 Edition. 2-37

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.


2 Drilling cycles and drilling patterns 08.97 2.1 Drilling cycles 2

If the values for the reference plane and the retraction plane are identical, a relative depth must not be programmed. The error message 61101 "Reference plane incorrectly defined" is output and the cycle is not executed. This error message is also output if the retraction plane lies behind the reference plane, i.e. the distance to the final drilling depth is smaller.

Programming example

Drilling_centering
You can use this program to make 3 holes using the drilling cycle CYCLE81, whereby this cycle is called with different parameter settings. The drilling axis is always the Z axis.

N10 G0 G90 F200 S300 M3	Specification of the technology values
N20 D3 T3 Z110	Traverse to retraction plane
N30 X40 Y120	Traverse to first drilling position
N40 CYCLE81 (110, 100, 2, 35)	Cycle call with absolute final drilling depth, safety clearance and incomplete parameter list
N50 Y30	Traverse to next drilling position
N60 CYCLE81 (110, 102, , 35)	Cycle call without safety clearance
N70 G0 G90 F180 S300 M03	Specification of the technology values
N80 X90	Traverse to next position
N90 CYCLE81 (110, 100, 2, , 65)	Cycle call with relative final drilling depth and safety clearance
N100 M30	End of program

2-38 © Siemens AG 1997 All rights reserved.
SINUMERIK 840D/810D/FM-NC Programming Guide, Cycles (PGZ) - 08.97 Edition.

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 SIEMENS 840D/840Di/810D or FM-NC 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** may 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
Before the control is started up, it should be ensured that the Operator' Guides have been read and understood by the people responsible. 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. All appropriate safety specifications must be observed.

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 can result in:

- personal injury or death,
- damage to the control, machine and other possessions of the plant and user.

Notes

Flexible NC Programming

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.1 Variables and arithmetic parameters

Function

Variables can be used instead of fixed values to increase the flexibility of a program. 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.

A skilled programmer can use variable calculation and program jumps to create a highly flexible program archive which will considerably reduce the programming work required.

Types of variables
The control distinguishes between three types of variable:

User-defined variables	Variables whose name and type are defined by the user, e.g. arithmetic parameters.
Arithmetic parameters	Special predefined arithmetic variables for which address R, followed by the number, is provided. The predefined arithmetic variables are type REAL.
System variable	Variables provided by the control which can be processed (read/written) in the program. 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)

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Types of variables

Type	Meaning	Value range
INT	Integers with leading sign	$\pm(2^{31} - 1)$
REAL	Real numbers (fractions with decimal point, LONG REAL to IEEE)	$\pm(10^{-300} \ldots 10^{+300})$
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 [...], maximum 200 characters	Sequence of values mit 0 … 255
AXIS	Axis names (axis addresses) only	All axis identifiers and spindles of a channel
FRAME	Geometrical parameters for translation, rotation, scaling, mirroring, see Chapter 4.	

Arithmetic variables

100 arithmetic variables of type REAL are available without any further definition under address R as standard.

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 are available and can be processed in all current programs.

System variables return machine and control states. Some of the system variables cannot be assigned values.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

The name of a system variable is always identified by the "$" character followed by the specific names.

Overview of system variable types

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

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

Example: `$AA_IM`
Meaning: Current axis-specific value in the machine coordinate system.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.2 Variable definition

User-defined variables

In addition to the predefined variables, programmers can also define their own variables and assign values to them.

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

Global variables (GUD) apply in all programs.

SW 4.4 and higher:

The local user variables (LUD) defined in the main program are redefined to program global user variables (PUD) via machine data.

Machine manufacturer

See machine manufacturer's specifications

If they are defined in the main program, they are also valid in all levels of the subprograms called. They are created with part program start and deleted with part 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          ;subprogram call SUB2
DEF INT VAR2       ;LUD DEFINITION
...
IF (VAR1==1)       ;read PUD
  VAR1=VAR1+1      ;read and write
                   ;PUD
  VAR2=1           ;write LUD
ENDIF
SUB3               ;subprogram call
...
M17
PROC SUB3          ;subprogram SUB3
...
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

```
IF (VAR1==1)       ;read PUD
   VAR1=VAR1+1     ;read and write
                   ;PUD
   VAR2=1          ;error: LUD from SUB2
                   ;not known
ENDIF
...
M17
```

If machine data `$MN_LUD_EXTENDED_SCOPE` is set, it is no longer possible to define a variable with the same name in main programs 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 "$" character cannot be used for user-defined variables, as it is reserved 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 index]=("A","B",...)`

`DEF STRING[string length] name`

`DEF AXIS name`
or `DEF AXIS name[array index]`

`DEF FRAME name`

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

If a value is not assigned to a variable when it is defined, the system initializes it with zero.

Variables must be defined at the beginning of the program before use. 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. fraction with decimal point
BOOL	Variable type Bool, i.e. 1 or 0 (TRUE or FALSE)
CHAR	Variable type Char, i.e. ASCII character specified by the code 0 to 255
STRING	Variable type String, i.e. character string
AXIS	Variable type Axis, i.e. axis addresses and spindles
FRAME	Variable type Frame, i.e. geometrical parameters
name	Variable name

Programming examples

Variable type INT

`DEF INT NUMBER`	A variable of type INTEGER is created with the name NUMBER. The system initializes the variable with zero.
`DEF INT NUMBER=7`	A variable of type INTEGER is created with the name NUMBER. The variable is initialized with the value 7.

Variable type REAL

`DEF REAL DEPTH`	A variable of type REAL is created with the name DEPTH. The system initializes the variable with zero (0.0).
`DEF REAL DEPTH=6.25`	A variable of type REAL is created with the name DEPTH. The initial value is 6.25.
`DEF REAL DEPTH=3.1,LENGTH=2,QUANTITY`	It is also possible to define several variables in a single line.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Variable type BOOL

`DEF BOOL IF_TOOMUCH`	A variable of type BOOL is created with the name IF_TOOMUCH. The system initializes the variable with zero (FALSE).
`DEF BOOL IF_TOOMUCH=1` or `DEF BOOL IF_TOOMUCH=TRUE` or `DEF BOOL IF_TOOMUCH=FALSE`	A variable of type BOOL is created with the name IF_TOOMUCH.

Variable type CHAR

`DEF CHAR GUSTAV_1=65`	You can assign a code for the ASCII character to the variable of type CHAR or
`DEF CHAR GUSTAV_1="A"`	assign the ASCII character directly (65 is the code for the letter "A").

Variable type STRING

`DEF STRING[6] SAMPLE_1="START"`	Variables of type STRING can store a string of characters. The maximum number of characters is enclosed in square brackets after the variable type.

Variable type AXIS

`DEF AXIS AXISNAME=(X1)`	The variables of type AXIS have the name AXISNAME and contain the axis identifier of a channel – here X1 (axis names with extended addresses are enclosed in parentheses).

Variable type FRAME

`DEF FRAME INCLINE_1`	The variables of type FRAME are called INCLINE_1.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Additional notes

A variable of type AXIS stores axis names and spindle identifiers of a channel.
Note:
Axis names with extended addresses must be enclosed in parentheses.

Programming example with local variables

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

Programming example

Scan for existing geometry axes

```
DEF AXIS ABSCISSA;                   ;1st 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)
…
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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]`	The data type STRING can only be defined with one-dimensional arrays
`NAME`	Variable name

The same memory size applies for type BOOL as for type CHAR.
SW 3 and higher:
The maximum size of an array is set via machine data.

Machine manufacturer

See machine manufacturer's specifications

Type	Memory size for each array element
BOOL	1 byte
CHAR	1 byte
INT	4 bytes
REAL	8 bytes
STRING	String length + 1
FRAME	~ 400 bytes depending on 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.
Standard: 812 bytes
If no large arrays are defined, please select: 256 bytes.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.
Standard: 256 bytes
Memory requirements per element: see above.
Example:
Global user data should contain PLC machine data for switching the control on/off (definition of BOOL arrays).

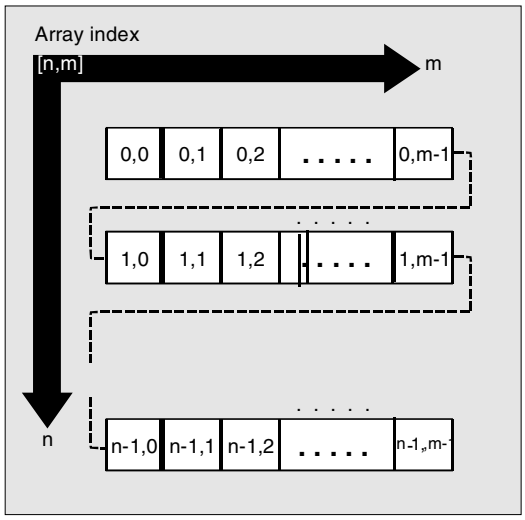
Additional notes

Arrays with a maximum of 2 dimensions can be defined.

Arrays with STRING variables may only be one-dimensional. The string length is specified after the data type String.

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

The first array element begins with the index [0,0]. With an array size of [3,4], for example, the maximum array index is [2,3].



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

In the above example, the initialization values match the index of the array element in order to illustrate the sequence of the individual array elements.

Initialization of arrays

Initialization values can be assigned to array elements during program execution or when arrays are defined.

The right-hand array index is incremented first on two-dimensional arrays.

Initialization with value lists, SET

1. Options during 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,
...)
```

- The number of array elements assigned corresponds to the number of initialization values programmed.
- Array elements without values (gaps in value list) are automatically assigned the value "0".
- There may be no gaps in the value list for variables of the AXIS type.
- If more values are programmed than remaining array elements exist, the system triggers an alarm.

Example:

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

You can specify SET optionally when defining arrays.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2. Options during program execution

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

- Field elements are initialized as described above for array definition.
- Expressions may also be used here as initialization values.
- Initialization starts at the programmed array indices. 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 is not processed for axis variables.
Example:
Initialization on one line

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

Corresponds 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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Initialization with identical values, REP

1. Options during 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. Options during program execution

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

- Expressions may also be used here as initialization values.
- All array elements are initialized with the same value.
- Initialization starts at the programmed array indices. Values can also be assigned selectively to subarrays.

Variables of the FRAME type are permitted and can be initialized very simply using this method.

Example:
Initialization of all elements with one value

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming example

Initialization of complete variable arrays.
The drawing shows the current assignment.

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

Array index 2

1

[1,2]	N10: Initialization 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.		

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.4 Indirect programming

Indirect programming enables programs to be used universally. The extended address (index) is substituted by a variable of suitable type.

All addresses can be configured, except for:

- N – block number
- G – G command
- L – subroutine

Indirect programming is not possible for any settable addresses.
Example: X[1] is not permitted instead of X1.

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 variable SPINU (1 in this example).

Feed

`FA[U]=300`	Direct programming
`DEF AXIS AXVAR2=U` `FA[AXVAR2]=300`	Indirect programming: Feed for positioning axis whose address name is stored in the variable of type AXIS with 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 axis whose name is stored in the variable AXVAR3.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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:
	Field sizes must always be specified as fixed values in array dimensions.

Axis instruction with axis variables

`X1=100 X2=200`	Direct programming

`DEF AXIS AXVAR1 AXVAR2`
`AXVAR1=(X1) AXVAR2=(X2)`
`AX[AXVAR1]=100 AX[AXVAR2]=200`
	Indirect programming:
	Define variables
	Assign axis names Traverse axes stored in the variables to 100 and 200.

Interpolation parameters with axis variables

`G2 X100 I20`	Direct programming
`DEF AXIS AXVAR1=X` `G2 X100 IP[AXVAR1]=20`	Indirect programming: Define and assign axis name Indirect programming of center point

Indirect subprogram call

`CALL "L" << R10`	Call the program with the number contained in R10

Additional notes

R parameters can also be interpreted as single-dimensional arrays with abbreviated notation (R10 corresponds to R[10]).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.5 Assignments

Values of matching types can be assigned to variables/arithmetic parameters in the program.

The assignment is always made in a separate block; up to two assignments are possible per block. Assignments to axis addresses (traversing instructions) always require a separate block to variable assignments.

Programming example

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

Assignment to string variables

A distinction is made between upper and lower case characters within a CHAR or STRING.
If ' or " are to be included in the character string, these should be enclosed in '...'.

Example:

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

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

Non-displayable characters can be stored in the string as binary or hexadecimal constants.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.6 Arithmetic operations/functions

Arithmetic functions are used predominantly for R parameters and variables (or constants and functions) of the type REAL. The types INT and CHAR are also permitted.

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

Operators/arithmetic functions

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division **Caution**: (Type `INT`)/(Type `INT`)=(Type `REAL`); Example: 3/4 = 0.75
`DIV`	Division, for variable type INT and REAL **Caution**: (Type `INT`)`DIV`(Type `INT`)=(Type `INT`); Example: 3 DIV 4 = 0
`MOD`	Modulo division (`INT` or `REAL`) produces the remainder of an 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 number
`POT()`	Square of Z (square)
`TRUNC()`	Truncate to integer
`ROUND()`	Round to integer
`LN()`	Natural logarithm
`EXP()`	Exponential function
`CTRANS()`	Translation
`CROT()`	Rotation
`CSCALE()`	Scale change
`CMIRROR()`	Mirroring

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

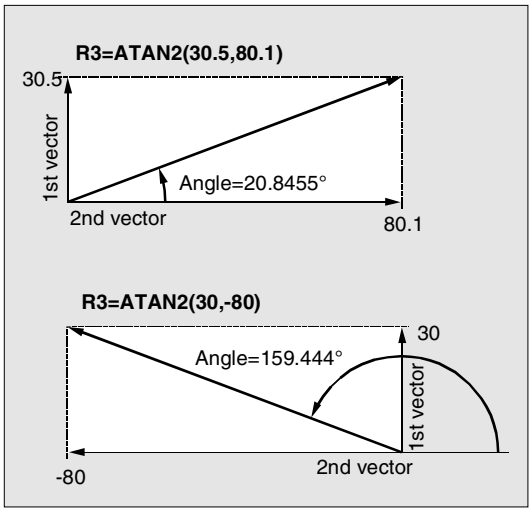
Programming examples

`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 and division have priority over addition and subtraction
`R14=(R1+R2)*R3`	Parentheses are calculated first
`R15=SQRT(POT(R1)+POT(R2))`	Inner parentheses are solved first R15 = square root of ($R1^2$+$R2^2$)

`RESFRAME= FRAME1:FRAME2`
 	
`FRAME3=CTRANS(…):CROT(…)`
	The chain operator combines frames in a resulting frame or assigns values to the frame components

Arithmetic function ATAN2(,)

The function calculates the angle of the resulting vector from two vectors at right angles to each other. 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.



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.7 Comparison and logic operators

Comparison operators

The comparison operators can be used for variables of types CHAR, INT, REAL and BOOL. The code value is compared with the CHAR type.

The following are possible with types STRING, AXIS and FRAME: == and <>.

The result of a comparison operation is always type BOOL.

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

Meaning of the comparison operators

==	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<<	Chaining of strings

Programming example

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

or

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

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Logic operators

Logic operators are used to logically combine truth values.
AND, OR, NOT and XOR can generally only be used on variables of type BOOL, however, they can also be used on the data types CHAR, INT and REAL by means of implicit type conversion.

Spaces must be inserted between Boolean operands and operators.

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

Meaning of the logic operators

`AND`	AND
`OR`	OR
`NOT`	NOT
`XOR`	Exclusive OR

Parentheses can be used in arithmetic expressions to define the order of execution for all operators and thus to override the normal priority rules.

Programming example

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

IF NOT R10 GOTOB START
```

NOT refers only to an operand.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Bit-for-bit logic operators

Bit-for-bit logic operations can also be performed on variables of the type CHAR and INT. Type conversion takes place automatically.

Meaning of the bit-for-bit logic operators

B_AND	Bit AND
B_OR	Bit OR
B_NOT	Bit NOT
B_XOR	Bit exclusive OR

The operator B_NOT refers only to an operand; this follows the operator.

Programming example

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.8 Priority of operators

Priority of 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.

Parentheses can be used in arithmetic expressions to define the order of execution for all operators and thus to override the normal priority rules.

Sequence of operators (highest to lowest)

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

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

The chain operator ":" for frames may not appear with other operators in an expression. A priority level is thus not required for this operator.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.9 Possible type conversions

Type conversion on assignment

The constant numeric value, variable or expression assigned to a variable must be compatible with the type of this variable. 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

* On type conversion from REAL to INT, a fraction >= 0.5 is rounded up, otherwise the fraction is rounded down (same effect as ROUND function)

[1)] Values <> 0 are TRUE, values == 0 are FALSE

[2)] If the value is in the permitted value range

[3)] If only 1 character

[4)] String length 0 = FALSE, otherwise TRUE

If a value is greater than the target range on conversion, an error message is generated.

Additional notes

If mixed types occur in an expression, a type conversion is performed automatically.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.10 String operations

Overview

In addition to the classic operations "assignment" and "comparison" described in this section, the following further string manipulations are possible:

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

Chaining 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 string:

`INT_ERG = STRLEN (STRING)`	Result type: INT

Search for character/string in 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 = STRINGFIELD [IDX_FIELD, IDX_CHAR]`	Result type: CHAR

[1] `"bel._Typ"` stands for variable types INT, REAL, CHAR, STRING and BOOL.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Special meaning of 0 character
The 0 character is interpreted internally as a string end identifier.
If a character is replaced by the 0 character, then the string will be shortened.

Example:

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

1.10.1 Type conversion

Type conversion allows variables of different types to be used as an integral part of a message (MSG).

Conversion to STRING
Results if the operator << is used implicitly for data types INT, REAL, CHAR and BOOL (see "Chaining strings").
An INT value is converted to the normal readable form. Up to 10 places after the decimal point are specified for REAL values.

Variables of the AXIS type can be converted to STRING by means of the AXSTRING function.
FRAME variables cannot be converted.

Example:

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Conversion from STRING

The NUMBER function converts from STRING to REAL.
If ISNUMBER returns the value FALSE, an alarm is output when NUMBER is CALLED with the same parameter.
A string can be converted to data type AXIS with the AXNAME function. 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 represents a semantically valid REAL number. It is thus possible to check whether the string can be converted to a valid number.
NUMBER (STRING) returns the value represented by the string as a REAL value.
AXSTRING (AXIS) supplies the specified axis identifier as a string.
AXNAME (STRING) converts the specified string into 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.10.2 Chaining of strings

This functionality makes it possible to compile a string with individual 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 specified strings (the implicitly converted other type in some cases) are chained.

This operator is also available as a unary variant. It is thus possible to perform an explicit type conversion to STRING (not for FRAME and AXIS).

Syntax:

`<< bel._Typ`	Result type: STRING

Semantics:
The specified type is converted implicitly to type STRING.

For example, this function can be used to compile a message or a command from text lists and to insert parameters (such as module name):
`MSG(STRG_TAB[LOAD_IDX]<<MODULE_NAME)`

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

The intermediate results of a string chaining 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 can be used to convert all letters in a string to a uniform case.

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 case or lower case letters.

Example:
Since user inputs can also be activated on the MMC, it is possible to achieve uniform representation of text (i.e. upper case or lower case):

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.10.4 Length of string

This functionality allows the length of a string to be specified.

Syntax:

`INT_ERG = STRLEN (STRING)`	Result type: INT

Semantics:
A number of characters is returned that – counting from the beginning of the string – are not 0 characters.

Example:
This function can be used to determine the end of the string, for example, in connection with the single character access described below:

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

1.10.5 Search for character/string in string

This functionality can be used to search for single characters or a whole string in another 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

INDEX searches from beginning of first parameter for character specified as second parameter.
RINDEX searches from end of first parameter for character specified as second parameter.
MINDEX corresponds to INDEX function except that a list of characters is transferred (as string). The index of the first character found in this list is returned.
MATCH looks for a string within a string.

Strings can therefore be broken down according to certain criteria, i.e. at positions with blanks or path separator (oblique) ("/").

Programming example

Example of how to break down an input into path and module name:

`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`	3 is returned as the value in LISTIDX; because "N" is the first character in parameter INPUT (starting at beginning of selection list).
`PATHIDX = INDEX (INPUT, "/") +1`	; PATHIDX is therefore 1
`PROGIDX = RINDEX (INPUT, "/") +1`	; PROGIDX = is therefore 12
	;Using the SUBSTR function introduced in the next section, the variable INPUT can be broken down into "Path" and "Module":
`VARIABLE = SUBSTR (INPUT, PATHIDX, PROGIDX-PATHIDX-1)`	Then supplies "_N_MPF_DIR"
`VARIABLE = SUBSTR (INPUT, PROGIDX)`	Then supplies "_N_EXECUTE_MPF"

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.10.6 Selection of a substring

This functionality makes it possible to separate 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`	Result type: INT
`STRING_ERG = SUBSTR (STRING,INT, INT)`	Result type: INT

Semantics:
In the first case, the substring from the position defined by the first parameter up to the end of the string is returned.
In the second case, the result string is limited to the maximum length as defined by the third parameter.
If the start position is after the string end, then the empty string ("") is returned.
If the start position or the length is negative, then an alarm is output.

Example:

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.10.7 Selecting a single character

Individual characters of a string can be selected by means of this function. This applies both to read access and write access operations.

Syntax:

`CHAR_ERG = STRINGVAR [IDX]`	Result type: CHAR
`CHAR_ERG = STRINGARRAY [IDX_ARRAY, IDX_CHAR]`	Result type: CHAR

Semantics:

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

Example of messages:
Insertion of an axis identifier in a pre-assembled string.

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

It is only possible to access single characters in variables that have been defined by the user (LUD, GUD and PUD data).
In addition, this mode of accessing data with subroutine calls can only be used for parameters of the "call-by-value" type.

Examples:

Accessing single character in a system data, machine data, ...:

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

Accessing single character with 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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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
`GOTOF`	Jump instruction with jump destination forwards (towards the end of program)
`GOTOB`	Jump instruction with jump destination backwards (towards the start of 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

1.12 Control structures

Explanation

IF-ELSE-IFENDIF	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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

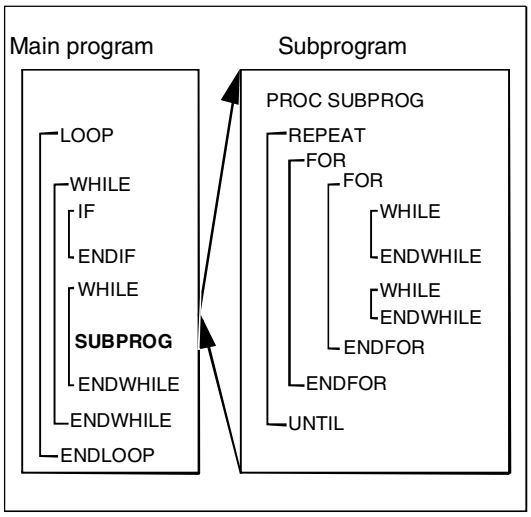
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Supplementary conditions

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Instructions for program coordination

• **Specification with absolute path**	

`INIT(n,"/_HUGO_DIR/_N_name_MPF")` or
`INIT (n,"/_N_MPF_DIR/_N_name_MPF")`
 	
Example:
`INIT(2,"/_N_WCS_DIR/_DRESSING_MPF")`
`G01 F0.1`
`START`
 	
`INIT (2,"/_N_WCS_DIR/_N_UNDER_1_SPF")`
	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
	SW 3 and lower:
	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.

• **Specification with relative path**	

Example:
`INIT(2,"DRESS")`
 	
`INIT(3,"UNDER_1_SPF")`
	The same rules apply to relative path definition as for program calls.
	
	With subprogram calls "_SPF" must be added to the program name.

`START (n,n)`	Starts the selected programs in the other channels. n,n: Number of the channel: 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

`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 N" 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.

Channel names

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

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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
```

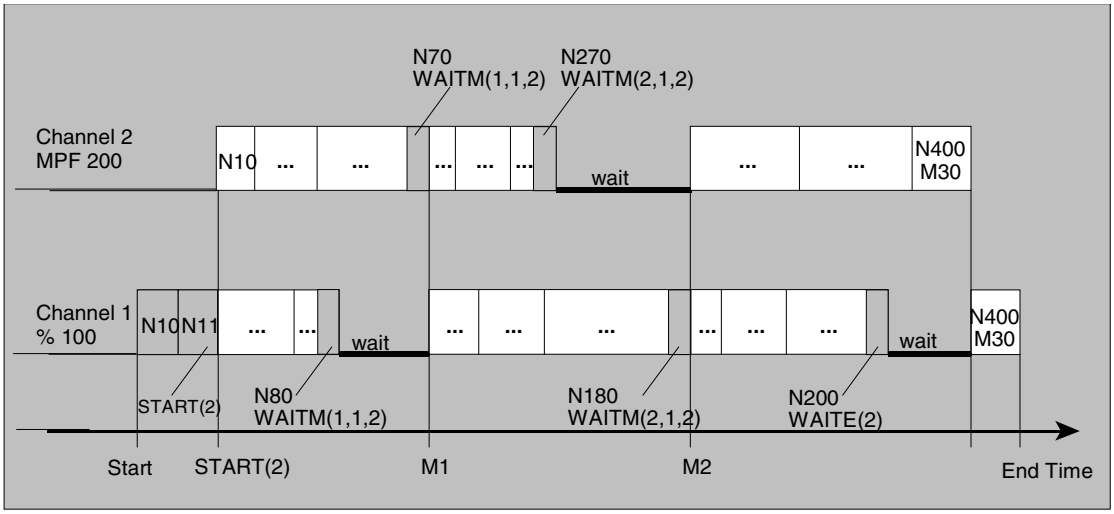
`N10 INIT(2,"MPF200")`	
`N11 START (2)` .	Program execution in channel 2
`N80 WAITM(1,1,2)` .	Wait for WAIT mark 1 in channel 1 and in channel 2 and execution continued in channel 1
`N180 WAITM(2,1,2)` .	Wait for WAIT mark 2 in channel 1 and in channel 2 and execution continued in channel 1
`N200 WAITE(2)`	Wait for end of program in channel 2
`N201 M30` ...	End of program channel 1, end all

Channel 2:

```
%_N_MPF200_MPF
```

`;$PATH=/_N_MPF_DIR`	
`N70 WAITM(1,1,2)` .	Program execution in channel 2 Wait for WAIT mark 1 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di



Example of program from workpiece

```
N10 INIT(2,"/_N_WCS_DIR/_N_SHAFT1_WPD/_N_STOKREM1_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.
```

Additional notes

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.

SW 3 and lower:
WAITE must not be scanned immediately after the START command or else a program end will be detected before the program is started.
Remedy: Programming a dwell time.
Example:

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

1.14 Interrupt routine

Programming

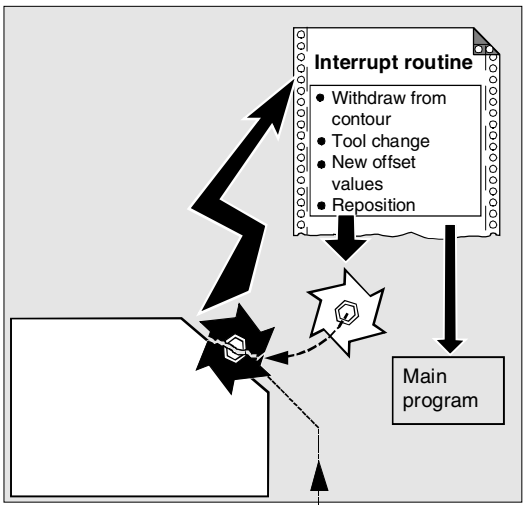
```
SETINT(3) PRIO=1 NAME
SETINT(3) PRIO=1 LIFTFAST
SETINT(3) PRIO=1 NAME  LIFTFAST
G… X… Y… ALF=…
DISABLE(3)
ENABLE(3)
CLRINT(3)
```

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 lift 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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
```

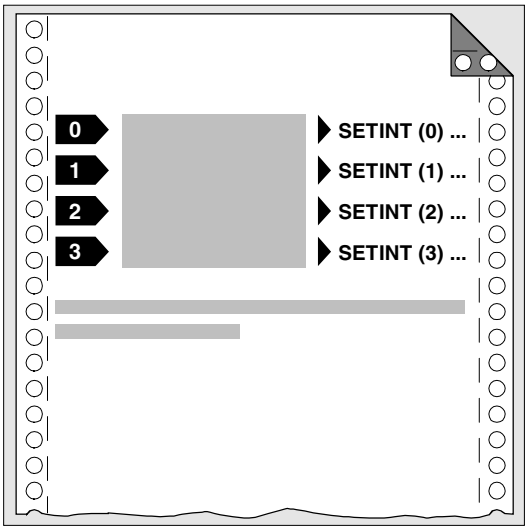
At the end of the interrupt routine the modal G functions are set to the value they had at the start of the interrupt routine by means of the SAVE attribute. The programmable zero offset and the basic offset are reestablished in addition to the settable zero offset (modal G function group 8). If the 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.

1.14 Interrupt routine

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Assign and start interrupt routine, SETINT
The control has eight signals
(inputs 1...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: Priority levels PRIO 1 to 128 are available, 1 has top 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 cancelled.

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).

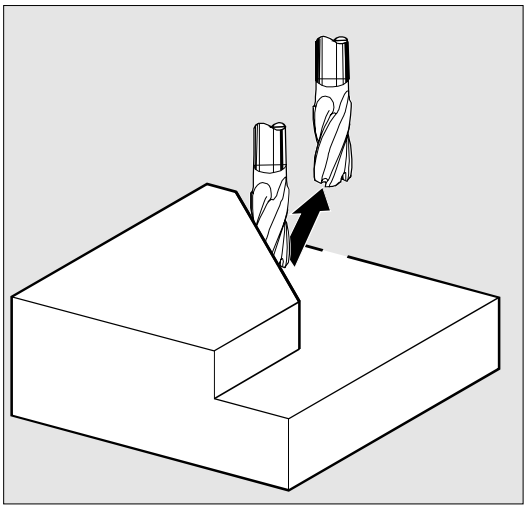
Example:

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

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

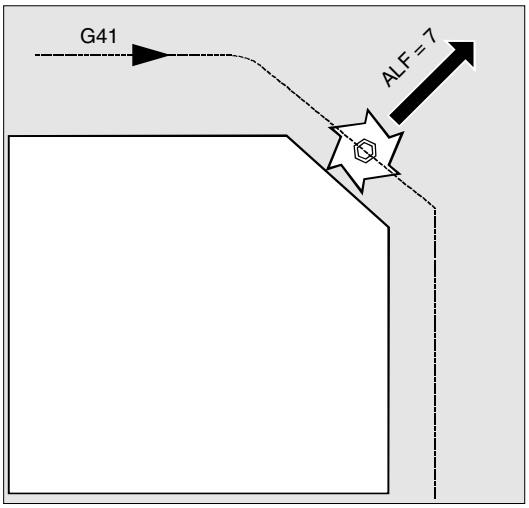
```
N30 SETINT(2) PRIO=1 LIFT_Z LIFTFAST
```

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, ASUP, was specified).
- The asynchronized subprogram "LIFT-Z" is executed with N30.

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.

Programmable traversing direction, ALF=...
You enter the direction in which the tool is to travel on liftfast in the NC program.



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

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

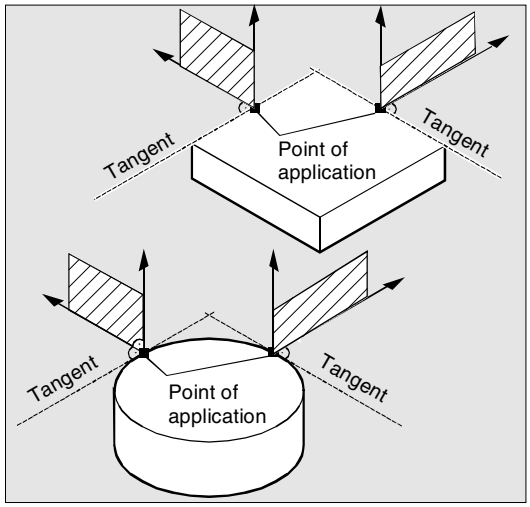
The tool moves – with G41 activated (direction of machining to the left of the contour) – away from the contour perpendicularly as seen from above.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Reference plane for describing the traversing directions

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

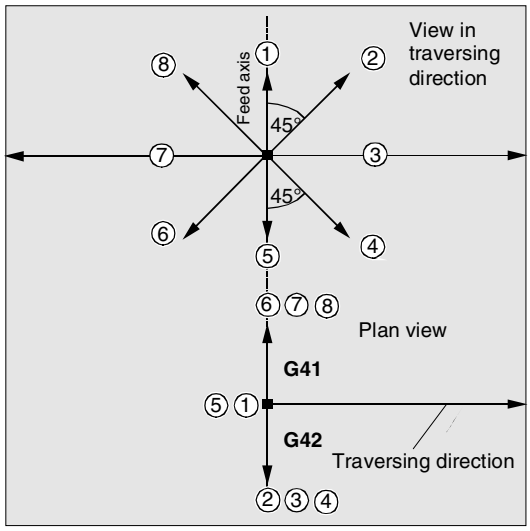
The reference plane is derived from the longitudinal tool axis (infeed direction) and a vector positioned perpendicular to this axis and perpendicular to the tangent at the point of application of the tool.



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.



Please 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.

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 presently used for programming the behavior for 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. Thus it is possible to program an axis-parallel fast lift.

In the retraction movement plane, **ALF** is used to program the direction in discrete steps of 45 degrees as was the case formerly. With **LFTXT** retraction in tool direction is defined for ALF=1.
With **LFWP** the direction in the working plane is according to the following:

- **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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

-> programmed in a single block.

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

1.15 Axis transfer, spindle transfer

Explanation of the commands

`RELEASE(axis name, axis name, …)`	Enable 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)`	Enable 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.

Sequence

Preconditions for axis transfer

- The axis must be defined in all channels via the machine data.
- The channel to which the axis is assigned after POWER ON is defined 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, coupled motion) must be enabled.
3. A concurrent positioning axis must not be transferred.
4. All the following axes of a gantry master axis are transferred with the master.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Axis transfer:Get 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)

N06 G01 F5000
N07 X20

N08 X30
N09 …
```

If synchronization not necessary, this is not an executable block.
Not an executable block.
Not an executable block because X position as for N04.
First executable block after N05.

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.

1.15 Axis transfer, spindle transfer

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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
An axis is taken directly from another channel with GETD (GET Directly). This means that no matching RELEASE has to be programmed in another channel for this GETD. It also means that other channel communication has to be established (e.g. wait markers).

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Programming example

Of the 6 axes, the following are used for machining in channel 1: 1st, 2nd, 3rd and 4th.
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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

1.16 NEWCONF: Setting machine data active (as from SW 4.3)

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 softkey "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 operation: Machining drilling position with different technologies

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

1.17 WRITE: Write file (as from SW 4.3)

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 being used 12 No free resources 13 No access rights 20 Other error
`filename`	Name of file in which the string is to be written. 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 no identifier (_MPF, _SPF or _CYC) is specified, _MPF is automatically added. 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.

1.17 WRITE: Write file (as from SW 4.3)

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 created files can be

- read, edited and deleted by all users,
- written in the part program that is currently 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 FROM 7.2.97")`	; Write text from LOG FROM 7.2.97 in the file TEST1
`N30 IF ERROR`	;
`N40 MSG ("Error with WRITE command:" <<ERROR)`	;
`N50 M0`	;
`N60 ENDIF`	;
`...`	
`WRITE(ERROR, "/_N_WCS_DIR/_N_PROT_WPD/_N_PROT_MPF", "LOG FROM 7.2.97")`	; Absolute path

Additional notes

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

- 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" softkey.

Machine manufacturer

Blocks from the part program can be stored in a file by means of the WRITE command. The file size for log files (KB) is specified in the machine data.

1.18 DELETE: Delete file (as from SW 4.3)

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 being used
	12	No free resources
	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. If no identifier (_MPF, _SPF or _CYC) is specified, _MPF is automatically added. 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/

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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                          ;
N20 DELETE(ERROR,"TEST1")                  ; Delete file TEST1
N30 IF ERROR                               ;
N40 MSG ("Error with DELETE command:"      ;
                              <<ERROR)
N50 M0                                     ;
N60 ENDIF                                  ;
...
```

1.19 READ: Read lines in file (as from SW 5.2)

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 not available (parameter "line" or "number" larger than number of lines in file) 22 Array length of "result" variable too small 23 Line range too large (parameter "number" has been selected so large, that reading goes beyond the end of the file)
`file`	Name/path of the file to be read (call-by-value parameter of type STRING with a max. length of 160 bytes). The file must be stored in the user memory of the NCK (passive file system). The file name can be preceded by the domain identifier _N_. If the domain identifier is missing, it is added correspondingly. The file identifier ("_" plus 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 before the end of the file which is specified by the parameter "number" is read. 1 to n Number of the first line to be read.
`number`	Number of lines to be read (call-by-value parameter of type INT).
`result`	Array of type STRING, where the read text is stored (call-by-reference parameter with a length of 255).

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.

1.19 READ: Read lines in file (as from SW 5.2)

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Additional notes

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

1.20 ISFILE: File available in user memory NCK (as from SW 5.2)

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
...
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

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 type of file 10 Checksum 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: in MY_CHECKSUM
`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.

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Additional notes

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"
```

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Notes

Subprograms, Macros

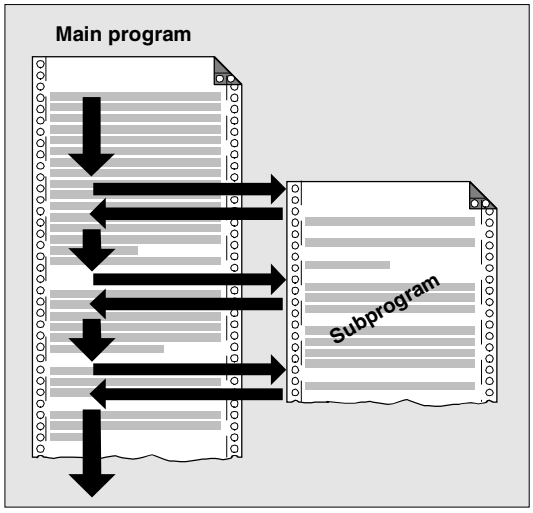
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.1 Using subprograms

What is a subprogram?

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

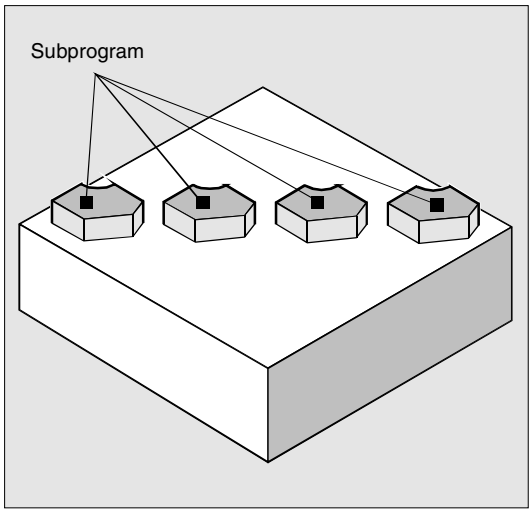
In principle, there is no difference between a main program and a subprogram. The subprogram contains either machining cycles or machining sections that must run more than once.



Use of subprograms

Machining sequences that recur are only programmed once in a subprogram. For example, certain contour shapes that occur more than once or machining cycles.

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



Structure of the subprogram

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

In a subprogram it is also possible to program a program header with parameter definitions.

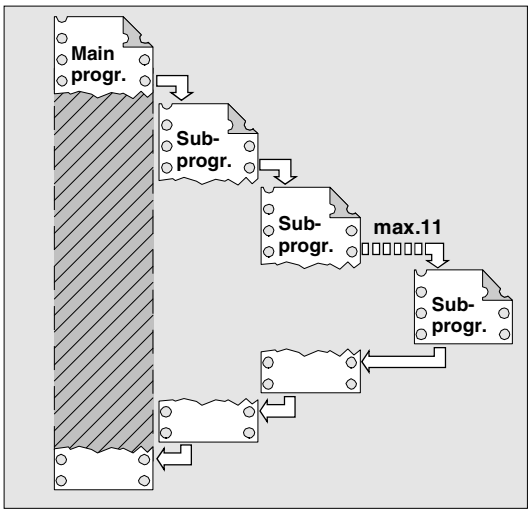
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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:
A main program can contain 11 nested subprogram calls.



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).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.2 Subprogram with SAVE mechanism

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

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.

Example:
Subprogram definition

```
PROC CONTOUR SAVE
N10 G91 …
N100 M17
```

Main program

```
%123
N10 G0  X… Y… G90
N20…
N50 CONTOUR
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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

Program 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.

RET must be programmed in a separate NC block.

Example:
```
PROC CONTOUR
N10…
…
N100 M17
```

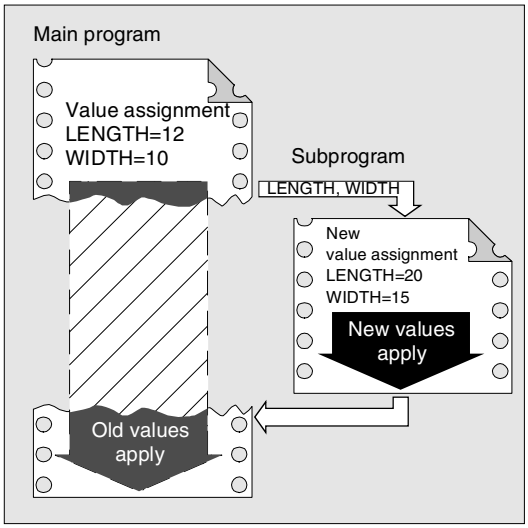
Parameter transfer between main program and subprogram
If you are working with parameters in the main program, you can use the values calculated or assigned in the subprogram as well.
For this purpose the values of the **current parameters** of the main program are passed to the **formal parameters**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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Example:

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

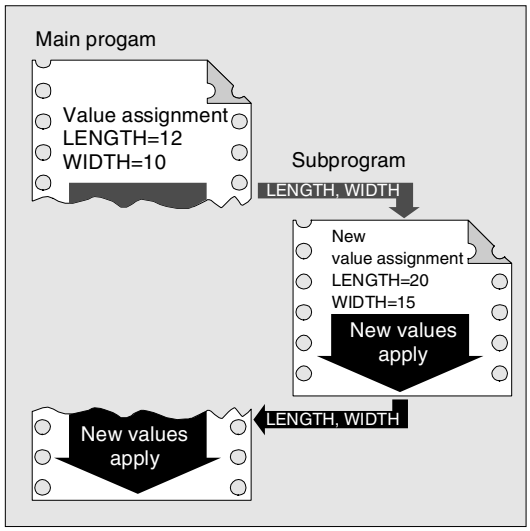
The values assigned in N20 in the main program are passed in N30 when the subprogram is called. Parameters are passed in the sequence stated. The parameter names do not have to be identical in the main programs and subprogram.



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.).



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

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])
```

Additional notes

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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)`	Parameters passed
`DEF INT COUNT`	
`STEP: G1 X=ARRAY[COUNT,0]-> -> Y=ARRAY[COUNT,1] F100`	Machining sequence
`Z=IC(-5)`	
`Z=IC(5)`	
`COUNT=COUNT+1`	
`IF COUNT<NUMBER GOTOB STEP`	
`RET`	End of subprogram

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.4 Calling subprograms

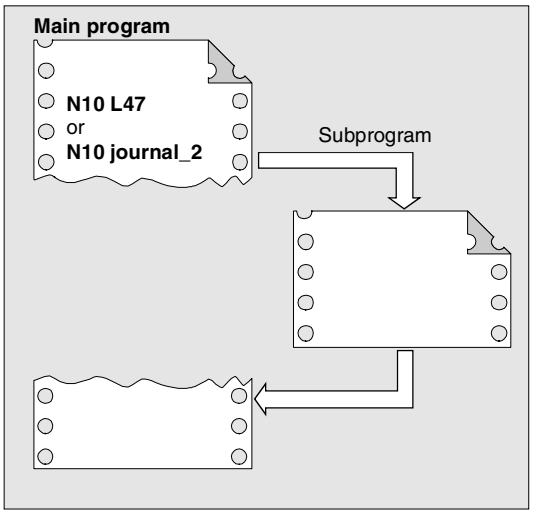
Subprogram call without parameter transfer

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

Example:

`N10 L47` or

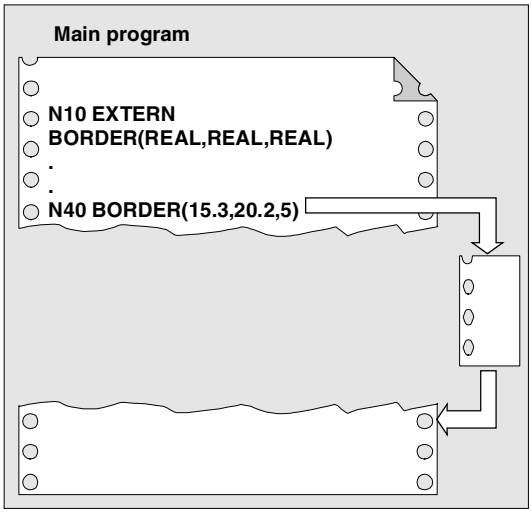
`N10 SPIGOT_2`



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(TYP1, TYP2, TYP3, …)` oder

`EXTERN NAME(VAR TYP1, VAR TYP2, …)`

Example:

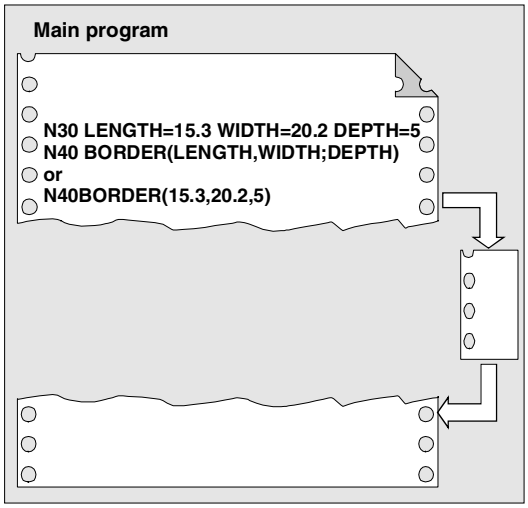
```
N10 EXTERN BORDER(REAL, REAL, REAL)
…
N40 BORDER(15.3,20.2,5)
```

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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)
```

Subprogram definition must match 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)
```

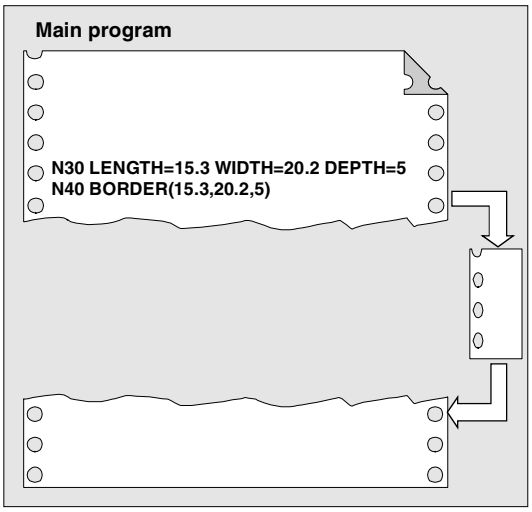
Call in the main program:

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

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, part program processing is aborted with alarm output.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Example:

Subprogram

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

IF $P_SUBPAR[1]==TRUE
   ;Parameter VAR1 was not
   ;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
   ;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
```

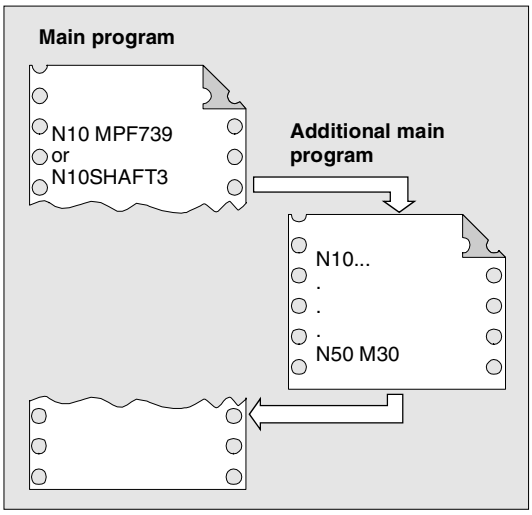
Calling the main program as a subprogram

A main program can also be called as 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).

Program the call by specifying the program name.

Example:

`N10 MPF739` or
`N10 SHAFT3`



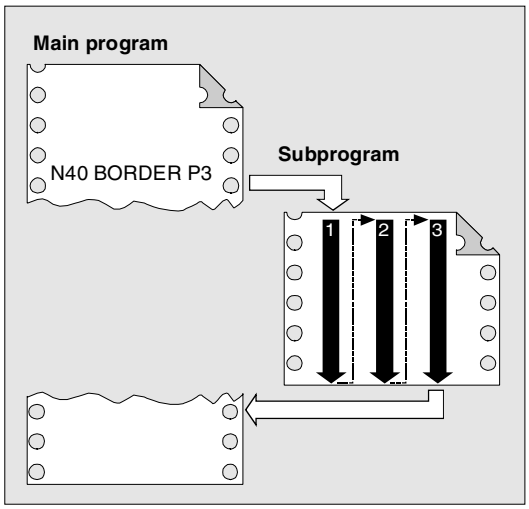
A subprogram can also be started as a main program.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.5 Subprogram with program repetition

Program repetition, P

If you want to execute a subprogram several times in succession, you can program the required number of program repetitions in the block in the subprogram call under address P.



Example:

```
N40 BORDER P3
```

The subprogram Border must be executed three 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 only transferred during the program call or the first pass. The parameters remain unchanged for the repetitions.

If you want to change the parameters in the program repetitions you must define declarations in the subprograms.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.6 Modal subprogram, MCALL

Modal subprogram call, MCALL

With this function the subprogram is automatically called and executed after every block with path motion.

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.

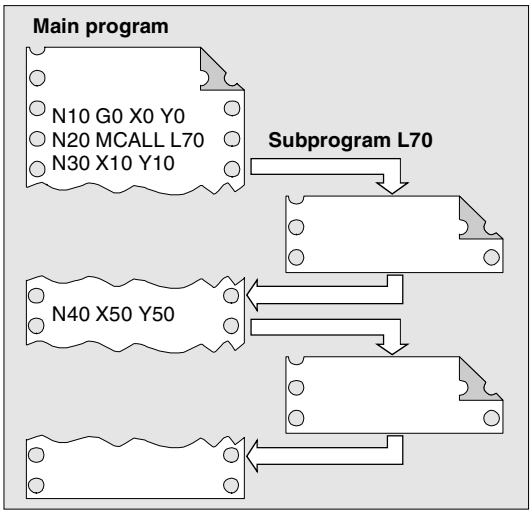
Examples:

```
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 stored 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.

Deactivating the modal subprogram call

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.7 Calling the subprogram indirectly

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.8 Calling subprogram with path specification and parameters, PCALL

With PCALL you can call subprograms 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

Example:

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2.9 Suppressing 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 asynchronized 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

Programming

`PROC ... SBLOF`	; The command can be programmed in a PROC block or in 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

With 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.

SBLOF is also valid in the called subprograms.

Example:

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

Single block suppression in the program

SBLOF can be alone in a block. From this block onwards, the single block mode is deactivated until

- the next SBLON or
- until the end of the active subprogram level.

Example:

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

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

Single block disable for asynchronized subprograms

The ASUP1.SYF and ASUP2.SYF subprograms run system-internally with REPOS must be executed step by step. In Software Version 4.3 and higher, the system ASUP can be executed in one step by programming SBLOF.

Example:

```
N10 SBLOF
N20 IF $AC_ASUP=='H200'
N30 RET
N40 ELSE
N50 REPOSA
N60 ENDIF
N70 RET
```

No REPOS with mode change

REPOS in all other cases

Supplementary conditions

- Display of the current block can be suppressed in cycles by means of DISPLOF.
- If DISPLOF is programmed together with SBLOF, then the cycle call is still displayed in single block stops within a cycle.
- The default setting made in MD 20117: IGNORE_SINGLEBLOCK_ASUP for the behavior of asynchronized subprograms in single block mode can be program-specifically overwritten by programming SBLOF.
- For testing purposes it is possible to suppress the effectiveness of SBLOF via OPI variable (see OEM documentation).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming example 1

Cycle is to act as a command for programmer

Main program

```
N10 G1 X10 G90 F200
N20 X-4 Y6
N30 CYCLE1
N40 G1 X0
N50 M30
```

Program cycle1

```
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 RET
```

The cycle CYCLE1 is executed as one step when single block is active.

Programming example 2

An ASUP run from the PLC for activating modified zero offsets and tool offsets should not be visible.

```
N100 PROC NV 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
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

2.11 Executing an external subprogram (SW 4.2 and higher)

This function only applies to MMC 102/103.

You can use EXTCALL to reload a program from the MMC 102/103 in "Execution from external" mode.

```
EXTCALL path/program name
```

Explanation

`EXTCALL`	Keyword for subprogram call
`Path name`	optional, not essential Constant/variable of type STRING. Absolute path name beginning "/",
`Program name`	The program identifier 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 "SHAFT"` bzw. `EXTCALL"/_N_WCS_DIR/_N_SHAFT_WPD/SHAFT"`

Function

During the machining of complex workpieces, program sequences may be generated for the separate machining stages that cannot be stored in main memory due to their memory space requirements.
You can use EXTCALL to reload a program from MMC 102/103 in "Execution from external" mode.
All programs that can be accessed via the directory structure of MMC102 can be reloaded.

SD 42700 EXT_PROG_PATH
The channel-specific setting data
SD 42700 EXT_PROG_PATH is available thanks to flexible setting options for the call 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

If the external subprogram is called without an absolute path name, the same search path is executed on the MMC as for calling a subprogram from user memory.

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. The size of the memory can be adjusted via MD 18360 EXT_PROG_BUFFER_SIZE.

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

Additional notes

External subprograms are not permitted to include jump commands such as GOTOF, GOTOB, CASE, IF - ELSE, FOR, LOOP, WHILE or REPEAT. Subprogram calls are possible.

Programming example

Setting data $SC_EXT_PROG_PATH contains the following path: "_N_WCS_DIR/_N_WPC1".
The main program _N_MAIN_MPF is in user memory and is selected.

```
%_N_MACHINE1_MPF
N10 PROC MAIN
N20 ...
N30 EXTCALL ROUGHING_SPF                ; Call of external subprogram
                                        ROUGHING_SPF
N40 ...
N50 M30
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Subprogram ROUGHING_SPF (located in the MMC directory structure under workpieces->WST1):

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

2.12 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 in the program: `;$PATH=/_N_CST_DIR)`.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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_CUS_DIR`	Path specification
`;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: Any integer, must not have been used in the file before;

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 (SPECIALCYCLE)
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Example of uc.com file – user cycle description

The explanation is based on the continuation of the example:

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	

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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_
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Line for description for each parameter:

```
(<Data type identification> / <Minimum value> <Maximum value>
(<Default 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

2.13 Macros

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 recurr are only programmed once as a macro in a separate macro block and once at the beginning of the program.
The macro can then be called in any main program or subprogram and executed.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

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

Three-digit M/G function (as of SW 5)

- up to **SW 4**
  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

Additional notes

Nesting of macros is not possible.

Two-digit H and L functions can be programmed.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Programming example

Example of macro definitions.

`DEFINE M6 AS L6`	On gear change, a subroutine is called to handle care of data transfer. The actual tool change 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 part 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 block 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)**.

File and Program Management

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

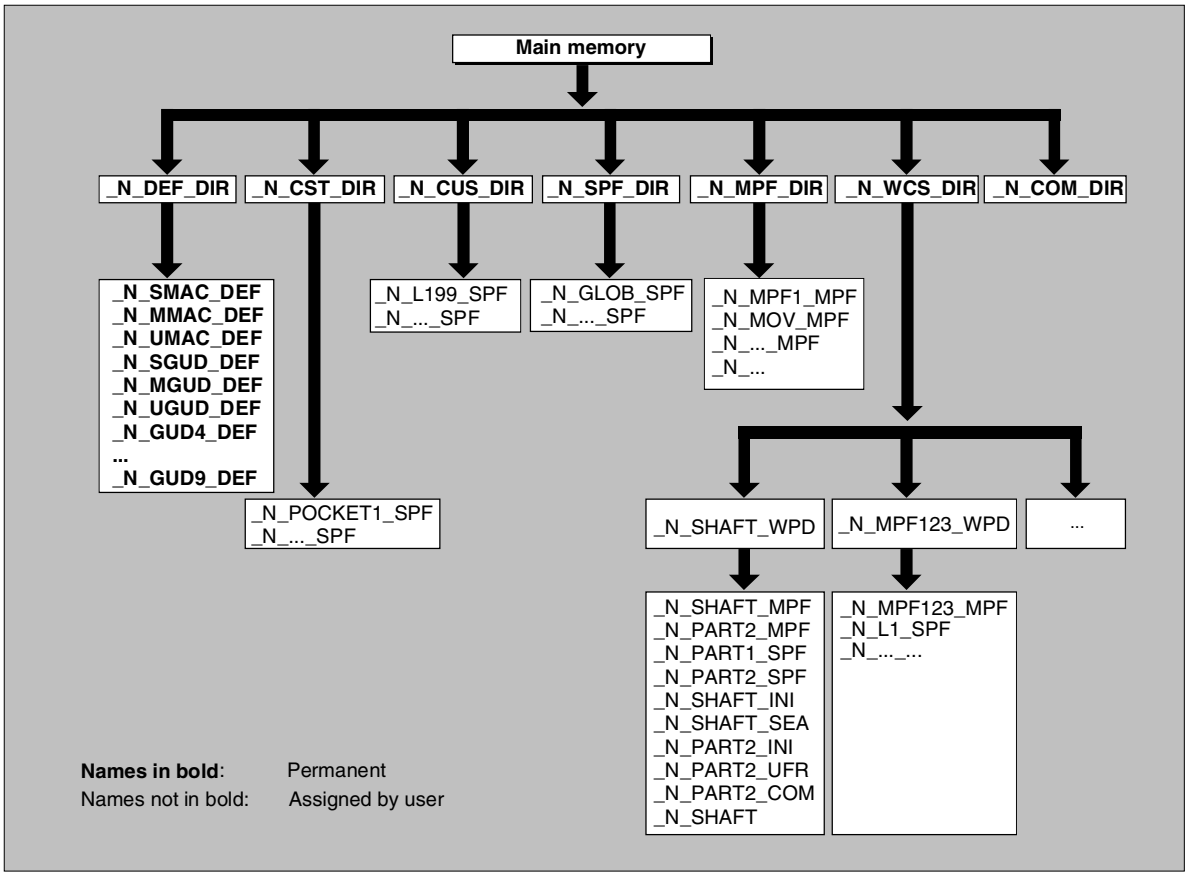
The files and programs are stored in the program memory and are thus permanently stored (passive file system).
Example:
Main programs and subprograms, macro definitions.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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).



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Directories

The following directories are provided as standard when a display and operator unit is connected:

1. _N_DEF_DIR	Data modules and macro modules (assigned on startup)
2. _N_CST_DIR	Standard cycles (assigned on startup)
3. _N_CUS_DIR	User cycles (assigned on startup)
4. _N_WCS_DIR	Workpieces
5. _N_SPF_DIR	Global subprograms
6. _N_MPF_DIR	Standard directory for main programs
7. _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 file
name_GUD	Global user data
name_RPA	R parameters
name_COM	Comments
name_DEF	Definitions for global user data and macros

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Workpiece directory, _N_WCS_DIR
The workpiece directory exists in the standard setup of the program directory under the name `_N_WCS_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.

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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_WCS_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
;$PATH=/_N_WCS_DIR/_N_SHAFT_WPD
N10 G0 X… Z…
•
M2

•
•
%_N_SHAFT_SPF
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Select workpiece for machining
A workpiece directory can be selected for execution in a channel.
If a main program with the **same name** is stored in this directory, this is automatically selected for execution.

Example:
The workpiece directory
`/_N_WCS_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 part 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_CST_DIR / name_SPF`	Standard cycles

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

3.3 User 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 the SINUMERIK FM-NC, 810D CCU1, 840D NCU 571) and can usually use several axes. The following areas are available:

Identifier	Data areas
NCK	NCK-specific data
CH<n>	Channel-specific data (n specifies the channel number)
AX<n>	Axis-specific data (n specifies the number of the machine axis)
TO	Tool data
COMPLETE	All data

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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`.
- An INI file: `INITIAL` can be created across all areas with `_N_INITIAL_INI`.

Loading initialization programs

INI programs can also be selected and called as part 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 part 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

3.4 Defining user data

Function

Definition of user data (GUD) implemented during start-up procedure.
The necessary machine data should be initialized accordingly.
The user memory must be configured, the necessary memory configuration must be defined in file `%_N_INITIAL_INI` which is loaded after the definition file. All relevant machine data have as a component of their name `GUD`.

- **SW 5** and higher (01.99):
  The user data (GUD) can be defined in the Services operating area. This means that lengthy reimport of data backup (`%_N_INITIAL_INI)` is not necessary.
  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 block names

The following modules can be stored in the directory `/_N_DEF_DIR`:

`_N_SMAC_DEF`	Contains macro definitions (Siemens, protection level 0)
`_N_MMAC_DEF`	Contains macro definitions (machine manufacturer, protection level 2)
`_N_UMAC_DEF`	Contains macro definitions (user, protection level 3)
`_N_SGUD_DEF`	Contains definitions for global data (Siemens, protection level 0)
`_N_MGUD_DEF`	Contains definitions for global data (machine manufacturer, protection level 2)
`_N_UGUD_DEF`	Contains definitions for global data (user, protection level 3)
`_N_GUD4_DEF`	Freely definable
`_N_GUD5_DEF`	Contains definitions for measuring cycles (Siemens, protection level 0)
`_N_GUD6_DEF`	Contains definitions for measuring cycles (Siemens, protection level 0)
`_N_GUD7_DEF`	Contains definitions for standard cycles (Siemens, protection level 0) or freely definable without standard cycles
`_N_GUD8_DEF`	Freely definable
`_N_GUD9_DEF`	Freely definable

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

The access authorization is assigned in the definition file with the APR or APW command.
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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Defining user data (GUD)

1. Save block _N_INITIAL_INI.
2. Creating a definition file for user data
   - on an external PC (up to **SW 4**)
   - in the Services operating area (**SW 5** and higher)

   Predefined file names are provided (see previous page):

   ```
   _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.

   An additional attribute is required to identify the variable as a GUD variable and to define the area in which the definition is to be valid:

   `NCK, CHAN.`

   An implicit preprocess stop can also be defined when the variables are read and/or written at a later stage:

   `SYNR`: Preprocess stop while reading

   `SYNRW`: Preprocess stop during read/write
3. Load the definition file in 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 V.24 interface.

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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

4. Activating definition files
   - Up to **SW4**
     The definition file only becomes active after the `_N_INITIAL_INI` file is read in.
   - **SW 5** and higher
     When the GUD definition file is loaded into the NC ("Load" softkey), it becomes active.

Save all programs, frames and machine data before reading in `_N_INITIAL_INI` because the static memory will be formatted.

5. Data storage
   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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

3.5 Defining protection levels for user data (GUD)

Explanation

APR n	Read access protection
APW n	Write access protection
n	Protection level n from 0/10 (highest level) to 7/17 (lowest level)

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

APW 10-17, APR 10-17:
The module variables can still be written or read via the NC program or in MDA mode.

Protection levels
0/10 = SIEMENS
1/11 = OEM_ HIGH
2/12 = OEM _LOW
3/13 = end user
4/14 to 7/17 = keyswitch position 3 to 0

Note

The command input sequence is as follows:
APR.. APW..
Any other sequence represents a syntax error.
In order to protect a complete file, the commands must be entered in the first line of the file!

Function

Access criteria can be defined for GUD modules to protect them against manipulation. Using such criteria, for example, it is possible to inhibit changes to cycles that the machine manufacturer has set up as GUD modules.
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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Sequence

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

Example of a definition file with read/write access protection (machine manufacturer):

`%_N_GUD6_DEF`	
`;$PATH=/_N_DEF_DIR`	
`APR 5 APW 2`	;Read/display with protection level keyswitch position 2 ;Write with protection level OEM_LOW ;Caution! This entry can alter the access rights of the file itself (see above)
`DEF CHAN REAL_CORRVAL`	
…	
`M30`	;

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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" → Then a prompt is displayed whether the currently active data are to be saved.
"Do you want to save the previous data in the definitions?"
"OK" → The GUD blocks of the definition file to be edited are saved, the new definitions are activated and the saved data are imported again.
"Cancel" → The new definitions are activated, the old data cleared.
"Cancel" → The changes in the definition file are discarded, 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 softkey 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.

Protection Zones

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

4.1 Defining the protection zones CPROTDEF, NPROTDEF

Programming

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

Explanation of the commands

DEF INT NOT_USED	Locale variable, define data type integer (cf. Chapter 10)
CPROTDEF	Define channel-specific protection zones (for NCU 572/573 only)
NPROTDEF	Define 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 limit in the 3rd 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

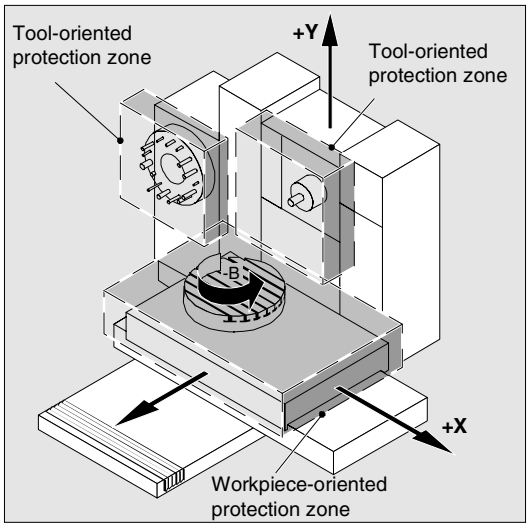
4.1 Defining the protection zones CPROTDEF, NPROTDEF

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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).



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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 travel motions programmed between CPROTDEF or NPROTDEF and EXECUTE are not executed, but merely define the protection zone.

Working 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 are permitted:
- G0, G1 for straight contour elements
- G2 for clockwise circle segments (only for tool-oriented protection zones)
- G3 for counterclockwise circle segments

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

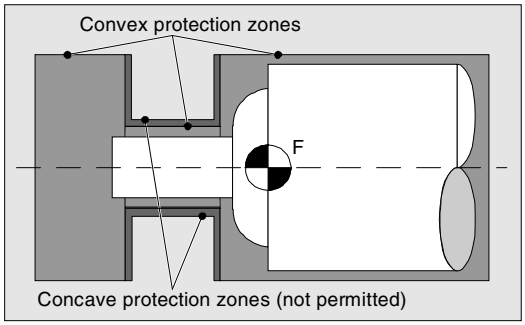
If a full circle describes the protection zone, it must be divided into two half circles. The order G2, G3 or G3, G2 is not permitted. A short G1 block must be inserted, if necessary.

The last point in the contour description must coincide with the first.

External protection zones (only possible for workpiece-oriented protection zones) should 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.



The following must not be active while the protection zones are defined:

- Cutter radius or tool nose radius compensation,
- Transformation,
- Frame.

Nor must reference point approach (G74), fixed point approach (G75), block search stop or program end be programmed.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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, deactivating protection zones or deactivate active protection zones for collision monitoring.

The maximum number of protection zones which can be active simultaneously on the same channel is defined in machine data.

If no tool-oriented protection zone is active, the tool path is checked against the workpiece-oriented protection zones.

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Sequence

Activation status

A protection zone is generally activated in the part 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.

Offset 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.

Additional notes

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.

4.2 Activating/deactivating protection zones: CPROT, NPROT

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 rules apply:

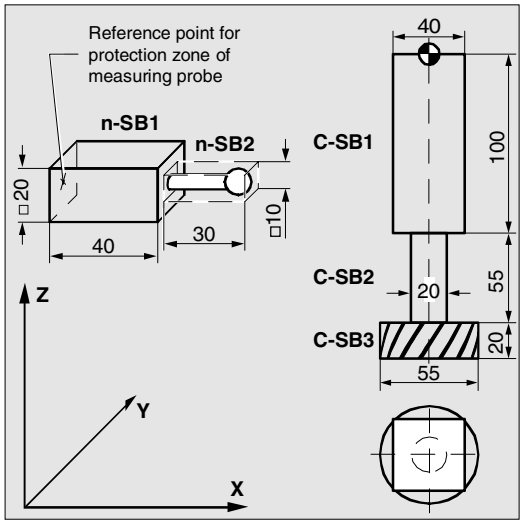
- The protection zone cannot be activated simultaneously with different offsets in a single channel.
- Machine-oriented protection zones must have the same orientation on both channels.

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

`DEF INT PROTECTB`	Definition of an auxiliary variable
Definition of protection zones `G17`	Set orientation

`NPROTDEF(1,FALSE,3,10,-10)`
`G01 X0 Y-10`
`X40`
`Y10`
`X0`
`Y-10`
`EXECUTE(PROTECTB)`
	Protection zone n–SB1

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

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

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

`CPROTDEF(3,TRUE,3,-150,-170)`
`G01 X0 Y-27,5`
`G03 X0 Y27,5 J27,5`
`X0 Y27,5 J-27,5`
`EXECUTE(PROTECTB)`
	Protection zone c–SB3

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Notes

Special Motion Commands

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

Explanation of the commands

`CAC(n)`	Approach coded positions absolutely
`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

You can enter a maximum of 60 (0 to 59) positions in special position tables for 2 axes in machine data.

For an example of a typical position table see diagram.

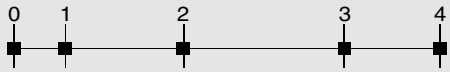


Table 1 (linear axis)

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

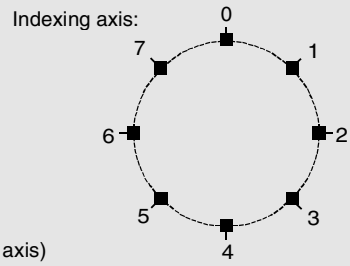


Table 1 (rotary axis)

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

Further details

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

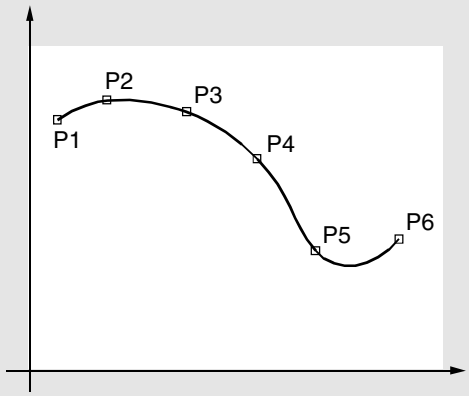
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 840Di

5.2 Spline interpolation

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

```
ASPLINEX 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)

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Additional notes

A, B and C splines are modally active and belong to the group of motion commands. The toolradius offseet 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).

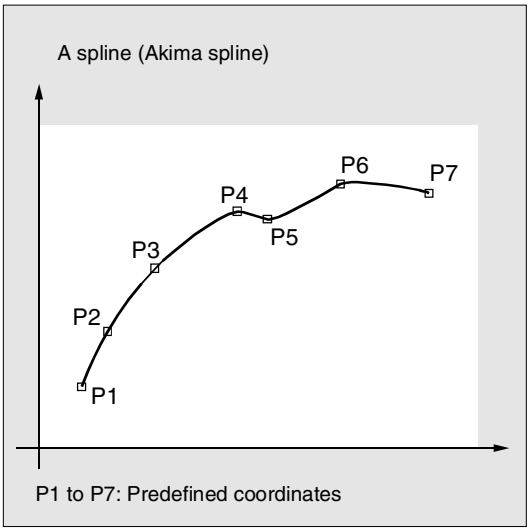
Sequence

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 6 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 3rd-degree polynomial is used for interpolation.

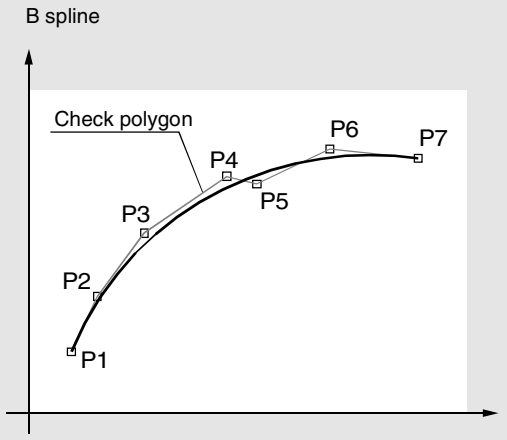


840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

B SPLINE

With a B spline, the programmed positions are not interpolation points, but merely check points of the spline, i.e. the curve is "drawn towards" the points, but does not pass directly 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 3rd-degree B spline does not produce any oscillations in spite of its continuously curved transitions.



P1 to P7: Predefined coordinates

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

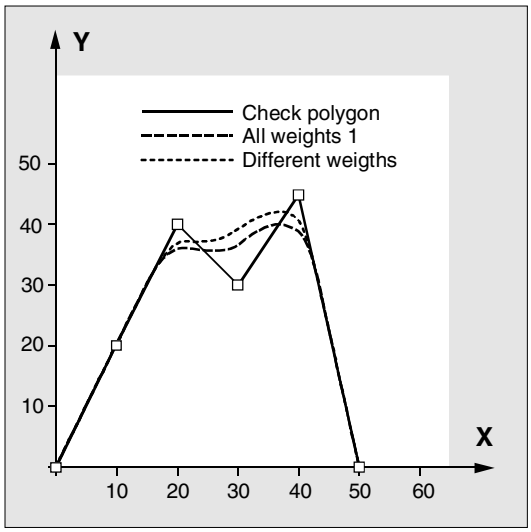
Spline degree:
A 3rd-degree polygon is used as standard, but a 2nd-degree polygon is also possible.

Programming:
`SD = 2`

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Distance between nodes:
Node distances are appropriately calculated internally in the control, but the system is also capable of processing user-programmed node distances.

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

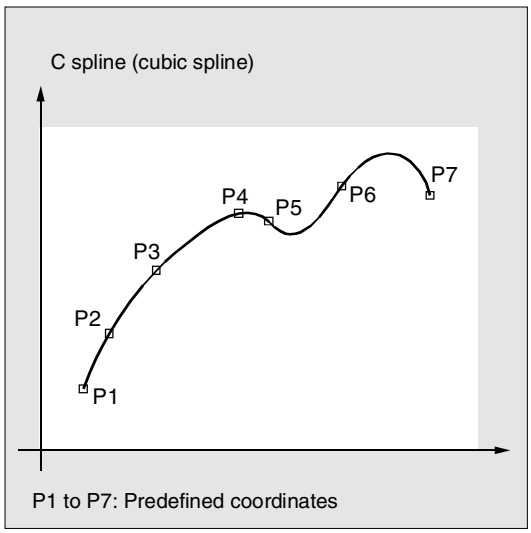


Example of B spline:

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 3rd-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).



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Supplementary conditions

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.

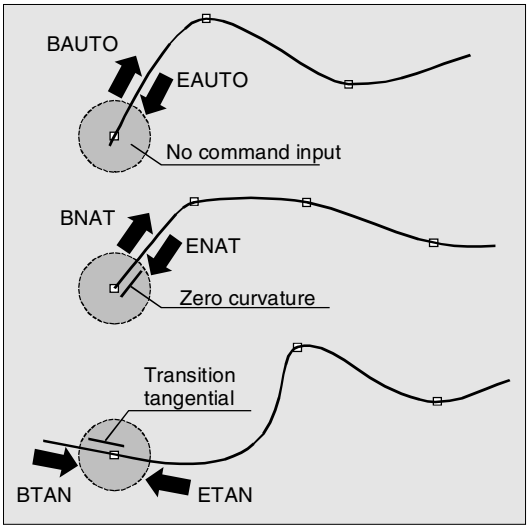
Explanation of the commands

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:

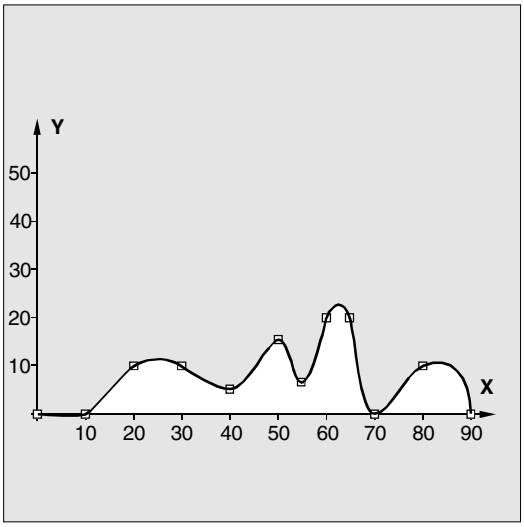
EAUTO	No command input; end is determined by the position of the last point
ENAT	Zero curvature
ETAN	Tangential transition to next block (initial setting)



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Example

C spline, zero curvature at start and end



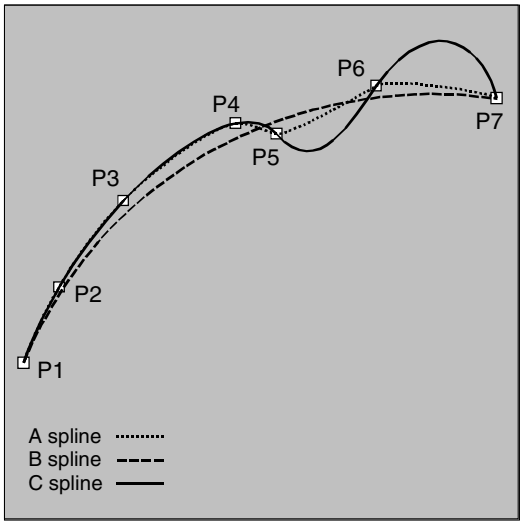
N10 G1 X0 Y0 F300	
N15 X10	
N20 BNAT ENAT	C spline, at start and end Zero curvature
N30 CSPLINE X20 Y10	
N40 X30	
N50 X40 Y5	
N60 X50 Y15	
N70 X55 Y7	
N80 X60 Y20	
N90 X65 Y20	
N100 X70 Y0	
N110 X80 Y10	
N120 X90 Y0	
N130 M30	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming

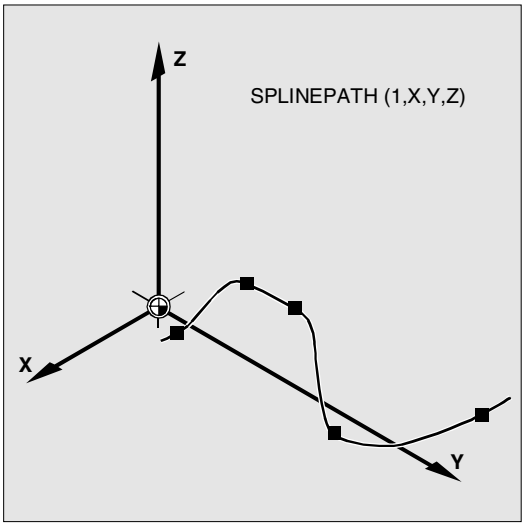
```
SPLINEPATH(n,X,Y,Z,…)
```

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 10 blocks as standard.

Not all block information is a spline endpoint. However, the control requires a certain number of spline endpoint blocks from 10 blocks.

They are as follows 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

5.3 Compressor COMPON/COMPCURV

As a rule, CAD/CAM systems provide linear blocks that meet the programmed accuracy.
With complex contours this leads to a considerable amount of data and to short path sections. These short path sections restrict the execution speed.
With the compressor a certain number (max. 10) of these short path sections can be joined together to form one path section.

The modal G code COMPON or COMPCURV activates an "NC block compressor".
With linear interpolation, this function groups a number of straight blocks (number is restricted to 10) and approaches them by means of third degree polynomials (COMPON), or five degree polynomials (COMPCURV), within an error tolerance range specified via machine data. In this way, the NC processes one large motion block rather than a large number of small ones.

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.
Blocks to be compressed may contain only block number, G1, axis addresses, feed and comment. This sequence is mandatory. Variables may not be used.

840 D NCU 571 | 840 D NCU 572 NCU 573 | FM-NC | 840Di

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.

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

Programming

`COMPON/COMPCURV`	Activate compressor
`COMPOF`	Deactivate compressor

Machine manufacturer

Three 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.
  Larger blocks are not compressed.
- $MA_COMPRESS_POS_TOL
  A tolerance can be set for each axis. The generated spline curve does not deviate by more than this value from the programmed end points. The higher these values are set, 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Example

`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
```

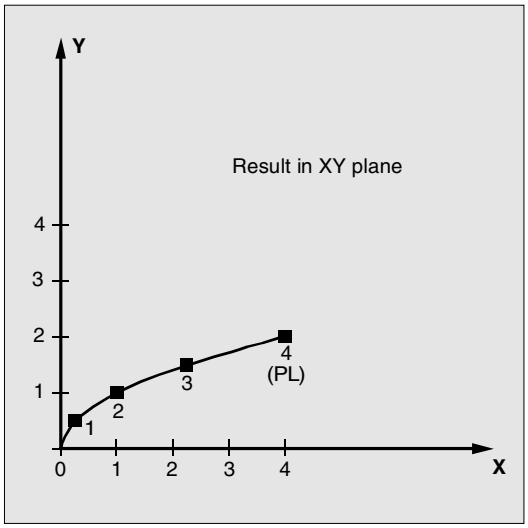
Not compressed are e.g.

extended addresses such as C=100 or A=ACNC.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

5.4 Polynomial interpolation, POLY

The control system is capable of traversing curves (paths) in which every selected path axis is operating according to a function (max. 3rd degree polynomial).



The equation used to express the polynomial function is generally as follows:
$f(p)= a_0 + a_1p + a_2p^2 + a_3p^3$

The letters have the following meaning:
a_n: Constant coefficients
p: Parameters

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

By setting the coefficients as $a_2 = a_3 = 0$,
it is possible to create, e.g. a straight line with
$f(p) = a_0 + a_1p$

Meanings:
a_0 = Axis position at end of preceding block
a_1 = Axis position at end of definition area (PL)

Definition

Polynomial interpolation (POLY) is not one of the real types of spline interpolation. 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

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

Polynomial coefficient

The PO value `(PO[]=)` 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Programming

POLY PO[X]=(x_{e1},a_2,a_3) PO[Y]=(y_{e1},b_2,b_3) PO[Z]=(ze1,c_2,c_3) PL=n

Explanation

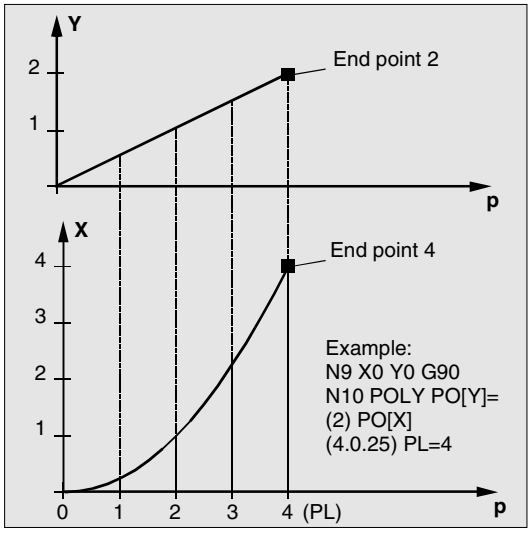
POLY	Activation of polynomial interpolation with a block containing POLY.
PO[]=(…,…,…)	End points and polynomial coefficients
x_e, y_e, z_e	Specification of end position for relevant axis; value range as for path dimension
a_2, a_3	Coefficients a_2 and a_3 are programmed with their value; value range as for path dimension. The last coefficient in each case can be omitted if it equals zero.
PL	Length of parameter interval over which the polynomials are defined (definition range of function f(p)). The interval always starts at 0. p can be set to values between 0 and PL. Theoretical value range for PL: 0.0001 ... 99 999.9999. The PL values applies to the block in which it is programmed. PL=1 is applied if no PL value is programmed.

Example

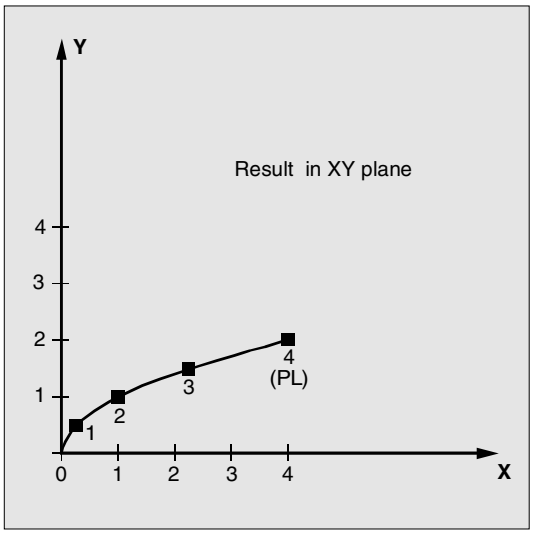
N10 G1 X… Y… Z… F600	
N11 POLY PO[X]=(1,2.5,0.7) -> -> PO[Y]=(0.3,1,3.2) PL=1.5	Polynomial interpolation ON
N12 PO[X]=(0,2.5,1.7) PO[Y]=(2.3,1.7) PL=3	
…	
N20 M8 H126 …	
N25 X70 PO[Y]=(9.3,1,7.67) PL=5	Mixed settings for axes
N27 PO[X]=(10.2.5) PO[Y]=(2.3)	No PL value programmed; PL=1 applies
N30 G1 X… Y… Z.	Polynomial interpolation OFF
…	

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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
```



840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Special case denominator polynomial

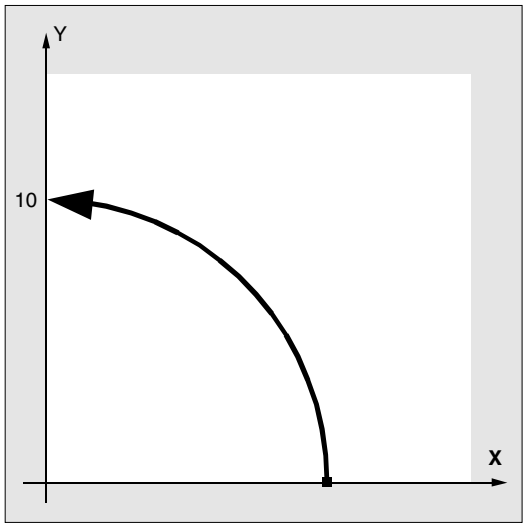
Command `PO[]=(...)` can be used to program a common denominator polynomial for the geometry axes (without specification of axes names), i.e. the motion of the geometry axes is then interpolated as the quotient of two polynomials.

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 (a_0) of the denominator polynomial is always assumed to be 1, the specified end point is not dependent on G90/G91.



The result obtained from the above example is as follows:

$X(p)=10(1-p^2)/(1+p^2)$ 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 = $1+2*p+1*p^2$

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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.

Additional notes

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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

Programming

SPATH	Path reference for FGROUP axes is length of arc
UPATH	The curve parameter is the path reference for FGROUP axes

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 Software Version 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 $p_i(U)$. Curve parameter U moves from 0 to 1 within an NC block, therefore it is standardized.

The axes to which the programmed path feed is to relate can be selected from the path axes by means of language command FGROUP. However, during polynomial interpolation, an interpolation with constant velocity on path S of these axes usually means a non constant change of curve parameter U.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 curve parameter U of the 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.

Expansion of rounding

If all of the path axes are not contained in FGROUP, the axes that are not included will often cause a sudden change in velocity at block transitions. By reducing the velocity at block change, the control can limit the extent of this change to the permissible value set in MD 32300: MAX_AX_ACCEL and MD 32310: _MAX_ACCEL_OVL_FACTOR. This deceleration can be prevented by rounding the specified position relationship of the path axes.

- Rounding with G641
  Rounding is activated modally by means of G641 and specifying a rounding radius ADIS (or ADISPOS in rapid traverse) for path functions. The control is now free to dissolve the path relationship within this radius around the block change point and replace it with a dynamically optimum path.
  Disadvantage: Only one ADIS value is available for all axes.
  See also: References /PG/, Programming Guide Fundamentals, Chapter 5, Path Action

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

- Rounding with G642
  Rounding with axial tolerances is activated modally by means of G642. Rounding does not take place within a defined ADIS area, instead it ensures the axial tolerances set in MD 33100: COMPRESS_POS_TOL are adhered to.
  The rest of the functionality is identical with G641.

References: /FB/, B1, Continuous Path Mode, Exact Stop and LookAhead

Supplementary conditions

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.

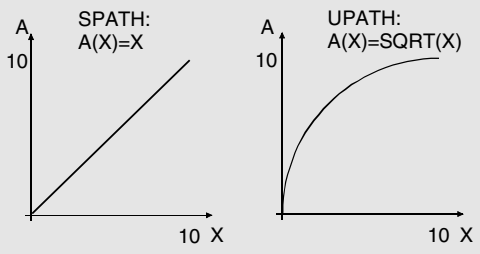
Activation

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 at Power ON, mode change, Reset, block search, REPOS

After 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).

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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)

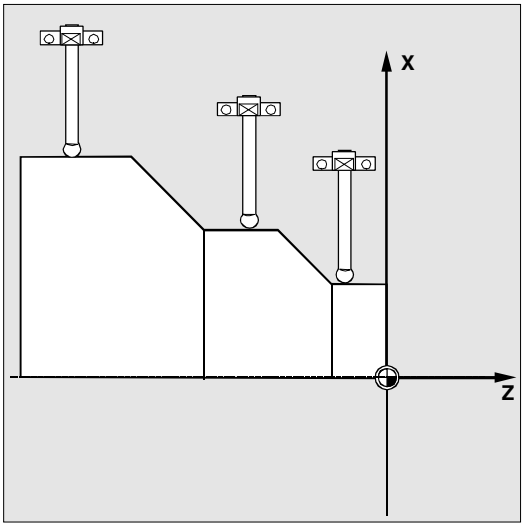
Explanation of the commands

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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.

Comment
If a GEO axis is programmed in a measurement block, the measured values for all current GEO axes are recorded.
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.

Additional notes

The MEAS and MEAW functions are active non-modally.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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

Programming

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

Explanation

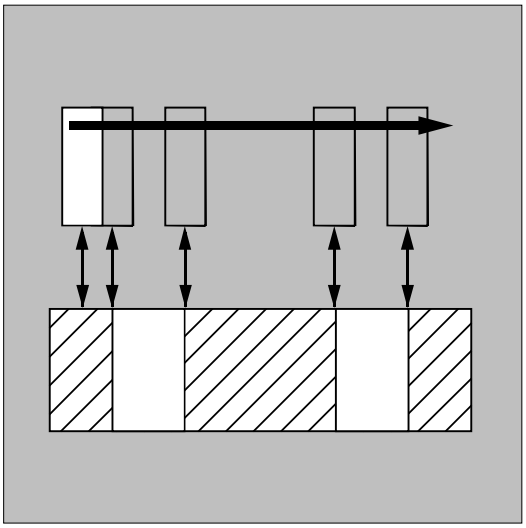
Axis	Name of channel axis used for measurement
Mode	Two-digit setting for operating mode consisting of **Measuring mode** (ones decade) and 0 Cancel measurement task 1 Mode **1**: Up to 4 trigger events that can be activated **simultaneously** 2 Mode **2**: Up to 4 trigger events that can be activated **sequentially** 3 Mode **3**: Up to 4 trigger events that can be activated **sequentially** However, no monitoring of trigger event 1 on START (alarms 21700/21703 are suppressed) Note: Mode 3 not possible with MEAC **Measuring system** (tens' decade) 0 or no setting: Active measuring system 1 Measuring system 1 2 Measuring system 2 3 Both measuring systems
TE 1…4	**Trigger event** 1 Rising edge, probe 1 –1 Falling edge, probe 1 2 Rising edge, probe 2 –2 Falling edge, probe 2
Measurement memory	Number of FIFO (circulating storage)

5.7 Extended measuring function MEASA, MEAWA, MEAC

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

Function

Axial measurement is available from SW version 4. 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 part 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.

Additional notes

- 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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.

Additional notes

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

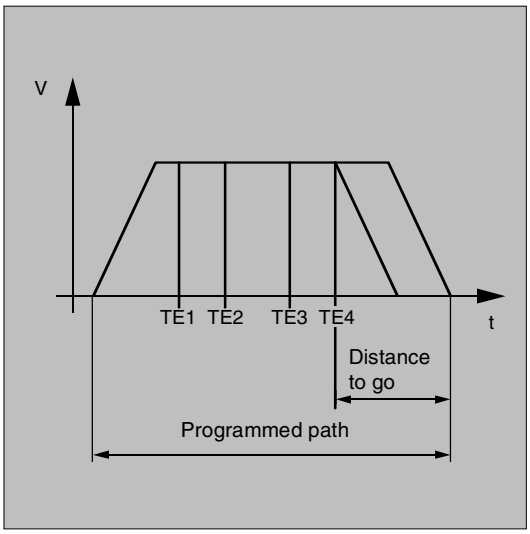
Measurement with and without delete distance-to-go

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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

Additional notes

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 2 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 2 on trigger event 1
$AA_MM4[axis]	or	$AA_MW4[axis]	Measured value of measuring system 2 on trigger event

Measuring probe status can be read via
$A_PROBE[n]
n=Probe
1==Probe deflected
0==Probe not deflected

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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: Measuring active
$AA_MEAACT==0: Measuring not active
References: /FB/ M5, Measurement

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

Additional notes

- 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

Programming example

Measurement with delete distance-to-go in mode 1

(evaluation in chronological sequence)

a) with 1 measuring system

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

Programming example

b) with 2 measuring systems

...	
`N200 MEASA[X] = (31,1-1) G01 X100 F100`	Measurement in mode 1 with both measuring systems. Wait for measuring signal with rising/falling edge from probe 1 on travel path to X = 100.
`N210 STOPRE`	Preprocessing stop
`N220 IF $AC_MEA[1] == FALSE gotof END`	Check success of measurement.
`N230 R10 = $AA_MM1[X]`	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:`	

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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`	Preprocessing stop
`N120 IF $AC_MEA[1] == FALSE gotof`	Check success of measurement with probe 1
`PROBE2`	
`N130 R10 = $AA_MM1[X]`	Store measured value acquired on first programmed trigger event (rising edge probe 1)
`N140 R11 = $AA_MM2[X]`	Store measured value acquired on second programmed trigger event (rising edge probe 1)
`N150 PROBE2:`	
`N160 IF $AC_MEA[2] == FALSE gotof END`	Check success of measurement with probe 2
`N170 R12 = $AA_MM3[X]`	Store measured value acquired on third programmed trigger event (rising edge probe 2)
`N180 R13 = $AA_MM4[X]`	Store measured value acquired on fourth programmed trigger event (rising edge probe 2)
`N190 END:`	

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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 INDEX = 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 INDEX = 0 TO R1-1`	
`N170 MEASVALUE[INDEX] = $AC_FIFO1[0]`	Read measured values from $AC_FIFO1 and store.
`N180 ENDFOR`	

Measurement with delete distance-to-go after 10 measured values

`...`	
`(x)`	Delete distance-to-go
`N20 MEAC[x]=(1,1,1,-1) G01 X100 F500`	
`N30 MEAC[X]=(0)`	
`N40 R1 = $AC_FIFO1[4]`	Number of measured values
`...`	

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

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
  ```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

5.8 Special functions for OEM users

OEM addresses

The meaning of OEM addresses is determined by the OEM user.
Their functionality is incorporated by means of compile cycles. 5 OEM addresses are reserved.
The address identifiers are settable.
OEM addresses can be programmed in any block.

OEM interpolations

The OEM user can define two additional interpolations. Their functionality is incorporated by means of compile cycles.
The names of G functions (OEMIPO1, OEMIPO2) are set by the OEM user.
OEM addresses (see above) can be used specifically for OEM interpolations.

Reserved G groups G800 – 819

Two G groups with 10 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

5.9 Programmable motion end criterion (SW 5.1 and higher)

Programming

```
FINEA[<axis>]
COARSEA[<axis>]
IPOENDA[<axis>]
```

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
Axis	Channel axis name (X, Y, ...)

Function

Similar to the block change criterion for continuous-path interpolation (G601, G602 and G603), the end of motion criiterion can be programmed in a part program for single axis interpolation or in synchronized action for the command-/PLC-axes. Depending on the end of motion criterion set, part program blocks or technology cycle blocks with single axis motion
take different times to comlete.
The same applies for PLC-positioning statements via FC15/ 16/ 18.

System variable $AA_MOTENDA
The set end of motion criterion can be polled using the system variables `$AA_MOTENDA[<axis>]`.

- `$AA_MOTENDA[<axis>] = 1` Motion end with "Exact stop fine"
- `$AA_MOTENDA[<axis>] = 2` Motion end with "Exact stop coarse"
- `$AA_MOTENDA[<axis>] = 3` End of motion with "IPO-Stop".

Additional notes

The last programmed value is retained after RESET.
References: /FB1/V1 feedrates

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

Programming example

```
...
N110 G01 POS[X]=100 FA[X]=1000 ACC[X]=90 IPOENDA[X]
```

Traversing to position X100 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

```
...
```

5.10 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 part 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 block definition locked through SCPARA".

840D NCU 571 | 840D NCU 572 NCU 573 | 810D CCU 2 | 840Di

In the case of a locked parameter block for SCPARA, an error message is produced if programmed.

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

Additional notes

- Up to SW 5.1, the servo-parameter block can be specified only by the PLC (DB3n DBB9 Bit0–2). 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 part program and in a synchronized action and the PLC, the PLC-application program mst be extended.
- **References:** /FB1/V1 feedrates

Programming example

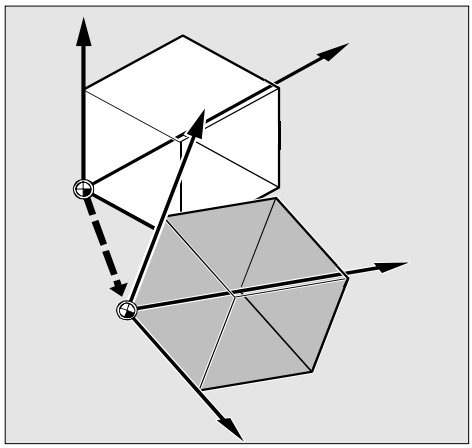
`...`	
`N110 SCPARA[X]= 3`	The 3rd parameter block is selected for axis X
`...`	

Frames

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.1 Coordinate transformation via frame variables

Definition of coordinate transformation 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 which can be processed in the NC program.
Possible frame variable:

- Base frame (basic offset)
- Settable frames
- Programmable frame

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

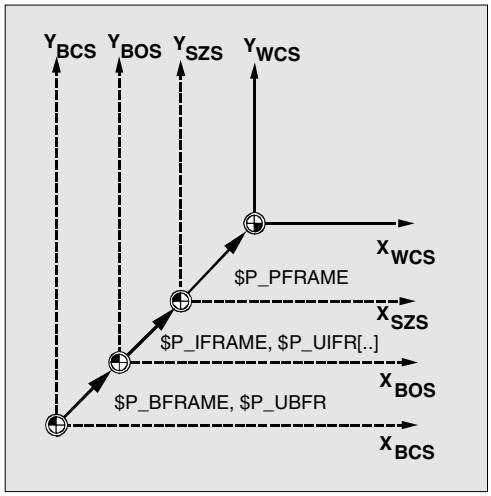
Frame variable/frame relationship

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 part 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

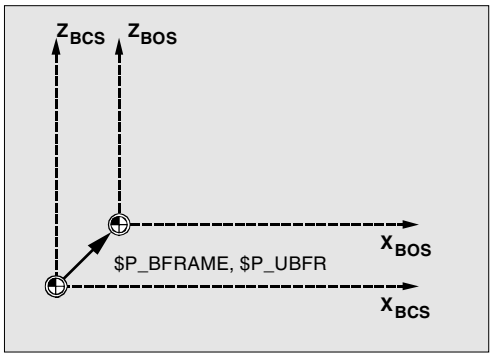
Overview of predefined Frame variables

$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.

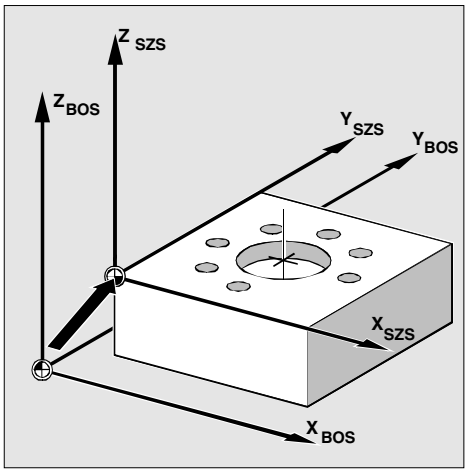


840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

$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]`

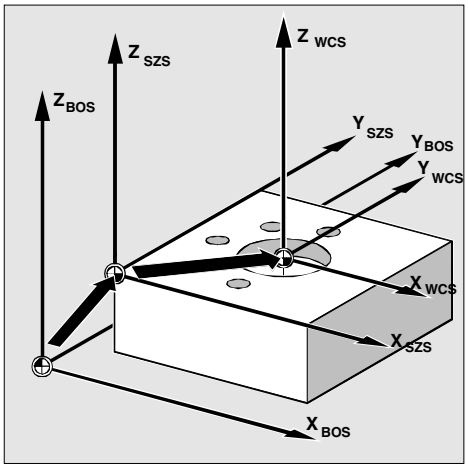
After G54 is programmed, for example, `$P_IFRAME` contains the translation, rotation, scaling and mirroring defined by G54.



$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.



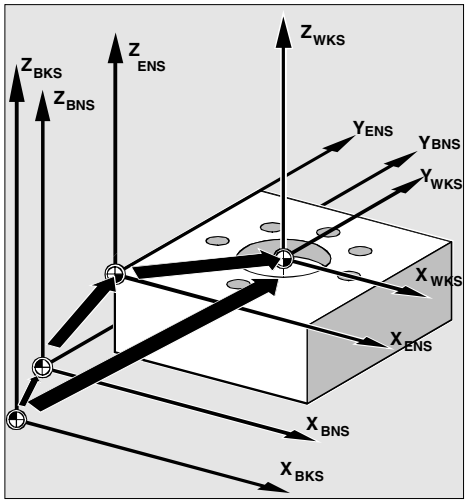
$P_ACTFRAME

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`.

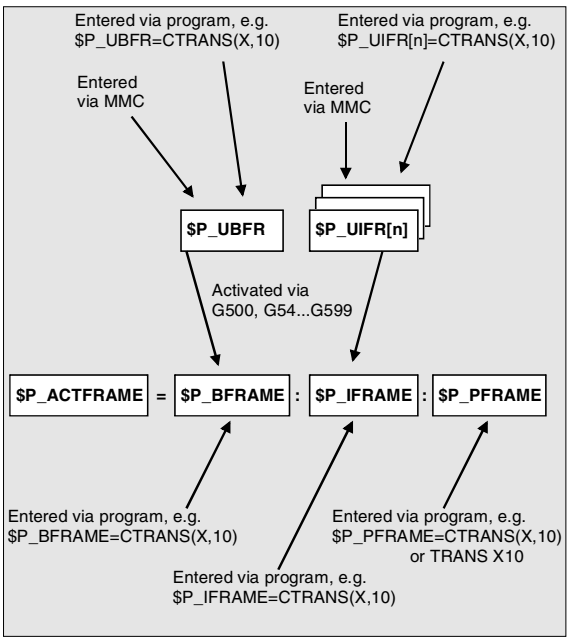
`$P_ACTFRAME` describes the currently valid workpiece zero.

If `$P_IFRAME`, `$P_BFRAME` or `$P_PFRAME` are changed, `$P_ACTFRAME` is recalculated.

`$P_ACTFRAME` corresponds to
`$P_BFRAME:$P_IFRAME:$P_PFRAME`



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di



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 part 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.

Predifined 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 part 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

`$P_IFRAME=$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, ...

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 `$P_BFRAME` is carried out analogous to `$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

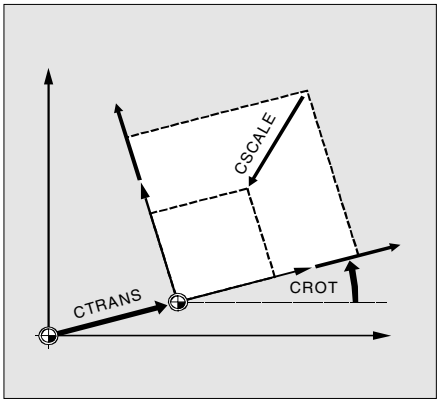
You can use these functions to assign frames or frame variables directly in the NC program.

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.



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

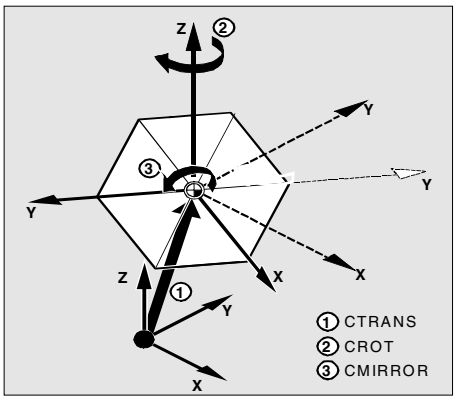
Additional notes

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)
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 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`
`SC`
`MI`
	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).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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]`.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

Additional notes

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]
```

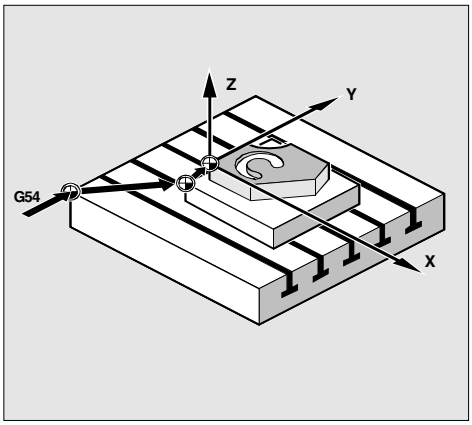
The settable frame 3 is created by chaining the settable frames 4 and 5.

6.2 Frame variables/assigning values to frames

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 linked to one another by the colon chain operator **:**.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Definition of new frames

Programming

```
DEF FRAME PALLET1

PALLET1=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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.3 Coarse/fine offset

Function

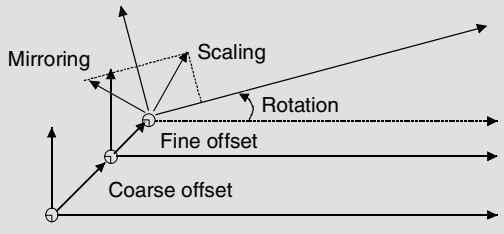
Fine offset

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)  ;chaining offset, fine offset and rotation
$P_UIFR[1]=CFINE(x, 0.5, y, 1.0, z, 0.1)             ;the total 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 the fine offset via variable FINEX
FINEX=$P_UIFR[3, X, FI]  ; Readout the fine offset of X axis in the 3rd frame via variable FINEX
```

Fine offset can only take place if MD18600: MM_FRAME_FINE_TRANS=1.

A fine offset changed via operator input is only active after the corresponding frame is activated, i.e. activation is conducted via G500, G54...G599. An activated fine offset of a frame is active for as long as the frame is active.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

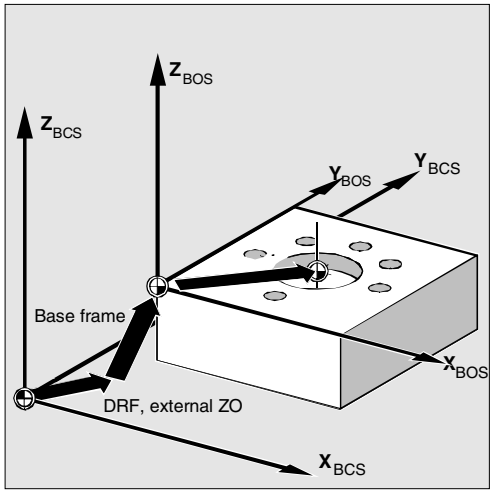
SW 5 and higher
The fine offset can be configured by means of MD18600 MM_FRAME_FINE_TRANS in the following variants:
0: Fine offset cannot be entered or programmed. G58 and G59 are not possible.
1: Fine offset for settable frames, base frames, programmable frames, G58 and G59 can be entered/programmed

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 acts on the basic coordinate system. See the diagram for the relationships.

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



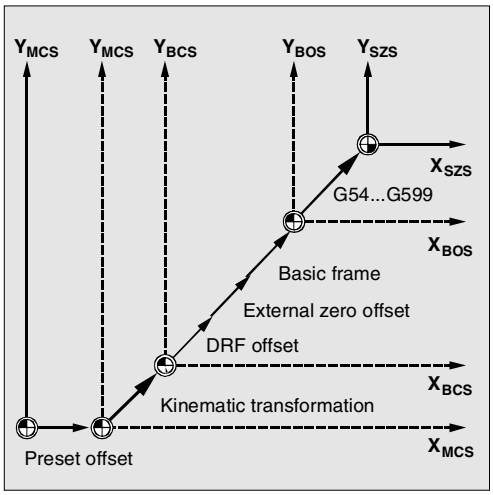
Clear DRF offset, DRFOF
DRFOF clears the handwheel offset for all axes assigned to the channel. DRFOF is programmed in a separate NC block.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 which contains the new value.

The external offset is generally set by the PLC and not specified in the part program.

The value entered in the part program only becomes active when the corresponding signal is enabled at the VDI interface (NCU-PLC interface).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.6 Programming Preset offset, PRESETON

Programming

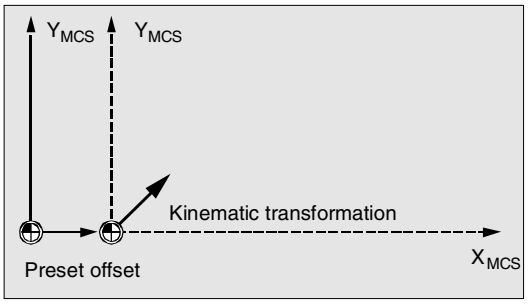
```
PRESETON(AXIS,VALUE,…)
```

Explanation of the commands

PRESETON	Set actual value
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).



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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.7 Deactivating frames

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)

Additional notes

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( )

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.8 Frame calculation from three measuring points in the area, 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 part program must be changed.
A frame can be determined by probing three points in the area for which the ideal positions are known.
Probing is performed with a tactile or optical sensor touching special holes or spheres that are precisely fixed to the backing plate.
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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming

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

Explanation of the commands

MEAFRAME	Frame calculation of 3 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 located approximately on a straight line: The frame could not be calculated. The frame variable returned contains a neutral frame. –2: The measured points are located approximately on a straight line: The frame could not be calculated. The frame variable returned contains a neutral frame. –4: The calculation of the rotation matrix failed for a different reason Positive value: Sum of the distortions (distances between the points) needed to reshape the measured triangle into one that is congruent to the ideal triangle.

Application example

```
; Part 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 ;
                                       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)
;
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

```
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]
N420 SHOW_MCS_POS1[1]=$AA_IM[Y]
N430 SHOW_MCS_POS1[2]=$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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.9 NCU-global frames (SW 5 and higher)

Function

NCU-global frames are only available once for all channels of each NCU. NCU-global frames can be written and read from all channels. The NCU-global frames are activated in the respective channel. Offsets, scalings and mirrorings can be applied to channel axes and machine axes by means of global frames.
With global frames there is no geometrical relationship between the axes. Therefore, it is not possible to perform rotations or program geometry axis identifiers.

- It is not possible to use global frames for rotations. If a rotation is programmed, it is rejected and alarm: "18310 channel%1block%2 frame: rotation not allowed" is issued.
- Chaining of global frames and channel-specific frames is possible. The resulting frame contains all frame elements including rotations for all axes. If a frame with rotation elements is assigned to a global frame, it is rejected and alarm "Frame: rotation not allowed" is issued.

NCU-global base frames: $P_NCBFR[n]
You can configure up to eight NCU-global base 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Global frames can be written and read from all channels of an NCU. When writing global frames, the user must pay attention to channel coordination, for example, by using Wait marks (WAITMC).

NCU-global settable frames: $P_UIFR[n]
The configuration of all settable frames G500, G54...G599 can be 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 for the 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 MD28081 MM_NUM_BASE_FRAMES.
The standard configuration provides at least one base frame per channel. A maximum of 8 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/base frames can be written and read from

- the PLC
- via the part program and

via the OPI.
Fine offset is also possible for global frames.
Suppression of global frames also takes place, as is the case with channel-specific frames, via G53, G153, SUPA and G500.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

$P_CHBFR[n]

The base frames can be read and written via system variable $P_CHBFR[n]. When writing a base frame, the chained total base frame is not activated; it is only activated when the G500, G54..G599 instruction is executed. The variable mainly serves as memory for writing processes to the MMC and PLC base frame. These frame variables are saved by data backup.

First base frame in the channel

Writing on the predefined variable $P_UBFR does not cause the base frame with array index 0 to be activated at the same time, rather activation occurs only when a G500, G54..G599 instruction is executed. The variable can also be written and read in the program.

$P_UBFR

$P_UBFR is identical to $P_CHBFR[0].
As standard, there is always a base frame in the channel making the system variable compatible with older versions. If there is no channel-specific base frame, an alarm is issued at read/write: "Frame: instruction not permissible".

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

6.9.2 Frames active in the channel

Function

$P_NCBFRAME[n]

Current NCU-global base frames

The current global base frame array elements can be written and read 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 only active in the channel in which the frame was programmed. If the frame is to be changed for all channels of an NCU, both [n] and $P_NCBFRAME[n] have to be programmed. The other channels must then still activate the frame with, for example, G54. When writing a base frame, the total base frame is calculated again.

$P_CHBFRAME[n]

Current channel base frames

The current channel base frame array elements can be written and read via system variable $P_CHBFRAME[n]. The resulting total base frame is calculated by means of the write process in the channel. When writing a base frame, the total base frame is calculated again.

$P_BFRAME

Current 1st base frame in the channel

The current base frame can be written and read in the part program via the predefined frame variable $P_BFRAME with the array index 0 that is valid in the part program. The written base frame is immediately included in the calculation.

$P_BFRAME is identical to $P_CHBFRAME[0]. By default, the system variable always a valid value. If there is no channel-specific base frame, an alarm is issued at read/write: "Frame: instruction not permissible".

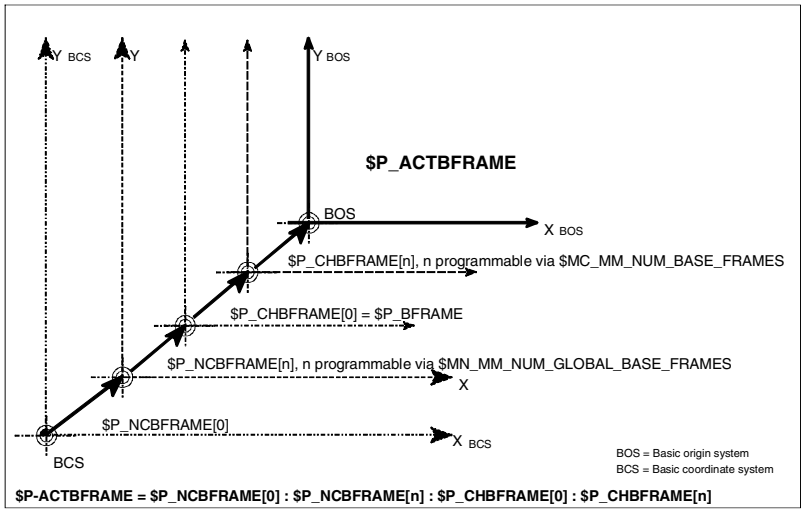
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

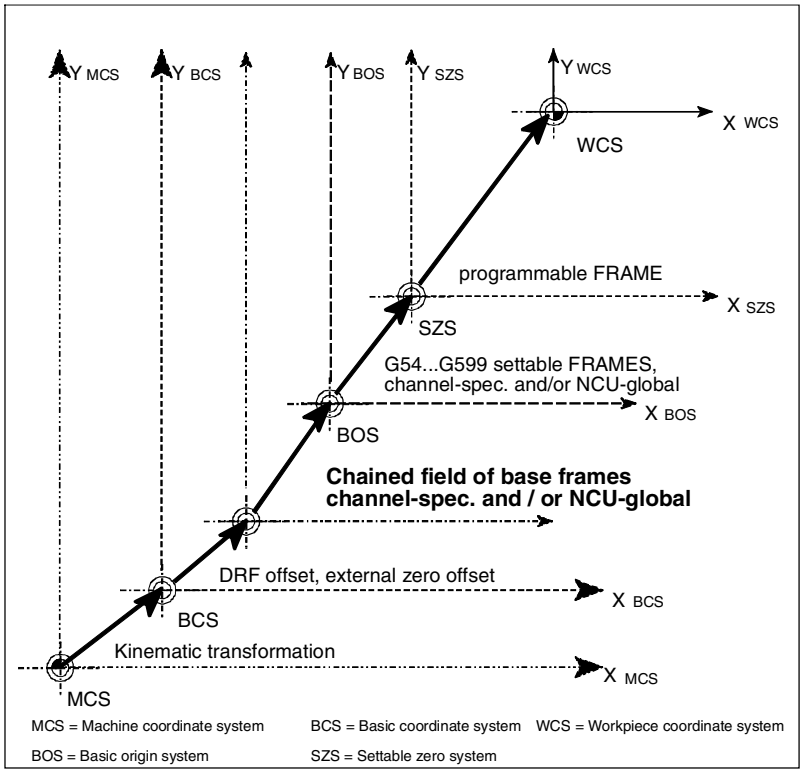
$P_ACTBFRAME

Total base frame

The variable $P_ACTBFRAME determines the chained total base frame. The variable can only be read.

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





840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming the total base frame

The user can select via system variables $P_CHBFRMASK and $P_NCBFRMASK which base frames are to be included in the calculation of the "Total" base frame. The variables can only be programmed in the program and read via OPI. The variable value is interpreted as a bit mask and specifies which base frame array element from $P_ACTBFRAME is to be included in the calculation.

You can specify with $P_CHBFRMASK which channel-specific base frames, and with $P_NCBFRMASK which NCU-global base frames, are to be included in the calculation.

By programming the variables the total base frame and the total frame are calculated again. After a Reset is performed, the basic setting value is

$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

The current settable frame that is valid in the channel can be written and read in the part program via the predefined frame variable $P_IFRAME. The written settable frame is immediately included in the calculation.

With NCU-global settable frames, the modified frame is only active in the channel in which the frame was programmed. If the frame is to be changed for all channels of an NCU, both $P_UIFR[n] and $P_IFRAME have to be programmed. The other channels must then still activate the respective frame with, for example, G54.

6.9 NCU-global frames (SW 5 and higher)

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

$P_PFRAME

Current programmable frame

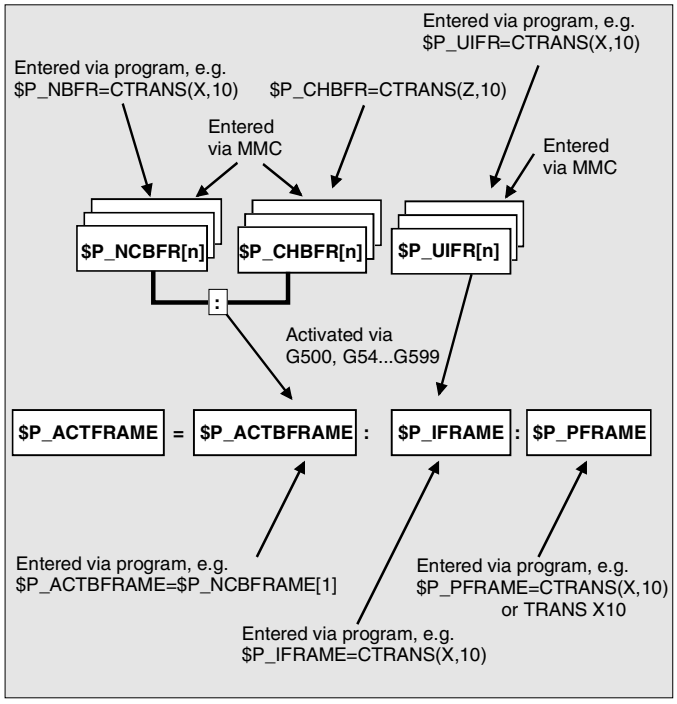
$P_PFRAME is the programmable frame resulting from the programming of TRANS/ATRANS, G58/G59, ROT/AROT, SCALE/ASCALE, MIRROR/AMIRROR or the assignment of CTRANS, CROT, CMIRROR, CSCALE to the programmable FRAME.

Current, programmable frame variable that establishes the reference between the settable zero system (SZS) and the workpiece coordinate system (WCS).

$P_ACTFRAME

The current resulting total frame $P_ACTFRAME results from the chaining of all base frames, the current settable frame and the programmable frame. The current frame is always updated if a frame element is modified.

$P_ACTFRAME corresponds to $P_ACTBFRAME : $P_IFRAME : $P_PFRAME



Transformations

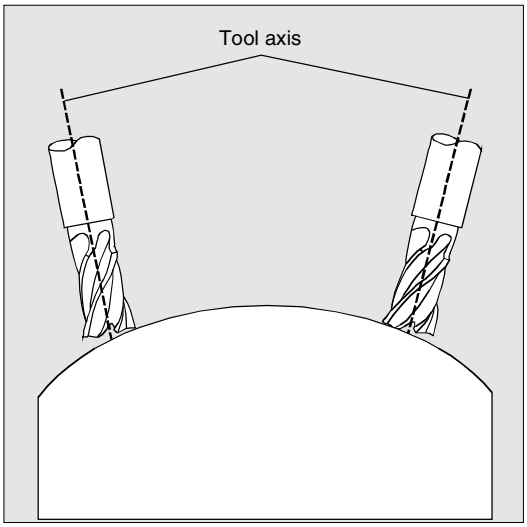
840D
NCU 572
NCU 573

840Di

7.1 Three, four and five-axes transformation: TRAORI

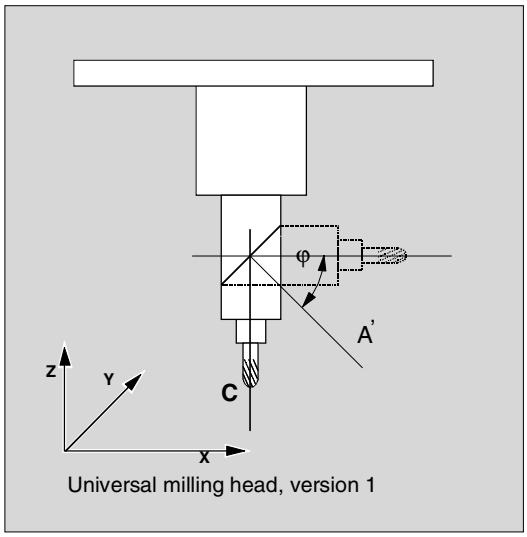
In order to achieve optimal cutting conditions in the machining of curved surfaces, the approach angle of the tool must be variable.

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



Universal milling head
Here three linear axes (X, Y, Z) and two orientation axes define the setting angle and machining point of the tool. One of the two orientation axes is applied as an inclined axis – in many cases, and in example A' – positioned at a 45° angle.

The axis sequence of the orientation axes and the direction of orientation of the tool are set via machine data as a function of the machine kinematics. In the examples on the right, the arrangements are illustrated by machine kinematics CA!



Universal milling head, version 1

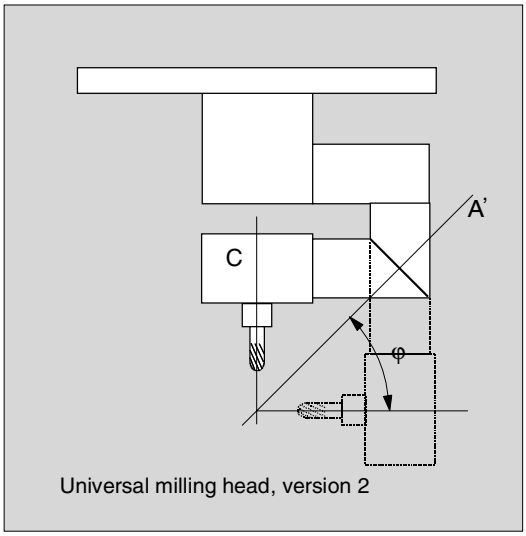
840D
NCU 572
NCU 573

840Di

The following interrelationships are possible:
A' is at angle φ to the X axis
B' is at angle φ to the Y axis
C' is at angle φ to the Z axis

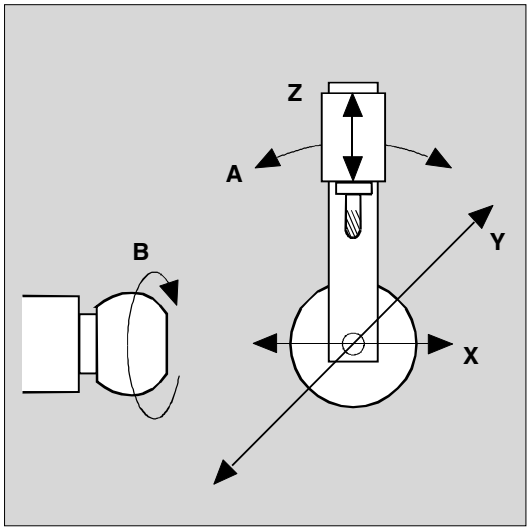
Angle φ can be configured in the 0° to +89° range via machine data.

Depending on the selected direction of tool orientation, the active machining plane (G17, G18, G19) must be set in the NC program such that the tool length compensation acts in the tool orientation direction.



Universal milling head, version 2

Transformation with linear swivel axis
This is an arrangement with a moving workpiece and a moving tool.
The kinematics comprise three linear axes (X, Y, Z) and two rotary axes at right angles. The first rotary axis is moved, for example, via the cross slide of two linear axes, the tool is positioned in parallel to the third linear axis.
The second rotary axis rotates the workpiece.
The third linear axis (swivel axis) is in the plane of the cross slide.



The axis sequence of the rotary axes and the direction of orientation of the tool are set via machine data as a function of the machine kinematics.

The following interrelationships are possible:

Axes:	Axis sequences:
1st rotary axis	A A B B C C
2nd rotary axis	B C A C A B
Linear swivel axis	Z Y Z X Y X

7.1 Three, four and five-axes transformation: TRAORI

840D
NCU 572
NCU 573

840Di

Three and four-axis transformations

The three-axis and four-axis transformations are special types of five-axis transformation.
The user can configure two or three linear axes and one rotary axis. The transformations operate on the assumption that the rotary axis is positioned orthogonally in the orientation plane.
The tool can only be orientated in the plane that is perpendicular to the rotary axis. The transformation supports machine types with moving tool and moving workpiece.

Three-axis and four-axis transformations are configured and programmed in the same way as five-axis transformations.

Programming

```
TRAORI(n)
TRAFOOF
```

Explanation of the commands

`TRAORI`	Activates the first selected orientation transformation
`TRAORI(n)`	Activates the orientation transformation assigned with n
`n`	Number of transformation (n = 1 or 2), TRAORI(1) corresponds to TRAORI
`TRAFOOF`	Deactivate transformation

Additional notes

When the transformation has been activated, the positional parameters (X, Y, Z) always refer to the tip of the tool.

Changes in the positions of the rotary axes participating in the transformation result in compensation movements on the other machine axes, such that the position of the tool tip remains the same.

840D
NCU 572
NCU 573

840Di

7.1.1 Programming the tool orientation

Five-axis programs are generally generated in CAD/CAM systems and not typed in on the control. The following description is therefore intended mainly for programmers of postprocessors.

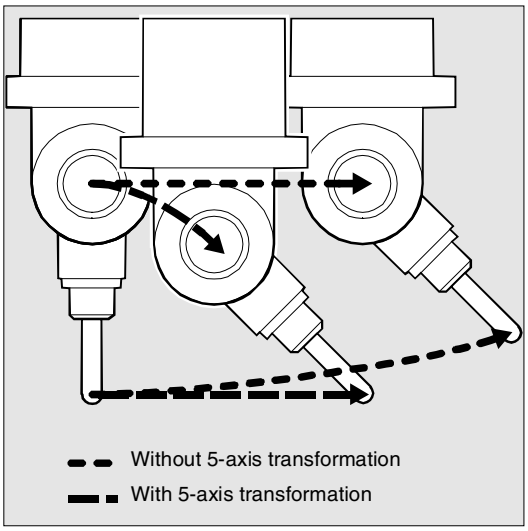
There are three ways of programming the orientation of the tool:

1. Program the movement of the rotary axes. The change in orientation always takes place in the basic or machine coordinate system. The orientation axes are traversed as synchronized axes.

2. Programming in Euler or in RPY angles with A2, B2, C2
   or
   programming the direction vector with A3, B3, C3. The direction vector points from the tool tip in the direction of the tool holder.

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

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



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

840D
NCU 572
NCU 573

840Di

Programming

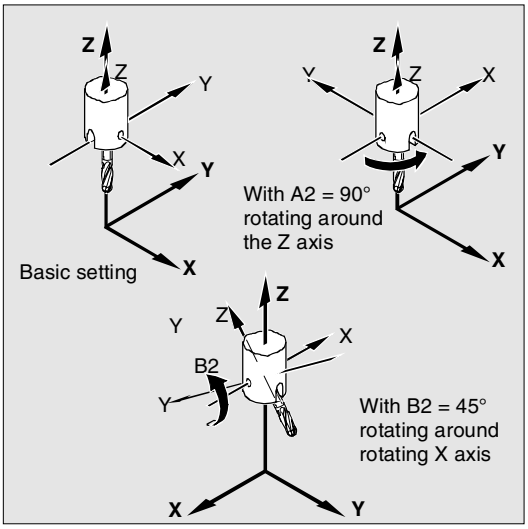
`G1 X Y Z A B C`	Programming of the movement of the rotary axes
`G1 X Y Z A2= B2= C2=`	Programming in Euler angles
`G1 X Y Z A3= B3= C3=`	Programming of the direction vector
`G1 X Y Z A4= B4= C4=`	Programming the surface normal vector at block start
`G1 X Y Z A5= B5= C5=`	Programming the surface normal vector at block end
`LEAD`	Lead angle for programming of the tool orientation
`TILT`	Side angle for programming of the tool orientation

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

Programming in Euler angles
The values programmed for the orientation with A2, B2, C2 are interpreted as Euler angles (in degrees).

The orientation vector is produced by rotating a vector in the Z direction first with A2 around the Z axis, then with B2 around the new X axis and finally with C2 around the new Z axis.

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



840D
NCU 572
NCU 573

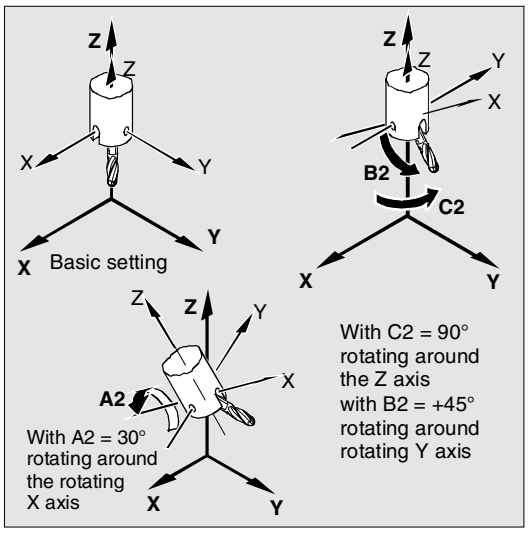
840Di

Programming in RPY angles

The values programmed for the orientation with A2, B2, C2 are interpreted as RPY angles (in degrees).

The orientation vector is produced by rotating a vector in the Z direction first with C2 around the Z axis, then with B2 around the new Y axis and finally with A2 around the new X axis.

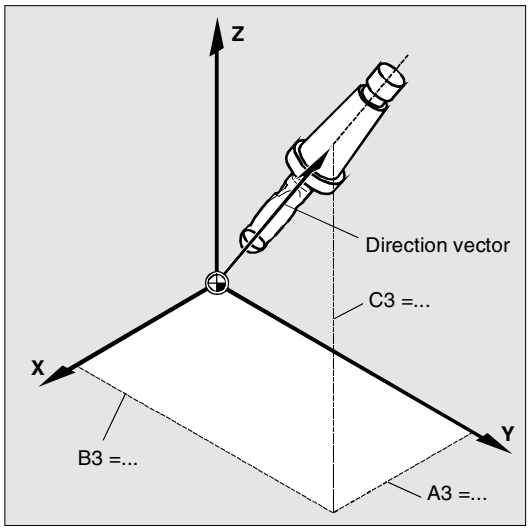
In contrast to Euler angle programming, all three values determine the orientation vector.



Programming the direction vector

The components of the direction vector are programmed with A3, B3, C3. The vector points in the direction of the tool fixture the length of the vector is irrelevant.

Unprogrammed vector components are set to zero.



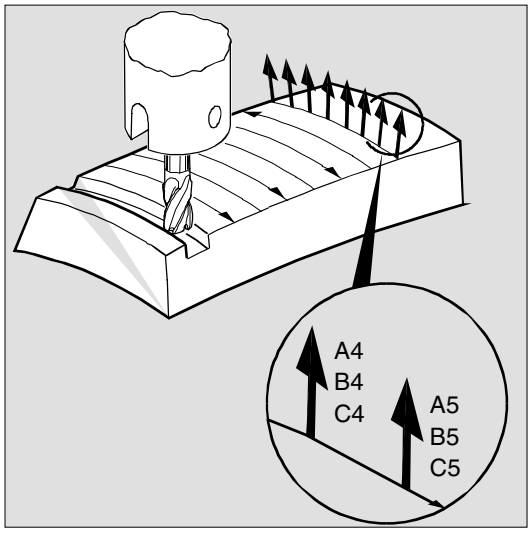
840D
NCU 572
NCU 573

840Di

Face milling
The face milling mode is used to machine surfaces with any degree of curvature.

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.
After they have been calculated, the NC blocks are read into the control via postprocessors.

Definition of surfaces
The path curvature is defined via surface normal vectors with the following components:
A4, B4, C4 Start vector at block beginning
A5, B5, C5 End vector at end of block



If a block only contains the start vector, the normal surface vector remains constant over the entire block.

If a block only contains the end vector, then large-circle interpolation is used to interpolate from the end value of the preceding block to the programmed end value.

If the start and end vectors are programmed, then large-circle interpolation is used to interpolate between the two directions, producing continuously smooth traversing paths. This means it is possible to create continuous smooth paths.

In the basic setting, the normal surface vectors point in the Z direction, regardless of active planes G17 to G19.

The length of a vector has no significance.

Unprogrammed vector components are set to zero.
When ORIWCS is active (see following pages), the normal surface vectors refer to the active frame and are rotated at the same time as the frame.

840D
NCU 572
NCU 573

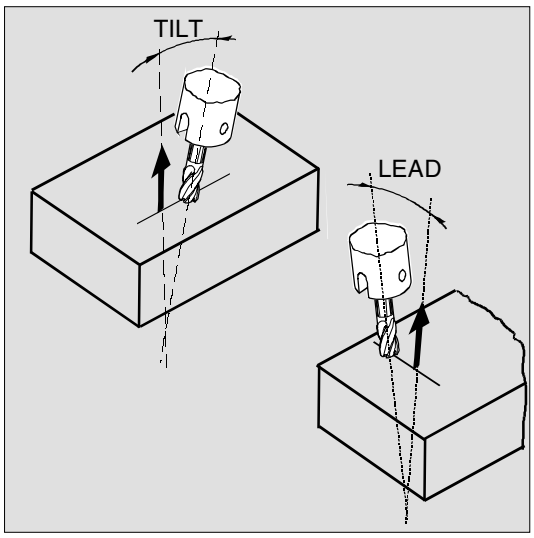
840Di

The surface normal vector must be perpendicular to the path tangent with a limit value set via machine data or else an alarm is output.

Programming the tool orientation: with LEAD and TILT

The resulting tool orientation is determined from:

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



Explanation of the commands

`LEAD`	Angle relative to the surface normal vector in the plane created from the path tangent and surface normal vector
`TILT`	Angle in the plane, perpendicular to the path tangent relative to the surface normal vector

Behavior at inside corners (with 3D tool offset)

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

840D
NCU 572
NCU 573

840Di

7.1.2 Orientation axes reference, ORIWCS, ORIMCS

Programming

```
N.. ORIMCS
```
or
```
N.. ORIWCS
```

Explanation of the commands

ORIMCS	Rotation in the machine coordinate system
ORIWCS	Rotation in the workpiece coordinate system

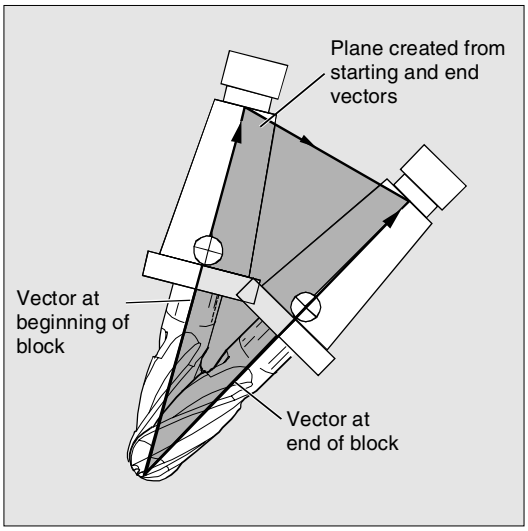
Function

When programming orientation in the workpiece coordinate system with Euler or RPY angles or orientation vector, the turning movement can be set via ORIMCS/ORIWCS.

Sequence

With ORIMCS, the tool movement depends on the machine kinematics. On a change in orientation where the tool tip is fixed in space, linear interpolation is performed between the rotary axis positions.

With ORIWCS, the tool movement is performed independently of the machine kinematics. On a change in orientation where the tool tip is fixed in space, the tool moves in the plane created from the starting and end vectors.



840D
NCU 572
NCU 573

840Di

Additional notes

ORIWCS is the default setting. If it is not clear from the outset on which machine a five-axis program is to run, ORIWCS should be selected. The movements that the machine actually performs depends on the machine kinematics.

With ORIMCS you can program actual machine movements, e.g. in order to avoid collisions with fixtures.

Which interpolation type is active is defined in machine data $MC_ORI_IPO_WITH_G_CODE: ORIMCS/ORIWCS or ORIMACHAX/ORIVIRTAX (see Section 7.1.4).

7.1.3 Singular positions and how they are handled

Notes on ORIWCS:

Orientation movements in the vicinity of the singular position of the five-axis machine call for large movements of the machine axes. (For example, for a rotation swivel head where C is the rotary axis and A is the swivel axis, all positions where A = 0 are singular).

To avoid overloading the machine, the velocity control considerably reduces the tool path velocity in the vicinity of the singular positions.

With machine data
`$MC_TRAFO5_NON_POLE_LIMIT`
`$MC_TRAFO5_POLE_LIMIT`
transformation can be parameterized in such a way that orientation movements in the vicinity of the pole run through the pole and thus speed up machining.

Notes on SW 5.2:

From SW 5.2 and higher, singular positions are treated only with MD `$MC_TRAFO5_POLE_LIMIT` (see Description of Functions, Part 3, Section 2.8.4).

840D
NCU 572
NCU 573

840Di

7.1.4 Orientation axes (SW 5.2 and higher)

Programming

```
N.. ORIEULER or ORIRPY
or
N.. ORIVIRT1 or ORIVIRT2
N.. G1 X Y Z A2= B2= C2=
```

Explanation of the commands

`ORIEULER`	Orientation programming using Euler angles
`ORIRPY`	Orientation programming using RPY angles
`ORIVIRT1`	Orientation programming using virtual orientation angles (Definition 1), definition according to MD $MC_ORIAX_TURN_TAB_1
`ORIVIRT2`	Orientation programming using virtual orientation angles (Definition 2), definition according to MD $MC_ORIAX_TURN_TAB_2
`G1 X Y Z A2= B2= C2=`	Angle programming of virtual axes

Programming

```
N.. ORIAXES or ORIVECT
N.. G1 X Y Z A B C
```

Explanation of the commands

`ORIAXES`	Linear interpolation of orientation axes
`ORIVECT`	Large-circle interpolation
`ORIMCS`	Rotation in the machine coordinate system See description in Section 7.1.2
`ORIWCS`	Rotation in the workpiece coordinate system See description in Section 7.1.2
`G1 X Y Z A B C`	Programming the machine position

Function

The orientation axes function describes the orientation of the tool in the area. This provides a third degree of freedom that describes the rotation around itself, which is necessary for six-axes transformations.

840D
NCU 572
NCU 573

840Di

MD $MC_ORI_DEF_WITH_G_CODE specifies how the programmed angles A2, B2 and C2 are defined:
Definition according to MD $MC_ORIENTATION_IS_EULER (standard) or
definition according to G_group 50
(ORIEULER, ORIRPY, ORIVIRT1, ORIVIRT2).

MD $MC_ORI_IPO_WITH_G_CODE defines which interpolation type is active:
ORIWCS/ORIMCS or ORIAXES/ORIVECT.

JOG mode

In this mode, the interpolation of the orientation angles is always linear. During continuous and incremental traversing via traversing keys, only one orientation axis can be traversed. Using the handwheels both orientation axes can be traversed at the same time.

For manual traversal of orientation axes, the channel-specific feedrate override switch or the rapid traverse override switch are active with rapid traverse override.

A separate velocity can be specified with the following machine data:
$MC_JOG_VELO_RAPID_GEO
$MC_JOG_VELO_GEO
$MC_JOG_VELO_RAPID_ORI
$MC_JOG_VELO_ORI

840D
NCU 572
NCU 573

840Di

Programming the feed

FORI1	Feed for swiveling the orientation vector on the large circle
FORI2	Feed for the overlaid rotation around the swiveled orientation vector

With orientation movements the programmed feed corresponds to an angle velocity [degrees/min].

Effectiveness of feeds over G code:

When programming ORIAXES the feed for an orientation axis can be limited by means of the FL[ ] instruction (feed limit).

When programming ORIVECT the feed must be programmed with FORI1 or FORI2. FORI1 and FORI2 may only be programmed once in the NC block. When programmed in this manner, the path is always traversed in the shortest way possible. With overlaid turning and swiveling movements, the smallest feedrate is always the one used. With orientation movements the programmed feed corresponds to an angle velocity [degrees/min].

If geometry axes and orientation axes are traversing a path together, the traversing movement is determined from the smallest feedrate.

840D
NCU 572
NCU 573

840Di

7.1.5 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 The movement is executed as a synchronized movement; the slowest axis participating in the movement is the dominating axis for the velocity.
CP	**c**ontinuous **p**ath (path movement) The movement is executed as Cartesian path movement
STAT=	Position of articulations; value dependent on the transformation.
TU=	TURN information This enables determinate approach of axis angles between –360 degrees and +360 degrees.

Function

This function allows a position to be programmed in a Cartesian coordinate system; the machine motion, however, takes place in machine coordinates.
The function can be used, for example, when changing the articulation position, if in doing so the movement passes through a singularity.

Note:
The function is only meaningful in conjunction with an active transformation. Further, "PTP travel" is only permissible together with G0 and G1.

Sequence

The switchover between the Cartesian traversal and traversal of the machine axes is carried out via the modal commands PTP and CP. The commands are modal. CP is set by default.

840D
NCU 572
NCU 573

840Di

Programming the position (STAT=)
A machine position cannot be uniquely defined solely by specifying the position with Cartesian coordinates and the tool orientation. According to which kinematics are used, there are up to 8 different articulation positions. They are therefore transformation-specific. In order to unambiguously convert a Cartesian position into an axis angle, you need to specify the position of the articulations by means of the STAT= command. The "STAT" command contains one bit as binary value for each of the possible positions.

References:
The following documentation provides a detailed description of the different transformations:
SINUMERIK 840D/FM-NC Description of Functions (Part 3), "Transformation Package Handling".

For a description of the position bits that need to be programmed for "STAT", please refer to:
SINUMERIK 840D/FM-NC Description of Functions (Part 3), "Three-Axis to five-Axis Transformation".

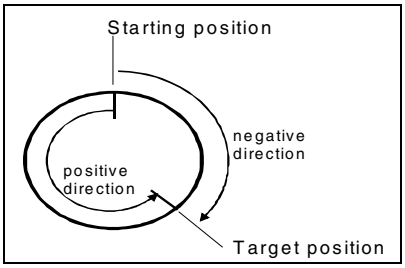
Programming the axes angles (TU=)
For a determinate approach of the axes angles < ±360 degrees, this information must be programmed with the "TU=" command.
The command is non-modal.

The axes traverse along the shortest path:
- if TU was not programmed with a position
- with axes that have a traversing range > ±360 degrees

840D
NCU 572
NCU 573

840Di

Example:
The target position displayed in the diagram can be approached in positive or negative direction. The direction is programmed under address A1.
A1=225°, TU=bit 0, → positive direction
A1=–135°, TU=bit 1, → negative direction



Corner rounding between the CP and PTP movements

Programmable corner rounding between the blocks is possible with G641.
The size of the rounding area is the path dimension in mm or Inch from which or to which the block transition is rounded. The size must be specified as follows:

- with ADISPOS for G0 blocks
- with ADIS for all other motion commands

The path distance calculation corresponds to the consideration of F addresses in non-G0 blocks. The feed is maintained on 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.

840D
NCU 572
NCU 573

840Di

Additional notes

Mode change

The function "Cartesian PTP travel" is only meaningful in AUTO and MDA mode. The current setting is retained when the mode is changed to JOG.
The axes are traversed in the MCS if the PTP G code is set. If the CP G code is set, the axes are traversed in the WCS.

Power On/reset

After Power ON or reset, the settings are according to the machine data $MC_GCODE_RESET_VALUES[48].
Traversing mode "CP" is set by default.

Repos

If the function "Cartesian PTP travel" was set during the interruption block, repositioning also takes place with PTP.

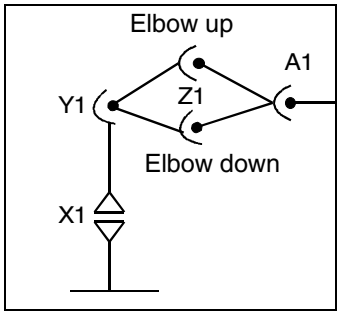
Overlaid movement

DRF offsets or external zero offsets are possible only with certain restrictions in conjunction with Cartesian PTP motion. Overrides must not exist in the BCS on a change from PTP to CP motion.

840D
NCU 572
NCU 573

840Di

Programming example



N10	G0 X0 Y-30 Z60 A-30 F10000	Starting position → 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	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

7.2 Milling machining on turned parts: TRANSMIT

Programming

`TRANSMIT` or `TRANSMIT(n)`
`TRAFOOF`

Explanation of the commands

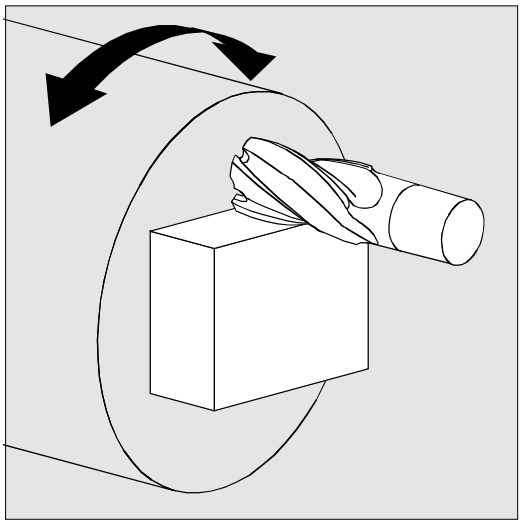
`TRANSMIT`	Activates the first declared TRANSMIT function
`TRANSMIT(n)`	Activates the nth declared TRANSMIT function; n may not be greater than 2 (TRANSMIT (1) corresponds to TRANSMIT).
`TRAFOOF`	Deactivates an active transformation

An active TRANSMIT transformation TRANSMIT transformation is also deactivated if one of the other transformations is activated in the respective channel (e.g. TRACYL, TRAANG, TRAORI).

The TRANSMIT function provides the following capabilities:

- End face machining of rotating parts in the rotating clamp (drill holes, contours).
- A Cartesian coordinate system can be used to program these machining operations.
- The control transforms the programmed traversing movements of the Cartesian coordinate system into traversing movements on the real machine axes (standard setting):
  - Rotary axis
  - Infeed axis perpendicular to the rotary axis.
  - Longitudinal axis parallel to the rotary axis.

  The linear axes are positioned perpendicular to each other.
- Tool center offset relative to the turning center is permitted.
- The speed control provides defined limits for the rotary movements.



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Rotary axis

The rotary axis cannot be programmed because it is assigned to a geometry axis and cannot therefore be programmed directly as a channel axis.

Pole

SW 3.x and lower

Traversing through the pole (origin of the Cartesian coordinate system) is inhibited. A movement leading through the pole stops at the pole and an alarm is output. With a cutter center offset, the movement stops at the edge of the unapproachable area.

SW 4 and higher

There are two ways to traverse the pole:

1. Traversing the linear axis only
2. Traversing in the pole with rotation of the rotary axis in the pole and travel from the pole

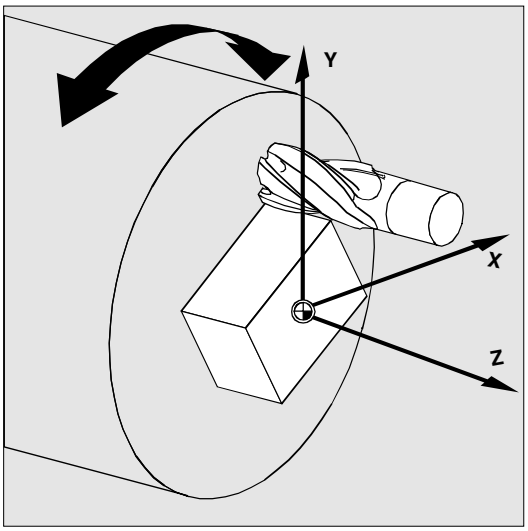
The setting is carried out in MD 24911 and 24951.

References

/FB/ M1 Kinematic Transformations

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming example



N10 T1 D1 G54 G17 G90 F5000 G94	Tool selection
N20 G0 X20 Z10 SPOS=45	Approach reference point
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	Rough-machine square; 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	Finish-machine square
N190 Z20 G40 N200 TRANS N210 TRAFOOF	Deselect frame
N220 G0 X20 Z10 SPOS=45	Approach reference point
N230 M30	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

7.3 Cylinder surface transformation: TRACYL

Programming

`TRACYL(d)` or `TRACYL(d,t)`
`TRAFOOF`

Explanation of the commands

`TRACYL(d)`	Activates the first declared TRACYL function
`TRACYL(d,n)`	Activates the nth declared TRACYL function n may not be greater than 2, TRACYL(d,1) corresponds to TRACYL(d).
`d`	Value for the current diameter of the cylinder to be machined.
`TRAFOOF`	Transformation off
`OFFN`	Contour offset – normal: Distance of the side of the groove from the programmed reference contour

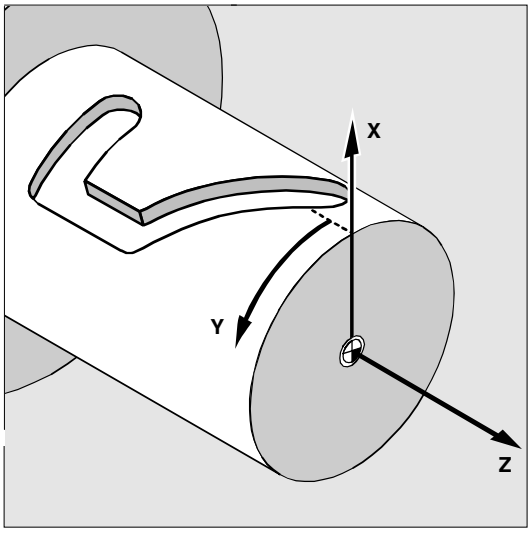
An active TRACYL transformation TRACYL transformation is also deactivated if one of the other transformations is activated on the same channel (e.g. TRANSMIT, TRAANG, TRAORI).

Function

Cylinder surface curve transformation TRACYL
The cylinder surface curve transformation TRACYL provides the following functions:

Machining of
- Longitudinal grooves on cylindrical bodies,
- Transverse grooves on cylindrical bodies,
- Any other groove shapes on cylindrical bodies.

The shape of the grooves is programmed with reference to the processed level cylinder surface area.



Workpiece coordinate system

7.3 Cylinder surface transformation: TRACYL

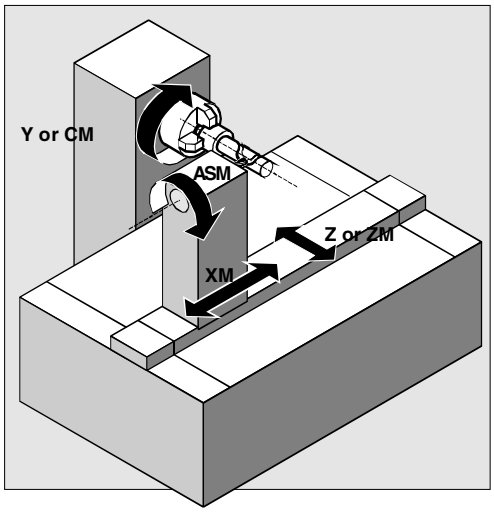
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

There are two types of cylinder surface coordinate transformation:

- without groove side compensation
- with groove side compensation

Without groove side compensation:
The control transforms the programmed traversing movements of the cylinder coordinate system into the traversing movements of the real machine axes:
– Rotary axis
– Infeed axis perpendicular to the rotary axis
– Longitudinal axis parallel to the rotary axis

The linear axes are positioned perpendicular to each other. The infeed axis intersects the rotary axis.

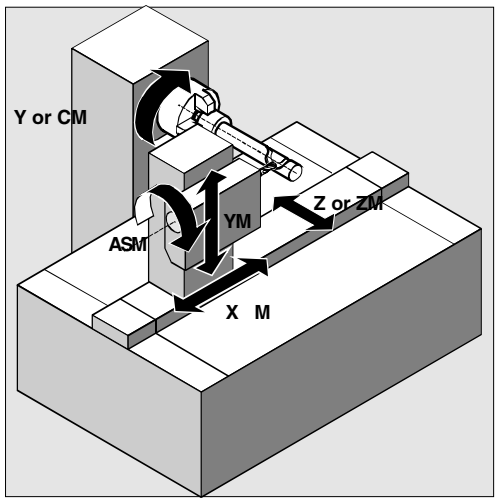


Machine coordinate system

With groove side compensation:
Same kinematics as above, plus:
– Longitudinal axis parallel to the direction of the circumference

The linear axes are positioned perpendicular to each other.

The speed control provides defined limits for the rotary movements.



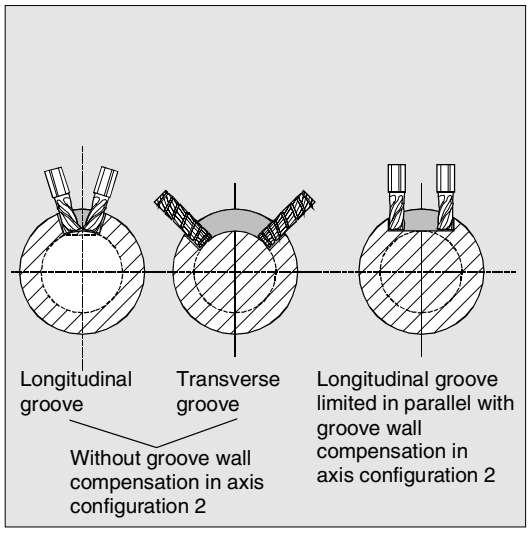
Machine coordinate system

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Groove cross section

With axis configuration 1, grooves along the rotary axis are only limited in parallel if the groove width matches the tool radius.

Grooves parallel to the circumference (transverse grooves) are not parallel at the start and end.



Offset contour normal OFFN

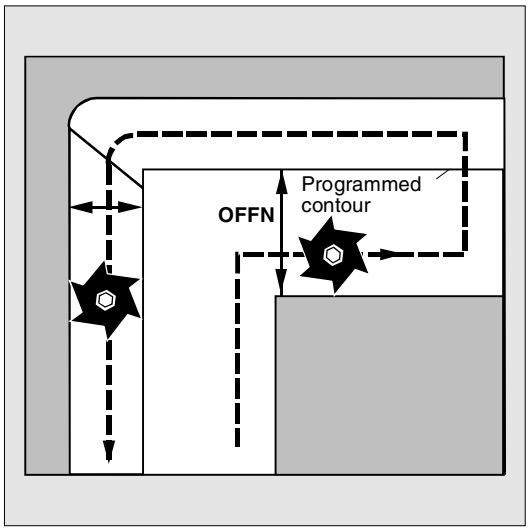
For milling grooves with TRACYL, the groove

- center line is programmed in the part
- program, the groove width via OFFN.

OFFN only becomes active when tool radius compensation
is selected, to protect the side of the groove from being damaged. Further, OFFN>=tool radius is advisable to exclude any possible damage to the opposite side of the groove.

A part program for milling a groove usually consists of the following steps:

1. Select tool
2. Select TRACYL
3. Select suitable coordinate offset (FRAME)
4. Positioning
5. Programming OFFN
6. Select TRC
7. Approach block (enter TRC and approach side of the groove)
8. Contour of groove center line
9. Deselect TRC
10. Retract block (exit TRC and retract from side of groove)
11. Positioning
12. TRAFOOF
13. Select original coordinate offset (FRAME) again



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Special cases:

- TRC selection:
  TRC is not programmed according to the TRC but relative to the programmed groove center line. G42 is programmed so that the tool travels to the left of the groove side (instead of G41). You do not have to do this if the groove width is specified with a negative leading sign in OFFN.
- OFFN with TRACYL has a different effect from without TRACYL. As OFFN is also included in the calculation without TRACYL when TRC is active, OFFN should be set back to zero after TRAFOOF.
- It is possible to change OFFN within the part program. This means that the groove center line could be moved from the center (see Figure).
- Control grooves:
  With TRACYL the same groove is not generated for control grooves (as if it were manufactured using a tool whose diameter is the same as the groove width).
  In principle, it is not possible to generate the same groove side geometry with a smaller cylindrical tool as with a larger one.
  TRACYL minimizes errors. In order to prevent inaccuracies from occuring, the tool radius should only be slightly smaller than half of the groove width.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

With cylinder surface curve transformation with groove side compensation, the axis used for the compensation should be set to zero (y = 0), in order that machining of the groove is aligned with the programmed groove center line. Cylinder surface curve transformation

Rotary axis

The rotary axis cannot be programmed because it is assigned to a geometry axis and cannot therefore be programmed directly as a channel axis.

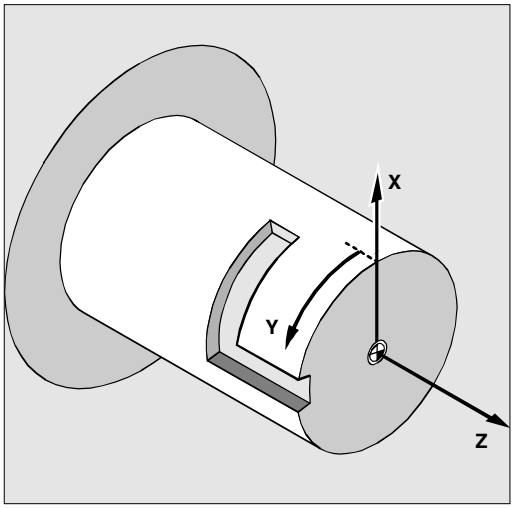
Axis use

The following axes cannot be used as positioning axes or reciprocating axes:

- The geometry axis in the circumferential direction of the cylinder surface curve (Y axis)
- The additional linear axis used in groove side compensation (Z axis)

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming example



N10 T1 D1 G54 G90 F5000 G94	Tool selection, clamping compensation
N20 SPOS=0 N30 G0 X25 Y0 Z105 CC=200	Approach reference point
N40 TRACYL (40)	Switch on cylinder surface curve transformation
N50 G19	Select plane

Producing a hook-shaped groove:

N60 G1 X20	Infeed tool to base of groove
N70 OFFN=12	Set groove side distance 12 mm relative to the groove center line
N80 G1 Z100 G42	Approach the right groove side
N90 G1 Z50	Groove section parallel to cylinder axis
N100 G1 Y10	Groove section parallel to circumference
N110 OFFN=4 G42	Approach left groove side; set groove side distance 4 mm from to the groove center line
N120 G1 Y70	Groove section parallel to circumference
N130 G1 Z100	Groove section parallel to cylinder axis
N140 G1 Z105 G40	Retract from the side of the groove
N150 G1 X25	Retract
N160 TRAFOOF	
N170 G0 X25 Y0 Z105 CC=200	Approach reference point
N180 M30	

840D
NCU 572
NCU 573

810D 840Di

7.4 Inclined axis: TRAANG

Programming

`TRAANG(α)` or `TRAANG(α,n)`
`TRAFOOF`

Explanation of the commands

`TRAANG(α)`	Activates the first declared inclined transformation axis
`TRAANG(α,n)`	Activates the nth declared inclined transformation axis. n may not exceed 2. TRAANG(α,1) corresponds to TRAANG(α).
α	Angle of the inclined axis
`TRAFOOF`	Transformation off

If α (angle) is omitted or a zero entered, the transformation is activated with the parameter settings of the previous selection. On the first selection, the defaults in the machine data are used.

An active TRAANG transformation is also deactivated if one of the other transformations (e.g. TRACYL, TRANSMIT, TRAORI) is activated on the same channel.

7.4 Inclined axis: TRAANG

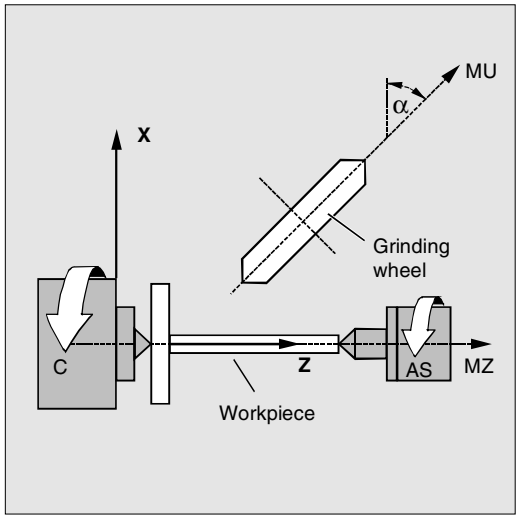
840 D
NCU 572
NCU 573

810D 840Di

Function

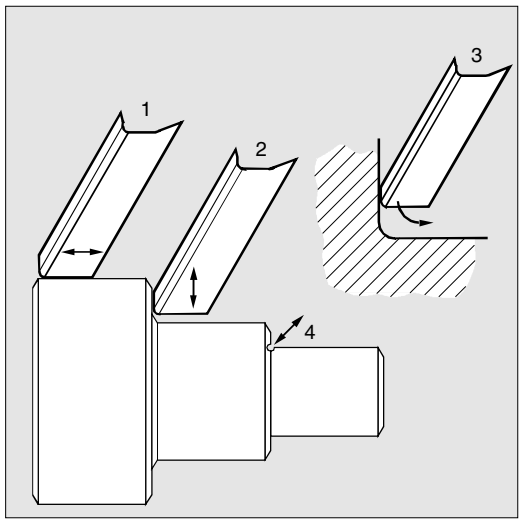
The inclined axis function is intended for grinding and provides the following capabilities:

- Machining with inclined infeed axis
- A Cartesian coordinate system can be used for programming.
- The control transforms the programmed traversing movements of the Cartesian coordinate system into traversing movements on the real machine axes (standard setting): inclined infeed axis.



The following machining operations are possible:

1. Longitudinal grinding
2. Transverse grinding
3. Grinding of a specific contour
4. Inclined recess grinding



840D
NCU 572
NCU 573

810D 840Di

The following settings are defined in machine data:

- The angle between a machine axis and the inclined axis.
- The position of the tool zero with reference to the origin of the coordinate system declared in the "inclined axis" function.
- The velocity reserve available on the parallel axis for the compensation movement.
- The axis acceleration reserve available on the parallel axis for the compensation movement.

Axis configuration

In order to program in the Cartesian coordinate system, the relationship between this coordinate system and the real machine axes (MU, MC) must be declared in the control:

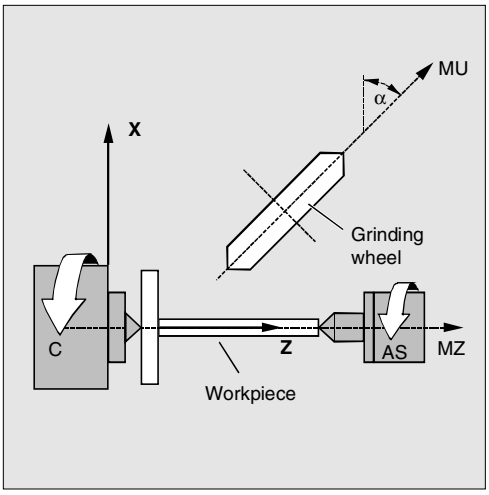
- Names of the geometry axes
- Assignment of the geometry axes to the channel axes
  – General case (inclined axis not active)
  – Inclined axis active
- Assignment of the channel axes to the machine axis numbers
- Identification of the spindles
- Assignment of machine axis names

The procedure is the same as for the normal axis configuration with the exception of "Inclined axis active".

840 D
NCU 572
NCU 573

810D 840Di

Programming example



`N10 G0 G90 Z0 MU=10 G54 F5000 ->` `-> G18 G64 T1 D1`	Select tool, clamping compensation, select plane
`N20 TRAANG(45)`	Activate inclined transformation axis
`N30 G0 Z10 X5`	Approach reference point
`N40 WAITP(Z)`	Enable axes for oscillation

`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`
	Oscillation performed to dimension (see Chapter 9 for oscillation)

`N90 WAITP(Z)`	Enable oscillation axes as positioning axes
`N100 TRAFOOF`	Switch off transformation
`N110 G0 Z10 MU=10`	Retract
`N120 M30`	

`->` program in one block

840D
NCU 572
NCU 573

810D 840Di

7.5 Supplementary conditions when selecting a transformation

Selection of transformations can be performed from the part program or with MDA. Please note:

- An intermediate motion block is not inserted (chamfers/radii).
- A spline block sequence must be completed, otherwise a message is output.
- Tool fine offset must be deactivated (FTOCOF), otherwise a message is output.
- Tool radius compensation must be deactivated (G40), otherwise a message is output.
- The control includes an activated tool length compensation in the transformation.
- The current frame that was active before the transformation is deactivated by the control.
- An active working area limitation is deselected for the axes involved in the transformation by the control (corresponds to WALIMOF).
- Protection zone monitoring is deselected.
- Continuous-path mode and approximate positioning are interrupted.
- DRF offsets of axes involved in the transformation may not change between execution of the preprocessing and main run routines (Software Version 3 and lower).
- All axes specified in the machine data must be block-synchronized.
- Exchanged axes are changed back again, otherwise a message is output.
- A message is output with dependent axes.

Tool change

A tool change can only be performed if tool radius compensation is deselected.
A change in the tool length compensation and selection/deselection of tool radius compensation must not be programmed in the same block.

7.5 Supplementary conditions when selecting a transformation

840 D
NCU 572
NCU 573

810D 840Di

Frame change

Any instructions that only refer to the basic coordinate system are allowed (FRAME, tool radius compensation). A frame change with G91 (incremental dimension) is not dealt with separately as is the case with inactive transformation. The increment to be traversed is calculated in the workpiece coordinate system of the new frame – irrespective **of the frame** active in the previous block.

Exceptions

Axes involved in transformation cannot be used

- as a preset axis (alarm)
- for approaching the fixed point (alarm)
- for referencing (alarm)

840D
NCU 572
NCU 573

810D 840Di

7.6 Deactivate transformation: TRAFOOF

Programming

```
TRAFOOF
```

Explanation of the commands

`TRAFOOF`	Deactivates all active transformations/frames

Function

When the TRAFOOF command is issued, all active transformations and frames are deactivated again.

Frames that are required after this must be programmed again to be activated.

Please note:
The same restrictions apply for deselection of a transformation as for its activation (see preceding Section "Supplementary conditions for selection of a transformation")

840 D
NCU 572
NCU 573

810D 840Di

7.7 Chained transformations

SW 5 and higher supports **two** transformations one after the other, such that the motion elements for the axes from the first transformation are input data for the chained second transformation. The motion elements from the second transformation are effective for the machine axes.

- With SW 5**,** the chain can contain **two** transformations.
- The **second** transformation must be "**Inclined axis**" (TRAANG).
- The first transformation can be any of the following:
  – Orientation transformations (TRAORI), incl. universal milling head
  – TRANSMIT
  – TRACYL
  – TRAANG

Applications

– Grinding contours that were programmed as side line of a cylinder operation (TRACYL) using an inclined grinding wheel e.g. tool grinding.

– Finish cutting of a contour which is not round and which was generated with TRANSMIT using an inclined grinding wheel.

Prerequisite for using the activation command for a chained transformation is that the individual transformations to be chained to one another and the chained transformation to be activated are defined by means of machine data.
The supplementary conditions and special cases specified in the individual descriptions for the transformations also apply to a chained transformation.

840D
NCU 572
NCU 573

810D 840Di

Additional notes

Information about how to configure the machine data for transformations can be found in the Description of Functions documentation: M1 and F2.

Machine manufacturer (MH7.1)

Please read the machine manufacturer's specifications regarding any transformations predefined by machine data.

Transformations and chained transformations are options. The Catalog provides information about the availability of specific transformations within a chain in the various controls.

The following commands are for chained transformations:
TRACON for activation and
TRAFOOF for deactivation.

Activation

Programming

`TRACON(trf, par)`	A chained transformation is activated.

Explanation of the parameters

`trf`	Number of the chained transformation: 0 or 1 for the first/only chained transformation. If nothing is programmed here, the value is the same as if 0 or 1 were programmed, that is, the first/only transformation is activated. 2 for the second chained transformation. (With values unequal to 0 – 2, an alarm is issued).

840 D
NCU 572
NCU 573

810D 840Di

`par`	One or several parameters separated by commas for the transformation in the chain, the parameters require input of, for example, the angle of the inclined axis. If the parameters are not set, the default settings or the parameters that were last used are effective. Commas must be inserted to ensure that the specified parameters are evaluated in the sequence in which they are required if default settings are to be effective for preceding parameters. In particular, when specifying at least one parameter, it must be preceded by a comma, even if it is not necessary to specify trf, for example TRACON(, 3.7).

Function

The chained transformation is activated. Any other previously activated transformation is implicitly deactivated with TRACON().

A tool is always assigned to the first transformation of a chain. The following transformation then behaves as if the active tool length were zero. Only the base lengths of a tool (_BASE_TOOL_) set in the machine data are valid for the **first** transformation of the chain.

Deactivation

Programming

```
TRAFOOF
```

Function

The command deactivates the (chained) transformation that was last activated.

840D
NCU 572
NCU 573

810D 840Di

7.8 Switchable geometry axes, GEOAX

Programming

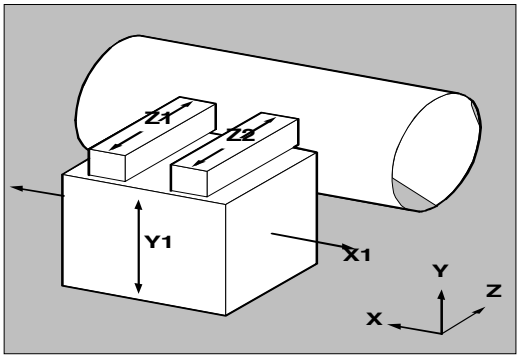
```
GEOAX(n,channel axis,n,channel axis,…)

GEOAX()
```

Explanation of the parameters

`GEOAX(n,channel axis,n,channel axis,…)`	Switches over the geometry axes.
`GEOAX()`	Calls the basic geometry axis configuration
`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 group without replacing it.
`Channel axis`	Name of the channel axis to be included in the gantry axis grouping.

Function

With the function "Switchable geometry axes" the geometry axis group configured in the machine data can be changed from the part program. A channel axis defined as a synchronized auxiliary axis can replace any geometry axis.



Example:
A tool slide can be traversed via channel axes X1, Y1, Z1, Z2. Axes Z1 and Z2 are to be used alternately as geometry axis Z in the part program. GEOAX programmed in the part program switches between the axes.
Following activation, the connection
X1, Y1, Z1 is effective (can be defined in machine data).

`N100 GEOAX (3,Z2)`	Channel axis Z2 operates as the Z axis.
`N110 G1 .....`	
`N120 GEOAX (3,Z1)`	Channel axis Z1 operates as the Z axis.

840 D
NCU 572
NCU 573

810D 840Di

Sequence

The geometry axis number
In the command GEOAX(n,channel axis...) number n denotes the geometry axis to which the channel axis subsequently specified is to be assigned.
Geometry axis numbers 1 to 3 (X, Y and Z axis) can be used to switch to a channel axis.
n = 0 removes an assigned channel axis from the geometry axis grouping without reassigning the geometry axis.

An axis replaced in the geometry axis group as a result of the switchover can be programmed as an auxiliary axis after the switchover via its channel name.

All frames, protection zones and working area limitations are cleared when a geometry axis is switched.
Polar coordinates:
Replacement of the geometry axes using the GEOAX command sets the modal polar coordinates to 0 analogous to changing the plane (G17–G19).
DRF, NPV:
Any handwheel offset (DRF) or an external zero offset remain active after the switchover.

Transferring axes positions
By allocating new axis numbers to channel axes already assigned it is also possible to program a position change within a geometry axis group.

N... GEOAX (1, XX, 2, YY, 3, ZZ)

Channel axis XX is the first geometry axis, YY the second and ZZ the third.

N... GEOAX (1, U, 2, V, 3, W)

Channel axis U is the first geometry axis, V the second and W the third.

840D
NCU 572
NCU 573

810D 840Di

Preconditions and limitations

1. Geometry axis switchover is not possible with:
   – active transformation,
   – active spline interpolation,
   – active tool radius compensation, (Programming Guide, Fundamentals: Chapter 8)
   – active tool fine compensation. (Programming Guide, Fundamentals: Chapter (8)
2. If a geometry axis and channel axis have the same name, the geometry axis cannot be switched.
3. None of the axes involved in the switchover must also be involved in an action that could continue beyond the block limit, as is possible with positioning axes of type A or following axes.
4. The GEOAX command can only be used to replace geometry axes with the ones that already existed on activation (i.e. it cannot be used to define new axes).
5. Replacement using the GEOAX command during preparation of the **contour table** (CONTPRON, CONTDCON) produces an alarm.

Deactivating the switchover
The command GEOAX() calls the basic configuration of the geometry axis group.

The basic configuration automatically becomes active after POWER ON and when switching over to the operating mode reference point approach.

Additional notes

Switchover procedure and tool length compensation
An active tool length compensation remains active after the switchover. It will affect any newly assigned axes or geometry axes whose position has been switched. When the first travel command is given for these geometry axes, the resulting path to be traversed is therefore the sum of the tool length compensation and the programmed travel path.

840 D
NCU 572
NCU 573

810D 840Di

Geometry axes that maintain their position in the axis group after a switchover also maintain their status as regards the tool length compensation.

Geometry axis configuration and transformation change

The geometry axis configuration that applies to an active transformation (defined in the machine data) cannot be changed with the function "Switchable geometry axes".

If you need to change the geometry axis configuration with transformation, this can only be done by programming another transformation.

A geometry axis configuration changed with GEOAX is cleared by activating a transformation.

If the machine data settings for the transformation and for the geometry axis switchover conflict, the settings for the transformation take priority.

Example:

A transformation is active. According to the settings in the machine data the transformation is to remain active after a RESET, at the same time, however, the RESET is to reestablish the basic configuration of the geometry axes. In this case the geometry axis configuration defined with the transformation is maintained.

840D
NCU 572
NCU 573

810D 840Di

Programming example

A machine has six channel axes called XX, YY, ZZ, U, V, W. The basic setting of the geometry axis configuration in 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()	Basic configuration of the geometry axes is active.
N20	G0 X0 Y0 Z0 U0 V0 W0	All axes to position 0 in rapid traverse.
N30	GEOAX(1,U,2,V,3,W)	Channel axis U is the first (X), V is the second (Y), W is the third geometry axis (Z).
N40	GEOAX(1,XX,3,ZZ)	Channel axis XX is the first (X), ZZ is the third geometry axis (Z). Channel axis V remains the second geometry axis (Y).
N50	G17 G2 X20 I10 F1000	Full circle in the X, Y plane. Channel axes XX and Y 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 setting
N100	GEOAX(1,U,2,V,3,W)	Channel axis U is the first (X), V is the second (Y), W is the third geometry axis (Z).
N110	G1 X10 Y10 Z10 XX=25	Channel axes U, V, W each travel to position 10, XX as the auxiliary axis travels to position 25.
N120	GEOAX(0,V)	V is removed from the geometry axes group. 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 remains the first (X), V becomes the second (Y) and W becomes the third geometry axis (Z).
N140	GEOAX(3,V)	V becomes the third geometry axis (Z), W is overwritten and therefore removed from the geometry axes group. The second geometry axis (Y) is still unassigned.

840 D
NCU 572
NCU 573

810D 840Di

Notes

Tool Offsets

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 generated with the associated tool offset blocks without being assigned to tools.
You can still program in the part program using T. However, this T does not relate 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Tool parameters number (DP)	Meaning	Comment
$TC_DP 1	Tool type	For overview see list
$TC_DP 2	Tool point direction	For turning tools only
Geometry	**Tool length compensation**	
$TC_DP 3	Length 1	Calculation depending
$TC_DP 4	Length 2	on type 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	Overhang 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	Overhang k	For slotting saw only
$TC_DP 20	Angle for cone milling tools	
Base dimensions/ adapter	**Tool length compensation**	
$TC_DP 21	Length 1	
$TC_DP 22	Length 2	
$TC_DP 23	Length 3	
Technology		
$TC_DP 24	Clearance angle	For turning tools

Additional notes

All other parameters are reserved.

Machine manufacturer

User cutting edge data can be configured via MD.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

8.2 Language commands for tool management

Explanation of the commands

`T="WZ"`	Select tool with name
`NEWT("WZ",DUPLO_NO)`	Create new tool, duplo number optional
`DELT("WZ",DUPLO_NO)`	Delete tool, duplo number optional
`GETT("WZ",DUPLO_NO)`	Determine T number
`SETPIECE(x,y)`	Set piece number
`GETSELT(x)`	Read preselected tool number (T No.)
`"WZ"`	Tool name
`DUPLO_NO`	Quantity
`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("WZ",DUPLO_NO)
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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("WZ", 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 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)

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 Preselect tool, i.e. the tool magazine can be brought into the tool position parallel to machining.

M6 Load preselected tool (depending on the setting in the machine data you can also program 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.

840D
NCU 572
NCU 573

840Di

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
```

Explanation of the commands

PUTFTOCF	Write online tool offsets continuously
FCTDEF	Define parameters for PUTFTOCF function
PUTFTOC	Write online tool offsets discretely
FTOCON	Activate online tool offsets
FTOCOF	Deactivate online tool offsets

Explanation of the parameters

Polynomial_No.	Values 1–3: a maximum of three polynomials can be programmed at the same time; polynomials up to 3rd degree
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	Lower limit
ULimit	Upper limit
a_0, a_1, a_2, a_3	Coefficients of polynomial function
Value	Value added in the wear parameter

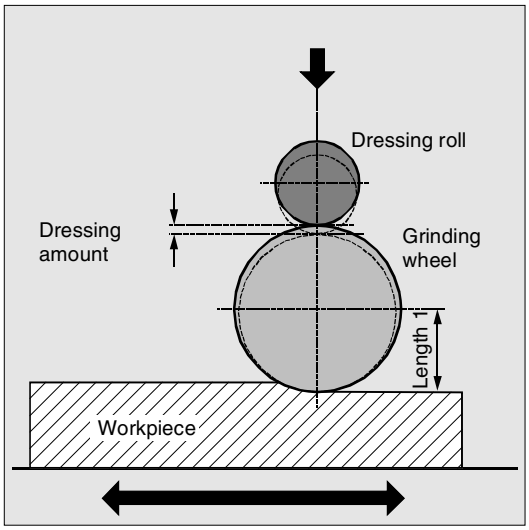
8.3 Online tool offset PUTFTOCF, PUTFTOC, FTOCON, FTOCOF

840D
NCU 572
NCU 573

840Di

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, modally: ID=1 DO FTOC (see Chapter 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").

8.3 Online tool offset PUTFTOCF, PUTFTOC, FTOCON, FTOCOF

840D
NCU 572
NCU 573

840Di

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 not spindle number is programmed, the offset applies to the active tool.

840D
NCU 572
NCU 573

840Di

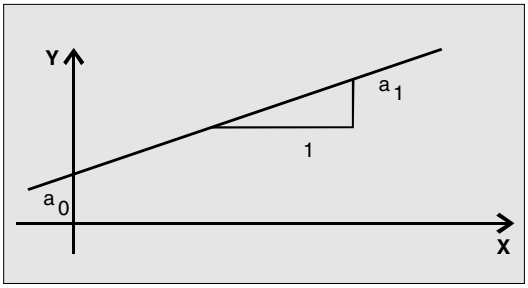
Set parameters for FCTDEF function

The parameters are defined in a separate block:

```
FCTDEF(Polynomial_NO., LLimit, ULimit,a0,a1,a2,a3)
```

The polynomial can be a 1st, 2nd or 3rd 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.

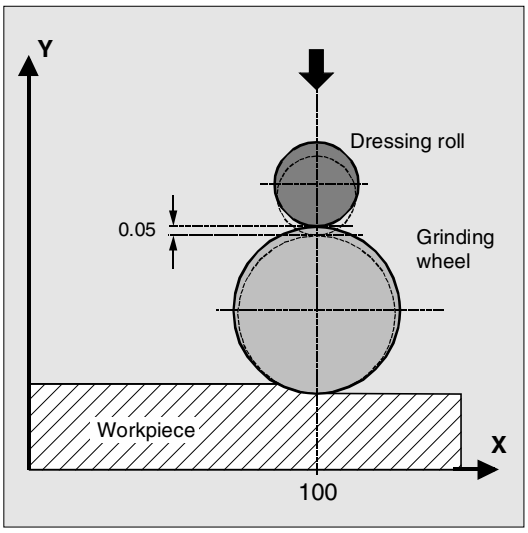
840D
NCU 572
NCU 573

840Di

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 the grinding wheel
V: Infeed axis for the dresser roll

Machine: Channel 1 with axes X, Z, Y
Dress: Channel 2 with axis V

Machining program in channel 1:

```
%_N_MACH_MPF
...
N110 G1 G18 F10 G90                    Basic position
N120 T1 D1                             Select current tool
N130 S100 M3 X100                      Spindle on, travel to starting position
N140 INIT (2, "DRESS", "S")            Select dressing program on channel 2
N150 START (2)                         Start dressing program on channel 2
N160 X200                              Travel to destination 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 line
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
```

840D
NCU 572
NCU 573

840Di

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 for dressing
`G1 V-0.05 F0.02`	
`...`	
`CANCEL(1)`	Deselect online offset
`...`	

840D
NCU 572
NCU 573

840Di

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 tool radius compensation constant function (default setting)

Function

The "tool radius compensation constant" function is used to suppress the tool radius compensation for a number of blocks while retaining the difference between the programmed and actual path of the tool center point accumulated in previous blocks as an offset.
This can be practical, for example, if several motion blocks are required at the reversal points during line-by-line milling but the contours (bypass strategies) generated by the tool radius compensation are not desirable.
It can be used according to the type of tool radius compensation (2 1/2D, 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.
The offset point at the end of block position is approached in the last motion block before CUTCONON.
All following blocks in which the compensation suppression is active are executed without compensation.

840D
NCU 572
NCU 573

840Di

They are displaced, however, by the vector from the end point of the last co block to its offset point.
The interpolation type of these blocks (linear, circular, polynomial) is arbitrary.
The deactivation block of the compensation suppression, i.e. the block containing CUTCONOF, is usually corrected; it begins at the offset point of the start point.
A linear block is inserted between this point and the end point of the previous block, i.e. the last programmed motion block with active CUTCONON.
Circle blocks in which the circle plane is perpendicular to the compensation plane (vertical circles) are treated as if CUTCONON had been programmed in the blocks.
This implicit activation of compensation suppression is automatically cancelled in the first motion block which is not a circle of this type but which contains a traversing movement in the compensation plane.
Vertical circles of this type can only occur with circumferential milling.

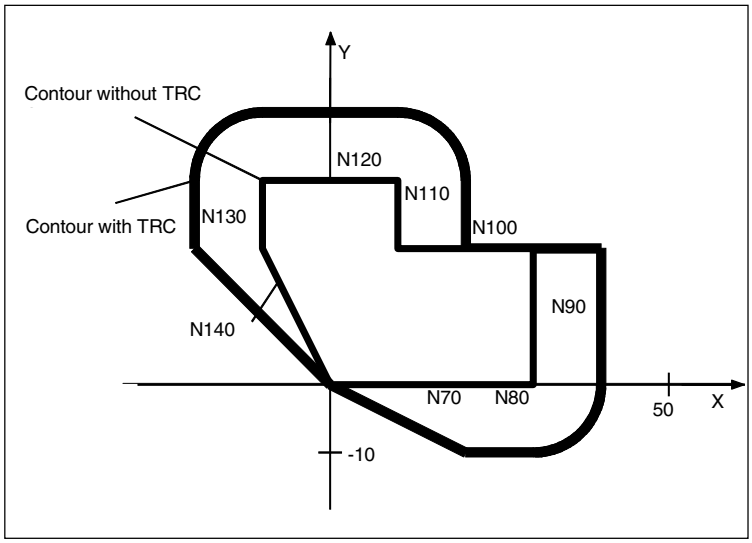
Example

`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 CONT ;`	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`	

8.4 Maintain tool radius compensation at constant level, CUTCONON

840D
NCU 572
NCU 573

840Di



Additional notes

1. CUTCONON has no effect if tool radius compensation is not active (G40). An alarm is output.
   The G code remains active, however. This is significant if tool radius compensation is to be activated in a subsequent block with G41 or G42.
2. It is possible to change the G code in the 7th G code group (tool radius compensation; G40 / G41 / G42) when CUTCONON is active. A change to G40 is effective immediately.
   The offset with which the previous blocks were traversed is applied.
3. If CUTCONON or CUTCONOF is programmed in a block without a traversing movement in the active compensation plane, the change does not become effective until the next block with such a traversing movement.

Further information: /FB/, W1 Tool Offset

840D
NCU 572
NCU 573

840Di

8.5 Activate 3D tool offsets

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 determined by G17–G19 and possibly turned by a Frame.
CUT3DF	3D tool offset for face milling with orientation change (only with active 5-axes transformation).
G40 X Y Z	To deactivate: linear block G0/G1 with geometry axes
ISD=Value	Insertion depth

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 compensationTool radius compensation, 3D 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 method is always NORM.

840D
NCU 572
NCU 573

840Di

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 …	

Additional notes

Intermediate blocks are permitted with 3D tool radius compensation. The rules for 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 DISC command is ignored.

8.5 Activate 3D tool offsets

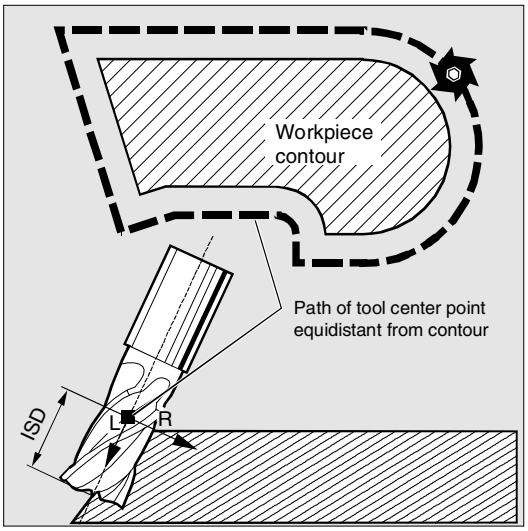
840D
NCU 572
NCU 573

840Di

Difference between 2 1/2D and 3D tool radius compensation

In 3D tool radius compensation tool orientation can be changed.

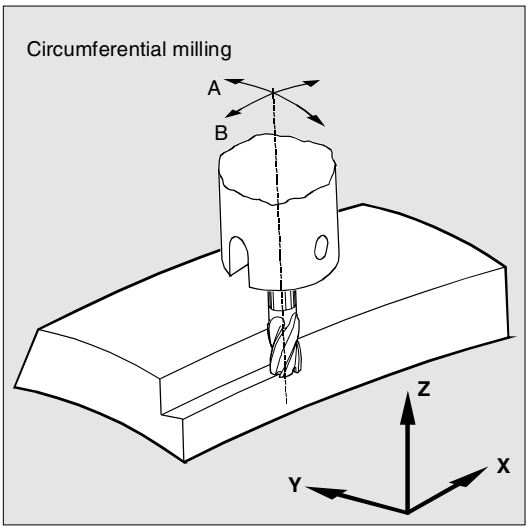
2 1/2D tool radius compensation assumes the use of a tool with constant orientation.



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.

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 deciding factor is the radius at the tool insertion point.



The 3D TRC function is limited to cylindrical tools.

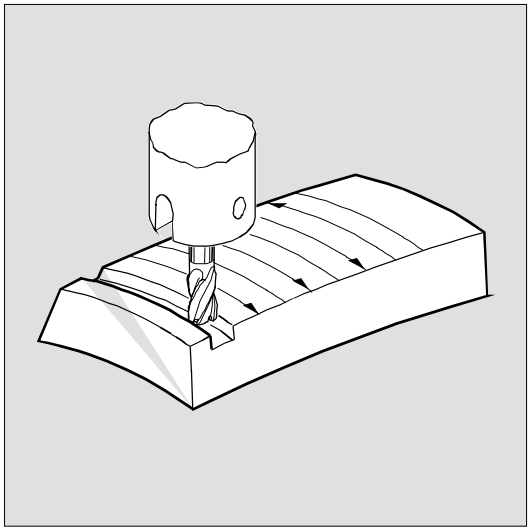
840D
NCU 572
NCU 573

840Di

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 part 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.

840D
NCU 572
NCU 573

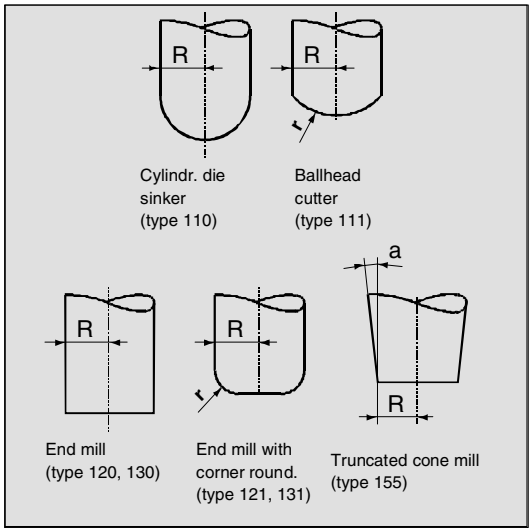
840Di

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.



Milling tool type	Type No.	R	r	a
Cylindrical miller	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

X=is not evaluated

Tool length compensation

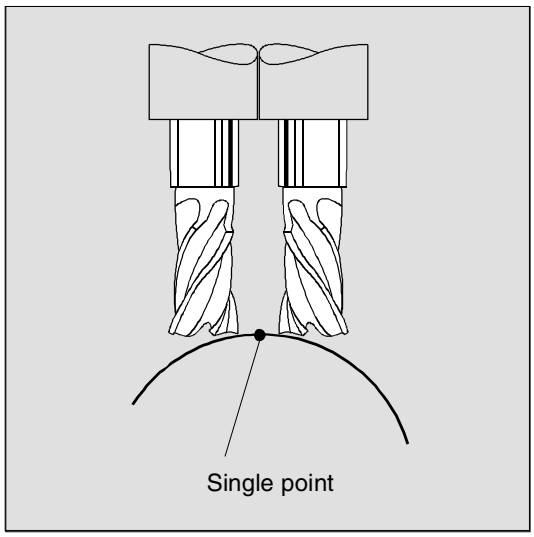
The tool tip is the reference point for length compensation (intersection longitudinal axis/surface).

840D
NCU 572
NCU 573

840Di

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.

Correction of the path
With respect to face milling, it is advisable to examine what happens when the contact point "jumps" on the tool surface as shown in the example on the right where a convex surface is being machined with a vertically positioned tool.



As a general rule, it is advisable to select a tool shape and tool orientation that are suitable for producing the required surface profile.

The application shown in the example should therefore 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.

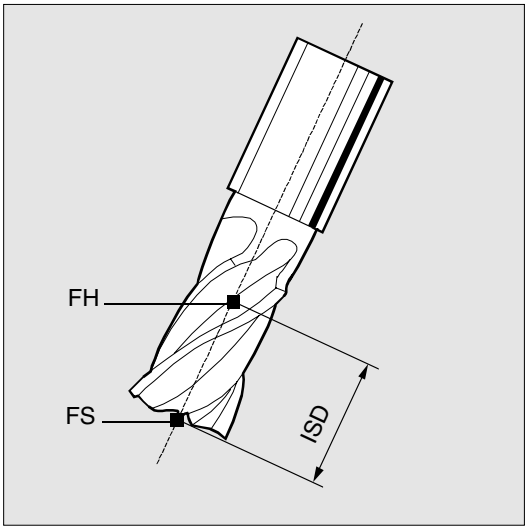
840D
NCU 572
NCU 573

840Di

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.

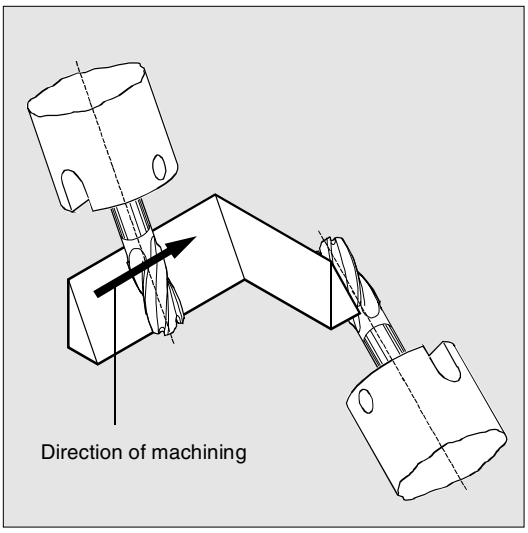
Insertion depth (ISD)
Program command ISD (insertion depth) is used to program the tool insertion depth for peripheral milling operations. This makes it possible to change the position of the machining point on the outer surface of the tool.

ISD specifies the distance between the cutter tip (FS) and the cutter reference point (FH). The point FH is produced by projecting the programmed machining point along the tool axis. ISD is only evaluated when 3D tool radius compensation is active.



Inside corners/outside corners
Inside and outside corners are handled separately. The term inside or outside corner depends on the tool orientation.

When changes occur in the orientation at a corner, the corner type can change during machining. If this happens, machining stops and an error message is generated.



840D
NCU 572
NCU 573

840Di

Intersection procedure for 3D compensation (from SW 5)

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. With SW 4 a circle was always inserted at the outside corners.

The new functionality is particularly advantageous for typical CAD-generated 3D programs. 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 1/2 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 both blocks and defining their intersection a 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

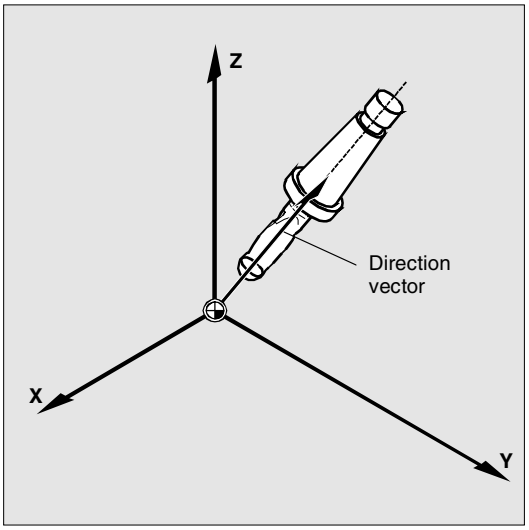
840D
NCU 572
NCU 573

840Di

8.6 Tool orientation

Tool orientation is the term given to the geometrical alignment of the tool in space.

On a 5-axis machine tool, the tool orientation can be controlled with program commands.



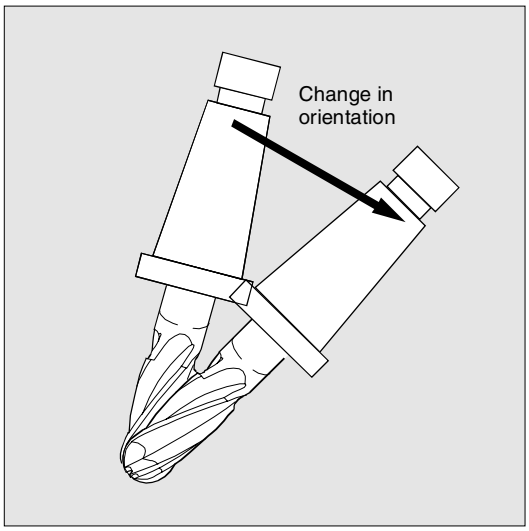
Programming tool orientation

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 (ORIMCS) or the current workpiece coordinate system (ORIWCS).

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; valid for OSS and OSSE

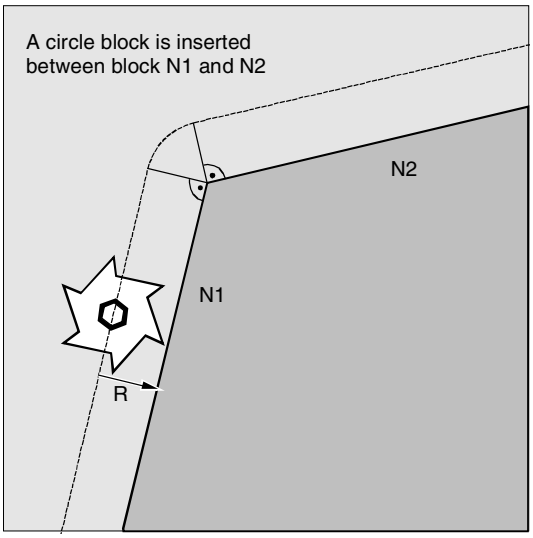
840D
NCU 572
NCU 573

840Di

Behavior at outside 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.

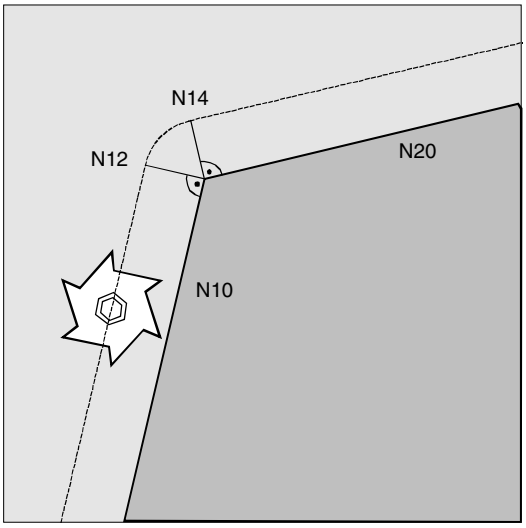
If several orientation blocks are inserted at an external corner and ORIC is selected, the circular movement is divided among the individual inserted blocks according to the values of the orientation changes.

840D
NCU 572
NCU 573

840Di

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.



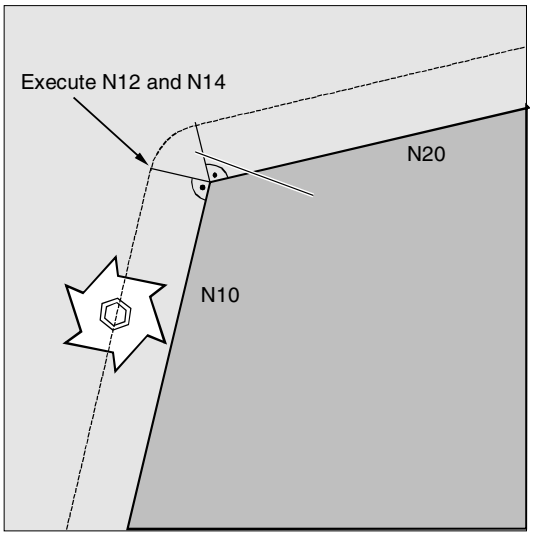
`ORIC`	
`N8 A2=… B2=… C2=…`	
`N10 X… Y… Z…`	
`N12 C2=… B2=…` `N14 C2=… B2=…`	The circle block inserted at the external corner is divided among N12 and N14 in accordance with the change in orientation. The circular movement and the orientation change are executed in parallel.
`N20 X =…Y=… Z=… G1 F200`	

840D
NCU 572
NCU 573

840Di

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 with the current orientation is subsequently traversed.
`N14 M20`	Auxiliary functions, etc.
`N20 X… Y… Z…`	

The program command which is active in the first traversing block of an external corner determines the type of orientation change.

840D
NCU 572
NCU 573

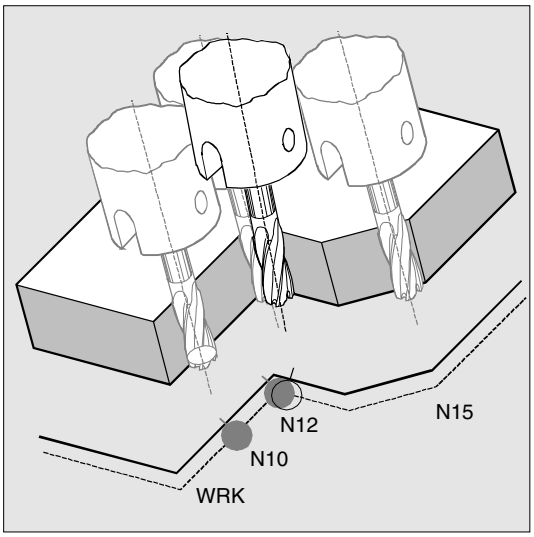
840Di

Without orientation change

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 the orientation at an internal corner




```
ORIC
N10 X …Y… Z… G1 F500
N12 X …Y… Z… A2=… B2=…, C2=…
N15 X Y Z A2 B2 C2
```

840D
NCU 572
NCU 573

840Di

8.7 Free assignment of D numbers, cutting edge number CE (as of SW 5)

As of SW5, 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 parameter $TC_DPCE to describe the cutting edge number.
Preset: offset number == cutting edge number
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). Please refer to the data of the machine tool manufacturer.

Additional notes

Besides the relative D number, you can also assign D-numbers al 'flat' or 'absolute' D-numbers (1–32000) without assigning a reference to a T-number (inside the function flat D number structure).

840D
NCU 572
NCU 573

840Di

8.7.1 Check 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 part program.
`CHKDNO(Tno1,Tno2)`	All D numbers of the part specified are checked.	
`CHKDNO(Tno1)`	All D numbers of Tno1 are checked against all other tools.	
`CHKDNO`	All D numbers of all tools are checked against all other tools.	

Function

CKKDNO checks whether the available D numbers assigned are unique.
The D numbers of all tools defined in a TO unit must only be present once. Replacement tools are not considered.

840D
NCU 572
NCU 573

840Di

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 cutting edge of the tool
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 there is no D number for the specified parameters, d is set to 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 produce an alarm. The D number cannot be set to 0 explicitly.

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)
```

840D
NCU 572
NCU 573

840Di

This assigns cutting edge CE=3 the new D value 17. Now, these data for the cutting edge are addressed via D number 17; both via the system parameters and in the programming with the NC address.

Additional notes

You must assign unique D numbers. Two different cuttting edges of a tool must not have the same D number.

8.7.3 T numbers for the specified D number (GETACTTD)

Programming

```
status = GETACTTD(Tno, Dno)
```

Explanation of the parameters

`Dno`	D number to be looked for for the T number.
`Tno`	T number found
`status`	0: The T number was found. Tno contains the value of the T number. –1: The specified D number does not have a T 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: Unable to perform the function for another reason.

Function

For an absolute D number, GETACTTD determines the assocated T number. There is not check for uniqueness. If there are several identical D numbers within a TO unit, the T number of the first tool found is returned. If 'flat' D numbers are used, it does not make sense to use the command because the value 1 is always returned (no T number in database).

840D
NCU 572
NCU 573

840Di

8.7.4 Set final D numbers to invalid

Programming

```
DZERO
```

Explanation

DZERO	Marks all D number of the TO unit as invalid

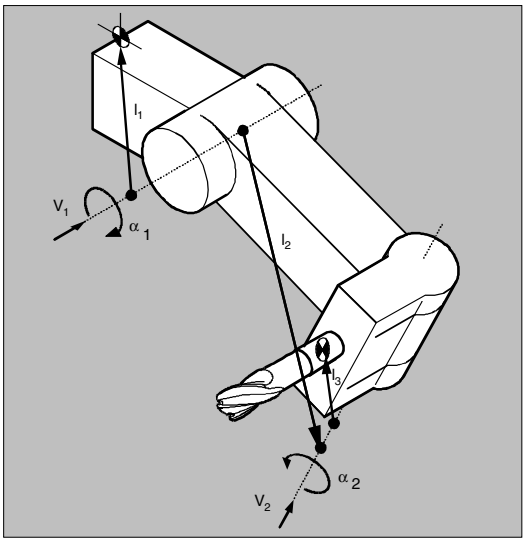
Function

The command is used for support during upgrading.
Offset block marked in this way are no longer
checked by the language command CHKDNO.
To regain access, you must set the D number to
SETDNO again

840D
NCU 572
NCU 573

840Di

8.8 Toolholder kinematics



The toolholder kinematics with up to two rotary axes is programmed by means of 17 system variables (see also Programming Guide Advanced) $TC_CARR1[m] to $TC_CARR17[m]. The description of the toolholder consists of:

- the vectorial distance between the first rotary axis and the toolholder reference point l_1, the vectorial distance between the first and the second rotary axis l_2, the vectorial distance between the second rotary axis and the tool reference point l_3;
- the reference vectors of both rotary axes V1,V2;
- the rotary 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 are described as follows:

The rotatable part consisting of:

- the vector distance between the second rotating axis v_2 and the reference point of a rotatable tool table l_4 of the 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 posiitons are accessed as required to determine the oritentation of the orientable toolholder.

The permissible types of kinematics consisitng of:

- 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

840D
NCU 572
NCU 573

840Di

Function of the system parameter for orientable toolholders

	x components	y components	z components
l_1	$TC_CARR1[m]	$TC_CARR2[m]	$TC_CARR3[m]
l_2	$TC_CARR4[m]	$TC_CARR5[m]	$TC_CARR6[m]
v_1	$TC_CARR7[m]	$TC_CARR8[m]	$TC_CARR9[m]
v_2	$TC_CARR10[m]	$TC_CARR11[m]	$TC_CARR12[m]
α_1 α_2	Angle of rotation = $TC_CARR13[m] Angle of rotation = $TC_CARR14[m]		
l_3	$TC_CARR15[m]	$TC_CARR16[m]	$TC_CARR17[m]
Offset vector l_4	$TC_CARR18[m]	$TC_CARR19[m]	$TC_CARR20[m]
Rotary axis v_1 Rotary axis v_2	$TC_CARR21[m] $TC_CARR22[m]		
Type of kinematics Preset T⇨P⇨M	$TC_CARR23[m]		
	Type of kinematics T or ⇨	Type of kinematics P or ⇨	Type of kinematics M
	Only the **T**ool ican be rotated	Only the **P**art can be rotated	Part and tool **M**ixed mode can be rotated

Additional notes

The number of the respective toolholder to be programmed is specified with "m".
The start/endpoints of the distance vectors on the axes can be freely selected. The rotation angles around the two axes are defined in the initial state of the toolholder with 0°. In this way, the kinematic description of a toolholder can be programmed unambiguously for any number of possibilities.
If the two rotary axes intersect, it is not necessary to specify the distance between the two axes.
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).

840D
NCU 572
NCU 573

840Di

Clearing the toolholder data
The data of all toolholder data sets is cleared via $TC_CARR1[0] = 0.
Ab SW 5.3
The type of kinematics $TC_CARR23[T] = T must be preassigned 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 part program.
As of SW 5.3
By preassigning the type of kinematics, you have only the following three options:
1. **T**: Only the tool (**T**ool) can rotate.
2. **P:** Only the workpiece (**P**art) can rotate.
3. **M**: Part and tool (**M**ixed mode) can rotate, corresponding to resolved kinematics.

Any other character, except for the three above, 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 part program.

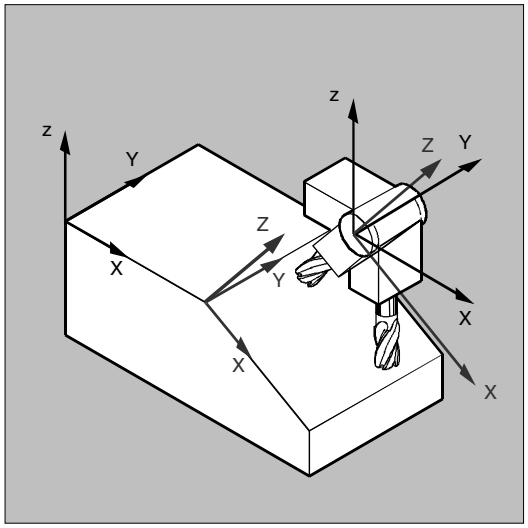
Supplementary conditions
A toolholder can only orientate a tool in every possible direction in space if
– there are two rotary axes.
– the rotary axes are positioned perpendicular to one another.
– the tool length axis is perpendicular to the second rotary axis.
As of SW 5.3
The following also applies to machine where both axes must rotate the table so that
– the tool orientation is perpendicular to the first rotary axis.

840D
NCU 572
NCU 573

840Di

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 initial 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`	

840D
NCU 572
NCU 573

840Di

Notes

Path Traversing Behavior

840D
NCU 572
NCU 573

810D 840Di

9.1 Tangential control TANG, TANGON, TANGOF

Programming

```
TANG (FAxisF,LAxis1,LAxis2,Coupling,CS)
TANGON (FAxis,Angle)
TANGOF (FAxis)
TLIFT (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
TANGOF	Deactivate tangential control specifying following axis
TLIFT	Insert intermediate block at contour corners

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
CSCS	Identifier for coordinate system "B" = basic coordinate system; "W" = workpiece coordinate system Parameter optional; default "B"
Angle	Offset angle of following axis

840D
NCU 572
NCU 573

810D 840Di

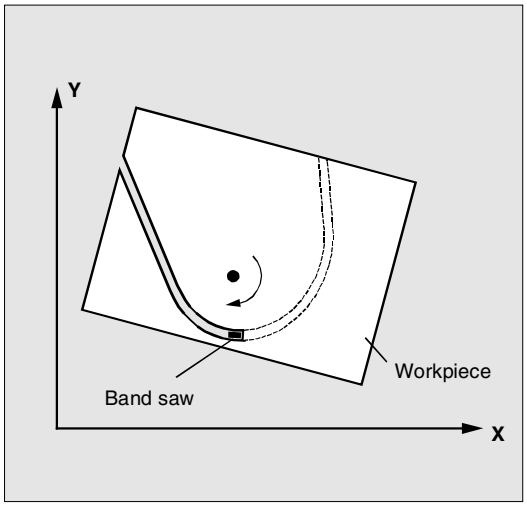
Function

A rotary axis (= following axis) follows the programmed path of two leading axes. The following axis is located at a defined offset angle to the path tangent.

Applications

Tangential control can be used in applications such as:

- Tangential positioning of a rotatable tool during nibbling
- Follow-up of the tool orientation on a band saw
- Positioning of a dresser tool on a grinding wheel (see diagram)
- 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.

9.1 Tangential control TANG, TANGON, TANGOF

840D
NCU 572
NCU 573

810D 840Di

Activating/deactivating tangential control TANGON, TANGOF

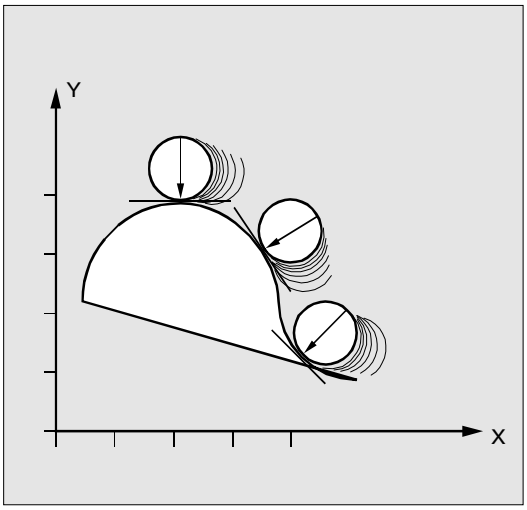
Tangential control is called with TANGON specifying the following axis and the desired offset angle of the following axis:

```
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)
```

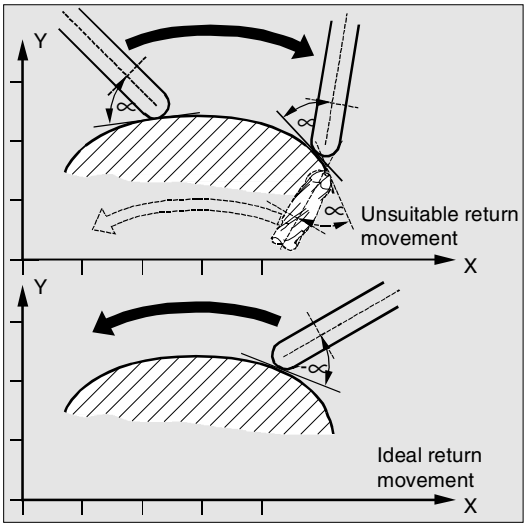


Angle limit through working area limitation

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 can be achieved 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.



840D
NCU 572
NCU 573

810D 840Di

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 axis is rotated at its maximum possible velocity.

The TLIFT(...) instruction must be programmed immediately after the axis assignment with TANG(...).

Example:

```
TANG(C,X,Y...)
TLIFT(C)
```

Deactivate TLIFT

To deactivate TLIFT, repeat the axis assignment TANG(...) without inserting TLIFT(...) afterwards.

The angular change limit at which an intermediate block is automatically inserted is defined via machine data `$MA_EPS_TLIFT_TANG_STEP`.

840D
NCU 572
NCU 573

810D 840Di

Additional notes

Influence on transformations

The position of the following rotary axis can be an 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.

Coupling status

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

9.2 Coupled motion TRAILON, TRAILOF

Programming

```
TRAILON(FAxis,LAxis,Coupling)
TRAILOF(FAxis,LAxis,Axis2)
```

Explanation of the commands and parameters

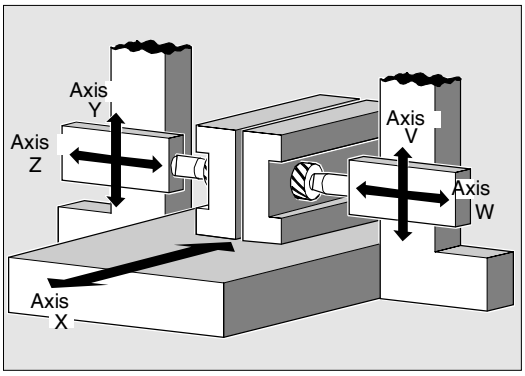
TRAILON	Activate and define coupled axes; modal
TRAILOF	Deactivate coupled axes
FAxis	Axis name of trailing axis
LAxis	Axis name of trailing axis
Coupling	Coupling factor = Path of coupled-motion axis/path of trailing axis Default = 1

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

- Traversing of an axis by a simulated axis. The leading axis is a simulated axis and the trailing axis is a real axis. The real axis can thus be traversed with allowance for the coupling factor.
- Two-sided machining with 2 combined axis pairs:
  1st leading axis Y, trailing axis V
  2nd leading axis Z, trailing axis W



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 axes that can be activated simultaneously is restricted only by the possible combinations of axes on the machine.

Coupled motion always takes place in the basic coordinate system (BCS).

Coupled axis types
A coupled-axis group can consist of any combination 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 trailing axis can be programmed with all the available motion commands (G0, G1, G2, G3, ...etc.). In addition to paths defined independently, the trailing axis also traverses the distances derived from its leading axes, allowing for the coupling factors.

A trailing axis can also act as a leading axis for other trailing axes. Various combinations of coupled axes can be set up in this way.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Coupling factor
The coupling factor specifies the desired ratio of the paths of trailing axis and leading axis.

$$\text{Coupling factor} = \frac{\text{Path of trailing axis}}{\text{Path of leading axis}}$$

If the coupling factor is not specified in the program, a coupling factor of 1 is automatically taken as the default.

The factor is entered as a decimal fraction (type REAL). The input of a negative value causes opposite traversing movements on the leading and trailing axes.

Deactivate coupled axes
The following language command deactivates the coupling with a leading axis:

```
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)
```

Additional notes

Acceleration and velocity
The acceleration and velocity limits of the combined axes are determined by the "weakest axis" in the combined axis pair.

Coupling status
You can query the status of the coupling in the NC program with the following system variable:

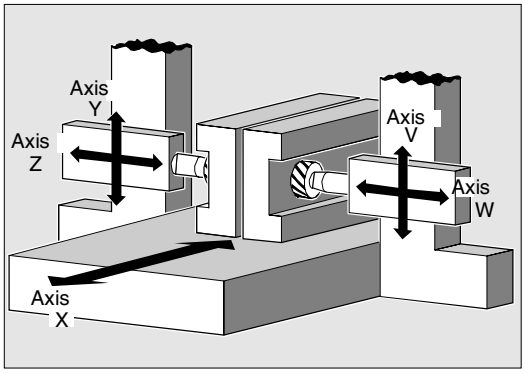
```
$AA_COUP_ACT[axis]
```

0	No coupling active
8	Coupled motion active

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 combined axis pair
`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 of Z and W axes in opposite axis 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
`TRAILOF(W,Z)`	Deactivate 2nd coupled axis

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

9.3 Curve tables, CTABDEF, CTABEND, CTAB, CTABINV

Programming

Curve tables are defined in a part program.

`CTABDEF(FAxis,LAxis,n,applim)`	Define beginning of curve table
`CTABEND()`	Define end of curve table
`CTABDEL(n)`	Delete a curve table
`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

For further information about leading and following values, see Section "Axial leading value coupling" and "Path leading value coupling" in this section.

Explanation

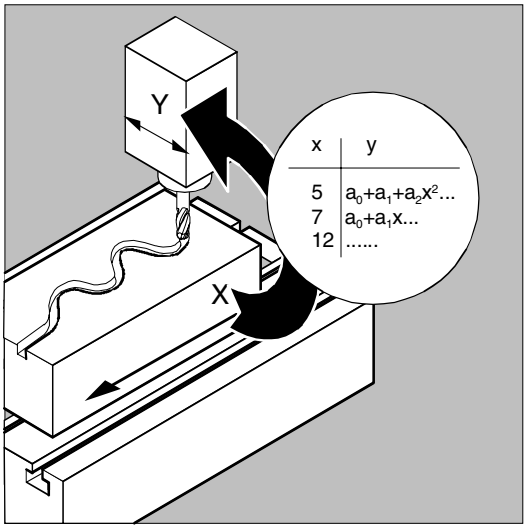
`FAxis`	Following axis: Axis that is programmed via the curve table.
`LAxis`	Leading axis Axis that is programmed with the leading value.
`n`	Number of the curve table
`applim`	Identifier for table periodicity: 0 Table is not periodic 1 Table is periodic
`LW`	Leading value Positional value of the leading axis for which a following value is to be calculated.
`degrees`	Parameter name for gradient parameter
`FW`	Following value Positional value of the following axis for which a leading value is to be calculated.
`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

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Function

You can use curve tables to program position and velocity relationships between 2 axes.

Example: Replace mechanical cam plates.
The curve table forms the basis for the axial leading value coupling by creating the functional relationship between the leading and the following value:
The control calculates a polynomial that corresponds to the cam plate from the relative positions of the leading and following axes.



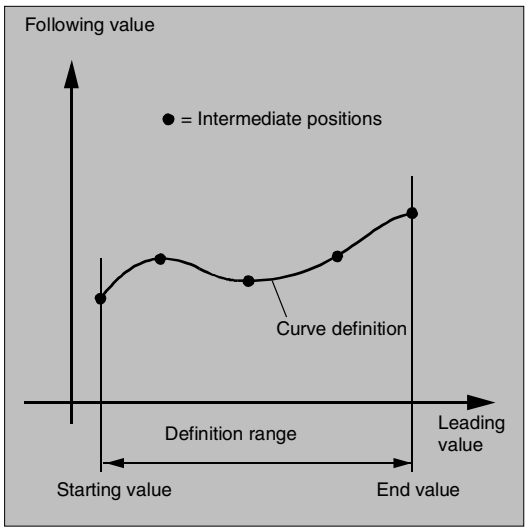
Additional notes

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 part program or a section of a part program which is enclosed by CTABDEF at the beginning and CTABEND at the end.

Within this part program section, unique following 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 the 3rd order.



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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Additional notes

The following are not permissible:

- Preprocess stop
- Tool radius compensation
- 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.

All modal statements that are made within the curve table definition are invalid when the table definition is completed. The part program in which the table definition is made is therefore located in front of and after the table definition in the same state.

R parameter assignments are reset.
Example:

```
...
R10=5 R11=20
...
CTABDEF
G1 X=10 Y=20 F1000
R10=R11+5      ;R10=25
X=R10
CTABEND
...         ;R10=5
```

Repeated use of curve tables

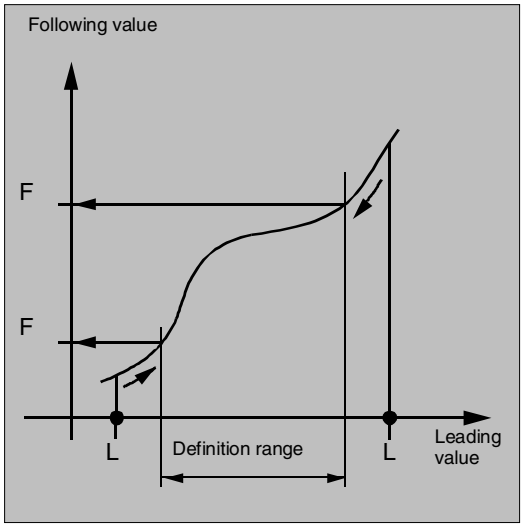
The function relation between the leading axis and the following axis calculated through the curve table is retained under the table number beyond the end of the part program and during power-off.

The curve table created can be applied to any axis combinations of leading and following axes whatever axes were used to create the curve table.

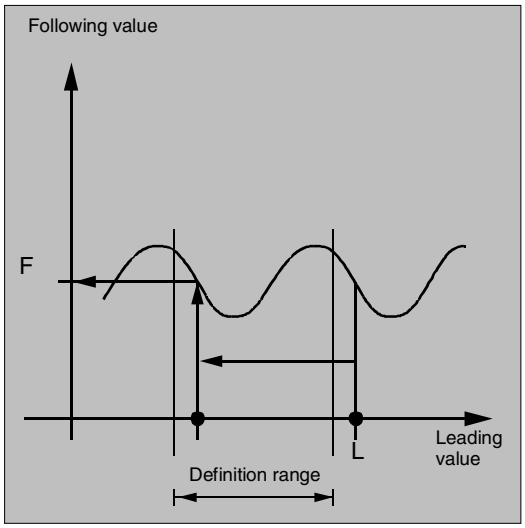
840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Behavior at the edges of the curve table

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.

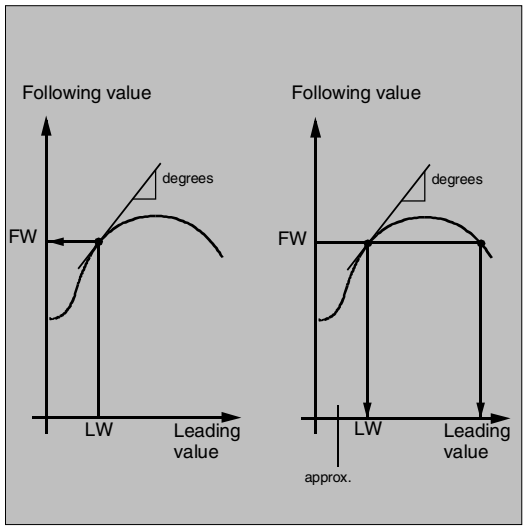


840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Reading table positions, CTAB, CTABINV

With CTAB you can read the following value for a leading value directly from the part 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.

Additional notes

Optional specification of the leading or following axis for CTAB/CTABINV is important if the leading and following axes are configured in different length units.

Deleting curve tables, **CTABDEL**
With CTABDEL you can delete the curve tables. Curve tables that are active in a coupling cannot be deleted.

Overwriting curve tables
A curve table is overwritten as soon as is number is used in another table definition. Active tables cannot be overwritten.

Additional notes

No warning is output when you overwrite curve tables!

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Additional notes

With the system variable $P_CTABDEF it is possible to query from inside a part program whether a curve table definition is active.

The part program section can be used as a curve table definition after excluding the statements and therefore as a real part program again.

Programming example

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
```

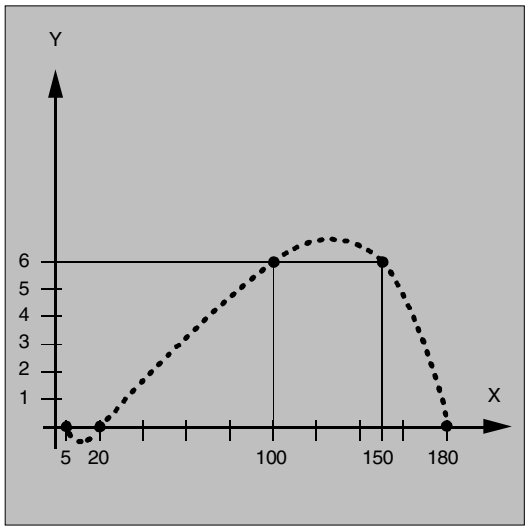
Curve tables and various operating states

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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: Leading value: 5; Following value: 0
`N120 X20 Y0`	2. Intermediate point: Leading value: 0...20; Following value: Starting value...0
`N130 X100 Y6`	3. Intermediate point: Leading value: 20...100; Following value: 0…6
`N140 X150 Y6`	4. Intermediate point: Leading value: 100...150; Following value: 6…6
`N150 X180 Y0`	5. Intermediate point: Leading 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 part program state before the beginning of the definition is restored.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

9.4 Axial leading value coupling, LEADON, LEADOF

Programming

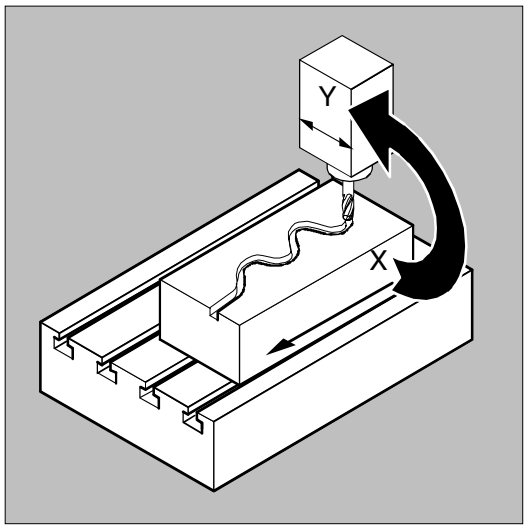
```
LEADON(FAxis,LAxis,n)
LEADOF(FAxis,LAxis,n)
```

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 part program and during the movement from synchronized actions.

The leading value coupling always applies in the basic coordinate system.

For information about creating curve table, see Chapter "Curve tables" in this chapter.
For information about leading value coupling, see /FB/, M3, Coupled Motion and Leading Value Coupling.

9.4 Axial leading value coupling, LEADON, LEADOF

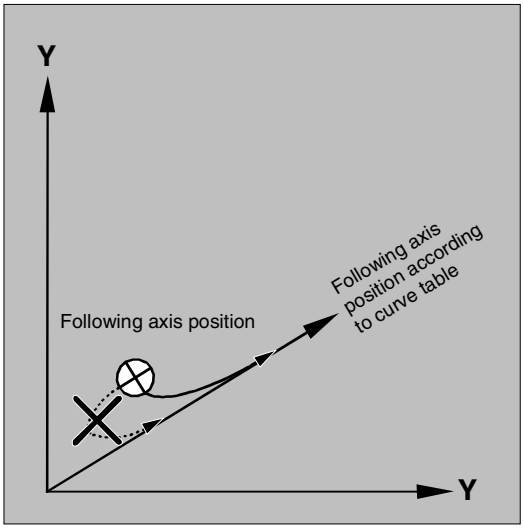
840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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).



Additional notes

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:

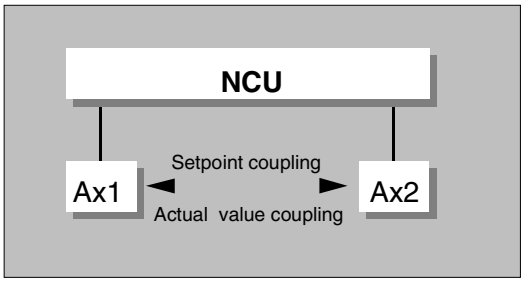
- Actual values of the leading axis position: Actual value coupling
- Setpoints of the leading axis position: Setpoint coupling

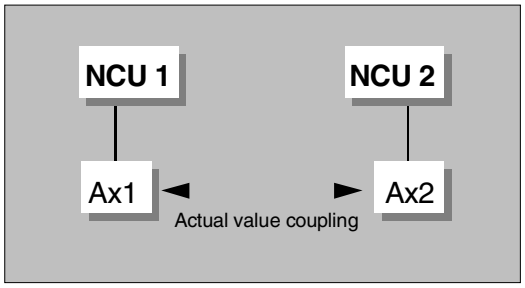
840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Additional notes

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 re-synchronize after switchover when the axis is motionless.

Application example:

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

Leading value simulation with setpoint coupling

Via machine data, you can disconnect the interpolator for the leading axis from the servo. In this way you can generate setpoints for setpoint coupling without actually moving the leading axis.

Leading values generated from a setpoint coupling 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

Additional notes

As an option, leading values can be generated with other self-programmed methods. The leading values generated in this way are written into the variables

- `$AA_LEAD_SP` Leading value position
- `$AA_LEAD_SV` Leading value velocity

and read from them. 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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 events 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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
```

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

9.5 Feed characteristic, 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.
FCUB	Path velocity profile cubic: The non-modally programmed F values are connected by means of a spline referred to the block end point. The spline begins and ends tangentially with the previous and the following feedrate specification. 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.

9.5 Feed characteristic, FNORM, FLIN, FCUB, FPO

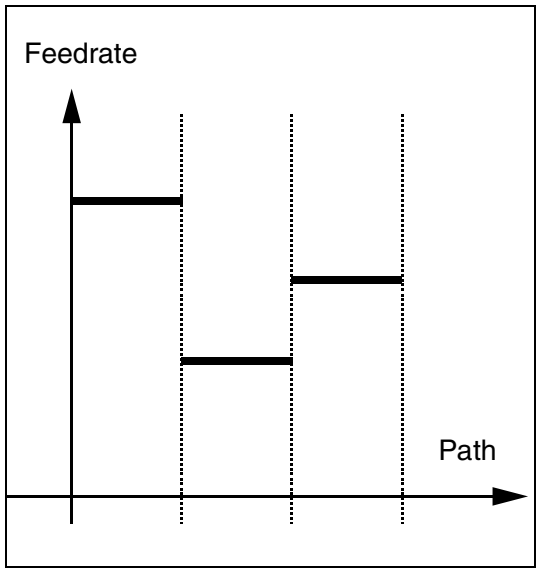
840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

These additional characteristics make it possible to program continuously smooth velocity characteristics depending on the curvature of the workpiece to be machined.

Sequence

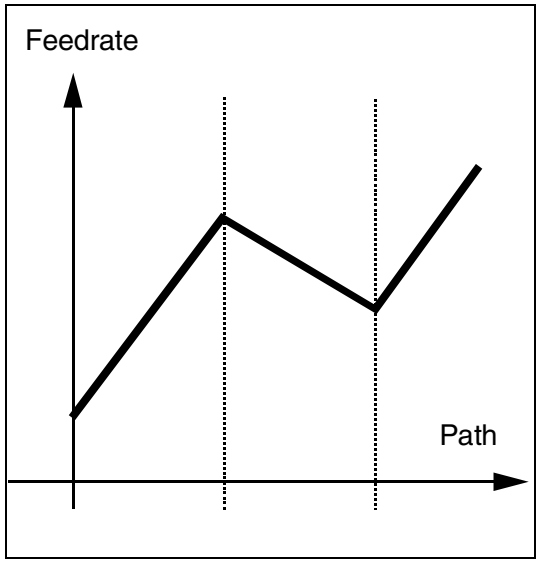
FNORM
The feed address F defines the path feed as a constant value according to DIN 66025.

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.

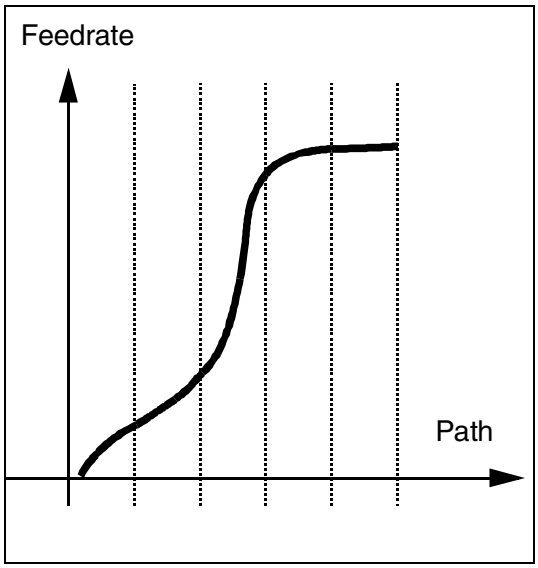
Example:
```
N30 F1400 FLIN X50
```



840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

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.

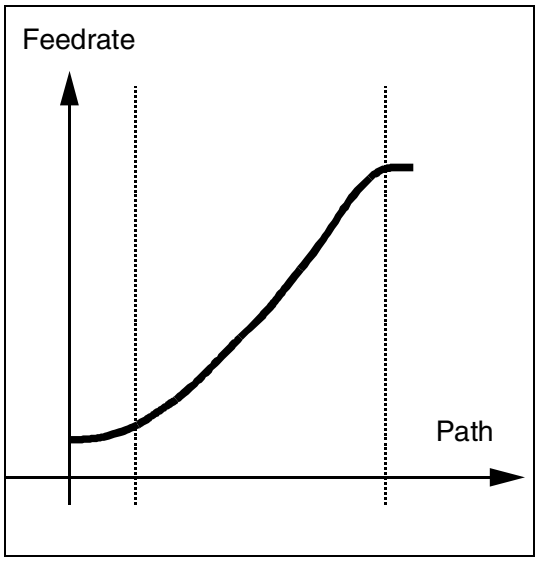


Example:

```
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.

Supplementary conditions

The functions for programming the path traversing characteristics apply regardless of the programmed feed characteristic.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

The programmed feed characteristic is always absolute regardless of G90 or G91.

Additional notes

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.

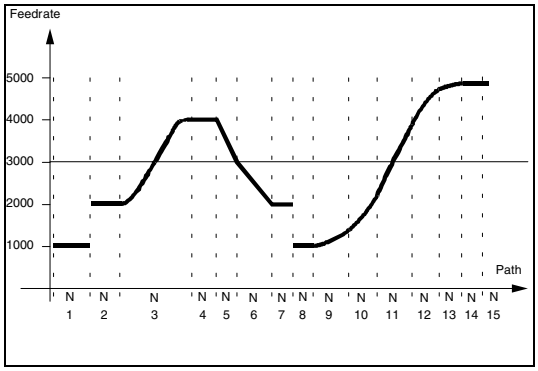
Feed optimization on curved path sections

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

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`	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

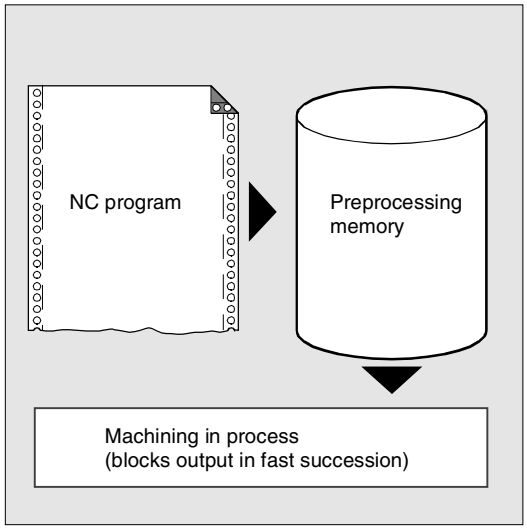
9.6 Program run with preprocessing memory, STARTFIFO, STOPFIFO, STOPRE

Explanation of the commands

STOPFIFO	Stop high-speed processing section, fill preprocessing memory, Preprocessing memory until STARTFIFO, "Preprocessing memory full" or "End of program" is detected.
STARTFIFO	Start of high-speed processing section, in parallel to filling the preprocessing memory
STOPRE	Preprocessing stop

Function

Depending on its expansion level, the control system has a certain quantity of so-called preprocessing memory in which preprepared 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Restrictions

The preprocessing memory is not filled or the filling process interrupted if the processing section contains commands that require unbuffered operation (reference point approach, measuring functions, ...).

Stop preprocessing

When **STOPRE** is programmed, the following block is not processed until all previously prepared and stored blocks have been fully executed. The previous block is halted with exact stop (as for G9).

Example:

```
N10 …
N30 MEAW=1 G1 F1000 X100 Y100 Z50
N40 STOPRE
```

The control system initiates an internal preprocessing stop while status data of the machine ($A...) are accessed.

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

9.7 Repositioning on contour, REPOSA, REPOSL, REPOSQ, REPOSH

Programming

`REPOSA RMI DISPR=…` or `REPOSA RMB` or `REPOSA RME`

`REPOSL RMI DISPR=…` or `REPOSL RMB` or `REPOSL RME`

`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

Repositioning point

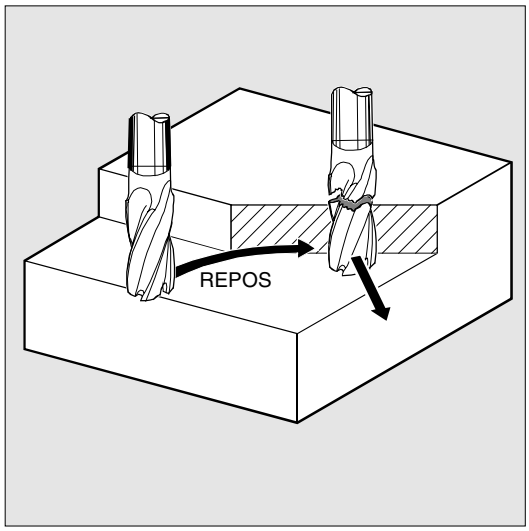
`RMI`	Approach interruption point
`RMI DISPR=…`	Entry point at distance DISPR in mm/inch **in front of** interruption point
`RMB`	Approach block start point
`RME DISPR=…`	Approach block end point at distance DISPR in front of end point
`A0 B0 C0`	Axes in which approach is to be made

9.7 Repositioning on contour, REPOSA, REPOSL, REPOSQ, REPOSH

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

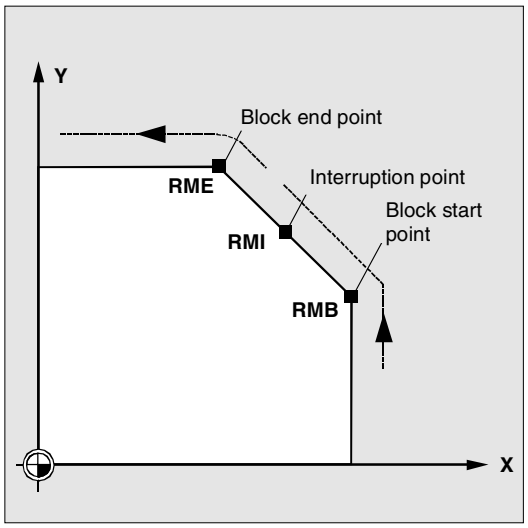
For information about interrupting program runs, see also Section "Interrupt routine" in Programming Guide "Advanced".

Sequence

Defining repositioning point

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



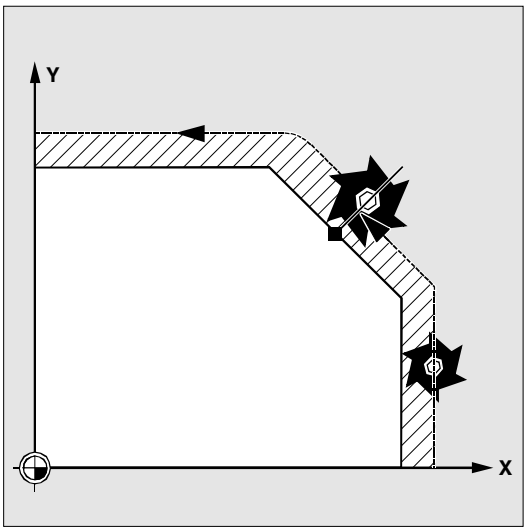
`RMI DISPR=…<F 6` or `RME DISPR=…` allows you to select a repositioning point which sits before the interruption point or the block end point.
`DISPR=...` allows you to describe the contour distance in mm/inch between the repositioning point and the interruption or **before** the end point. Even for 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`).
SW 5.2 and higher:
The sign before `DISPR` is evaluated.
In the case of a plus sign, the behavior is as previously.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

Application example:
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.

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.

9.7 Repositioning on contour, REPOSA, REPOSL, REPOSQ, REPOSH

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

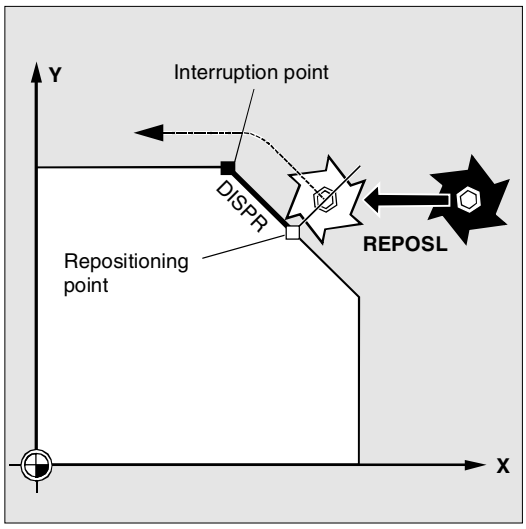
All axes are automatically traversed with command REPOSA. With REPOSL you can specify which axes are to be moved.

Example:

```
REPOSL RMI DISPR=6 F400
```

or

```
REPOSA RMI DISPR=6 F400
```

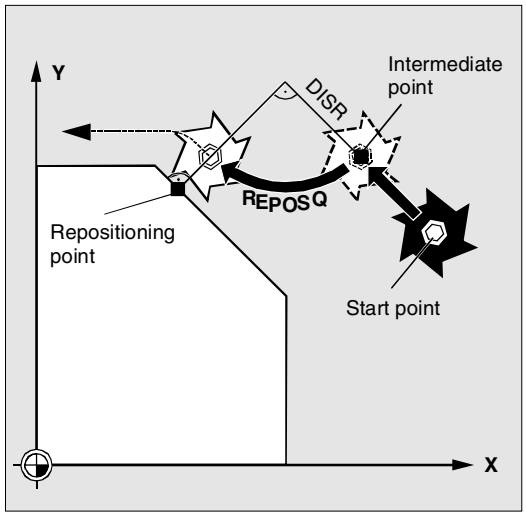


Approach along quadrant, REPOSQ, REPOSQA

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:

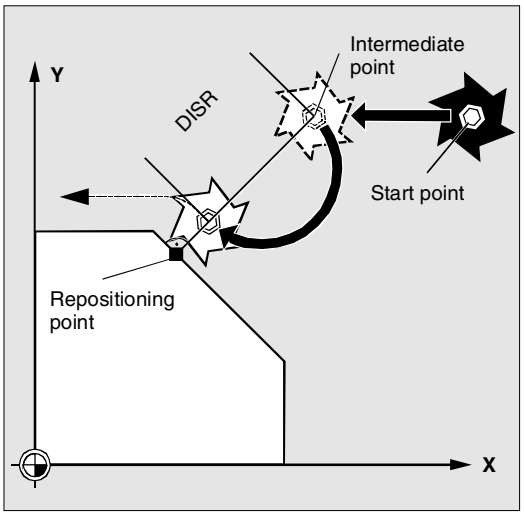
```
REPOSQ RMI DISR=10 F400
```



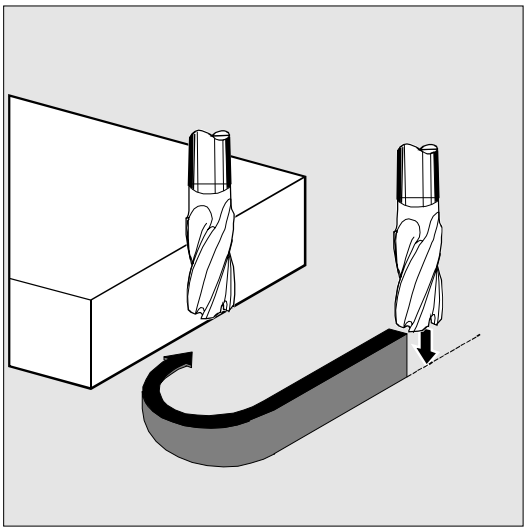
840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

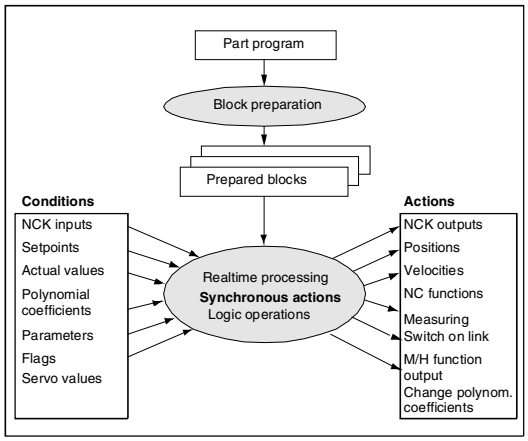
Motion-Synchronous Action

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

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.

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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]	Enable 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.

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

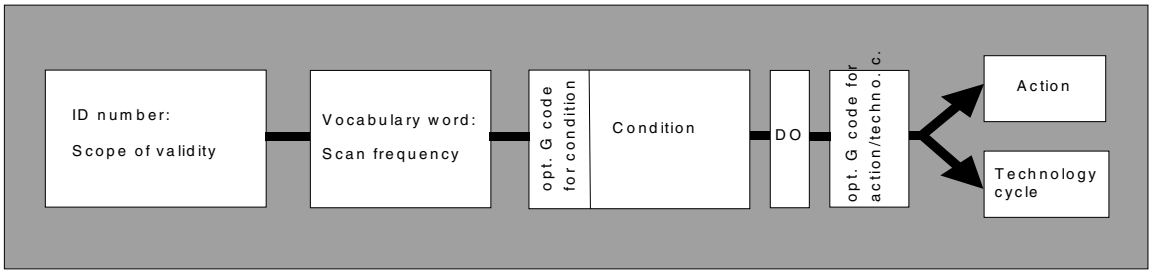
Possible applications:

- 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

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 (ASUP) started by the PLC. In this way, actions can be activated that are executed regardless of the mode selected in the NC.

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

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.
- 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.

Example:

```
DO $A_OUTA[1]=$AA_IN[X]
```

; Output of actual value on analog output

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

- FROM
  The condition is tested in each interpolation cycle until it is fulfilled once. The action is then executed as long as the synchronized 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.
  Pulse edge control:
  The action is initiated again when the condition changes from FALSE to TRUE.

Example:

```
ID=1 EVERY $AA_IM[B]>75 DO
POS[U]=IC(10) FA[U]=900;
```

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.

SW 4 and higher:

Results of comparisons can also be gated by Boolean operators in the condition ().

The condition is tested in interpolation cycles. If it is fulfilled, the associated action is executed.

SW 5 and higher:

Conditions can be specified with a G code. This means that it is possible to have defined settings for condition evaluation and the action/technology cycle irrespective of the currently active part program state. It is necessary to decouple synchronized actions from the programming environment because synchronized actions are to execute their actions in the defined initial state at any time when the trigger conditions are fulfilled.

Application cases:

Defining the measurement systems for condition assessment and action via G codes G70, G71, G700, G710.

In SW 5 only these G codes are allowed.

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

A specified G code for the condition applies for assessment of the condition **as well as** for the action if there is no separate G code specified for the action.

Only one G code of the G code group may be programmed for each condition part.

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

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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. This G code specifies another G code from the one set for the condition for all actions in the block and technology cycles if necessary. If there are technology cycles in the action part, then after completion of the technology cycle the G code continues to apply modally for all subsequent actions 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 M, S and H auxiliary functions
- STOPREOF: Cancel preprocessing stop
- FTOC: Online tool offset
- Definition of evaluation functions (polynomials)
- SYNFCT: Activate evaluation functions: AC 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: Set actual value
- Activate or deactivate coupled-axis motion/leading value coupling

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

- Measurement
- 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.

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.1.5 Overview of synchronized actions

SW 3.x and lower

- 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 synchronized 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

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

- Activation, deactivation, parameterization of axis coupling: Leading value coupling, coupled-axis motion
- Activation/deactivation of axial measuring function
- Software cams
- Delete distance-to-go without stopping preprocessing
- 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

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

Calculations in real time

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)

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

- **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 cannot 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])
```

840 D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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) DO $AA_OVR[S1]=80`	Selection of a position window
`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

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

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], as from SW 4

Function

(not 840D NCU 571, FM-NC)

The system variable $AC_TIMER[n] allows actions to be started following defined waiting times.
Data type: REAL
Units: s
n: Number of the timer variable

- **Set timer**
  A timer variable is incremented via value assignment $AC_TIMER[n]=value

840D
NCU 572
NCU 573

810D
CCU2

840Di

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. When the timer is stopped by assigning the value –1, the most up-to-date timer value is retained 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 per channel are defined using machine data MD 28254: MM_NUM_AC_PARAM.
The parameters are available once per channel under the same name. The $AC_PARAM flags are stored in the dynamic memory.

840D NCU 572 NCU 573 | 810D CCU2 | 840Di

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`	
`STOPRE`	Preprocessing stop
`IF R10>20`	Evaluation of the arithmetic variable

840 D
NCU 572
NCU 573

810 D
CCU2

840Di

10.3.5 Machine and setting data read/write, as from SW 4

Function

From SW 4 upwards, 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
```

;In this example, reverse position 2 for oscillation is addressed assumed to be unmodifiable.

- **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 only appropriate to modify MD and SD from the synchronized action when the change is active **immediately**. 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
```

;Changing the switching position of SW cams. Note: The switching positions must be changed two to three interpolation cycles before they reach their position.

840D
NCU 572
NCU 573

810D
CCU2

840Di

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:

- Cyclic measurement
- Pass execution

Each element can be accessed in read or write mode.
The number of available FIFO variables is defined using machine data MD 28260: NUM_AC_FIFO.
The number of values that can be written into an FIFO variable is defined using the machine data MD 28264: LEN_AC_FIFO. All FIFO variables are of the same length.

Indices 0 to 5 have a special significance:

n=0: While writing: New value is stored in FIFO
While reading: Oldest element is read and removed from FIFO
n=1: Accessing the oldest stored element
n=2: Accessing the most recently stored element
n=3: Sum of all FIFO elements
n=4: Number of elements available in FIFO.
Read and write access to each element is possible.
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

840 D
NCU 572
NCU 573

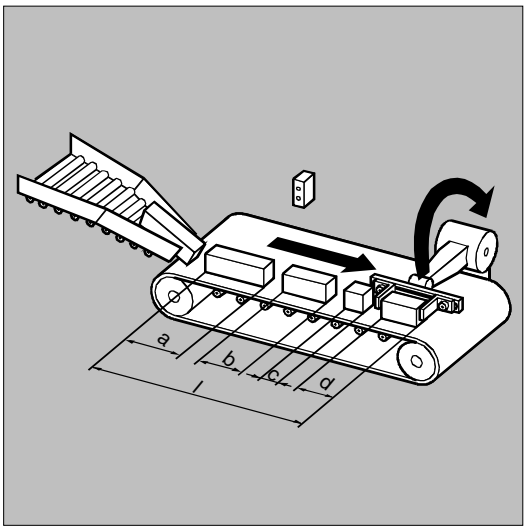
810 D
CCU2

840Di

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 out the product length out of the FIFO.

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

10.4 Actions within 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 G1 X20
F250
```

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 function
- 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 5 mm, auxiliary functions M172 and H510 are output to the PLC.

840 D NCU 571	840 D NCU 572 NCU 573	FM-NC	810 D CCU2	840Di

10.4.2 Read-in disable set 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`	
`...`	

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

10.4.3 Preprocessing stop cancel 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 the end of the block.

`WHEN $AC_DTEB<5 DO STOPREOF`	;Cancel the preprocess stop when distance to block end is less than 5 mm.
`G01 X100`	;The preprocessing stop is cancelled after execution of the linear interpolation.
`IF $A_INA[7]>500 GOTOF MARKER1=X100`	;If the voltage 5 V is exceeded at input 7, jump to label 1.

840 D NCU 571	840 D NCU 572 NCU 573	FM-NC	810 D CCU2	840Di

10.4.4 Delete 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, DELTG (axis1,..)

Function

Prepared delete distance-to-go with DELTDG 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.

The distance-to-go can be retrieved with the system variable $AC_DELT or $AC_DELT[axis].

840 D	840D	FM-NC	810D	840Di
NCU 571	NCU 572 NCU 573		CCU2	

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 cancelled
N110 G01 X…
IF $AC_DELT>50…
```

Programming example

Rapid axial delete distance-to-go

```
POS[X1]=100 G1 Z100 F1000
```

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 8 V, delete distance-to-go for axis X1.
                Path motion continues.
```

Restriction

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).

840D NCU 571 | 840D NCU 572 NCU 573 | FM–NC | 810D | 840Di

10.4.7 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Notes

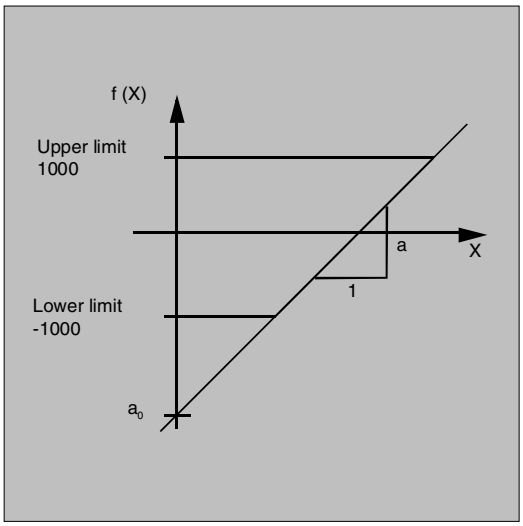
- The system variables can be written from the part program or from a synchronized action. When writing from part programs, program STOPRE to ensure that writing is block synchronized.
- SW 4 and higher:
  The system variables $AC_FCTLL[n], $AC_FCTUL[n], $AC_FCT0[n] to $AC_FCTn[n] can be modified from 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)
```

840D
NCU 572
NCU 573

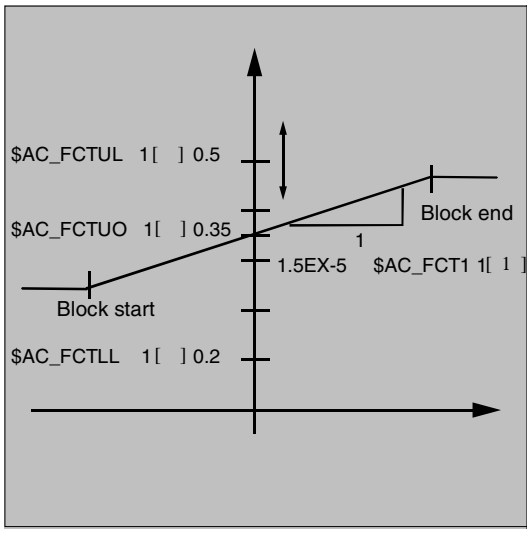
840Di

10.4.8 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)`	
;in dependence 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.

840D
NCU 572
NCU 573

840Di

10.4.9 Evaluation function SYNFCT

Programming

```
SYNFCT(Polynomial_No., real-time variable output, real-time variable
input)
```

Explanation

`Polynomial_No.`	With polynomial defined with FCTDEF (see Subsection "Polynomial definition").
`Real-time variable output`	Write real-time variable
`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 to real-time variables and subjected to upper and lower limits with FCTDEF (see Section 10.4.7).

As real-time variables, variables can be selected and directly included in the processing operation

- with additive influencing
- with multiplicative influencing
- as position offset.

Application

The evaluation function is used

- in AC control (Adaptive Control)
- in laser output control
- with position feedforward

840D
NCU 572
NCU 573

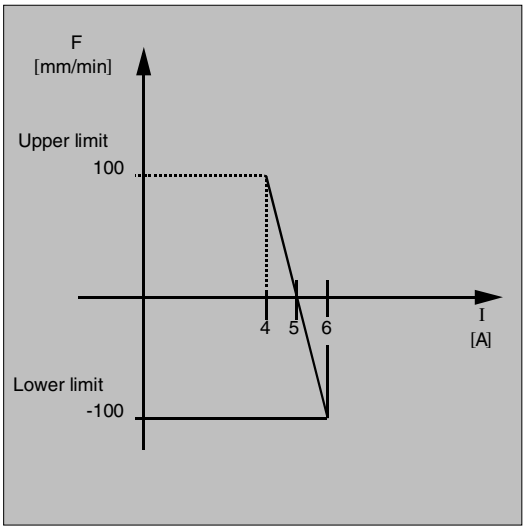
840Di

10.4.10 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

$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$

$a_1 = -100$mm/1 min A

$a_0 = -(-100)*5 = 500$

$a_2 = a_3 = 0$ (not quadratic or cubic element)

Upper limit = 100

Lower limit = –100

Therefore:

```
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.

840D
NCU 572
NCU 573

840Di

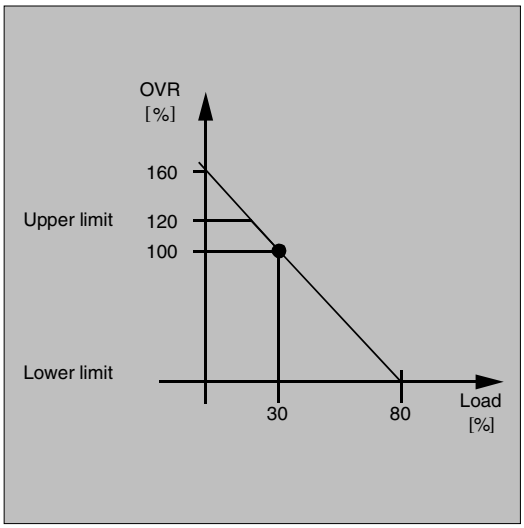
10.4.11 Adaptive control (multiplicative)

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

Therefore:

```
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.

840D
NCU 572
NCU 573

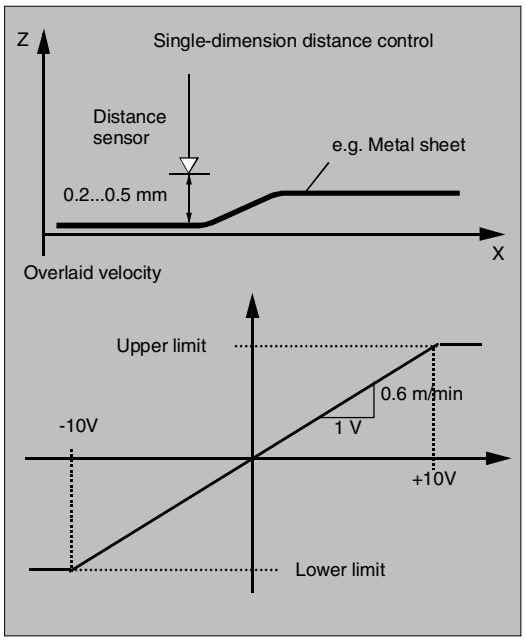
810D 840Di

10.4.12 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:

$$\frac{0.120mm}{0.6ms} / mV = 0.6\frac{m}{\min} / V$$



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 off limits check
`RET`	
`ENDPROC`	

840D
NCU 572
NCU 573

810D 840Di

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.
From SW 4 upwards, it is possible to limit the absolute value to be corrected (real-time variable output) to the variable in setting data SD 43350: AA_OFF_LIMIT.
The manner of overlaying the distance is defined in machine data MD 36750: AA_OFF_MODE:

0 Proportional valuation
1 Integrating valuation

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 within limits
1 Limit of offset value reached in the positive direction
–1 Limit of the offset value reached in the negative direction

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.13 Online tool offset FTOC

Programming

```
FTOC(Polynomial_No., RV, Length_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

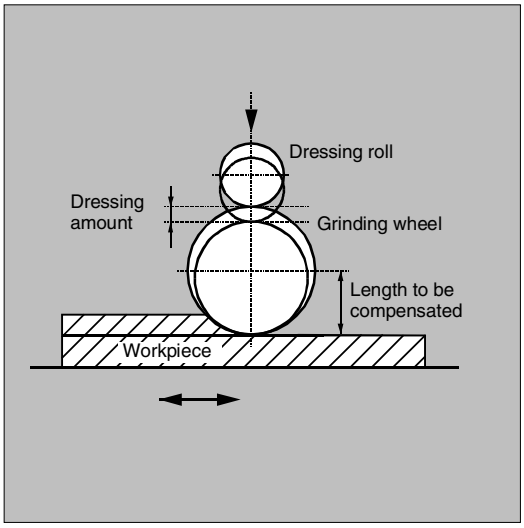
FTOC permits overlaid movement for a geometry axis after a polynomial programmed with FCTDEF depending on a reference value that might, for example, be the actual value of an axis.
This means that you can also program modal, Online tool compensations or clearance controls as synchronized actions.

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".

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 compensation: 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 for dressing
`G1 V-0.05 F0.02`	
`...`	
`CANCEL(1)`	Deselect online offset
`...`	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.14 Positioning movements

Function

Axes can be positioned completely asynchronizedly with respect to the part program from synchronized actions. The programming of 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 part program context.

Example 1

`N100 R1=0`		
`N110 G0 X0 Z0`		
`N120 WAITP(X)`		
`N130 ID=1 WHENEVER $R==1 DO POS[X]=10`		
`N140 R1=1`		
`N150 G71 Z10 F10`	Z=10 mm	X=10 mm
`N160 G70 Z10 F10`	Z=254 mm	X=254 mm
`N170 G71 Z10 F10`	Z=10 mm	X=10 mm
`N180 M30`		

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Example 2

`N100 R1=0`		
`N110 G0 X0 Z0`		
`N120 WAITP(X)`		
`N130 ID=1 WHENEVER $R==1 DO G71 POS[X]=10`		
`N140 R1=1`		
`N150 G71 Z10 F10`	Z=10 mm	X=10 mm
`N160 G70 Z10 F10`	Z=254 mm	X=10 mm (X is always positioned to 10 mm)
`N170 G71 Z10 F10`	Z=10 mm	X=10 mm
`N180 M30`		

Programming example

Disabling 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 until digital input 1 =0

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.15 Position axis POS

Function

```
POS[axis]=value
```

Unlike programming from the part program, the positioning axis movement has no effect on execution of the part program.

Explanation

`Axis:`	Name of the axis to be traversed
`Value:`	Value to be traversed

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.

10.4.16 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 movement

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

10.4.17 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
```

10.4.18 SW 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 part program has no effect on the command axes.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.19 Axis coordination

Function

Typically, an axis is either moved from the part 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 part program as a path or positioning axis and from synchronized actions, however, a coordinated transfer takes place between both axis movements.
If a command axis is subsequently traversed from the part program, preprocessing must be reorganized. This, in turn, causes an interruption in the part program processing comparable to a preprocessing stop.

Programming example

Move the X axis from either the part program or the synchronized actions:

`N10 G01 X100 Y200 F1000`	X axis programmed in the 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)`	Deselect synchronized action
…	
`N100 G01 X240 Y200 F1000`	
;X becomes the path axis; before motion, delay occurs because of axis transfer if digital input was 1 and X was positioned from the synchronized action.	

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 >= 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.

840D
NCU 572
NCU 573

810D
CCU2

840Di

10.4.20 Set actual value

Function

When PRESETON (axis, value) is executed, the current axis position is not changed but a new value is assigned to it.

Notes

PRESETON can be executed from within a synchronized action in the following cases:

- Modulo rotary axes that have been started from the part program
- All command axes that have been started from the synchronized action

Restriction:
PRESETON is not possible for axes that participate 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.
```

Restriction

One and the same axis can by moved from the part program and from a synchronized action, only at different times. For this reason, delays can occur in the programming of an axis from the part program if the same axis has been program in a synchronized action first.
If the same axis is used alternately, transfer between the two axis movements is coordinated. Part program execution must be interrupted for that.

840D
NCU 572
NCU 573

810D
CCU2

840Di

10.4.21 Spindle motions

Function

Spindles can be positioned completely asynchronizedly with respect to the part program from synchronized actions. This type of programming is advisable for cyclic sequences or operations that are strongly dependent on events.

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

840D
NCU 572
NCU 573

810D
CCU2

840Di

10.4.22 Coupled-axis 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 asynchronized coupled motion:

```
... DO TRAILON(FA, LA, CF)
```

where: FA: Following axis
LA: Leading axis
CF: Coupling factor

Deactivate asynchronized coupled motion:

```
... DO TRAILOF(FA, LA, LA2)
```

where: FA: Following axis
LA: Leading axis
LA2: Leading axis 2, optional

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 combined axis pair
`G0 Z10`	Infeed of Z and W axes in opposite axis directions
`G0 Y20`	Infeed of Y and V axes in same axis directions
`...`	
`G1 Y22 V25`	Superimpose dependent and independent movement of coupled-motion axis "V"
`...`	
`TRAILOF(Y,V)`	Deactivate 1st coupled axis
`TRAILOF(Z,W)`	Deactivate 2nd coupled axis

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.23 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 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

A continuous material that runs continuously through the work area of parting device is to be separated into pieces of equal length.

X axis: Axis in which the continuous material runs. WCS
X1 axis: Machine axis of the continuous material, MCS
Y axis: Axis in which the parting device "travels" with the continuous material

It is assumed that the positioning and control of the parting tool is controlled by the PLC. The signals of the PLC interface can be evaluated for the purpose of determining the degree of synchronism between the continuous material and the parting tool.

Actions: Activate coupling, LEADON
Deactivate coupling, LEADOF
Set actual value, PRESETON

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

```
%_N_SHEARS1_MPF
;$PATH=/_N_WCS_DIR/_N_DEMOFBE_WPD
N100 R3=1500                                  ;Length of a section to be parted
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 pair
N900 X=R13 Y=R13
N1000 X=2*R13 Y=30
N1100 CTABEND                                 ;End of table definition
N1200 PRESETON(X1,0)                          ;PRESET to begin
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 $R1==1 DO MOV[X]=1            ;Place material axis in continuous motion
FA[X]=$R4
N2200 M30
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.24 Measurement

Compared with use in traverse blocks of the part 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, measurement memory, trigger event_1, ..._4

For further information on measuring: see Chapter 5, "Extended Measuring Function"

10.4.25 Wait markers set/clear: SETM, CLEARM

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 part program and in the action part of a synchronized action. It sets the marker MarkerNumber for the channel in which the command executes.
CLEARM
The CLEARM command can be written in the part program and in the action part of a synchronized action. It resets the flag MarkerNumber for the channel in which the command executes.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.4.26 Error 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 $AA_OVR[X2]=0
                ;If the safety distance between axes X1 and X2 is to small, stop axis X2.
ID=67 WHENEVER ($AA_IM[X1]-$AA_IM[X2])<4.567 DO SETAL(61000)
                ;If the safety distance between axes X1 and X2 is to small, set an alarm.
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 of this type 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.

The program end is programmed with M02/M17/M30/RET. A maximum of one axis movement per block can be programmed.

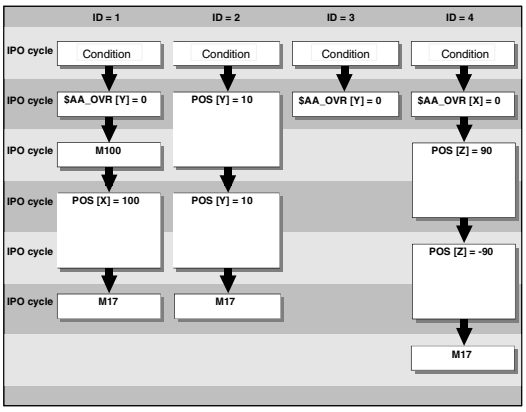
Application

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 part program is now only used for the management of synchronized actions in extreme cases.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Technology cycles are started as soon as their conditions are fulfilled. In the case of positioning axes, several interpolation cycles are necessary for their execution. 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)
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 cancelled directly from the part 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 cancelled synchronized action are completed as programmed.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

10.7 Supplementary conditions

- **Power ON**
  With Power ON no synchronized actions are active.
  However, static synchronized actions can be activated on Power On with an asynchronized subroutine (ASUP) 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.
  Synchronized actions of the WHEN type that have already been executed are not executed again following RESET.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Response following RESET		
Synchronized action / technology cycle	**Modal/non-modal**	**Static (IDS)**
	Active actions are reset, synchronized actions are cancelled	Active action is cancelled, technology cycle is reset
Axis / positioning spindle	Movement is reset	Movement is reset
Speed-controlled spindle	$MA_SPIND_ACTIVE_AFTER_RESET==1: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==0: Spindle is stopped.	$MA_SPIND_ACTIVE_AFTER_RESET==1: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==0: Spindle is stopped.
Leading value coupling	$MC_RESET_MODE_MASK, Bit13 == 1: Leading value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: Leading value coupling is disconnected	$MC_RESET_MODE_MASK, Bit13 == 1: Leading value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: Leading value coupling is disconnected
Measuring procedures	Measurements started from synchronized actions are cancelled.	Measurements started from static synchronized actions are cancelled.

- **NC Stop**
  Static synchronized actions remain active on NC stop. Movements started from static synchronized actions are not cancelled.
  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 this, cancel with CANCEL before the end of the program (see previous subsection).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.	Movement continues
Speed-controlled spindle	End of program: $MA_SPIND_ACTIVE_AFTER_RESET==1: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==0: Spindle is stopped Spindle remains active following a change in operating mode	Spindle remains active
Leading value coupling	$MC_RESET_MODE_MASK, Bit13 == 1: Leading value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: Leading value coupling is disconnected	A coupling started from a static synchronized action remains
Measuring procedures	Measurements started from synchronized actions are cancelled.	Measurements started from static synchronized actions remain active.

- **Block search**
  Synchronized actions found during a block search Block searchare 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**
  ASUP start:
  Modal and static motion-synchronized actions remain active and are also active in the asynchronized subroutine.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

- **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 cancelled 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 spark-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 path movement or a positioning axis can be used to achieve a constant infeed, however, there is **no relationship** between the oscillating movement and the infeed movement.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

The oscillating axis

For the oscillating axis the following applies:

- Any axis can be used as an oscillating axis.
- Several oscillating axes can be active simultaneously (maximum: 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)
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 asynchronized oscillationSetting datacan 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
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 block-synchronizedly during the main run and remain active until changed again.

Activate, deactivate 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 is continued without waiting 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 synchronized 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

This timing is affected by the feedrate override.
Under certain circumstances, an oscillating stroke is performed before the spark 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).

Set options
These options are switched over. When OSE (end position) is programmed, option 4 is implicitly activated.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 cancelled by deletion of the distance-to-go: then execute spark-out strokes and approach end position if appropriate
16	If the oscillation movement is cancelled 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)
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming example

Oscillating axis Z is to oscillate between 10 and 100. Approach reversal point 1 with exact stop fine, reversal point 2 with exact stop coarse. Machining takes place 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 feed for the infeed axis is 1, the end of the infeed in the X direction is at 15.

`WAITP(X,Y,Z)`	Starting position
`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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

11.2 Oscillation controlled via synchronized actions

Programming

1. **Define parameters for oscillation**
2. **Define motion-synchronized actions**
3. **Assign axes, define infeed**

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 spark-out strokes
`OSE[OscillationAxis]=`	End position
`WAITP(OscillationAxis)`	Enable axis for oscillation

Axis assignment, infeed

```
OSCILL[OscillationAxis] = (InfeedAxis1, InfeedAxis2, InfeedAxis3)
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-synchronized actions

`WHEN… … DO`	when ... , do
`WHENEVER … DO`	whenever ... , do

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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**.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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[InfeedAxis] = (EndPosition, Part, 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 area 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 area 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%.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Infeed at reversal point

As long as the oscillating axis has not reached the reversal point, no movement takes place on the infeed axis.

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%.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Stop oscillation motion 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.

This synchronized action can also 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	the position of reversal point 1
then	set the axial override of the oscillating 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	the position of reversal point 2
then	set the axial override of the oscillating axis to 0%
and	set the axial override of the infeed axis to 100%.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

Restart oscillation movement

This synchronized action is used to continue the oscillating movement when the partial infeed movement is complete.

The following instructions are used subject to the above assumptions:

```
WHENEVER $AA_DTEPW[X]==0 DO $AA_OVR[Z]= 100
```

Whenever	the distance-to-go for the partial infeed on infeed axis X in the WCS
equal to	zero
then	set the axial override of the oscillating axis to 100%.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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. A synchronized action is then used to inhibit the next infeed movement.

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

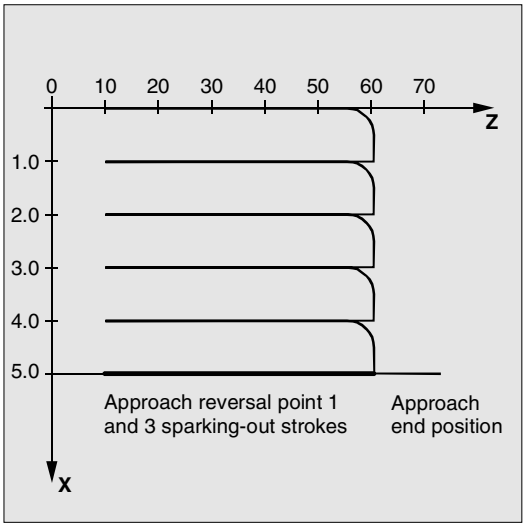
```
WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0
```

Whenever	marker 1 is
equal to	1,
then	set the axial override of the infeed axis to 0%.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Programming example

No infeed is to 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 oscillating axis and axis X the infeed axis.



Program extract

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 spark-out strokes
`OSE[Z]=70`	End position = 70
`ii2=2`	Set reversal area
`WAITP(Z)`	Enable oscillation for Z axis

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

2. Motion-synchronized 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	the position of reversal point 2
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 partial infeed is
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 marker with index 0 is
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 separate 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Notes

,

Punching and Nibbling

840 D
NCU 572
NCU 573

840Di

12.1 Activation, deactivation

12.1.1 Language commands

Programming

```
PDELAYON
PON G... X... Y... Z...
PONS G... X... Y... Z...
PDELAYOF
SON G... X... Y... Z...
SONS G... X... Y... Z...
SPOF
```

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 on with delay
PDELAYOF	Punching off with delay

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.

840 D
NCU 572
NCU 573

840Di

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.

Otherwise PONS and SONS work in exactly the same way as PON and SON.

Punching with delay
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.

840 D
NCU 572
NCU 573

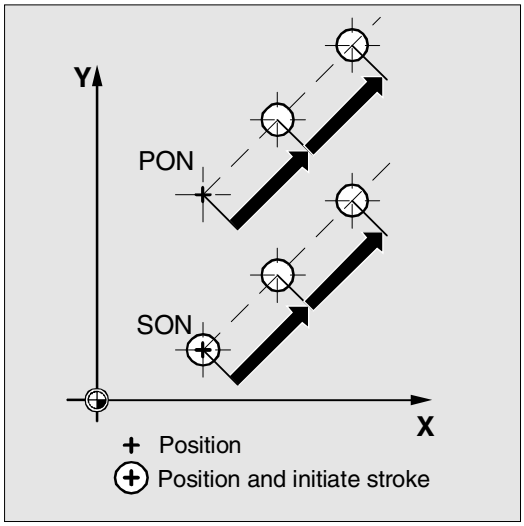
840Di

Initiation of stroke

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.

Additional notes

Machining with rotatable tools
Use the tangential control function if you wish to position rotatable tools at a tangent to the programmed path.

840 D
NCU 572
NCU 573

840Di

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 on with delay
`DEFINE M20 AS SPOF`	Punching, nibbling off
`DEFINE M23 AS SPOF`	Punching, nibbling off

840 D
NCU 572
NCU 573

840Di

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.

840 D
NCU 572
NCU 573

840Di

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.

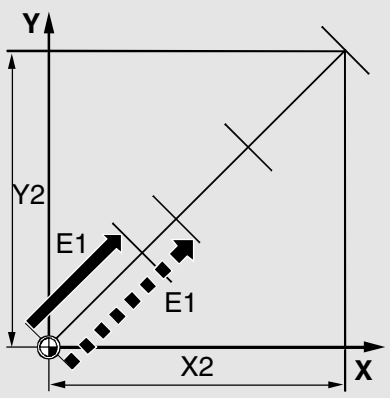
The command is deactivated with SPOF or SPP=0.

Example:
N10 G1 SON X0 Y0
N20 **SPP=2** X10

In this example, the total traversing distance of 10 mm is divided into 5 path segments of 2 mm (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

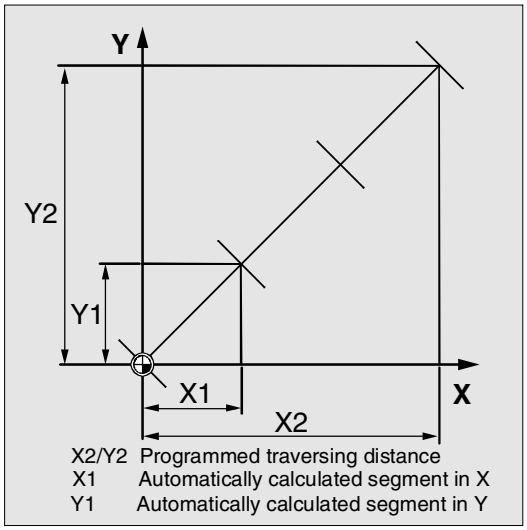
When the total traversing distance is 15 mm and the path segment length 3.5 mm, the quotient is not an integer value (4.28).
In this case, the SPP value is reduced down to the next possible integer quotient. The result in this example would be a path segment length of 3 mm.

840 D
NCU 572
NCU 573

840Di

Number of SPN path segments
SPN defines the number of path segments to be generated from the total traversing distance. The length of the segments is calculated automatically.

Since SPN is non-modal, punching or nibbling must be activated beforehand with PON or SON respectively.



SPP and SPN in the same block
If you program both the path segment length (SPP) and the number of path segments (SPN) in the same block, then SPN applies to this block and SPP to all the following blocks.

If SPP was activated before SPN, then it takes effect again after the block with SPN.

Additional notes

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.

840 D
NCU 572
NCU 573

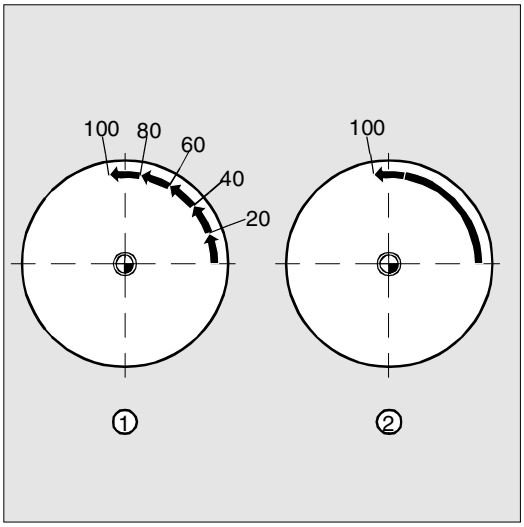
840Di

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. Default setting
The path traversed by the single axis is distributed evenly among the intermediate blocks generated by SPP.

Example:
N10 G1 SON X10 A0
N20 SPP=3 X25 A100

As a result of the programmed distance between strokes of 3 mm, five blocks are generated for the total traversing distance of the X axis (path axis) of 15 mm.
The A axis thus rotates through 20° in every block.



2. Single axis without path segmentation
The single axis traverses the total distance in the first of the generated blocks.

3. With/without path segmentation
The response of the single axis depends on the interpolation of the path axes:
- Circular interpolation: With path segmentation
- Linear interpolation: Without 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.
Precondition: The single axis is defined as a punching/nibbling axis.

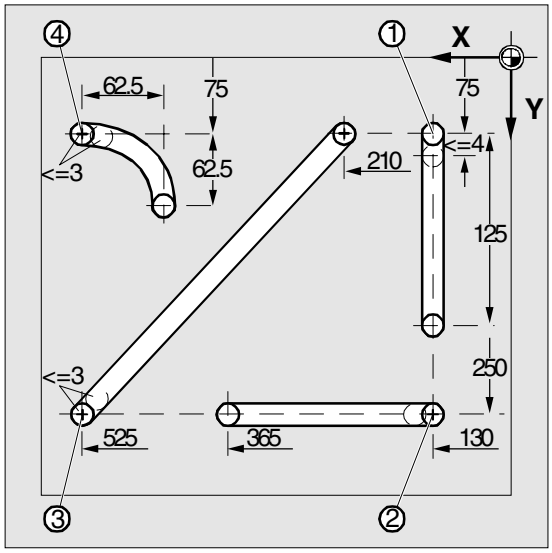
840 D
NCU 572
NCU 573

840Di

12.2.3 Programming examples

Programming example 1

The programmed nibbling paths must be divided automatically into equidistant path segments.



Program extract

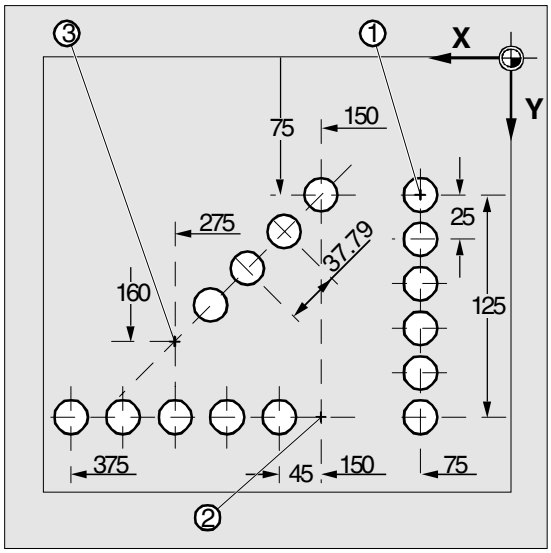
`N100 G90 X130 Y75 F60 SPOF`	Position at starting point 1
`N110 G91 Y125 SPP=4 SON`	Nibbling on, maximum path segment length for automatic path segmentation: 4 mm
`N120 G90 Y250 SPOF`	Nibbling off, position at starting point 2
`N130 X365 SON`	Nibbling on, maximum path segment length for automatic path 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

840 D
NCU 572
NCU 573

840Di

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 extract

`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

840 D
NCU 572
NCU 573

840Di

Notes

Additional Functions

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

13.1 Axis functions AXNAME, SPI, ISAXIS

Programming

```
AXNAME("TRANSVERSE AXIS")
AX[AXNAME("String")]
SPI(spindle number)
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 a spindle number to an axis identifier. The parameter transferred must contain a valid spindle number.
AX	Variable axis identifier
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.

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:
```

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

13.2 Learn compensation characteristics: QECLRNON, QECLRNOF

Explanation of the commands

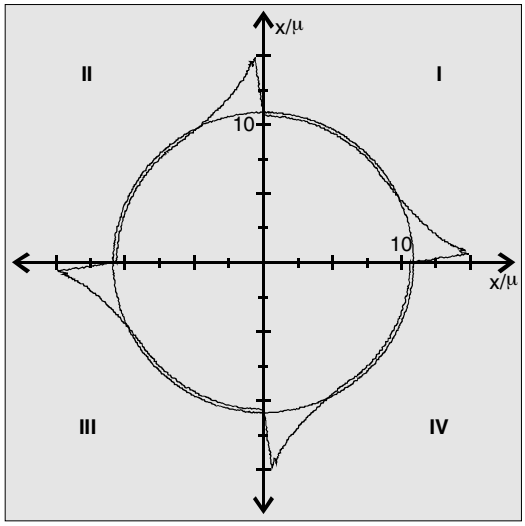
`QECLRNON (axis.1,…4)`	Activate "Learn quadrant error compensation" function
`QECLRNOF`	Deactivate "Learn quadrant error compensation" function

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:

840D NCU 571 | 840D NCU 572 NCU 573 | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

13.3 Synchronized spindle

Programming

```
COUPDEF (FS,LS,SRFS,SRLS, block change beh., coupling)
COUPDEL (FS,LS)
COUPRES (FS,LS)
COUPON (FS,LS,PSFS)
COUPOF (FS,LS,POSFS,POSLS)
WAITC (FS, block change beh., FS, 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"	Immediate (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

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

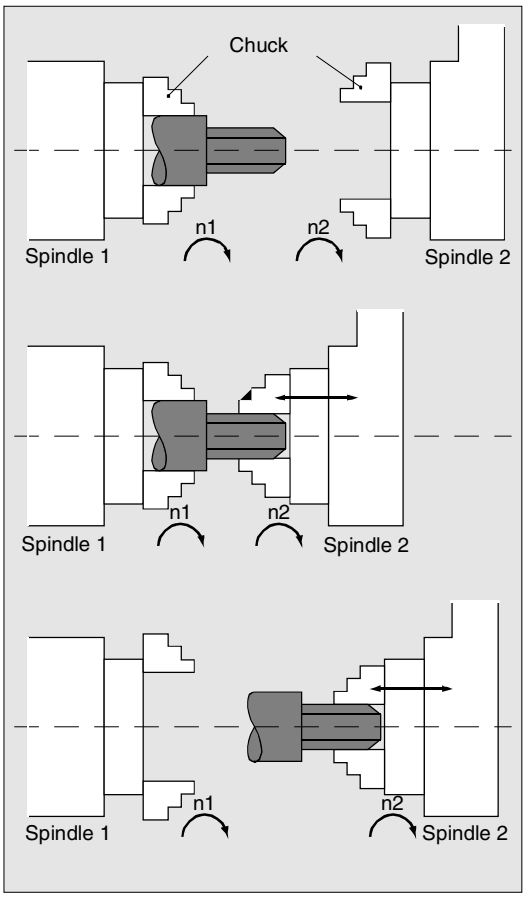
Function

In synchronized mode, there is a leading spindle (LS) and a following spindle (FS). They are referred to as the **synchronized 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.

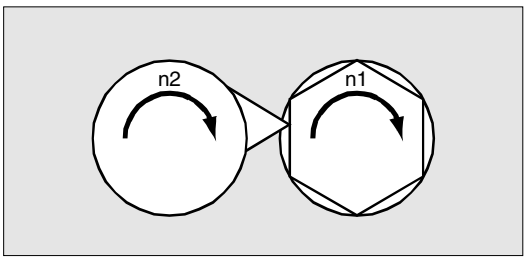
This function enables turning machines to perform workpiece transfer from spindle 1 to spindle 2 on-the-fly, e.g. for final machining. This avoids downtime caused, for example, by rechucking.

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 $k_Ü$ can also be specified between the main spindle and a "tool spindle" for multi-edge machining (polygon turning).



The synchronized spindle pair can be defined permanently for each machine with channel-specific machine data or defined by the user in the CNC part program.
Up to two synchronized spindle pairs can be operated simultaneously on each NC channel.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

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 part program. The coupling can nevertheless be parameterized in the NC part 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 part programs. If a new coupling relationship is to be defined, any existing user-defined coupling must be deleted with COUPDEL.

Define 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,$Ü_{FS}$,$Ü_{LS}$)
Meaning:
S2 = following spindle, S1 = leading spindle

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

Positioning 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 :
Speed, 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)] or 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

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 $k_Ü$

The speed ratio is defined with parameters for FS (numerator) and LS (denominator).

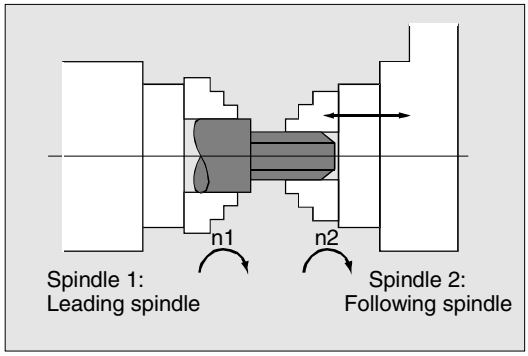
Options:

- 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
- The following and leading spindles rotate at different speeds
  ($n_{FS} = k_Ü \bullet n_{LS}$; $k_Ü \bullet 1$)
  Application: Multi-sided 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 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.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

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)

It is sufficient to specify the characters typed in bold when specifying the block change method.

The block change method is modal!

Coupling type

"DV"	Setpoint coupling between FS and LS (default)
"AV"	Actual-value coupling between FS and LS

The coupling type is modal.

Caution

The coupling type may only be changed when the coupling is deactivated!

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

Activate 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.

Deactivate 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)
```

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

Delete couplings, COUPDEL

An existing user-defined synchronized spindle coupling must be deleted if a new coupling relationship is to be defined and all user-configurable couplings (1 or 2) are already defined.

```
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.

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

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

System variables

Current coupling status following spindle

The current coupling status of the following spindle can be read in the NC part 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 which is read has the following meaning for the following spindle:
0: No coupling active
4: synchronized spindle coupling active

Current angular offset

The setpoint of the current position offset of the FS to the LS can be read in the part 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 part program.

840D NCU 571 | 840D NCU 572 NCU 573 | 840Di

Programming example

Working with master and slave spindles.

	;Leading spindle = master spindle = spindle 1
	;Slave spindle = spindle 2
`N05 M3 S3000 M2=4 S2=500`	;Master spindle rotates at 3000 rpm, slave spindle at 500 rpm
`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 rpm. With project 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`	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.4 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 the leading axis and the following axis are defined by the coupling factor for each leading axis.
The following axis motion part is calculated by an addition of the individual leading axis motion parts multiplied by their respective coupling factors.
When activating an EG axis grouping, the following axis can be sychronized according to a defined position.
A gear group can be

- defined,
- activated,
- deactivated, and
- 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 the leading axes

13.4.1 Define 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 axis 1, coupling type 1, leading axis 2, coupling type 2, ...)

Explanation

`Following axis`	Axis that is influenced by the leading axes
`Leading axis1, ... leading axis5`	Axes that influence the following axis

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

`Coupling type1, ... coupling type5`	Following axis is influenced by: 0: actual value 1: setpoint of the respective leading axis

Programming

`EGDEF(C, B,1, Z, 1, Y, 1)`	B, Z, Y inflluence C via setpoint

The coupling type does not need to be identical for all leading axes and is therefore specified for each leading axis individually.
The coupling factors are preset with zero for definition of the EG coupling group.
Prerequirement for an EG axis grouping definition:
An axis coupling may not yet be defined for the following axis (if necessary, any existing one must first be deleted with EGDEL).

Note

EGDEF triggers preprocessing stop.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.4.2 Activate electronic gear

There are two 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

It is only permissible to program the leading axes which have previously been specified with EGDEF. At least one leading axis must be programmed. The positions of the leading axes and following axis at the time of activation are saved as "synchronized positions". The "synchronized positions" can be read via 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])
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

It is only permissible to program leading axes that have previously been specified with EGDEF.
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 synchronized state when it is activated, the following axis will traverse 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 (so-called relative synchronization: e.g. the next tooth gap). The synchronized position is only approached if "Enable following axis override" interface signal DB(30 + axis number), DBB26 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.4.3 Deactivate 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 decelerated until it is motionless.
The call triggers 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 leading axis must be specified. The influence of the specified leading axes on the following axis is selectively disabled.
The call triggers preprocessing stop.
If leading axes are still active, the following axis will continue to operate under their control. If all leading axis influences have been disabled in this manner, the following axis is decelerated until it reaches a standstill.

Variant 3:

```
EGOFC(following spindle)
```

The electronic gear is deactivated. The following spindle continues to operate with the current speed that was valid at the time of deactivation.
The call triggers preprocessing stop.

Note

This functions is only allowed for spindles.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.4.4 Delete 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 coupling definition 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.
The call triggers preprocessing stop.

13.4.5 Revolutional feedrate (G95)/electronic gear (SW 5.2)

In SW 5 and higher, using the FPR() command, it is also possible to define the following axis of an electronic gear as the axis determining the revolutional feedrate. The following applies in this case:

- The feed is dependent on the setpoint speed of the following axis of the electronic gear.
- The setpoint speed is calculated from the speed of the leading spindles and modulo leading axes (that are not path axes) and their assigned coupling factors.
- Speed parts of linear or non-modulo leading axes and overlaid movement of the following axis are not taken into account.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.4.6 Response of EG at Power ON, RESET, mode change, block search

After Power ON there are **no** active couplings.
Active couplings are retained after reset and mode change.
With block search, commands for switching, deleting and defining the electronic gear are not executed or retained, instead they are skipped.

13.4.7 The electronic gear's system variables

By means of the electronic gear's system variables, the part program can determine the current states of an EG axis grouping and react to them if required.

Additional notes

The system variables for the electronic gear are listed in the Annex. They are characterized by names beginning with:
`$AA_EG_ ...`
or
`$VA_EG_ ...`

13.5 Extended stopping and retract (as of SW 5)

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 three 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

As an independent drive mode, it provides the drive DC link with the necessary power to perform an orderly "stop" and "retract" in the event of a power failure or similar occurrence.

All reactions can be used independently from one another.

For further information, see
/FB/ M 3, Axis Couplings and ESR

13.5.1 Drive-independent reactions

Function

Drive-independent reactions are defined axially; if activated, each drive processes its stop/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

Generator operation is

- Configured: via MD
- 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.

Stop (independent drive)

Drive-independent stop is

- Configured: via MD as well as time specification via MD;
- Enabled ($AA_ESR_ENABLE) and
- Triggered: system variable $AN_ESR_TRIGGER.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Retract (drive-independent)

Drive-independent retract is

- configured: via MD; time specification and return 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.

13.5.2 Possible trigger sources

Function

The following error sources for starting "Extended stop and retract" are possible:

- General sources (NC-external/global or mode group/channel-specific):
    - Digtial inputs (e.g. on NCU modules or terminal blocks) or mapping the digital outputs within the control ($A_IN, $A_OUT)
    - Channel status ($AC_STAT)
    - VDI signals ($A_DBB)
    - Group messages from a number of alarms ($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].

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.5.3 Logic gating functions: Source/reaction 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. Users can evaluate the source system variable as a whole or also selectively by means of bitmasks and gate their desired reactions to them.
The static synchronized 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

13.5.4 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 is "automatically" active in the drive when a pending DC link undervoltage is detected.
- Drive-independent stop and/or retract are active when a communications failure (between the NC and drive) is detected, as well as when a DC link undervoltage is detected in the drive (providing it is configured and enabled).

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

- 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).
- NC-controlled retract via LIFT_FAST

13.5.5 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. Drive deceleration (default speed setpoint = 0) causes regeneration of energy in the DC link.

For more information see
/FB/ M 3, Coupled Motion and Leading Value Coupling

13.5.6 Drive-independent stop

Function

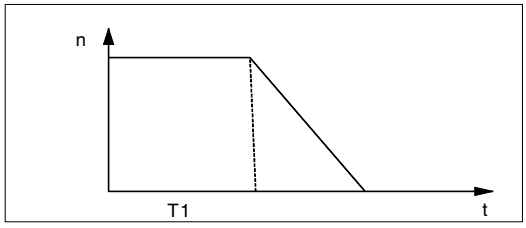
The drives of a previously coupled grouping can be stopped by time-controlled cutout delay keeping the difference between them to a minimum, if the control is unable to achieve this.
Drive-independent stop is configured and enabled via MD (delay time T1 in MD) and is enabled by system variable $AA_ESR_ENABLE and started with $AN_ESR_TRIGGER.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Reactions

For time T1 the speed setpoint that was active when the error occurred is still output. This is an attempt to maintain the movement that was active before the failure until the physical contact is annulled or the retraction movement initiated simultaneously in other drives is completed. This can be necessary for all leading/following drives or for drives that are coupled or in a grouping.

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).



13.5.7 Drive-independent retract

Function

Axes with digital 611D drives can (if configured and released) also execute a retraction movement independently

- at control failure (sign-of-life detection)
- if the DC link voltage falls below a warning threshold
- if triggered by the system variable $AN_ESR_TRIGGER.

The retraction movement is performed independently by drive 611D.

Once the retraction phase is initiated, the drive independently maintains its enables at the values that were previously valid.

For more information see /FB/ M 3, Coupled Motion and Leading Value Coupling

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.5.8 Example: Using the drive-independent reaction

Example configuration

- Axis A is to operate as generator drive,
- axis X is to retract by 10 mm at maximum speed in event of an error and
- axes Y and Z are to stop with a time delay of 100 ms, such that the retraction axis has 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:
   MD1639 RETRACT_SPEED[X] =400000H in pos. direction (max. speed),
   =FFC00000H in neg. direction,
   D1638 RETRACT_TIME[X] =10ms (retract time),
   MD1637 GEN_STOP_DELAY[Y] =100ms,
   MD1637 GEN_STOP_DELAY[Z] =100ms,
   MD1635 GEN_AXIS_MIN_SPEED[A] =Generator min. speed (rpm).
4. Function enable (from part program or synchronized actions):
   $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 synchronized action(s), e.g.:
   - dependent on intervention of the 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

- and dependent on EU synchronization monitoring (if e.g. Y is defined as EU following axis and the maximum permissible synchronization difference is to be 100 µm):
  IDS=03 WHENEVER ABS($VA_EG_SYNCDIFF[Y])>0.1
  DO $AN_ESR_TRIGGER=1

13.6 Link communication (SW 5.2 and higher)

Function

The NCU link, which connects several NCU units from an installation, is used in configurations with a distributed system design. 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

- Relatively small data volume
- Very high transfer speed,
  therefore: Use is intended for time-critical information.
- These system variables can be accessed from the **part program** and from **synchronized actions**. The size of the memory area for global NCU system variables is 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 type in question, 1, 2, 4 or 8 bytes are addressed when the link variables are written/read.
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 up.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Value ranges

The different 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 sharing access to the link memory **at the same time** must use the link memory in a **uniform manner**. When the process is completely separate in time, the link memory can be occupied differently.

Caution

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

13.7 Axis container (SW 5.2 and higher)

Function

With revolving machines/multi-spindle machines the axes holding the workpiece move from one machining station 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. The **axis container** is 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 compiles the possible connections with all clamping axes/spindles, of which only exactly **one** is always **activated** for the machining station.
The following can be assigned via axis containers:

- 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 **part program**.
The axis containers with link axes are a tool that is valid across NCUs (NCU global) and is coordinated by the control.
It is also possible to have axis containers in which only local axes are managed.

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 (CT1, CT 2, ...)`	AXIS CONTAINER SWITCH ENABLE
`AXCTSWED(CT1, CT 2, ...)`	AXIS CONTAINER SWITCH ENABLE DIRECT

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

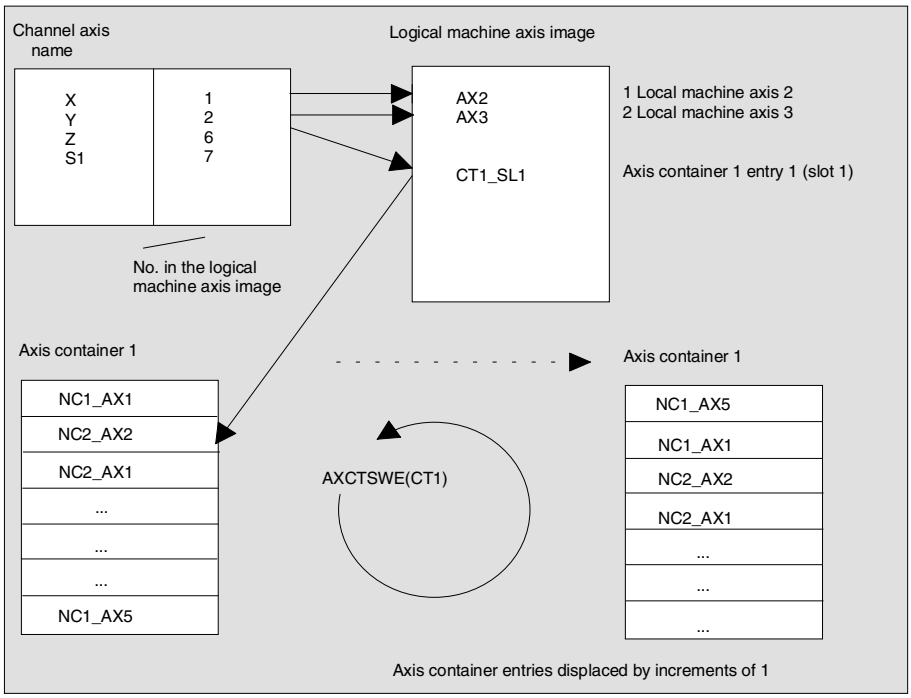
Explanation

CT1, CT 2 ... or	Numbers of the axis containers whose contents are to be switched or
e.g. A_CONT1	individual names of axis containers set via MD.

Function

`AXCTSWE` ()

Each channel whose axes are contained 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(CT1, ...) can be used to simplify start-up. Under the sole effect of the active channel, the axis container rotates around the increment stored in the SD.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

13.8 Program execution time/Workpiece counter (as from SW 5.2)

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 in the operator panel interface.

13.8.1 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

$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'
```

13.8.2 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Machine data can be used to control counter activation, counter reset timing and the counting algorithm.

Explanation

The following counters are provided:

`$AC_REQUIRED_PARTS`	Number of workpieces required In this counter you can define the number of workpieces at which the actual workpiece counter $AC_ACTUAL_PARTS is reset to zero. Machine data can be used to configure the generation of the display alarm "Required number of workpieces reached" and the channel VDI signal "Required number of workpieces reached".
`$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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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. Activate all counters, 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

User Stock Removal Programs

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 (as before SW 4.3, default value) 1 = Contour preparation in both directions
EXECUTE	Terminate contour preparation
ERROR	Variable for error check-back, 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(…)
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Additional notes

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)

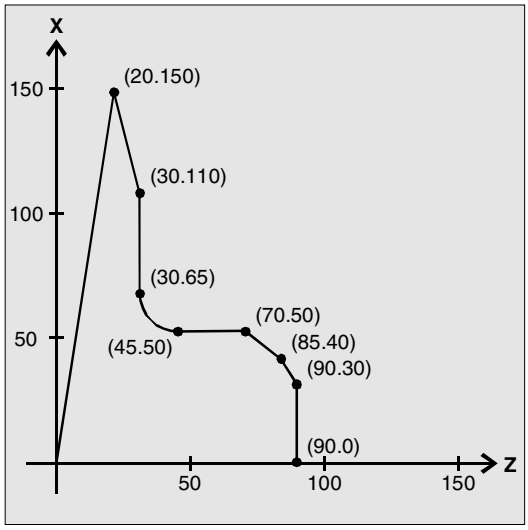
In SW 4.4 and higher, contour preparation has been expanded. Now when CONTPRON is called, the contour table is available irrespective of the programmed direction.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 part 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 in current plane

(5), (6) Starting point of contour elements
(5) = abscissa, (6) = ordinate in 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.

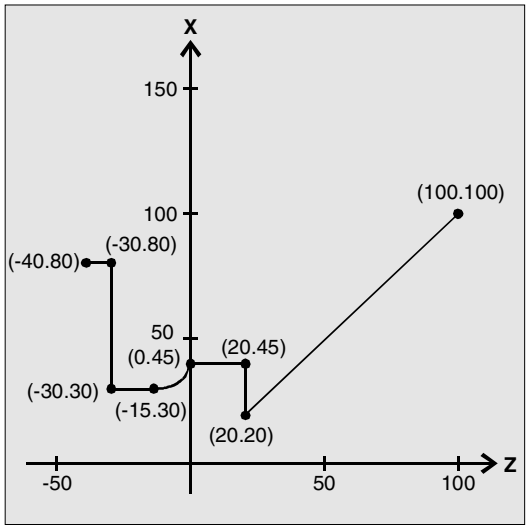
(9), (10) Center point coordinates of contour element, if it is a circle block.
(9) = abscissa, (10) = ordinate

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 part 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 check-back 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
`...`	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Table KTAB

After contour preparation is finished, the contour is available in both directions.

Row	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 in current plane

(5), (6) Starting point of contour elements
(5) = abscissa, (6) = ordinate in 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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

(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)

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

14.3 Contour decoding: CONTDCON (as of SW 5.2)

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 check-back, 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(…)
```

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Additional notes

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
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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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.

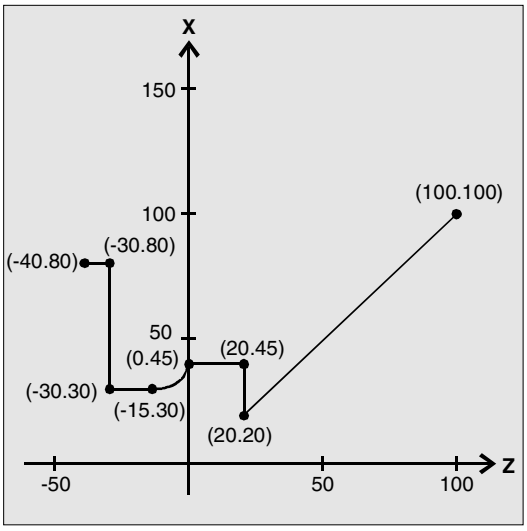
Stock removal in the programmed contour direction

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



840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

NC part 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 check-back 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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

Explanation of column contents

Line 0 coding for **starting point**:

Column 0:

10^0 (ones): G0 = 0

10^1 (tens): 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 (ones): G0 = 0, G1 = 1, G2 = 2, G3 = 3

10^1 (tens): G70 = 0, G71 = 1, G700 = 2, G710 = 3

10^2 (hundreds): G90 = 0, G91 = 1

10^3 (thousands): G93 = 0, G94 = 1, G95 = 2, G96 = 3

10^4 (ten thousands): G60 = 0, G44 = 1, G641 = 2, G642 = 3

10^5 (hundred thousands): G9 = 1

Column 1: End point abscissa

Column 2: End point ordinate

Column 3: Center point abscissa for circular interpolation

Column 4: Center point ordinate for circular interpolation

Column 5: Feedrate

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	

Function

INTERSEC calculates the intersection of two normalized contour elements from the contour table generated with CONTPRON.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

The indicated status specifies whether or not an intersection exists (TRUE = intersection, FALSE = no intersection).

Additional notes

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 CUTCUT (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
...	

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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
PT[n,2]	Points for calculation n = number of points (3 or 4); 2 = point coordinates
NO.	Number of points used for calculation: 3 or 4
RES[3]	Variable for result: specification of circle center point coordinates and radius; 0 = abscissa, 1 = ordinate of circle center point; 2 = radius

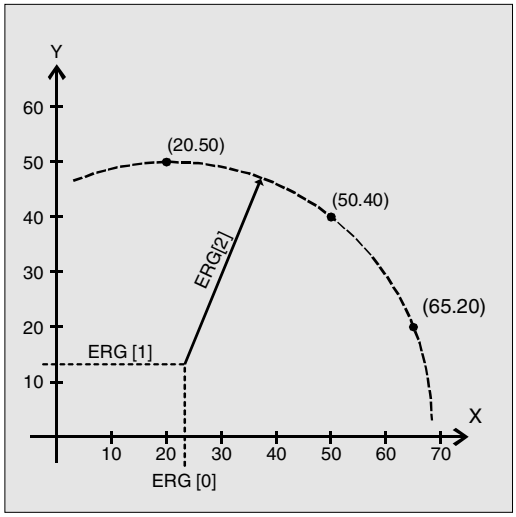
Function

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.

840D NCU 571 | 840D NCU 572 NCU 573 | FM-NC | 810D | 840Di

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 start of program (in delivery state of control system provided that another setting is not 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 depending on syntax of G function

[4] IPO parameters act incrementally as arc centres. They can be programmed in absolute mode with AC. When they have other meanings (e.g. pitch), the address modification is ignored.

[5] Vocabulary word does not apply to SINUMERIK FM-NC/810D

[6] Vocabulary word does not apply to SINUMERIK FM-NC/810D/NCU571

[7] Vocabulary word does not apply to SINUMERIK 810D

[8] The OEM user can incorporate two extra interpolation types and modify their names.

[9] Vocabulary word applies only to SINUMERIK FM-NC

[10] The extended address block format may not be used for these functions.

Name	Meaning	Value assignment	Description, comment	Syntax	Modal/ non-modal	Group[2]
:	Block number – main block (see N)	0 ... 9999 9999 integer values only, no sign	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 angle	Real			s	
A3 [5]	Tool orientation: Direction vector component	Real			s	
A4 [5]	Tool orientation for block beginning	Real			s	
A5 [5]	Tool orientation for block end; Normal vector component	Real			s	
ABS	Absolute value	Real				
AC	Dimension input, absolute	0, ..., 359.9999°		X=AC(100)	s	
ACC [5]	Axial acceleration	Real, without sign			m	
ACN	Absolute dimension setting for rotary axes, approach position in negative direction			A=ACN(...) B=ACN(...) C=ACN(...)	s	

ACP	Absolute dimension setting for rotary axes, approach position in positive direction			A=ACP(...) B=ACP(...) C=ACP(...)	s	
ACOS	Arccosine (trigon. function)	Real				
ADIS	Resurfacing distance for path functions G1, G2, G3, ...	Real, without sign			m	
ADISPOS	Resurfacing distance for rapid traverse G0	Real, without sign			m	
ALF	Angle lift fast	Integer, without sign			m	
AMIRROR	Programmable mirroring (additive mirror)			AMIRROR X0 Y0 Z0 ; separate block	s	3
AND	Logical AND					
ANG	Contour definition angle	Real				
AP	Polar angle (Angle Polar)	0, ..., ± 360°			m,s[3]	
APR	Read/display access protection (access protection read)	Integer, without sign				
APW	Write access protection (access protection write)	Integer, without sign				
AR	Aperture angle (angle circular)	0, ..., 360°			m,s[3]	
AROT	Programmable rotation (additive rotation)	Rotation around 1st geom. axis: –180° .. 180° 2nd geom. axis: –89.999° .. 90° 3rd geom. axis: –180° .. 180°		AROT X... 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	Arcsine (trigon. function)	Real				
ASPLINE [7]	Akima spline				m	1
ATAN2	Arctangent 2	Real				
ATRANS	Additive programmable offset (additive translation)			ATRANS X... Y... Z... ; separate block	s	3
AX	Integer without sign	Real			m,s[3]	
AXCSWAP	Switch container axis			AXCSWAP(CTn,CTn+1,...)		25
AXIS	Data type: Axis name		Name of file can be added			

AXNAME	Converts the input string to an axis name (get axname)	String	An alarm is generated if the input string does not contain a valid axis name			
AXSTRING	Convert axis name to string (get axis as string)	AXIS	Name of file can be added			
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 angle	Real			s	
B3 [5]	Tool orientation: Direction vector component	Real			s	
B4 [5]	Tool orientation for block beginning	Real			s	
B5 [5]	Tool orientation for block end; Normal vector component	Real			s	
BAUTO [7]	Definition of first spline segment by means of following 3 points (begin not a knot)				m	19
BLSYNC	Processing of interrupt routine is only to start with the next block change					
BNAT [1,7]	Natural transition to first spline block (begin natural)				m	19
BOOL	Data type: Boolean value TRUE / FALSE or 0 / 1					
BRISK [1]	Brisk path acceleration				m	21
BRISKA	Activate brisk axis acceleration for the programmed axes					
BSPLINE [7]	B spline				m	1
BTAN [7]	Tangential transition to first spline block (begin tangential)				m	19
C	Axis	Real			m,s[3]	
C2 [5]	Tool orientation: Euler angle	Real			s	
C3 [5]	Tool orientation: Direction vector component	Real			s	
C4 [5]	Tool orientation for block beginning	Real			s	
C5 [5]	Tool orientation for block end; 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 subroutine call			CALL PROGVAR		
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
CFC [1]	Constant feed on contour				m	16
CFIN	Constant feed at internal radius, acceleration at external radius (constant feed at internal 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 block			
CHAR	Data type: ASCII character	0, ..., 255				
CHF software Version 3.5 and higher CHR	Chamfer; value = length of chamfer in direction of movement (chamfer) Chamfer; value = length of chamfer	Real, without sign			s	
CHKDNO	D number check					
CIC	Incremental approach of position (coded position: incremental coordinate)		See CAC			
CIP	Circular interpolation through intermediate points			CIP X... Y... Z... I1=... J1=... K1=...	m	1

CLAL	Clear alarm	INT	Parameter: Alarm number			
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				
COMPOF [1,6]	Compressor OFF				m	30
COMPON [6]	Compressor ON				m	30
COMPCURV	Compressor ON constant curve polynomials				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, without sign			s	
CROT	Rotation of the current coordinate system.	FRAME	Maximum number of parameters: 6			
CSCALE	Scale factor for multiple axes.	FRAME	Maximum number of parameters: 2 * axis number $_{max}$			
CSPLINE [7]	Cubic spline				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					
CTABINV	Define leading axis position according to following axis position from curve table	Real	See CTAB			
CT	Circle with tangential transition			CT X... Y.... Z...	m	1
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 tool offset peripheral milling (cutter compensation type 3-dimensional circumference)				m	22
CUT3DF [5]	3D tool offset face milling (cutter compensation type 3-dimensional face)				m	22
CUT3DFF [5]	3D tool offset face milling with constant tool orientation as a function of active frame (cutter compensation type 3-dimensional face frame)				m	22
CUT3DFS [5]	3D tool offset face milling with constant tool orientation irrespective of active frame (cutter compensation type 3-dimensional face frame)				m	22
CUTCONO[1]	Constant radius compensation OFF				m	40
CUTCONON	Constant radius compensation ON				m	40
D	Tool offset number	1, ..., 9 Software Version 3.5 and higher 1, ... 32 000	Contains offset data for a specific tool T... ; D0 SpecSym → offset values for a tool	D...		
DC	Absolute dimension setting for rotary axes, approach position directly			A=DC(...) B=DC(...) C=DC(...) SPOS=DC(...)	s	

DEF	Variable definition	Integer, without sign				
DEFAULT	Branch in CASE branch		Jump to if expression does not fulfill any of the specified values			
DEFINE	Define macro					
DELDTG	Delete distance-to-go					
DELT	Delete tool		Duplo number can be omitted			
DIAMOF[1]	Diametral programming: OFF				m	29
DIAMON	Diametral programming: ON				m	29
DIAM90	Diameter program for G90, radius progr. for G91				m	29
DILF	Rapid lift length				m	
DISABLE	Interrupt OFF					
DISC	Transition circle overshoot in tool radius compensation	0, ..., 100			m	
DISPLOF	Suppress current block display (display OFF)					
DISPR	Distance path for repositioning	Real, without sign			s	
DISR	Distance for repositioning	Real, without sign			s	
DITE	Thread run-out path	Real			m	
DITS	Thread run-in path	Real			m	
DIV	Integer division					
DL	Tool sum compensation	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 [7]	Definition of last spline segment by 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			
ELSE	Program branch, if IF condition not fulfilled					
ENABLE	Interrupt ON					
ENAT [1,7]	Natural curve 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 [7]	Tangential curve transition to next traversing block at beginning of spline (end tangential)				m	20
EVERY	Execute synchronized action if condition changes from FALSE to TRUE					
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	Exponent function e^x	Real				
EXTERN	Broadcast a subroutine with parameter passing					
F	Feed value (dwell time is also programmed under F in conjunction with G4)	0.001, ..., 99 999.999	Tool/workpiece path velocity; Dimension in mm/min or mm/revolution as a function of G94 or G95	F=100 G1 ...		

FA	Axial feed (feed axial)	0.001, ..., 999999.999 mm/min, degree/min; 0.001, ..., 39999.9999 inch/min		FA[X]=100	m	
FAD	Infeed feedrate for smooth approach and retraction (Feed approach / depart)	Real, without sign				
FALSE	Logical constant: False	BOOL	Can be replaced with integer constant 0			
FCTDEF	Define polynomial function		Is evaluated in SYFCT or PUTFTOCF.			
FCUB [6]	Feed variable according to cubic spline (feed cubic)				m	37
FD	Path feed for handwheel override (feed DRF)	Real, without sign			s	
FDA	Axial feed for handwheel override (feed DRF axial)	Real, without sign			s	
FFWOF [1]	Feedforward control OFF (feed forward OFF)				m	24
FFWON	Feedforward control ON (feed forward ON)				m	24
FGREF	Reference radius				m	
FGROUP	Define axis(es) with path feed		F applies to all axes programmed under FGROUP	FGROUP (Axis1, [Axis2], ...)		
FIFOLEN	Programmable preprocessing depth					
FL	Limit velocity for synchronous axes (feed limit)	Real, without sign	The unit set with G93, G94, G95 applies (max. rapid traverse)	FL [Axis] =...	m	
FLIN [6]	Linearly variable feed (feed linear)				m	37
	Feed multiple axial	Real, without sign			m	
FNORM [1,6]	Normal feed acc. to DIN66025 (feed normal)				m	37
FOR	Counter loop with fixed number of passes					
FORI1	Feed for swiveling the orientation vector on the large circle				m	
FORI2	Feed for the overlaid rotation around the swiveled orientation vector				m	
FP	Fixed point: numb. of fixed points to be approached	Integer, without sign		G75 FP=1	s	

FPO	Feed characteristic programmed via a polynomial (feed polynomial)	Real	Quadratic, cubic polynomial coefficient			
FPR	Rotary axis identification	0.001 ... 999999.999		FPR (rotary axis)		
FPRAOF	Deactivate revolutional feedrate					
FPRAON	Activate revolutional feedrate					
FRAME	Data type to define the coordinate system		Contains for each geometry axis: Offset, rotation, angle of shear, scaling, mirroring; For each special axis: Offset, scaling, mirroring			
FRC	Feed for radius and chamfer				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 (fine tool offset OFF)				m	33
FTOCON [6]	Online fine tool offset ON (fine tool offset ON)				m	33
FXS	Travel to fixed stop ON (fixed stop)	Integer, without sign	1 = select, 0 = deselect		m	
FXST	Torque limit for travel to fixed stop (fixed stop torque)	%	Optional setting		m	
FXSW	Monitoring window for travel to fixed stop (fixed stop window)	mm, inch or degree	Optional setting			

G functions						
G	G function (preparatory function) G functions are divided into G groups. Only one of the G functions in a group may be programmed in a block. A G function can be modally active (until it is cancelled by another function in the same group) or it is active only in the block in which it is programmed (non-modal).	Integer, preset values only		G...		
G0	Linear interpolation with rapid traverse		Motion commands	G0 X... Z...	m	1
G1[1]	Linear interpolation with feed			G1 X... Z... F...	m	1
G2	Circular interpolation clockwise			G2 X... Z... I... K... F... ; center and end points G2 X... Z... CR=... F... ; radius and end points G2 AR=... I... K... F... ; aperture angle and center point G2 AR=... X... Z... F... ; aperture angle and end point	m	1
G3	Circular interpolation counterclockwise			G3 ... ; otherwise as for G2	m	1
G4	Predefined dwell time		Special motion	G4 F... ; dwell time in s or G4 S... ; dwell time in spindle rotations ; 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 assignment in channel axes	G25 X.. Y.. Z.. ; separate block	s	3
G26	Upper working area limitation			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 (progessive 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 to left of contour				m	7
G42	Tool radius compensation to right of contour				m	7
G53	Suppression of current zero offset (non-modal)		incl. programmed offsets		s	9
G54	1st settable zero offset				m	8
G55	2nd settable zero offset				m	8
G56	3rd settable zero offset				m	8
G57	4th settable zero offset				m	8
G58						
G59						
G60 [1]	Exact stop deceleration				m	10
G63	Tapping with compensating chuck			G63 Z... G1	s	2
G64	Exact stop – contouring mode				m	10
G70	Dimension in inches				m	13
G71 [1]	Metric dimension				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]	Dimension setting, absolute			G90 X... Y... Z...(...) Y=AC(...) or X=AC Z=AC(...)	m s	14
G91	Incremental dimension setting			G91 X... Y... Z... or X=IC(...) Y=IC(...) Z=IC(...)	m s	14
G94 [1]	Linear feed F in mm/min or inch/min and °/min				m	15
G95	Revolutional feedrate F in mm/rev or inch/rev				m	15
G96	Constant cutting speed ON			G96 S... LIMS=... F...	m	15

G97	Constant cutting speed OFF				m	15
G110	Polar programming relative to last programmed set position			G110 X.. Y.. Z..	s	3
G111	Pole programming relative to zero point of current workpiece coordinate system			G110 X.. Y.. Z..	s	3
G112	Polar programming relative to last valid pole			G110 X.. Y.. Z..	s	3
G140 [1]	Direction of approach WAB defined by G41/G42				m	43
G141	Direction of approach WAB left of contour				m	43
G142	Direction of approach WAB right of contour				m	43
G143	Direction of approach WAB dependent on tangent				m	43
G147	Smooth approach with straight line				s	2
G148	Smooth retraction with straight line				s	2
G153	Suppression of current frame incl. base frame				s	9
G247	Smooth approach with quadrant				s	2
G248	Smooth retraction with quadrant				s	2
G331	Tapping	± 0.001, ...,	Motion		m	1
G332	Retraction (tapping)	2000.00 mm/rev	commands		m	1
G340 [1]	Approach block spatial (depth and in plane at same time (helix)		for smooth approach and retract		m	44
G341	Approach in the perpendicular axis (z), then approach in plane		for smooth approach and retract		m	44
G347	Smooth approach with semi-circle				s	2
G348	Smooth retract with semi-circle				s	2
G450 [1]	Transition circle		Tool compensation response		m	18
G451	Intersection of equidistant paths		at corners		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]	Deactivation of all settable frames, if no value in G500				m	8
G505 .... G599	5. ... 99. Settable zero offset				m	8
G601 [1]	Block change in response to exact stop fine		Effective only in conjunction with		m	12
G602	Block change in response to exact stop coarse		active G60 or		m	12
G603	Block change in response to IPO end of block		G9 with programmable transition		m	12
G641	Exact stop – contouring mode		rounding	G641 ADIS=...	m	10
G642	Rounding with axial precision				m	10

G643	Block-internal corner rounding				m	10
G700	Dimensions in inch and inch/min				m	13
G710[1]	Metric dimensions in mm and mm/min				m	13
G810[1], ..., G819	G group reserved for OEM users					31
G820[1], ..., G829	G group reserved for OEM users					32
G961	Constant cutting speed ON		without additional spindle rotation	G961 S... LIMS=... F...	m	15
G971	Constant cutting speed OFF				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					
GETT	Get T number for tool name					
GOTOF	Jump instruction forwards (towards the end of program)					
GOTOB	Jump instruction back (towards start of program)					
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 PLC	Real/INT	Settable via MD (machine manufact.)	H100 or H2=100		
I [4]	Interpolation parameter	Real			s	
I1	Intermediate point coordinate	Real			s	
IC	Incremental dimension setting	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: Parameter 1, character: Parameter 2			
INIT	Select block 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				
ISAXIS	Check if geometry axis 1 – 3 specified as parameter exist	BOOL				
ISD	Insertion depth	Real			m	
ISNUMBER	Check whether the input string can be converted to a number	BOOL				
J [4]	Interpolation parameter	Real			s	
J1	Intermediate point coordinate	Real			s	
JERKA	Activate acceleration response set via machine data for programmed axes					
K [4]	Interpolation parameter	Real			s	
K1	Intermediate point coordinate	Real			s	
KONT	Traverse around contour for tool compensation				m	17
L	Subprogram number	Integer, up to 7 places	Leading zeros are relevant!	L10	s	
LEAD [5]	Lead angle	Real			m	
LEADOF	Leading value coupling OFF (lead off)					
LEADOFP	Path leading value coupling OFF (lead off path)					
LEADON	Leading value coupling ON (lead on)					
LEADONP	Path leading value coupling ON (lead on path)					
LFOF [1]	Interruption of thread cutting OFF				m	41
LFON	Interruption of thread cutting ON				m	41
LFTXT [1]	Tool direction tangential at lift				m	46
LFWP	Tool direction not tangential at lift				m	46
LIFTFAST	Rapid lift before interrupt routine call					
LIMS	Spindle speed limitation (limit spindle speed) with G96	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	Max. of 5 free special functions to be defined by machine manufacturer			
M0 [10]	Programmed stop					

M1 [10]	Optional stop					
M2 [10]	Program end, main program with reset to program start					
M3	Clockwise spindle rotation for master spindle					
M4	Counter-clockwise spindle rotation for master spindle					
M5	Spindle stop for master spindle					
M6	Tool change					
M17 [10]	End of subprogram					
M19	Spindle positions					
M30 [10]	Program end, as for M2					
M40	Automatic gear change					
M41... M45	Gear stage 1, ..., 5					
M70	Transition to axis operation					
MCALL	Modal subprogram call		Without subprogram name: Deselection			
MEAC	Continuous measurement without deletion of distance-to-go	Integer, without sign			s	
MEAFRAME	Frame calculation from measuring points	FRAME				
MEAS	Measurement with touch trigger probe (measure)	Integer, without sign			s	
MEASA	Measurement with deletion of distance-to-go				s	
MEAW	Measurement with touch trigger probe without deletion of distance-to-go (measure without deleting distance-to-go)	Integer, without sign			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: Parameter 1, character: Parameter 2			
MIRROR	Programmable mirror			MIRROR X0 Y0 Z0 ; separate block	s	3
MMC	Command to MMC command interpreter	STRING				
MOD	Modulo division					
MOV	Start positioning axis (start moving positioning axis)	Real				
MSG	Programmable messages			MSG("message")	m	
N	Subblock number	0, ..., 9999 9999 integer values only, no sign	Can be used to identify blocks with a number; position at beginning of block	E.g. N20		

NCK	Specify validity range for data		once per NCK			
NEWCONF	Accept modified machine data					
NEWT	Create new tool		Duplo number can be omitted			
NORM [1]	Normal setting at start and end points for 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 for 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 compensation – normal	Real			m	
OR	Logical OR					
ORIC [1,6]	Changes in orientation at outer corners are overlaid on the circular block to be inserted (orientation change continuously)				m	27
ORID [6]	Changes in orientation are performed before the circular block (orientation change discontinuously)				m	27
ORIEULER	Orientation angles using Euler angles				m	50
ORIMACHAX	Linear interpolation of machine axes or orientation axes				m	51
ORIMCS [6]	Tool orientation in machine coordinate system				m	25
ORIRPY	Orientation angles using RPY angles				m	50
ORIS [5]	Change in orientation (orientation smoothing factor)	Real	Referred to path		m	
ORIVIRT1	Orientation angles using virtual orientation axes (definition 1)				m	50
ORIVIRT2	Orientation angles using virtual orientation axes (definition 1)				m	50
ORIVIRTAX	Large-circle interpolation				m	51

ORIWCS [1,6]	Tool orientation in workpiece coordinate system				m	25
OS	Oscillation ON / OFF	Integer, without sign				
OSC [6]	Constant smoothing for tool orientation				m	34
OSCILL	Axis assignment for oscillation – activate oscillation		Axis: 1–3 infeed axes		m	
OSCTRL	Oscillation control options	Integer, without sign			m	
OSE	Oscillation: End point				m	
OSNSC	Oscillation: Number of spark-out cycles number spark out cycles)				m	
OSOF [1,6]	Constant smoothing for tool orientation OFF				m	34
OSP1	Oscillation: Left-hand reversal point (oscillating: position 1)	Real			m	
OSP2	Oscillation: Right-hand reversal point (oscillating: position 2)	Real			m	
OSS [6]	Tool orientation smoothing at end of block				m	34
OSSE [6]	Tool orientation smoothing at beginning and end of block				m	34
OST1	Oscillation: Stop in left-hand reversal point	Real			m	
OST2	Oscillation: Stop in right-hand reversal point	Real			m	
OVR	Spindle override	1, ..., 200%			m	
OVRA	Axial spindle override	1, ..., 200%			m	
P	Number of subprogram passes	1 ... 9999, integer without sign		E.g. L781 P... ; separate block		
PDELAYOF [6]	Delay for punching OFF (punch with delay OFF)				m	36
PDELAYON [1,6]	Delay for punching ON (punch with delay ON)				m	36
PL	Parameter interval length	Real, without sign			s	
PM	per minute			Feed per minute		
PO	Polynomial	Real, without sign			s	
POLF	Position LIFTFAST	Real, without sign		POLF[Y]=10	m	
POLY [5]	Polynomial interpolation				m	1
PON [6]	Punching ON (punch ON)				m	35
PONS [6]	Punching ON in IPO cycle (punch ON slow)				m	35
POS	Position axis			POS[X]=20		
POSA	Position axis across block boundaries			POSA[Y]=20		

POSP	Positioning in part sections (oscillation) (Position axis in parts)	Real: End position, part length; Integer: option				
POT	Square (arithmetic function)	Real				
PR	Per revolution			Revolutional feedrate		
PRESETON	Set actual value for programmed axes		An axis name is programmed with the corresponding 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 movement				m	49
PUTFTOC	Tool offset fine for parallel dressing (continuous dressing)					
PUTFTOCF	Put fine tool correction function dependent: Fine tool offset dependent on a function for continuous dressing defined with FCtDEF					
PW	Point weight	Real, without sign			s	
QECLRNOF	Quadrant error compensation learning OFF					
QECLRNON	Quadrant error compensation learning ON					
QU	Fast additional (auxiliary) function output					
R...	Arithmetic parameters Software Version 5 and higher: also as settable address identifier and with numerical extension	±0.0000001, ..., 9999 9999	R parameter number can be set via MD	R10=3 ;R parameter assignment X=R10 ;Axis value R[R10]=6 ;indirect programming		
RDISABLE	Read-in disable					
READAL	Read alarrm		Alarms are searched according to ascending numbers			

REAL	Data type: floating point variable with leading sign (real numbers)	Correspond s to the 64-bit floating point format of the processor				
REDEF	Setting for machine data, 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	Reposition all axes linearly				s	2
REPOSH	Reposition along semi-circle				s	2
REPOSHA	Reposition all axes along semi-circle: Reposition all axes, geometry axes along quadrant				s	2
REPOSL	Reposition linearly				s	2
REPOSQ	Reposition along quadrant				s	2
REPOSQA	Reposition all axes along quadrant Reposition all axes linearly, geometry axes along quadrant				s	2
RESET	Reset technology cycle		One or several IDs can be programmed			
RET	End of subprogram		Used instead of M2 to maintain continuous-path mode	RET		
RINDEX	Define index of character in input string	0, ..., INT	String: Parameter 1, character: Parameter 2			
RMB	Reposition at beginning of block (Repos mode begin of block)				m	26
RME	Reposition at end of block (Repos mode end of block)				m	26
RMI [1]	Reposition at interruption point (Repos mode interrupt)				m	26

RND	Round contour corner	Real, without sign		RND=...	s	
RNDM	Modal rounding	Real, without sign		RNDM=... RNDM=0: M. V. deactivation	m	
ROT	Programmable rotation	Rotation around 1st geometry axis: –180° .. 180° 2nd G axis: –89.999°, ..., 90° 3rd G axis: –180° .. 180°		ROT X... Y... Z... ROT RPL= ; separate block	s	3
ROUND	Round decimal places	Real				
RP	Polar radius (radius polar)	Real			m,s[3]	
RPL	Rotation in plane (rotation plane)	Real, without sign			s	
RT	Parameter for access to frame data: Rotation					
S	Spindle speed or (with G4, G96) another meaning	0.1 ... 99999999.9	Spindle speed in rev/min G4: Dwell time in spindle rotations G96: Cutting rate in m/min	S...: Spindle speed for master spindle S1...: Spindle 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)					
SCALE	Programmable scaling (scale)			SCALE X... Y... Z... ; separate block	s	3
SD	Spline degree	Integer, without sign			s	
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 is evaluated → 1			
SETM	Set one/several markers for channel coordination		Machining in the local channel is not influenced by this.			
SETMS	Switch back to master spindle programmed in machine data					
SETMS(n)	Spindle n must act as master spindle					
SETPIECE	Set piece number for all tools assigned to the spindle.		Without spindle number: Valid for master spindle			
SF	Start point offset for thread cutting (spline offset)	0.0000, ..., 359.999°			m	
SIN	Sine (trigon. function)	Real				
SOFT	Soft axis 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 length of arc				m	45
SPCOF	Switch master spindle or spindle (n) from speed control over to position control			SPCON SPCON (n)		
SPCON	Switch master spindle or spindle (n) from position control over to speed control			SPCON SPCON (n)		
SPIF1 [1,6]	High-speed NCK inputs/outputs for punching/nibbling byte 1 (stroke/punch interface 1)		see /FB/, N4: Punching and Nibbling		m	38
SPIF2 [6]	High-speed NCK inputs/outputs for punching/nibbling byte 2 (stroke/punch interface 2)		see /FB/, N4: Punching and Nibbling		m	38
SPLINE PATH [7]	Define spline grouping		Max. of 8 axes			
SPOF [1,6]	Stroke OFF, punching, nibbling OFF (stroke/punch OFF)				m	35
SPN [6]	Number of path sections per block (stroke/punch number)	Integer			s	
SPP [6]	Length of a 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, without sign			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, without sign			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 articulated joints	Integer			s	
STARTFIFO[1]	Execute; fill preprocessing buffer in parallel				m	4
STOPFIFO	Stop processing; fill preprocessing buffer until STARTFIFO, preprocessing buffer "full" or end of program is detected				m	4
STOPRE	Stop preprocessing until all prepared blocks are executed in main run.					
STOPREOF	Stop preprocessing OFF					
STRING	Data type: String	Max. 200 characters				
STRLEN	Define string length	INT				
SUBSTR	Define index of character in input string	Real	String: Parameter 1, character: Parameter 2			
SUPA	Suppression of current 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 (change only if so defined in machine data; otherwise M6 command required)	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 current tool orientation		Required after resetting machine, e.g.		m	42
TCOFR	Determine tool length components from orientation of active frame		by manual setting		m	42
TILT [5]	Side angle	Real			m	
TMOF	Deselect tool monitoring function		T number required only if tool with this number is not active.	TMOF (T No.)		
TMON	Select tool monitoring function		T No. = 0: Deactivate monitoring function for all tools	TMON (T No.)		
TO	Defines the end value in a FOR counter loop					
TOFRAME	Set current programmable frame to tool coordinate system				s	3
TOLOWER	Convert letters of the string into lowercase					
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	Switch off transformation			TRAFOOF( )		
TRAILOF	Synchronous coupled motion of axes OFF (trailing OFF)					
TRAILON	Synchronous coupled motion of axes ON (trailing ON)					

TRANS	Programmable offset (translation)			TRANS X. Y. Z. ; separate block	s	3
TRANSMIT	Polar transformation		see TRAANG			
TRAORI	4-axis, 5-axis transformation (transformation oriented)		see TRAANG			
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	No. of turns for helix	0, ..., 999			s	
UNLOCK	Enable synchronized action with ID (continue technology cycle)					
UNTIL	Condition for end of REPEAT loop					
UPATH	Curve parameter is path reference for FGROUP axes				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.	WAITM(1,1,2)		
WAITM	Wait for marker in specified channel; terminate previous block with exact stop.			WAITM(1,1,2)		
WAITMC	Wait for marker in specified channel; exact stop only if other channels have not yet reached the marker			WAITMC(1,1,2)		
WAITP	Wait for end of travel			WAITP(X) ; separate block		
WAITS	Wait until spindle position is reached			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					
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 start of program (in delivery state of control system provided that another setting is not 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 depending on syntax of G function

[4] IPO parameters act incrementally as arc centres. They can be programmed in absolute mode with AC. When they have other meanings (e.g. pitch), the address modification is ignored.

[5] Vocabulary word does not apply to SINUMERIK FM-NC/810D

[6] Vocabulary word does not apply to SINUMERIK FM-NC/810D/NCU571

[7] Vocabulary word does not apply to SINUMERIK 810D

[8] The OEM user can incorporate two extra interpolation types and modify their names.

[9] Vocabulary word applies only to SINUMERIK FM-NC

[10] The extended address block format may not be used for these functions.

15.2 List of system variables

Legend:

Part pro	Part program
Sync	Synchronized action
O	The index can be calculated online in synchronized actions. (+)
S	Software version
R	Read access possible
W	Write access possible
RS	A preprocessor stop takes place implicitly on read access
WS	A preprocessor stop takes place implicitly on write access
+	In column O: The index can be calculated online in synchronized actions.

15.2.1 R parameters

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
R **$R**	REAL	Rn or R[n] The max. number of R parameters is defined in machine data	R	W	R	W		1

15.2.2 Frames 1

$P_UIFR	FRAME	$P_UIFR[n] Settable frames, can be activated via G500, G54 .. G599. 5 to 100 settable frames can be configured with MD $MC_MM_NUM_USER_FRAMES.	R	W				2
$P_CHBFR	FRAME	$P_CHBFR[n] Channel base frames, can be activated via G500, G54 .. G599. 0 to 8 channel base frames can be configured via MD $MC_MM_NUM_BASE_FRAMES.	R	W				5
$P_NCBFR	FRAME	$P_NCBFR[n] NCU base frames, can be activated via G500, G54 .. G599. 0 to 8 NCU base frames can be configured via MD $MN_MM_NUM_GLOBAL_BASE_FRAMES.	R	W				5

15.2.3 Toolholder data

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_CARR1	REAL	$TC_CARR1[n] x component of offset vector l1 Caution! All system parameters with the '$TC_' prefix are contained in the TOA area. The specialty in this area is that various channels of the NCI can access these parameters when machine data 28085 = MM_LINK_TOA_UNIT. If this parameterization mode is chosen for the NCK, you must be aware of the fact that changes may interfere with an other channel; i.e. you must make sure that the change only affects the local channel. The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR2	REAL	$TC_CARR2[n] y component of offset vector l1 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR3	REAL	$TC_CARR3[n] z component of offset vector l1 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR4	REAL	$TC_CARR4[n] x component of offset vector l2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR5	REAL	$TC_CARR5[n] y component of offset vector l2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR6	REAL	$TC_CARR6[n] z component of offset vector l2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR7	REAL	$TC_CARR7[n] x component of axis of rotation v1 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_CARR8	REAL	$TC_CARR8[n] y component of axis of rotation v1 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR9	REAL	$TC_CARR9[n] z component of axis of rotation v1 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR10	REAL	$TC_CARR10[n] x component of axis of rotation v2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR11	REAL	$TC_CARR11[n] y component of axis of rotation v2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR12	REAL	$TC_CARR12[n] z component of axis of rotation v2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR13	REAL	$TC_CARR13[n] Angle of rotation alpha1 (in degrees) The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR14	REAL	$TC_CARR14[n] Angle of rotation alpha2 (in degrees) The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				4
$TC_CARR15	REAL	$TC_CARR15[n] x component of basic vector b The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5
$TC_CARR16	REAL	$TC_CARR16[n] y component of basic vector b The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5
$TC_CARR17	REAL	$TC_CARR17[n] z component of basic vector b The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_CARR18	REAL	$TC_CARR18[n] x component of offset vector l4 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5.3
$TC_CARR19	REAL	$TC_CARR19[n] y component of offset vector l4 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5.3
$TC_CARR20	REAL	$TC_CARR20[n] z component of offset vector l4 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5.3
$TC_CARR21	AXIS	$TC_CARR21[n] axis identifier for axis of rotation v1 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5.3
$TC_CARR22	AXIS	$TC_CARR22[n] axis identifier for axis of rotation v2 The max. number of toolholders can be set via machine data. Default setting is = 0; i.e. NCI has no such data.	R	W				5.3
$TC_CARR23	CHAR	$TC_CARR23[n] Kinematic type 0:Alarm (14153) n:Alarm (14153) Permissible options: T:only the tool can rotate (default) P:only the part can rotate M: tool and part can rotate (mixed mode) Uppercase and lowercase are permissible.	R	W				5.3

15.2.4 Channel-specific protection zones

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$SC_PA_ACTIV_IMMED	BOOL	$SC_PA_ACTIV_IMMED[n] Protection zone active immediately? TRUE: The protection zone is active immediately upon start-up of the control and referencing of the axes FALSE: The protection zone is not active immediately n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SC_PA_T_W	CHAR	$SC_PA_T_W[n] Workpiece/tool-oriented prot. zone 0: Workpiece-oriented protection zone 3: Tool-oriented protection zone n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SC_PA_ORI	INT	$SC_PA_ORI[n] Orientation of protection zone 0: Polygon in plane from 1st and 2nd geo axis 1: Polygon in plane from 3rd and 1st geo axis 2: Polygon in plane from 2nd and 3rd geo axis n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SC_PA_LIM_3DIM	INT	$SC_PA_LIM_3DIM[n] Code for restricting the protection zone in the axis that lies parallel to the polygon definition 0: = No limit 1: = Limit in positive direction 2: = Limit in negative direction 3: = Limit in both directions n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SC_PA_PLUS_LIM	REAL	$SC_PA_PLUS_LIM[n] Positive limit for the protection zone in the axis that lies perpendicular to the polygon definition n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SC_PA_MINUS_LIM	REAL	$SC_PA_MINUS_LIM[n] Negative limit for prot. zone in minus direction that lies perpendicular to polygon definition n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SC_PA_CONT_NUM	INT	$SC_PA_CONT_NUM[n] Number of valid contour elements n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$SC_PA_CONT_TYP	INT	$SC_PA_CONT_TYP"[n,m] Contour element type (G1, G2, G3) n: No. of protection zone 0 – (maximum value can be set via MD) m: Number of contour element 0 – 10	R	W				2
$SC_PA_CONT_ORD	REAL	$SC_PA_CONT_ORD[n,m] End point of contour element (ordinate) n: No. of protection zone 0 – (maximum value can be set via MD) m: Number of contour element 0 – 10	R	W				2
$SC_PA_CONT_ABS	REAL	$SC_PA_CONT_ABS[n,m] End point of contour element (abscissa) n: No. of protection zone 0 – (maximum value can be set via MD) m: Number of contour element 0 – 10	R	W				2
$SC_PA_CENT_ORD	REAL	$SC_PA_CENT_ORD[n,m] Center point of contour element (ordinate) n: No. of protection zone 0 – (maximum value can be set via MD) m: Number of contour element 0 – 10	R	W				2
$SC_PA_CENT_ABS	REAL	$SC_PA_CENT_ABS[n,m] Center point of contour element (abscissa) n: No. of protection zone 0 – (maximum value can be set via MD) m: Number of contour element 0 – 10	R	W				2

15.2.5 Tool parameters

$TC_DP1	INT	$TC_DP1[t,d] Tool type With active 'Flat D number management' function, the syntax is as follows: $TC_DP1[d] t: T number 1–32000 d: Cutting edge number/D number 1–9	R	W				2
$TC_DP2	REAL	$TC_DP2[t,d] Tool edge position With active 'Flat D number management' function, the syntax is as follows: $TC_DP2[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP3	REAL	$TC_DP3[t,d] Geometry – Length 1 With active 'Flat D number management' function, the syntax is as follows: $TC_DP3[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_DP4	REAL	$TC_DP4[t,d] Geometry – Length 2 With active 'Flat D number management' function, the syntax is as follows: $TC_DP4[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP5	REAL	$TC_DP5[t,d] Geometry – Length 3 With active 'Flat D number management' function, the syntax is as follows: $TC_DP5[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP6	REAL	$TC_DP6[t,d] Geometry – Radius With active 'Flat D number management' function, the syntax is as follows: $TC_DP6[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP7	REAL	$TC_DP7[t,d] Slotting saw: Corner radius With active 'Flat D number management' function, the syntax is as follows: $TC_DP7[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP8	REAL	$TC_DP8[t,d] Slotting saw: Length With active 'Flat D number management' function, the syntax is as follows: $TC_DP8[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP9	REAL	$TC_DP9[t,d] Reserved With active 'Flat D number management' function, the syntax is as follows: $TC_DP9[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_DP10	REAL	$TC_DP10[t,d] Angle between face of tool and torus surface With active 'Flat D number management' function, the syntax is as follows: $TC_DP10[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP11	REAL	$TC_DP11[t,d] Angle between tool longitudinal axis and upper end of torus surface With active 'Flat D number management' function, the syntax is as follows: $TC_DP11[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP12	REAL	$TC_DP12[t,d] Wear – Length 1 – $TC_DP3 With active 'Flat D number management' function, the syntax is as follows: $TC_DP12[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP13	REAL	$TC_DP13[t,d] Wear – Length 2 – $TC_DP4 With active 'Flat D number management' function, the syntax is as follows: $TC_DP13[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP14	REAL	$TC_DP14[t,d] Wear – Length 3 – $TC_DP5 With active 'Flat D number management' function, the syntax is as follows: $TC_DP14[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP15	REAL	$TC_DP15[t,d] Wear – Radius – $TC_DP6 With active 'Flat D number management' function, the syntax is as follows: $TC_DP15[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP16	REAL	$TC_DP16[t,d] Slotting saw: Wear, corner radius – $TC_DP7 With active 'Flat D number management' function, the syntax is as follows: $TC_DP16[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_DP17	REAL	$TC_DP17[t,d] Slotting saw: Wear – Length – $TC_DP8 With active 'Flat D number management' function, the syntax is as follows: $TC_DP17[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP18	REAL	$TC_DP18[t,d] Wear – Reserved – $TC_DP9 With active 'Flat D number management' function, the syntax is as follows: $TC_DP18[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP19	REAL	$TC_DP19[t,d] Wear – Angle between face of tool and torus surface – $TC_DP10 With active 'Flat D number management' function, the syntax is as follows: $TC_DP19[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP20	REAL	$TC_DP20[t,d] Wear – Angle between tool longitudinal axis and upper end of torus surface – $TC_DP11 With active 'Flat D number management' function, the syntax is as follows: $TC_DP20[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP21	REAL	$TC_DP21[t,d] Base – Length 1 With active 'Flat D number management' function, the syntax is as follows: $TC_DP21[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP22	REAL	$TC_DP22[t,d] Base – Length 2 With active 'Flat D number management' function, the syntax is as follows: $TC_DP22[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_DP23	REAL	$TC_DP23[t,d] Base – Length 3 With active 'Flat D number management' function, the syntax is as follows: $TC_DP23[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP24	REAL	$TC_DP24[t,d] Clearance angle With active 'Flat D number management' function, the syntax is as follows: $TC_DP24[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DP25	REAL	$TC_DP25[t,d] Reserved With active 'Flat D number management' function, the syntax is as follows: $TC_DP25[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				4
$TC_DPCE	INT	$TC_DPCE[t,d] = 'Cutting edge number' of offset data block t,d With active 'Flat D number management' function, the syntax is as follows: $TC_DPCE[d] CE stands for <C>utting<E>dge t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_DPH	INT	$TC_DPH[t,d] = 'H cutting edge number' of offset data block t,d for Fanuc0 M With active 'Flat D number management' function, the syntax is as follows: $TC_DPH[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.1

Cutting edge data OEM user

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_DPC1	REAL	The type can be defined in the machine data. The default is REAL $TC_DPC1[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_DPC1[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DPC2	REAL	The type can be defined in the machine data. The default is REAL $TC_DPC2[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_DPC2[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DPCi	REAL	The type can be defined in the machine data. The default is REAL TC_DPC_i$[t,d] With active 'Flat D number management' function, the syntax is as follows: TC_DPC_i$[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DPC10	REAL	The type can be defined in the machine data. The default is REAL $TC_DPC10[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_DPC10[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_DPCS1	REAL	The type can be defined in the machine data. The default is REAL $TC_DPCS1[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_DPCS1[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2
$TC_DPCS2	REAL	The type can be defined in the machine data. The default is REAL $TC_DPCS2[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_DPCS2[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_DPCSi	REAL	The type can be defined in the machine data. The default is REAL TC_DPCS_i$[t,d] With active 'Flat D number management' function, the syntax is as follows: TC_DPCS_i$[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2
$TC_DPCS10	REAL	The type can be defined in the machine data. The default is REAL $TC_DPCS10[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_DPCS10[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2
$TC_SCP13	REAL	Offset for $TC_DP3: $TC_SCP13[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP13[d] t: T number 1–32000 d: Cutting edge number/D number 1– 32000	R	W				5
$TC_SCP14	REAL	Offset for $TC_DP4: $TC_SCP14[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP14[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...	...	...						
$TC_SCP21	REAL	Offset for $TC_DP11: $TC_SCP21[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP21[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP23	REAL	Offset for $TC_DP3: $TC_SCP23[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP23[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP24	REAL	Offset for $TC_DP4: $TC_SCP24[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP24[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
...	...	...						
$TC_SCP31	REAL	Offset for $TC_DP11: $TC_SCP31[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP31[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP33	REAL	Offset for $TC_DP3: $TC_SCP33[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP33[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP34	REAL	Offset for $TC_DP4: $TC_SCP34[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP34[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_SCP41	REAL	Offset for $TC_DP11: $TC_SCP41[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP41[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP43	REAL	Offset for $TC_DP3: $TC_SCP43[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP43[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP44	REAL	Offset for $TC_DP4: $TC_SCP44[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP44[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_SCP51	REAL	Offset for $TC_DP11: $TC_SCP51[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP51[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_SCP53	REAL	Offset for $TC_DP3: $TC_SCP53[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP53[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP54	REAL	Offset for $TC_DP4: $TC_SCP54[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP54[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_SCP61	REAL	Offset for $TC_DP11: $TC_SCP61[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP61[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

$TC_SCP63	REAL	Offset for $TC_DP3: $TC_SCP63[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP63[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_SCP64	REAL	Offset for $TC_DP4: $TC_SCP64[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP64[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_SCP71	REAL	Offset for $TC_DP11: $TC_SCP71[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_SCP71[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_ECP13	REAL	Offset for $TC_DP3: $TC_ECP13[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP13[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_ECP14	REAL	Offset for $TC_DP4: $TC_ECP14[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP14[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_ECP21	REAL	Offset for $TC_DP11: $TC_ECP21[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP21[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

$TC_ECP23	REAL	Offset for $TC_DP3: $TC_ECP23[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP23[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_ECP24	REAL	Offset for $TC_DP4: $TC_ECP24[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP24[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_ECP31	REAL	Offset for $TC_DP11: $TC_ECP31[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP31[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_ECP33	REAL	Offset for $TC_DP3: $TC_ECP33[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP33[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_ECP34	REAL	Offset for $TC_DP4: $TC_ECP34[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP34[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_ECP41	REAL	Offset for $TC_DP11: $TC_ECP41[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP41[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

$TC_ECP43	REAL	Offset for $TC_DP3: $TC_ECP43[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP43[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_ECP44	REAL	Offset for $TC_DP4: $TC_ECP44[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP44[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_ECP51	REAL	Offset for $TC_DP11: $TC_ECP51[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP51[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

$TC_ECP53	REAL	Offset for $TC_DP3: $TC_ECP53[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP53[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_ECP54	REAL	Offset for $TC_DP4: $TC_ECP54[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP54[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_ECP61	REAL	Offset for $TC_DP11: $TC_ECP61[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP61[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

$TC_ECP63	REAL	Offset for $TC_DP3: $TC_ECP63[t,d] analogous to $TC_DP12[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP63[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_ECP64	REAL	Offset for $TC_DP4: $TC_ECP64[t,d] analogous to $TC_DP13[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP64[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
...		...						
$TC_ECP71	REAL	Offset for $TC_DP11: $TC_ECP71[t,d] analogous to $TC_DP20[t,d] With active 'Flat D number management' function, the syntax is as follows: $TC_ECP71[d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

15.2.6 Monitoring data for tool management

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MOP1	REAL	$TC_MOP1[t,d] Prewarning limit for tool life t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_MOP2	REAL	$TC_MOP2[t,d] Remaining tool life t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_MOP3	INT	$TC_MOP3[t,d] Prewarning limit for number of workpieces t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_MOP4	INT	$TC_MOP4[t,d] Remaining number of workpieces t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_MOP5	REAL	$TC_MOP5[t,d] Prewarning limit wear t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_MOP6	REAL	$TC_MOP6[t,d] Remaining wear t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_MOP11	REAL	$TC_MOP11[t,d] Service life setpoint t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_MOP13	INT	$TC_MOP13[t,d] Workpiece count setpoint t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5
$TC_MOP15	REAL	$TC_MOP15[t,d] Wear setpoint t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5

15.2.7 Monitoring data for OEM users

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MOPC1	INT	The type can be defined in the machine data. The default is INT $TC_MOPC1[t,d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_MOPC2	INT	The type can be defined in the machine data. The default is INT $TC_MOPC2[t,d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
...		...						
$TC_MOPC10	INT	The type can be defined in the machine data. The default is INT $TC_MOPC10[t,d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				2
$TC_MOPCS1	INT	The type can be defined in the machine data. The default is INT $TC_MOPCS1[t,d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2
$TC_MOPCS2	INT	The type can be defined in the machine data. The default is INT $TC_MOPCS2[t,d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2
...		...						
$TC_MOPCS10	INT	The type can be defined in the machine data. The default is INT $TC_MOPCS10[t,d] t: T number 1–32000 d: Cutting edge number/D number 1–32000	R	W				5.2

15.2.8 Tool-related data

$TC_TP1	INT	$TC_TP1[t] Duplo number t: T number 1–32000	R	W				2
$TC_TP2	STRING	$TC_TP2[t] Tool name t: T number 1–32000	R	W				2
$TC_TP3	INT	$TC_TP3[t] Size to left t: T number 1–32000	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_TP4	INT	$TC_TP4[t] Size to right t: T number 1–32000	R	W				2
$TC_TP5	INT	$TC_TP5[t] Size toward top t: T number 1–32000	R	W				2
$TC_TP6	INT	$TC_TP6[t] Size toward bottom t: T number 1–32000	R	W				2
$TC_TP7	INT	$TC_TP7[t] Magazine location type t: T number 1–32000	R	W				2
$TC_TP8	INT	$TC_TP8[t] Status t: T number 1–32000	R	W				2
$TC_TP9	INT	$TC_TP9[t] Type of tool monitoring t: T number 1–32000	R	W				2
$TC_TP11	INT	$TC_TP11[t] Replacement strategy t: T number 1–32000	R	W				2
$TC_TP10	INT	$TC_TP10[t] Tool info t: T number 1–32000	R	W				2
$TC_TPC1	REAL	The type can be defined in the machine data. The default is INT $TC_TPC1[t] t: T number 1–32000	R	W				2
$TC_TPC2	REAL	The type can be defined in the machine data. The default is INT $TC_TPC2[t] t: T number 1–32000	R	W				2
...	...	...						
$TC_TPC10	REAL	The type can be defined in the machine data. The default is INT $TC_TPC10[t] t: T number 1–32000	R	W				2
$TC_TPCS1	REAL	The type can be defined in the machine data. The default is INT $TC_TPCS1[t] t: T number 1–32000	R	W				5 . 2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_TPCS2	REAL	The type can be defined in the machine data. The default is INT $TC_TPCS2[t] t: T number 1–32000	R	W				5 . 2
...		...						
$TC_TPCS10	REAL	The type can be defined in the machine data. The default is INT $TC_TPCS10[t] t: T number 1–32000	R	W				5 . 2

15.2.9 Tool-related grinding data

$TC_TPG1	INT	$TC_TPG1[t] Spindle number t: T number 1–32000	R	W				2
$TC_TPG2	INT	$TC_TPG2[t] Chaining rule t: T number 1–32000	R	W				2
$TC_TPG3	REAL	$TC_TPG3[t] Minimum grinding wheel radius t: T number 1–32000	R	W				2
$TC_TPG4	REAL	$TC_TPG4[t] Minimum grinding wheel width t: T number 1–32000	R	W				2
$TC_TPG5	REAL	$TC_TPG5[t] Current grinding wheel width t: T number 1–32000	R	W				2
$TC_TPG6	REAL	$TC_TPG6[t] Maximum rotation speed t: T number 1–32000	R	W				2
$TC_TPG7	REAL	$TC_TPG7[t] Maximum surface speed t: T number 1–32000	R	W				2
$TC_TPG8	REAL	$TC_TPG8[t] Inclination angle for oblique grinding wheel t: T number 1–32000	R	W				2
$TC_TPG9	INT	$TC_TPG9[t] Parameter number for radius calculation t: T number 1–32000	R	W				2

15.2.10 Magazine location data

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MPP1	INT	$TC_MPP1[n,m] Location class n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPP2	INT	$TC_MPP2[n,m] Location type n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPP3	BOOL	$TC_MPP3[n,m] Adjacent location consideration on/off n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPP4	INT	$TC_MPP4[n,m] Location status n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPP5	INT	$TC_MPP5[n,m] Buffer magazine: Location class index Real magazines: Wear group number n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPP6	INT	$TC_MPP6[n,m] T-no. of the tool at this location n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPP7	INT	$TC_MPP7[n,m] Adapter number of tool adapter at this location n: Physical Magazine number m: Physical location number	R	W				5

15.2.11 Magazine location data for OEM users

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MPPC1	INT	The type can be defined in the machine data. The default is INT $TC_MPPC1[n,m] n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPPC2	INT	The type can be defined in the machine data. The default is INT $TC_MPPC2[n,m] n: Physical Magazine number m: Physical location number	R	W				2
...		...						
$TC_MPPC10	INT	The type can be defined in the machine data. The default is INT $TC_MPPC10[n,m] n: Physical Magazine number m: Physical location number	R	W				2
$TC_MPPCS1	INT	The type can be defined in the machine data. The default is INT $TC_MPPCS1[n,m] n: Physical Magazine number m: Physical location number	R	W				5 . 2
$TC_MPPCS2	INT	The type can be defined in the machine data. The default is INT $TC_MPPCS2[n,m] n: Physical Magazine number m: Physical location number	R	W				5 . 2
...		...						
$TC_MPPCS10	INT	The type can be defined in the machine data. The default is INT $TC_MPPCS10[n,m] n: Physical Magazine number m: Physical location number	R	W				5 . 2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MDP1	INT	$TC_MDP1[n,m] Distance between change position of magazine n and location m of 1st internal magazine internal mag. 1 distance parameter n: Physical Magazine number m: Physical location number	R	W				2
$TC_MDP2	INT	$TC_MDP2[n,m] Distance between change position of magazine n and location m of 2nd internal magazine internal mag. 2 distance parameter n: Physical Magazine number m: Physical location number	R	W				2
$TC_MLSR	INT	$TC_MLSR[n,m]=0 Assignment between buffer location n and buffer location m m must identify a location of type 'spindle'. N must identify a location not of type 'spindle'. This permits definition of the grippers are assigned to which spindles. The parameter value fix = 0. The write process defines a relation, the read process checks whether a particular relation applies. If not, an alarm is produced during a read operation. Define links of grippers,... to spindles. N: Physical magazine location number of location class not equal to SPINDLE m: Physical magazine location number of location class equal to SPINDLE	R	W				3
$TC_MPTH	INT	$TC_MPTH[n,m] Magazine location type hierarchy mag.location (place)types hierarchy parameter n: Hierarchy 0 – 7 m: Location type 0 – 7	R	W				3

15.2.12 Magazine description data for tool management

$TC_MAP2	STRING	$TC_MAP2[n] Identifier of the magazine n: Magazine number 1 to ...	R	W				2
$TC_MAP1	INT	$TC_MAP1[n] Type of magazine n: Magazine number 1 to ...	R	W				2
$TC_MAP3	INT	$TC_MAP3[n] State of magazine n: Magazine number 1 to ...	R	W				2
$TC_MAP4	INT	$TC_MAP4[n] Chaining with following magazine n: Magazine number 1 to ...	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MAP5	INT	$TC_MAP5[n] Chaining with previous magazine n: Magazine number 1 to ...	R	W				2
$TC_MAP6	INT	$TC_MAP6[n] Number of rows n: Magazine number 1 to ...	R	W				2
$TC_MAP7	INT	$TC_MAP7[n] Number of columns n: Magazine number 1 to ...	R	W				2
$TC_MAP8	INT	$TC_MAP8[n] Current magazine position with reference to the change position n: Magazine number 1 to ...	R	W				2
$TC_MAP9	INT	$TC_MAP9[n] Current wear group number n: Magazine number 1 to ...	R	W				5

15.2.13 Tool management magazine description data for OEM users

$TC_MAPC1	INT	The type can be defined in the machine data. The default is INT $TC_MAPC1[n] n: Magazine number 1 to ...	R	W				2
$TC_MAPC2	INT	The type can be defined in the machine data. The default is INT $TC_MAPC2[n] n: Magazine number 1 to ...	R	W				2
...	...	...						
$TC_MAPC10	INT	The type can be defined in the machine data. The default is INT $TC_MAPC10[n] n: Magazine number 1 to ...	R	W				2

$TC_MAPCS1	INT	The type can be defined in the machine data. The default is INT $TC_MAPCS1[n] n: Magazine number 1 to ...	R	W				5 . 2
$TC_MAPCS2	INT	The type can be defined in the machine data. The default is INT $TC_MAPCS2[n] n: Magazine number 1 to ...	R	W				5 . 2
...	...	...						
$TC_MAPCS10	INT	The type can be defined in the machine data. The default is INT $TC_MAPCS10[n] n: Magazine number 1 to ...	R	W				5 . 2

15.2.14 Magazine module parameter

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$TC_MAMP1	STRING	$TC_MAMP1 Identifier of the magazine module Scalar variable	R	W				2
$TC_MAMP2	INT	$TC_MAMP2 Type of tool search Scalar variable	R	W				2
$TC_MAMP3	INT	$TC_MAMP3 Handling of tools with wear groups Scalar variable	R	W				5

Adapter data

$TC_ADPTT	INT	$TC_ADPTT[a] Adapter transformation number a: Adapter number 1–32000	R	W				5
$TC_ADPT1	REAL	$TC_ADPT1[a] Adapter geometry: Length 1 a: Adapter number 1–32000	R	W				5
$TC_ADPT2	REAL	$TC_ADPT2[a] Adapter geometry: Length 2 a: Adapter number 1–32000	R	W				5
$TC_ADPT3	REAL	$TC_ADPT3[a] Adapter geometry: Length 3 a: Adapter number 1–32000	R	W				5

15.2.15 Measuring system compensation values

$AA_ENC_COMP	REAL	$AA_ENC_COMP[n,m,a] Compensation values a: Machine axis n: Encoder no. 0–1 m: Point no. 0 – <MD value> Axes: Machine axis	R	W				2
$AA_ENC_COMP_STEP	REAL	$AA_ENC_COMP_STEP[n,a] Step width a: Machine axis n: Encoder no. 0–1 Axes: Machine axis	R	W				2
$AA_ENC_COMP_MIN	REAL	$AA_ENC_COMP_MIN[n,a] Compensation start position a: Machine axis n: Encoder no. 0–1 Axes: Machine axis	R	W				2
$AA_ENC_COMP_MAX	REAL	$AA_ENC_COMP_MAX[n,a] Compensation end position a: Machine axis n: Encoder no. 0–1 Axes: Machine axis	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_ENC_COMP_IS_MODULO	BOOL	$AA_ENC_COMP_IS_MODULO[n,a] Compensation is modulo a: Machine axis n: Encoder no. 0–1 Axes: Machine axis	R	W				2

15.2.16 Quadrant error compensation

$AA_QEC	REAL	$AA_QEC[n,m,a] Result of learning process a: Machine axis n: 0 m: No. of point: 0 – $MN_MM_QEC_MAX_POINTS	R	W				2
$AA_QEC_COARSE_STEPS	INT	$AA_QEC_COARSE_STEPS[n,a] Compensation value: Coarse quantization of the characteristic a: Machine axis n: 0	R	W				2
$AA_QEC_FINE_STEPS	INT	$AA_QEC_FINE_STEPS[n,a] Fine quantization of characteristic a: Machine axis n: 0	R	W				2
$AA_QEC_ACCEL_1	REAL	$AA_QEC_ACCEL_1[n,a] Acceleration in 1st knee-point according to definition [mm/s^2 o. inch/s^2 o. degrees/s^2] a: Machine axis n: 0	R	W				2
$AA_QEC_ACCEL_2	REAL	$AA_QEC_ACCEL_2[n,a] Acceleration in 2nd knee-point according to definition [mm/s^2 o. inch/s^2 o. degrees/s^2] a: Machine axis n: 0	R	W				2
$AA_QEC_ACCEL_3	REAL	$AA_QEC_ACCEL_3[n,a] Acceleration in 3rd knee-point according to definition [mm/s^2 o. inch/s^2 o. degrees/s^2] a: Machine axis n: 0	R	W				2
$AA_QEC_MEAS_TIME_1	REAL	$AA_QEC_MEAS_TIME_1[n,a] Measuring time for range $AA_QEC_ACCEL_1 a: Machine axis n: 0	R	W				2
$AA_QEC_MEAS_TIME_2	REAL	$AA_QEC_MEAS_TIME_2[n,a] Measuring time for range $AA_QEC_ACCEL_2 a: Machine axis n: 0	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_QEC_MEAS_TIME_3	REAL	$AA_QEC_MEAS_TIME_3[n,a] Measuring time for range $AA_QEC_ACCEL_3 a: Machine axis n: 0	R	W				2
$AA_QEC_TIME_1	REAL	$AA_QEC_TIME_1[n,a] 1st filter time for feedforward element a: Machine axis n: 0	R	W				2
$AA_QEC_TIME_2	REAL	$AA_QEC_TIME_2[n,a] 2nd filter time for feedforward element a: Machine axis n: 0	R	W				2
$AA_QEC_LEARNING_RATE	REAL	$AA_QEC_LEARNING_RATE[n,a] Learning rate for network a: Machine axis n: 0	R	W				2
$AA_QEC_DIRECTIONAL	BOOL	$AA_QEC_DIRECTIONAL[n,a] TRUE: Compensation is directional FALSE: Compensation is not directional a: Machine axis n: 0	R	W				2

15.2.17 Interpolatory compensation

$AN_CEC	REAL	$AN_CEC[n,m] Compensation value n: No. of compensation table 0 – (maximum value settable via MD) m: No. of interpolation point, 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_INPUT_AXIS	AXIS	$AN_CEC_INPUT_AXIS[n]: Name of axis whose setpoint is to act as the compensation table input n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_OUTPUT_AXIS	AXIS	$AN_CEC_OUTPUT_AXIS[n]: Name of axis which is influenced by the compensation table output n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_STEP	REAL	$AN_CEC_STEP[n] Distance between compensation values n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AN_CEC_MIN	REAL	AN_CEC_MIN[n] Start position of compensation table n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_MAX	REAL	AN_CEC_MAX[n] End position of compensation table n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_DIRECTION	INT	$AN_CEC_DIRECTION[n] Activates directional action of compensation table n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_MULT_BY_TABLE	INT	$AN_CEC_MULT_BY_TABLE[n] Number of table for which the initial value is to be multiplied by the initial value of the compensation table 0: Both traversing directions of basic axis 1: Positive traversing direction of basic axis –1: Negative traversing direction of basic axis n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2
$AN_CEC_IS_MODULO	BOOL	$AN_CEC_IS_MODULO[n] TRUE: Cycl. repetition of compensation table FALSE: No cycl. repetition of compensation table n: No. of compensation table 0 – (maximum value settable via MD)	R	W				2

15.2.18 NCK-specific protection zones

$SN_PA_ACTIV_IMMED	BOOL	$SN_PA_ACTIV_IMMED[n] Protection zone active immediately? TRUE: The protection zone is active immediately upon start-up of the control and referencing of the axes FALSE: The protection zone is not active immediately n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SN_PA_T_W	CHAR	$SN_PA_T_W[n] Workpiece/tool-oriented prot. zone 0: Workpiece-oriented protection zone 3: Tool-oriented protection zone n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SN_PA_ORI	INT	$SN_PA_ORI[n] Orientation of protection zone 0: Polygon in plane from 1st and 2nd geo axis 1: Polygon in plane from 3rd and 1st geo axis 2: Polygon in plane from 2nd and 3rd geo axis n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$SN_PA_LIM_3DIM	INT	$SN_PA_LIM_3DIM[n] Code for restricting the protection zone in the axis that lies parallel to the polygon definition 0: = No limit 1: = Limit in positive direction 2: = Limit in negative direction 3: = Limit in both directions n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SN_PA_PLUS_LIM	REAL	$SN_PA_PLUS_LIM[n] Positive limit for the protection zone in the axis that lies perpendicular to the polygon definition n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SN_PA_MINUS_LIM	REAL	$SN_PA_MINUS_LIM[n] Negative limit for protection zone in minus axis direction that lies perpend. to the polygon definition n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SN_PA_CONT_NUM	INT	$SN_PA_CONT_NUM[n] Number of valid contour elements n: Number of protection zone 0 – (maximum value settable via MD)	R	W				2
$SN_PA_CONT_TYP	INT	$SN_PA_CONT_TYP"[n,m] Contour element type (G1, G2, G3) n: Number of protection zone 0 – (maximum value settable via MD) m: Number of contour element 0 – 10	R	W				2
$SN_PA_CONT_ORD	REAL	$SN_PA_CONT_ORD[n,m] End point of contour element (ordinate) n: Number of protection zone 0 – (maximum value settable via MD) m: Number of contour element 0 – 10	R	W				2
$SN_PA_CONT_ABS	REAL	$SN_PA_CONT_ABS[n,m] End point of contour element (abscissa) n: Number of protection zone 0 – (maximum value settable via MD) m: Number of contour element 0 – 10	R	W				2
$SN_PA_CENT_ORD	REAL	$SN_PA_CENT_ORD[n,m] Center point of contour element (ordinate) n: Number of protection zone 0 – (maximum value settable via MD) m: Number of contour element 0 – 10	R	W				2
$SN_PA_CENT_ABS	REAL	$SC_PA_CENT_ABS[n,m] Center point of contour element (abscissa) n: Number of protection zone 0 – (maximum value settable via MD) m: Number of contour element 0 – 10	R	W				2

15.2.19 System data

$AN_SETUP_TIME	REAL	IF $AN_SETUP_TIME > 60000 GOTOF MARK01 Time since last power up of control with default values (in minutes)	RS		R			5.2
$AN_POWERON_TIME	REAL	IF $AN_POWERON_TIME == 480 GOTOF MARK02 Time since last standard power-on of control (in minutes)	RS		R			5.2

15.2.20 Frames 2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_UBFR	FRAME	$P_UBFR **1.** Base frame in channel activated after G500, G54..G599. Corresponds to $P_CHBFR[0].	R	W				4
$P_CHBFRAME	FRAME	$P_CHBFRAME[n] Current base frame in the channel. Configurable from 0 to 8 via MD $MC_MM_NUM_BASE_FRAMES. The dimension is checked on variable access.	R	W				5
$P_NCBFRAME	FRAME	$P_NCBFRAME[n] Current NCU base frame. 0 to 8 NCU base frames can be configured via MD $MN_MM_NUM_GLOBAL_BASE_FRAMES. The dimension is checked on variable access.	R	W				5
$P_ACTBFRAME	FRAME	$P_ACTBFRAME Current linked overall basic frame	R					5
$P_BFRAME	FRAME	$P_BFRAME Current 1st base frame in the channel. Corresponds to $P_CHBFRAME[0].	R	W				4
$P_IFRAME	FRAME	$P_IFRAME Current settable frame	R	W				2
$P_PFRAME	FRAME	$P_PFRAME Current programmable frame	R	W				2
$P_ACTFRAME	FRAME	$P_ACTFRAME Current total frame	R					2
$P_UIFRNUM	INT	$P_UIFRNUM Number of the active $P_UIFR	R					2
$P_NCBFRMASK	INT	$P_NCBFRMASK Bit screenform is used for definition of NCU-global base frames, which are included in the calculation of the whole base frame.	R	W				5
$P_CHBFRMASK	INT	$P_CHBFRMASK Bit screenform is used for definition of channel-specific base frames, which are included in the calculation of the whole base frame.	R	W				5

15.2.21 Tool data

$P_AD	REAL	$P_AD[n] Active tool offsets n: Parameter number 1 – 27	R	W				2
$P_TOOL	INT	$P_TOOL Active tool cutting edge D0 – D'max.'; 'max'= value of $MN_MM_MAX_CUTTING_EDGE_NO	R					2
$P_TOOLNO	INT	$P_TOOLNO Active tool number T0 – T32000; T can be 8-digit if the 'flat D number' is active	R					2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_TOOLL	REAL	$P_TOOLL[n] Active total tool length n: Length 1 – 3	R					2
$P_TCANG	REAL	$P_TCANG[n] Active angle of toolholder axis n: Angle 1 – 2	R					5
$P_TOOLR	REAL	$P_TOOLR Active tool radius (total)	R					2
$P_TOOLND	INT	$P_TOOLND[t] Number of cutting edges of tool t t: T number 1 – 32000	R					4
$P_TOOLEXIST	BOOL	$P_TOOLEXIST[t] Tool with T No. t exists t: T number 1 – 32000	R					4
$P_D	INT	$P_D Current D number in ISO_2-language mode	R					5.2
$P_H	INT	$P_H Current H number in ISO_2-language mode	R					5.2
$A_TOOLMN	INT	$A_TOOLMN[t] Magazine number of tool t t: T number 1 – 32000			R			4
$A_TOOLMLN	INT	$A_TOOLMLN[t] Magazine number of tool t t: T number 1 – 32000			R			4
$A_MONIFACT	REAL	$A_MONIFACT Factor for tool length monitoring	R	W S	R	W		4
$AC_MONMIN	REAL	$AC_MONMIN Ratio between tool monitoring actual value and setpoint. Threshold for tool search strategy "load only tools with actual value greater than threshold"	R	W S	R	W		5.2
$P_VDITCP	INT	$P_VDITCP[n] Available parameters for tool management on VDI interface n: Index 1 – 3	R	W				2
$A_DNO	INT	$A_DNO[i] Read a D number defined by the PLC via VDI interface i: Index 1 – 9 for table location in D number table			R			4
$P_ATPG	REAL	$P_ATPG[n] Current tool-related grinding data n: Parameter number 1 – 9	R	W				2

15.2.22 Programmed values

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_AXN1	AXIS	$P_AXN1 Current address of the geometry axis – abscissa	R					3
$P_AXN2	AXIS	$P_AXN2 Current address of the geometry axis – ordinate	R					3
$P_AXN3	AXIS	$P_AXN3 Current address of the geometry axis – applicate	R					3
$P_ACTGEOAX	AXIS	$P_ACTGEOAX[1] Current geometry axis assignment, dependent on plane Returns the current geometry axis assignment programmed with GEOAX(1,X,2,Y,3,Z) Array index 1–3 for 1st to 3rd geometry axis n: Number of input 1 – ...	R					4

15.2.23 G groups

$P_GG	INT	$P_GG[n] Current G function of a G group (index as PLC interface) n: Number of the G group	R					2
$P_EXTGG	INT	$P_EXTGG[n] Can only be used in Siemens mode: Current G function of a G group with external NC languages (index as PLC interface) n: Number of the G group	R					5
$A_GG	INT	$A_GG[n] Read current G function of a G group (index as PLC interface) from SA (index as PLC interface) n: Number of the G group			R			5

$P_SEARCH	BOOL	$P_SEARCH Block search is active if TRUE (1)	R					2
$P_SEARCH1	BOOL	$P_SEARCH1 Block search with calculation is active if TRUE (1)	R					2
$P_SEARCH2	BOOL	$P_SEARCH2 Block search without calculation is active if TRUE (1)	R					2
$P_SEARCHL	INT	R1 = $P_SEARCHL Returns the last selected search type: (coding analogous to PIservice _N_FINDBL) 0 : No block search 1 : Block search without calculation 2 : Block search with calculation on contour 3 : Reserved 4 : Block search with calculation at end of block position	R					5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_SUBPAR	BOOL	$P_SUBPAR[n] Queries whether the subprogram with parameter transfer for parameter n has actually been programmed (TRUE) or whether the system has used a default parameter (FALSE). n: Parameter number 1 to n according to definition in PROC instruction	R					5
$P_CTABDEF	BOOL	$P_CTABDEF Definition of curve tables is active if TRUE (1)	R					4
$P_MC	INT	$P_MC Status of modal subprogram call FALSE (0) -> Subprogram call not modal TRUE (1) -> Subprogram call modal	R					2
$P_REPINF	INT	$P_REPINF Status info for repositioning with REPOS command (0) -> Repositioning with REPOS not possible for following reasons – Call is not executed in an ASUP – Call is executed in an ASUP, which was started in the reset state – Call is executed in an ASUP, which was started in Jog mode (1) -> Repositioning with REPOS possible	R					4
$P_SIM	BOOL	$P_SIM Simulation runs if TRUE (1)	R					2
$P_DRYRUN	BOOL	$P_DRYRUN Dry run on if TRUE, else FALSE	R					2
$P_OFFN	REAL	$P_OFFN Programmed offset contour normal	R					5.1
$PI	REAL	$PI Circle constant PI = 3.1415927	R					2
$P_PROGPATH	STRING	PCALL ($P_PROGPATH << _N_MYSUB_SPF) Call a subprogram from the current directory Example: The current directory is /_N_WCS_DIR/_N_SHAFT_DIR/. The above call starts the subprogram /_N_WCS_DIR/_N_SHAFT_DIR/_N_MYSUB_SPF.	R					3
$P_PROG	STRING	mmcNum = 474 NAME = $P_PROG[0] Returns the program name of the program in program level 0 = main program name, in string variable NAME defines the program level from which the program name is to be read	R					5.1
$P_STACK	INT	$P_STACK Returns the program level in which a part program is active. progLevel = $P_STACK , writes the number of the current program level into the integer variable	R					5.1

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_PATH	STRING	$P_PATH[0] returns the directory of the current main program, e.g. "_N_WCS_DIR/_N_SHAFT_WPD" The variable is used to store a subroutine generated with WRITE, for example, in the same directory where the calling program resides: DEF INT ERROR WRITE (ERROR, $P_PATH[$P_STACK – 1] << _N_LIST_MPF, "X10 Y20") If the current program is called from the main program directory, a new file /_N_MPF_DIR/_N_LIST_MPF is created Defines the program level from which the program path is to be read	R					5.1
$P_ACTID	BOOL	$P_ACTID[n] Modal synchronized action with ID n active if TRUE n: 1–16	R					2

15.2.24 Channel statuses

$AC_STAT	INT	$AC_STAT –1: Invalid 0: Channel in reset mode 1: Channel interrupted 2: Channel active			R			4
$AC_PROG	INT	$AC_PROG –1: Invalid 0: Program in reset mode 1: Program stopped 2: Program active 3: Program waiting 4: Program interrupted			R			4
$AC_SYNA_MEM	INT	$AC_SYNA_MEM Free memory for motion-synchronized actions indicated how many elements of the memory occupied by $MC_MM_NUM_SYNC_ELEMENTS are still free, can be read from the part program and motion-synchronized actions			R			4
$AC_IPO_BUF	INT	$AC_IPO_BUF Level of interpolation buffer, can be read from the part program and motion-synchronized actions The status is read from the part program without feedforward stop while interpreting the block			R			4
$AC_IW_STAT	INT	$AC_IW_STAT Position information of the articulated joints (transformation-specific) for PTP travel	RS		R			5.2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_IW_TU	INT	$AC_IW_TU Position information of the axes (MCS) for PTP travel	RS		R			5.2
$A_PROBE	INT	$A_PROBE[1]: Status of first probe $A_PROBE[2]: Status of second probe 0 => not deflected 1 => deflected n: Number of probe	RS		R			4
$AC_MEA	INT	$AC_MEA[n] Probe has been triggered if TRUE (1) n: Number of probe 1 – MAXNUM_PROBE			R			2
$AC_TRAFO	INT	$AC_TRAFO Code number of the active transformation according to machine data $MC_TRAFO_TYPEn	RS		R			3
$AC_LIFTFAST	INT	$AC_LIFTFAST Information about execution of liftfast. 0: Initial state. 1: Liftfast has been executed. The variable is set to 1 by the NC at the start of the list fast process. The program evaluating the variable (if one exists) is set back to the basic setting ($AC_LIFTFAST=0) in order to be able to detect a subsequent list fast.	RS	W S	R	W		4

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_ASUP	INT	$AC_ASUP Code number for the cause of the ASUP activation. The reasons bit-coded and have the following meanings: **BIT0:** Activation due to: User interrupt "ASUP with Blsync" Activation by: VDI signal, digital/analog interface Continuation by: user-selectable Reorg or Ret **BIT1:** Activation due to: User interrupt "ASUP" For program continuation with Repos, the position after stopping is stored. Activation by: VDI signal, digital/analog interface Continuation by: user-selectable **BIT2**: Activation due to: User interrupt "ASUP from Ready channel status" Activation by: VDI signal, digital/analog interface Continuation by: user-selectable **BIT3:** Activation due to: User interrupt "ASUP in a manual mode and Ready channel status" Activation by: VDI signal, digital/analog interface Continuation by: user-selectable **BIT4:** Activation due to: User interrupt "ASUP" For program continuation with Repos, the current position where the interrupt occurred is stored. Activation by: VDI signal, digital/analog interface Continuation by: user-selectable **BIT5:** Activation due to: Cancelation of subprogram repetition Activation by: VDI signal Continuation by: using system ASUP REPOS **BIT6:** Activation due to: Activation of decoding single block Activation by: VDI signal (+OPI) Continuation by: using system ASUP REPOS **BIT7:** Activation due to: Activation of delete distance to go Activation by: VDI signal Continuation by: using system ASUP Ret **BIT8:** Activation due to: Activation of axis synchronization Activation by: VDI signal Continuation by: using system ASUP REPOS **BIT9:** Activation due to: Mode change Activation by: VDI signal Continuation by: using system ASUP REPOS or RET (see MD.) **BIT10:** Activation due to: Program continuation with teach-in or after teach-in deactivation Activation by: VDI signal Continuation by: using system ASUP Ret **BIT11:** Activation due to: Overstore selection Activation by: PI selection Continuation by: using system ASUP REPOS **BIT12:** Activation due to: Alarm with reaction of compensation block with Repos (COMPBLOCKWITHREORG) Activation by: Internal Continuation by: using system ASUP REPOS **BIT13:** Activation due to: Retraction movement on G33 and Stop	RS		R			4

		Activation by: Internal Continuation by: using system ASUP Ret **BIT14:** Activation due to: Activation of dry run feedrate Activation by: VDI Continuation by: using system ASUP REPOS **BIT15:** Activation due to: Deactivation of dry run feedrate Activation by: VDI Continuation by: using system ASUP REPOS **BIT16:** Activation due to: Activation of block suppression Activation by: VDI Continuation by: using system ASUP REPOS **BIT17:** Activation due to: Deactivation of block suppression Activation by: VDI Continuation by: using system ASUP REPOS **BIT18:** Activation due to: Set machine data active Activation by: PI Continuation by: using system ASUP REPOS						

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_ISTEST	BOOL	$P_ISTEST Check test mode in part program TRUE = Program test active FALSE = Program test not active	R					4
$P_MMCA	STRING	$P_MMCA MMC task acknowledgment	R	W				2
$A_PROTO	BOOL	$A_PROTO Activate/deactivate log function	RS	WS	R	W		4

15.2.25 Synchronized actions

$AC_MARKER	INT	$AC_MARKER[n] Marker variable for motion-synchronized actions The dimension is defined in MD $MC_MM_NUM_AC_MARKER.	RS	WS	R	W	+	2
$AC_PARAM	REAL	$AC_PARAM[n] Arithmetic variable for motion-synchronized actions The dimension is defined in MD $MC_MM_NUM_AC_PARAM.	RS	WS	R	W	+	3

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_FIFO1	REAL	$AC_FIFO1[n] FIFO for motion-synchronized actions and cyclic measurements n: Parameter number 0 – max. FIFO element Special meaning: n=0: On write accesses with index 0, a new value is stored in the FIFO, On read accesses with 0, the oldest element is read and deleted from the FIFO n=1: Read access to oldest element n=2: Read access to latest element n=3: Total of all elements in the FIFO if bit 0 is set in MD $MC_MM_MODE_FIFO n=4: Read access to current number of FIFO elements n=5–m: Read access to individual FIFO elements 5 is the oldest element 6 is the second-oldest, etc.	RS	W	R	W	+	4
$AC_FIFO2	REAL	$AC_FIFO2[n] FIFO for motion-synchronized actions and cyclic measurements n: Parameter number 0 – max. FIFO element Special meaning: n=0: On write accesses with index 0, a new value is stored in the FIFO, On read accesses with 0, the oldest element is read and deleted from the FIFO n=1: Read access to oldest element n=2: Read access to latest element n=3: Total of all elements in the FIFO if bit 0 is set in MD $MC_MM_MODE_FIFO n=4: Read access to current number of FIFO elements n=5–m: Read access to individual FIFO elements 5 is the oldest element 6 is the second-oldest, etc.	RS	W	R	W	+	4
...	...	...						

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
...	...	...						
$AC_FIFO10	REAL	$AC_FIFO10[n] FIFO for motion-synchronized actions and cyclic measurements n: Parameter number 0 – max. FIFO element Special meaning: n=0: On write accesses with index 0, a new value is stored in the FIFO, On read accesses with 0, the oldest element is read and deleted from the FIFO n=1: Read access to oldest element n=2: Read access to latest element n=3: Total of all elements in the FIFO if bit 0 is set in MD $MC_MM_MODE_FIFO n=4: Read access to current number of FIFO elements n=5–m: Read access to individual FIFO elements 5 is the oldest element 6 is the second-oldest, etc.	RS	W	R	W	+	4

15.2.26 I/Os

$A_IN	BOOL	$A_IN[n] Digital input NC n: Number of input 1 – ... The max. input number is determined by MD $MN_FASTIO_DIG_NUM_INPUTS	RS		R			2
$A_OUT	BOOL	$A_OUT[n] Digital output NC n: Number of output 1 – ... The max. input number is determined by MD $MN_FASTIO_DIG_NUM_OUTPUTS	RS	W	R	W		2
$A_INA	REAL	$A_INA[n] Analog input NC n: Number of input 1 – ... The max. input number is determined by MD $MN_FASTIO_ANA_NUM_INPUTS	RS		R			2
$A_OUTA	REAL	$A_OUTA[n] Analog output NC When writing, the value does not become active until the next IPO cycle at which point it is read back. n: Number of output 1 – ... The max. input number is determined by MD $MN_FASTIO_ANA_NUM_OUTPUTS	RS	W	R	W		2
$A_INCO	BOOL	$A_INCO[n] Comparator input n: Number of output 1 – ... The max. input number is determined by MD	RS		R			2

15.2.27 Reading and writing PLC variables

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$A_DBB	INT	$A_DBB[n] Read/write data byte (8 bits) from/to PLC n: Position offset within I/O area 0 – ...	RS	W	R	W		4
$A_DBW	INT	$A_DBW[n] Read/write data word (16 bits) from/to PLC n: Position offset within I/O area 0 – ...	RS	W	R	W		4
$A_DBD	INT	$A_DBD[n] Read/write double data word (32 bits) from/to PLC n: Position offset within I/O area 0 – ...	RS	W	R	W		4
$A_DBR	REAL	$A_DBR[n] Read/write Real data (32 bits) from/to PLC n: Position offset within I/O area 0 – ...	RS	W	R	W		4

15.2.28 NCU link

$A_DLB	INT	$A_DLB[n] Read/write data byte (8 bits) from/to NCU link n: Position offset within the link memory area 0 – ... synchronized with main run	RS	W	R	W		5
$A_DLW	INT	$A_DLW[n] Read/write data word (16 bits) from/to NCU link n: Position offset within the link memory area 0 – ... synchronized with main run	RS	W	R	W		5
$A_DLD	INT	$A_DLD[n] Read/write data doubleword (32 bits) from/to NCU link n: Position offset within the link memory area 0 – ... synchronized with main run	RS	W	R	W		5
$A_DLR	REAL	$A_DLR[n] Read/write Real data (32 bits) from/to NCU link n: Position offset within the link memory area 0 – ... synchronized with main run	RS	W	R	W		5
$A_LINK_TRANS_RATE	INT	$A_LINK_TRANS_RATE Number of bytes that can still be transferred via the NCU link communication in the current IPO cylce.			R			5

15.2.29 Direct PLC I/O

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$A_PBB_IN	INT	$A_PBB_IN[n] Read data byte (8 bits) directly from PLC I/O n: Byte offset within PLC input area 0 – ...	RS		R			5
$A_PBW_IN	INT	$A_PBW_IN[n] Read data word (16 bits) directly from PLC I/O n: Byte offset within PLC input area 0 – ...	RS		R			5
$A_PBD_IN	INT	$A_PBD_IN[n] Read data doubleword (32 bits) directly from PLC I/O n: Byte offset within PLC input area 0 – ...	RS		R			5
$A_PBR_IN	REAL	$A_PBR_IN[n] Read Real data (32 bits) directly from PLC I/O n: Byte offset within PLC input area 0 – ...	RS		R			5 . 2
$A_PBB_OUT	INT	$A_PBB_OUT[n] Write data byte (8 bits) directly to PLC I/O n: Byte offset within the PLC output area 0 – ... synchronized with main run	RS	W	R	W		5
$A_PBW_OUT	INT	$A_PBW_OUT[n] Write data word (16 bits) directly to PLC I/O n: Byte offset within the PLC output area 0 – ... synchronized with main run	RS	W	R	W		5
$A_PBD_OUT	INT	$A_PBD_OUT[n] Write data doubleword (32 bits) directly to PLC I/O n: Byte offset within the PLC output area 0 – ... synchronized with main run	RS	W	R	W		5
$A_PBR_OUT	REAL	$A_PBR_OUT[n] Write Real data (32 bits) directly to PLC I/O n: Byte offset within the PLC output area 0 – ... synchronized with main run	RS	W	R	W		5

15.2.30 Tool management

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_TC_FCT	INT	$AC_TC_FCT Command number. Specifies which action is desired.	RS		R			5
$AC_TC_STATUS	INT	$AC_TC_STATUS Status of command – read via $AC_TC_FCT.	RS		R			5
$AC_TC_THNO	INT	$AC_TC_THNO Number of the toolholder (spec. the spindle no.) where the new tool is to be changed.	RS		R			5
$AC_TC_TNO	INT	$AC_TC_TNO NCK-internal T number of new tool (to be changed). 0: There is no new tool.	RS		R			5
$AC_TC_MFN	INT	$AC_TC_MFN Source magazine number of new tool. 0: There is no new tool.	RS		R			5
$AC_TC_LFN	INT	$AC_TC_LFN Source location number of new tool. 0: There is no new tool.	RS		R			5
$AC_TC_MTN	INT	$AC_TC_MTN Target magazine number of new tool. 0: There is no new tool.	RS		R			5
$AC_TC_LTN	INT	$AC_TC_LTN Target location number of new tool. 0: There is no new tool.	RS		R			5
$AC_TC_MFO	INT	$AC_TC_MFO Source magazine number of old tool (to be changed). 0: There is no old tool.	RS		R			5
$AC_TC_LFO	INT	$AC_TC_LFO Source location number of old tool (to be changed). 0: There is no old tool.	RS		R			5
$AC_TC_MTO	INT	$AC_TC_MTO Target magazine number of old tool (to be changed). 0: There is no old tool.	RS		R			5
$AC_TC_LTO	INT	$AC_TC_LTO Target location number of old tool (to be changed). 0: There is no old tool.	RS		R			5

15.2.31 Timers

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$A_YEAR	INT	$A_YEAR System time, year	RS		R			3
$A_MONTH	INT	$A_MONTH System time, month	RS		R			3
$A_DAY	INT	$A_DAY System time, day	RS		R			3
$A_HOUR	INT	$A_HOUR System time, hour	RS		R			3
$A_MINUTE	INT	$A_MINUTE System time, minute	RS		R			3
$A_SECOND	INT	$A_SECOND System time, second	RS		R			3
$A_MSECOND	INT	$A_MSECOND System time, millisecond	RS		R			3
$AC_TIME	REAL	$AC_TIME Time from the beginning of block in seconds This variable can only be accessed from synchronized actions	RS		R			2
$AC_TIMEC	REAL	$AC_TIMEC Time from the beginning of block in IPO clock cycles This variable can only be accessed from synchronized actions	RS		R			3
$AC_TIMER	REAL	$AC_TIMER[n] Timer – unit in seconds Time is counted internally in multiples of interpolation cycle cycle; To start the counter variable, assign the value $AC_TIMER[n]=<starting value> To stop the counter variable, assign a negative value: $AC_TIMER[n]=–1 The current time value can be read when the timer is running or when it has stopped. When the timer is stopped by assigning the value –1, the most up-to-date timer value is retained and can be read. The dimension is defined in MD $MC_MM_NUM_AC_TIMER.	RS	W S	R	W	+	4
$AC_PRTIME_M	REAL	$AC_PRTIME_M "ProgramRunTIME-Main" Set (initialize) the accumulated program runtime (main time)		W				4
$AC_PRTIME_A	REAL	$AC_PRTIME_A "ProgramRunTIME-Auxiliary" Set (initialize) the accumulated program runtime (auxiliary time)		W				4
$AC_PRTIME_M_INC	REAL	$AC_PRTIME_M_INC "ProgramRunTIME-Main-INCrement" Increment the accumulated program runtime (main time)		W				4
$AC_PRTIME_A_INC	REAL	$AC_PRTIME_A_INC "ProgramRunTIME-Auxiliary-INCrement" Increment the accumulated program runtime (auxiliary time)		W				4

15.2.32 Path movement

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_PATHN	REAL	$AC_PATHN Normalized path parameter value between 0=start of block and 1=end of block This variable can only be accessed from synchronized actions	RS		R			2
$AC_DTBW	REAL	$AC_DTBW Geometric distance from start of block in workpiece coordinate system The programmed position is decisive for computing the distance; if the axis is a couppling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions	RS		R			2
$AC_DTBB	REAL	$AC_DTBB Geometric distance from start of block in basic coordinate system The programmed position is decisive for computing the distance; if the axis is a couppling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions	RS		R			2
$AC_DTEW	REAL	$AC_DTEW Geometric distance from end of block in workpiece coordinate system The programmed position is decisive for computing the distance; if the axis is a couppling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions	RS		R			2
$AC_DTEB	REAL	$AC_DTEB Geometric distance from end of block in basic coordinate system The programmed position is decisive for computing the distance; if the axis is a couppling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions	RS		R			2
$AC_PLTBB	REAL	$AC_PLTBB Path distance from start of block in basic coordinate system This variable can only be accessed from synchronized actions	RS		R			3
$AC_PLTEB	REAL	$AC_PLTEB Path distance from end of block in basic coordinate system This variable can only be accessed from synchronized actions	RS		R			3
$AC_DELT	REAL	$AC_DELT Unlatched residual path distance in workpiece coordinate system after delete distance-to-go with motion-synchronized actions			R			3
$P_APDV	BOOL	$P_APDV Returns True if the positional values which can be read with $P_APR[X] or $P_AEP[X] (the start point or contour point for soft approach and retraction) are valid.	R					4

15.2.33 Velocities

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_F	REAL	$P_F Path feed F last programmed	R					2
$AC_OVR	REAL	$AC_OVR: Path override for synchronized actions Multiplicative override component acting in addition to the user OV, programmed OV and transformation OV. The total factor is restricted to 200%, however. It must be rewritten in every interpolator cycle, otherwise the value applies 100%. $AA_OVR[S1] changes the spindle override. The override defined by machine data $MN_OVR_FACTOR_LIMIT_BIN, $MN_OVR_FACTOR_FEEDRATE[30], $MN_OVR_FACTOR_AX_SPEED[30], is not exceeded This variable can only be accessed from synchronized actions			R	W		2
$AC_VC	REAL	$AC_VC Additive path feed compensation for synchronized actions The compensation is not effective with G0, G33, G331, G332 and G63. It must be rewritten in every interpolator cycle, otherwise the value is 0. With an override of 0, the compensation value has no effect, otherwise the override has no impact on the compensation value. The compensation value cannot make the total feedrate negative. The upper value is limited such that the maximum axis velocities and accelerations are not exceeded. The computation with different feedrate components is not affected by $AC_VC. The override values defined by machine data $MN_OVR_FACTOR_LIMIT_BIN, $MN_OVR_FACTOR_FEEDRATE[30], $MN_OVR_FACTOR_AX_SPEED[30] and $MN_OVR_FACTOR_SPIND_SPEED cannot be exceeded. The additive feedrate override is limited such that the resulting feedrate does not exceed the maximum override value of the programmed feedrate. This variable can only be accessed from synchronized actions			R	W		2
$AC_VACTB	REAL	$AC_VACTB Path velocity in the base coordinate system This variable can only be accessed from synchronized actions	RS		R			2
$AC_VACTW	REAL	$AC_VACTW Path velocity in workpiece coordinate system This variable can only be accessed from synchronized actions	RS		R			2

15.2.34 Spindles

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$P_GWPS	BOOL	$P_GWPS[n] Constant grinding wheel surface speed on if TRUE n: Spindle number, 0 – max. spindle number	R					2
$P_NUM_SPINDLES	INT	$P_NUM_SPINDLES[n] Number of spindles in the channel	R					5 3
$P_MSNUM	INT	$P_MSNUM Return value: 0: No spindle exists 1..n: Number of the master spindle	R					5 . 2
$AC_MSNUM	INT	$AC_MSNUM Return value: 0: No spindle exists 1..n: Number of the master spindle	RS		R			3
$P_S	REAL	$P_S[n] Last programmed spindle speed n: Spindle number, 0 – max. spindle number	R					2
$AA_S	REAL	$AA_S[n] Spindle act. speed. The sign corresponds to the direction of rotation. n: Spindle number, 0 – max. spindle number	RS		R			4
$P_SDIR	INT	$P_SDIR[n] Last direction of spindle rotation to be programmed. 3: Clockwise rotation, 4: Counterclockwise rotation, 5: Spindle stop n: Spindle number, 0 – max. spindle number	R					3
$AC_SDIR	INT	$AC_SDIR[n] Spindle rotation direction currently active 3: Clockwise rotation, 4: Counterclockwise rotation, 5: Spindle stop n: Spindle number, 0 – max. spindle number	RS		R			3
$P_SEARCH_S	REAL	$P_SEARCH_S[n] accummulated last progr. spindle speed for block search (SSL) 0: Spindle standstill, 0 – last programmed spindle speed	R					5 . 3
$P_SEARCH_SDIR	INT	$P_SEARCH_SDIR[n] accum. last programmed spindle direction of rotation for block search 3: M3 output speed control mode 4: M4 output speed control mode 5: M5 output speed control mode –5: Preset spindle not programmed at time of SSL start –19: M19 output positioning mode 70: M70 output axis mode n: Spindle number, 0 – max. spindle number	R					5 . 3
$P_SEARCH_SGEAR	INT	$P_SEARCH_SDIR[n] accumm. last programmed spindle gear stage M function for SSL 40: M40 automatic gear stage change 41: M41 as predefined gear stage in part program 42: M42 as predefined gear stage in part program 43: M43 as predefined gear stage in part program 44: M44 as predefined gear stage in part program 45: M45 as predefined gear stage in part program n: Spindle number, 0 – max. spindle number	R					5 . 3

$P_SEARCH_POS	REAL	$P_SEARCH_SPOS[n] accum. last programmed spindle position or –path for SSL Value range:from –100000000 to 100000000. –100000000 to –0,001: possible spindle path in negative range 100000000 to 0,000: possible spindle path in positive range Path and position definitions may be positive or negative and defined up to three decimal places. n: Positioning data must lie inside the modulo range	R	W				5.3
$P_SEARCH_POSMODE	INT	$P_SEARCH_SMODE[n] accummulated last programmed position approach mode for SSL 0: DC (default) 1: AC 2: IC 3: DC 4: ACP 5: ACN n: Spindle number, 0 – max.	R	W				5.3
$P_SAUTOGEAR	BOOL	$P_SAUTOGEAR[n] programmed gear stage change 0: no automatic gear stage change 1: automatic gear stage change is active	R					5.3
$P_SGEAR	INT	$P_SGEAR[n] last programmed/requested gear stage GS 1:1st gear stage is programmed/requested 2: 2. Requested gear stage 3: 3. Requested gear stage 4: 4. Requested gear stage 5: 5. Requested gear stage n: Gear stage, 0 – max. gear stage	R					5.3
$AC_SGEAR	INT	$AC_SGEAR[n] currently active gear stage 1: 1. Requested gear stage 2: 2. Requested gear stage 3: 3. Requested gear stage 4: 4. Requested gear stage 5: 5. Requested gear stage n: Gear stage, 0 – max. gear stage	RS		R			5.3
$P_SMODE	INT	$P_SMODE[n] Last programmed spindle mode: 0: No spindle or in other channel or PLC spindle 1: Speed control mode 2: Positioning mode 3: synchronized mode 4: Axis mode n: Spindle number, 0 – max. spindle number	R					3
$AC_SMODE	INT	$AC_SMODE[n] Spindle mode currently active 0: No spindle exists 1: Speed control mode 2: Positioning mode 3: synchronized mode 4: Axis mode n: Spindle number, 0 – max. spindle number	RS		R			3

15.2.35 Polynomial values for synchronized actions

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_FCT1LL	REAL	$AC_FCT1LL Lower limit value for evaluation function FCTDEF 1	RS	W S	R	W	+	2
$AC_FCT2LL	REAL	$AC_FCT2LL Lower limit value for evaluation function FCTDEF 2	RS	W S	R	W	+	2
$AC_FCT3LL	REAL	$AC_FCT3LL Lower limit value for evaluation function FCTDEF 3	RS	W S	R	W	+	2
$AC_FCT1UL	REAL	$AC_FCT1UL Upper limit value for evaluation function FCTDEF 1	RS	W S	R	W	+	2
$AC_FCT2UL	REAL	$AC_FCT2UL Upper limit value for evaluation function FCTDEF 2	RS	W S	R	W	+	2
$AC_FCT3UL	REAL	$AC_FCT3UL Upper limit value for evaluation function FCTDEF 3	RS	W S	R	W	+	2
$AC_FCT1C	REAL	$AC_FCT1C[n] Polynomial coefficient a0 – a3 for evaluation function FCTDEF 1 n: Degree of coefficient 0 – 3	RS	W S	R	W	+	2
$AC_FCT2C	REAL	$AC_FCT2C[n] Polynomial coefficient a0 – a3 for evaluation function FCTDEF 2 n: Degree of coefficient 0 – 3	RS	W S	R	W	+	2
$AC_FCT3C	REAL	$AC_FCT3C[n] Polynomial coefficient a0 – a3 for evaluation function FCTDEF 3 n: Degree of coefficient 0 – 3	RS	W S	R	W	+	2
$AC_FCTLL	REAL	$AC_FCTLL[n] Lower limit of polynomial for synchronized actions (SYNFCT) n: Number of polynomial, limited by machine data	RS	W S	R	W	+	4
$AC_FCTUL	REAL	$AC_FCTUL[n] Upper limit of polynomial for synchronized actions (SYNFCT) n: Number of polynomial, limited by machine data	RS	W S	R	W	+	4
$AC_FCT0	REAL	$AC_FCT0[n] a0 coefficient of polynomial for synchronized actions (SYNFCT) n: Number of polynomial, limited by machine data	RS	W S	R	W	+	4
$AC_FCT1	REAL	$AC_FCT1[n] a1 coefficient of polynomial for synchronized actions (SYNFCT) n: Number of polynomial, limited by machine data	RS	W S	R	W	+	4
$AC_FCT2	REAL	$AC_FCT2[n] a2 coefficient of polynomial for synchronized actions (SYNFCT) n: Number of polynomial, limited by machine data	RS	W S	R	W	+	4
$AC_FCT3	REAL	$AC_FCT3[n] a3 coefficient of polynomial for synchronized actions (SYNFCT) n: Number of polynomial, limited by machine data	RS	W S	R	W	+	4

15.2.36 Channel statuses

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_ALARM_STAT	INT	$AC_ALARM_STAT (Selected) alarm reactions for synchronized actions (SYNFCT)	RS		R			5
$AN_ESR_TRIGGER	BOOL	$AN_ESR_TRIGGER = 1 Trigger "Extended stop and retract"			R	W		5
$AC_OPERATING_TIME	REAL	IF $AC_OPERATING_TIME < 12000 GOTOB STARTMARK Total runtime of NC programs in Automatic mode (in seconds)	RS		R			5.2
$AC_CYCLE_TIME	REAL	IF $AC_CYCLE_TIME > 2400 GOTOF ALARM01 Runtime of selected NC program (in seconds)	RS		R			5.2
$AC_CUTTING_TIME	REAL	IF $AC_CUTTING_TIME > 6000 GOTOF ACT_M06 Tool operation time (in seconds)	RS		R			5.2
$AC_REQUIRED_PARTS	REAL	$AC_REQUIRED_PARTS = ACTUAL_LOS Definition of number of workpieces required, e.g. for definition of a batch size, daily production target, etc.	RS	WS	R	W		5.2
$AC_TOTAL_PARTS	REAL	IF $AC_TOTAL_PARTS > SERVICE_COUNT GOTOF MARK_END Total number of workpieces produced (total)	RS	WS	R	W		5.2
$AC_ACTUAL_PARTS	REAL	IF $AC_ACTUAL_PARTS == 0 GOTOF NEW_RUN Actual number of parts produced For $AC_ACTUAL_PARTS == $AC_REQUIRED_PARTS, $AC_ACTUAL_PARTS = 0 automatically.	RS	WS	R	W		5.2
$AC_SPECIAL_PARTS	REAL	$AC_SPECIAL_PARTS = R20 Number of workpieces counted according to a user strategy. Without internal impact.	RS	WS	R	W		5.2

15.2.37 Positions

$P_EP	REAL	$P_EP[X] Setpoint last programmed Axes: Channel axis	R					2
$P_APR	REAL	$P_APR[X] Position of axis in the workpiece coordinate system at the start of the approach motion for soft approach to the contour. Axes: Channel axis	R					4
$P_AEP	REAL	$P_AEP[X] Approach point: first contour point in the workpiece coordinate system for soft approach to contour Axes: Channel axis	R					4
$AA_IW	REAL	$AA_IW[X] Actual value in workpiece coordinate system (WCS) Axes: Channel axis	RS		R			2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_IEN	REAL	$AA_IEN[X] Actual value in the settable origin system (SOS). Axes: Channel axis	RS		R			5
$AA_IBN	REAL	$AA_IBN[X] Actual value in the basic origin system (BOS). Axes: Channel axis	RS		R			5
$AA_IB	REAL	$AA_IB[X] Actual value in basic coordinate system (BCS) Axes: Channel axis	RS		R			2
$AA_IM	REAL	$AA_IM[X] Actual value in machine coordinate system (MCS). Axes: GEOAX, channel axis, machine axis	RS		R			2

15.2.38 Indexing axes

$AA_ACT_INDEX_AX_POS_NO	INT	$AA_ACT_INDEX_AX_POS_NO[X] 0: No indexing axis, therefore no indexing position available. > 0: Number of indexing position last reached or crossed Axes: Geometry axis, channel axis, machine axis	RS		R			5
$AA_PROG_INDEX_AX_POS_NO	INT	$AA_PROG_INDEX_AX_POS_NO[X] 0: Not an indexing axis, therefore no indexing position available or the indexing axis is not currently approaching an indexing position > 0: Number of programmed indexing position Axes: Geometry axis, channel axis, machine axis	RS		R			5

15.2.39 Encoder limit frequency

$AA_ENC_ACTIVE	BOOL	$AA_ENC_ACTIVE[X] Active measuring system is operating below encoder limit frequency Axes: Geometry axis, channel axis, machine axis	RS		R			4
$AA_ENC1_ACTIVE	BOOL	$AA_ENC1_ACTIVE[X] Encoder 1 is operating below encoder limit frequency Axes: Geometry axis, channel axis, machine axis	RS		R			4
$AA_ENC2_ACTIVE	BOOL	$AA_ENC2_ACTIVE[X] Encoder 2 is operating below encoder limit frequency Axes: Geometry axis, channel axis, machine axis	RS		R			4

15.2.40 Encoder values

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$VA_IM	REAL	$VA_IM[X] Encoder actual value in machine coordinate system (measured active measuring system), actual value compensations are corrected (leadscrew error compensation, backlash compensation, quadrant error compensation) Axes: Machine axis	RS		R			4
$VA_IM1	REAL	$VA_IM1[X] Actual value in machine coordinate system (measured on encoder 1), compensations corrected Axes: Machine axis	RS		R			4
$VA_IM2	REAL	$VA_IM2[X] Actual value in machine coordinate system (measured on encoder 2), compensations corrected Axes: Machine axis	RS		R			4
$AA_MW	REAL	$AA_MW[X] Measured value in workpiece coordinate system Axes: Channel axis	R	W S	R	W		2
$AA_MM	REAL	$AA_MW[X] Measured value in machine coordinate system Axes: Machine axis	R	W S	R	W		2
$AA_MW1	REAL	$AA_MW1[X] Measurement result of axial measurement Trigger event 1 in WCS Axes: Channel axis	R	W S	R	W		4
$AA_MW2	REAL	$AA_MW2[X] Measurement result of axial measurement Trigger event 2 in WCS Axes: Channel axis	R	W S	R	W		4
$AA_MW3	REAL	$AA_MW3[X] Measurement result of axial measurement Trigger event 3 in WCS Axes: Channel axis	R	W S	R	W		4
$AA_MW4	REAL	$AA_MW4[X] Measurement result of axial measurement Trigger event 4 in WCS Axes: Channel axis	R	W S	R	W		4

15.2.41 Axial measurement

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_MM1	REAL	$AA_MM1[X] Measurement result of axial measurement Trigger event 1 in MCS Axes: Channel axis	R	W S	R	W		4
$AA_MM2	REAL	$AA_MM2[X] Measurement result of axial measurement Trigger event 2 in MCS Axes: Channel axis	R	W S	R	W		4
$AA_MM3	REAL	$AA_MM3[X] Measurement result of axial measurement Trigger event 3 in MCS Axes: Channel axis	R	W S	R	W		4
$AA_MM4	REAL	$AA_MM4[X] Measurement result of axial measurement Trigger event 4 in MCS Axes: Channel axis	R	W S	R	W		4
$AA_MEAACT	BOOL	$AA_MEAACT[X] Value is TRUE if axial measurement is active for X Axes: Geometry axis, channel axis, machine axis			R			4

15.2.42 Offsets

$AC_DRF	REAL	$AC_DRF[X] DRF offset Axes: Channel axis	RS		R			2
$AC_PRESET	REAL	$AC_PRESET[X] Preset value last specified Axes: Channel axis	RS		R			2
$AA_ETRANS	REAL	$AA_ETRANS[X] External zero offset Axes: Channel axis	R	W				2
$AA_OFF	REAL	$AA_OFF[X] Overlaid motion for programmed axis This variable can only be accessed from synchronized actions Axes: Channel axis	RS	W	R	W		3
$AA_OFF_LIMIT	INT	$AA_OFF_LIMIT[axis] Limit value for axial offset $AA_OFF[axis] 0: Limit value not reached 1: Limit value reached in positive axis direction –1: Limit value reached in negative axis direction Axes: Channel axis	RS		R			4

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AC_RETPOINT	REAL	$AC_RETPOINT[X] Reset point on the contour for reapproach Axes: Channel axis	RS		R			2
$AA_SOFTENDP	REAL	$AA_SOFTENDP[X] Software limit position, positive direction Axes: Machine axis	RS		R			2
$AA_SOFTENDN	REAL	$AA_SOFTENDN[X] Software limit position, negative direction Axes: Machine axis	RS		R			2

15.2.43 Axial distances

$AA_DTBW	REAL	$AA_DTBW[X] axial path from start of block in the workpiece coordinate system for positioning and synchronized axes for motion synchronized action The programmed position is decisive for computing the path; if the axis is a coupling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2
$AA_DTBB	REAL	$AA_DTBB[X] Axial distance from start of block in basic coordinate system for positioning and synchronized axes with motion-synchronized actions The programmed position is decisive for computing the path; if the axis is a coupling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2
$AA_DTEW	REAL	$AA_DTEW[X] Axial distance to end of block in workpiece coordinate system for positioning and synchronized axes with motion-synchronized actions The programmed position is decisive for computing the path; if the axis is a coupling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2
$AA_DTEB	REAL	$AA_DTEB[X] Axial distance to end of block in basic coordinate system for positioning and synchronized axes with motion-synchronized actions The programmed position is decisive for computing the path; if the axis is a coupling axis, the position part that results from axis coupling is not considered here. This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2

15.2.44 Oscillation

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_DTEPW	REAL	$AA_DTEPW[X] Axial distance-to-go for infeed oscillation in workpiece coordinate system This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2
$AA_DTEPB	REAL	$AA_DTEPB[X] Axial distance-to-go for infeed oscillation in basic coordinate system This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2
$AA_OSCILL_RE VERSE_POS1	REAL	$AA_OSCILL_REVERSE_POS1[X] Current reversal position 1 for oscillation In synchronized actions, the setting data value $SA_OSCILL_REVERSE_POS1 is evaluated online This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			3
$AA_OSCILL_RE VERSE_POS2	REAL	$AA_OSCILL_REVERSE_POS2[X] Current reversal position 2 for oscillation In synchronized actions, the setting data value $SA_OSCILL_REVERSE_POS2 is computed online This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			3

$AA_DELT	REAL	$AA_DELT[X] Latched axial residual path distance in workpiece coordinate system after axial delete distance-to-go with motion-synchronized actions Axes: Geometry axis, channel axis, machine axis			R			2
$P_FA	REAL	$P_FA[X] Axis feed last programmed Axes: Channel axis	R					2

15.2.45 Axial velocities

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_OVR	REAL	$AA_OVR[X] Axial override for motion-synchronized actions Multiplicative override component acting in addition to the user OV, programmed OV and transformatory OV; the overall factor is limited to a maximum of 200% however. Must be rewritten in every interpolator cycle, otherwise the value is 100%. The spindle override is changed with $AA_OVR[S1]. The override defined by machine data $MN_OVR_FACTOR_LIMIT_BIN, $MN_OVR_FACTOR_FEEDRATE[30], $MN_OVR_FACTOR_AX_SPEED[30] and $AA_OVR_FACTOR_SPIND_SPEED is not exceeded This variable can only be accessed from motion-synchronized actions Axes: Channel axis			R	W		2
$AA_VC	REAL	$AA_VC[X] Additive axial feed compensation for motion-synchronized actions It must be rewritten in every interpolator cycle, otherwise the value is 0. With an override of 0, the compensation value has no effect, otherwise the override has no impact on the compensation value. The compensation value cannot make the total feedrate negative. The upper value is limited such that the maximum axis velocities and accelerations are not exceeded. The computation of the other feedrate components is not affected by $AA_VC. The override values defined by machine data $MN_OVR_FACTOR_LIMIT_BIN, $MN_OVR_FACTOR_FEEDRATE[30], $MN_OVR_FACTOR_AX_SPEED[30] and $MN_OVR_FACTOR_SPIND_SPEED cannot be exceeded. The additive feedrate override is limited such that the resulting feedrate does not exceed the maximum override value of the programmed feedrate. Axes: Channel axis			R	W		2
$AA_VACTB	REAL	$AA_VACTB[X] Axis velocity in the base coordinate system This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_VACTW	REAL	$AA_VACTW[X] Axis velocity in workpiece coordinate system This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			2
$AA_VACTM	REAL	$AA_VACTM[X] Axis velocity, setpoint-related in machine coordinate system Can also be read for replacement and PLC axes This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			4
$VA_VACTM	REAL	$VA_VACTM[X] Axis velocity, actual value-related in machine coordinate system The variable returns an undefined value if the encoder limit frequency is exceeded This variable can only be accessed from synchronized actions Axes: Channel axis	RS		R			4

15.2.46 Drive data

$AA_LOAD	REAL	$AA_LOAD[X] Drive utilization in % (for 611D only) Axes: Channel axis	RS		R			2
$VA_LOAD	REAL	$VA_LOAD[X] Drive utilization in % (for 611D only) Axes: Channel axis	RS		R			5.1
$AA_TORQUE	REAL	$AA_TORQUE[X] Drive torque setpoint in Nm (for 611D only) Axes: Channel axis	RS		R			2
$VA_TORQUE	REAL	$VA_TORQUE[X] Drive torque setpoint in Nm (for 611D only) Axes: Channel axis	RS		R			5.1
$AA_POWER	REAL	$AA_POWER[x] Drive active power in W (for 611D only) Axes: Channel axis	RS		R			2
$VA_POWER	REAL	$VA_POWER[x] Drive active power in W (for 611D only) Axes: Channel axis	RS		R			5.1
$AA_CURR	REAL	$AA_CURR[X] Actual current value of axis or spindle in A (for 611D only) Axes: Channel axis	RS		R			2
$VA_CURR	REAL	$VA_CURR[X] Actual current value of axis or spindle in A (for 611D only) Axes: Channel axis	RS		R			5.1
$VA_VALVELIFT	REAL	$VA_VALVELIFT[X] Actual valve stroke in mm (for 611D hydraulics only) Axes: Channel axis	RS		R			5.1

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$VA_PRESSURE_A	REAL	$VA_PRESSURE_A[X] Pressure on A side of cylinder in bar (for 611D hydraulics only) Axes: Channel axis	RS		R			5.1
$VA_PRESSURE_B	REAL	$VA_PRESSURE_B[X] Pressure on B side of cylinder in bar (for 611D hydraulics only) Axes: Channel axis	RS		R			5.1

15.2.47 Axis statuses

$AA_STAT	INT	$AA_STAT[X] Axis status: 0: No axis status available 1: Traversing motion in progress 2: Axis has reached IPO end (applies only to axes in channel) 3: Axis in position (exact stop coarse) for all axes 4: Axis in position (exact stop fine) for all axes Axes: Geometry axis, channel axis, machine axis	RS		R			4
$AA_REF	INT	$AA_REF[X] Axis status: 0: Axis is not referenced 1: Axis is referenced Axes: Geometry axis, channel axis, machine axis	RS		R			5
$AA_TYP	INT	$AA_TYPT[X] Axis type: 0: Axis on other channel 1: Channel axis of local channel 2: Neutral axis 3: PLC axis 4: Oscillating axis 5: Neutral axis currently traversing in JOG mode 6: Lead. value coupled follow. ax. 7: Coupled motion follow. axis 8: Command axis 9: Compile cycle axis Axes: Geometry axis, channel axis	RS		R			4

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_FXS	INT	$AA_FXS[X] Status "Travel to fixed stop" 0: Axis not at fixed stop 1: Fixed stop successfully approached 2: Fixed stop approach has failed 3: Selection of travel to fixed stop active 4: Deselection of travel to fixed stop active Axes: Geometry axis, channel axis, machine axis	RS		R			2
$AA_COUP_ACT	INT	$AA_COUP_ACT[SPI(2)] Current coupling status of following spindle/following axis: 0: Axis/spindle is not coupled to a leading spindle/leading axis 3: Tangential follow-up of axis 4: synchronized spindle coupling 8: Axis is trailing 16: Following axis of master value coupling The respective values apply to one coupling. If several couplings are active for a following axis, this is represented by the sum of the relevant numerical values. Axes: Geometry axis, channel axis, machine axis	RS		R			2

15.2.48 Electronic gear 1

$AA_EG_SYNFA	REAL	$AA_EG_SYNFA[a] a: Following axis Synchronized position of following axis Axes: Geometry axis, channel axis, machine axis	RS		R			5
$P_EG_BC	STRING	$P_EG_BC[a] Block change condition for EGONSYN, EGON, WAITC. Axes: Channel axis	R					5.2
$AA_EG_NUM_LA	INT	$AA_EG_NUM_LA[a] a: Following axis Number of leading axes specified with EGDEF Axes: Geometry axis, channel axis	RS		R			5
$VA_EG_SYNCDIFF	REAL	$VA_EG_SYNCDIFF[a] a: Following axis Synchronization difference Axes: Geometry axis, channel axis, machine axis	RS		R			5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_EG_AX	AXIS	$AA_EG_AX[n,a] n: Index for leading axis a: Following axis Identifier for nth leading axis n: Index for leading axis (nth leading axis) Axes: Geometry axis, channel axis, machine axis	RS		R			5.2

15.2.49 Leading value coupling

$AA_LEAD_SP	REAL	$AA_LEAD_SP[LW] Simulated master value – position	RS	W S	R	W		4
$AA_LEAD_SV	REAL	$AA_LEAD_SV[LW] Simulated master value – velocity	RS	W S	R	W		4
$AA_LEAD_P_TURN	REAL	$AA_LEAD_P_TURN[LW] current leading value position parts lost through modulo reduction. The actual leading value position (used for internal control computation) is $AA_LEAD_P[LW] + $AA_LEAD_P_TURN[LW] If MV is a modulo axis, $AA_LEAD_P_TURN is an integral multiple of $MA_MODULO_RANGE. If MV is not a modulo axis, $AA_LEAD_P_TURN is always 0. Example_1: $MA_MODULO_RANGE[LW]=360 $AA_LEAD_P[LW] =290 $AA_LEAD_P_TURN[LW] =720 The actual leading value position (used for internal control computation) is 1010. Example_2: $MA_MODULO_RANGE[LW]=360 $AA_LEAD_P[LW] =290 $AA_LEAD_P_TURN[LW] =–360 The actual master value position (used internally by the control in calculations) is –70.	RS		R			4
$AA_LEAD_P	REAL	$AA_LEAD_P[LW] Current master value – position (modulo-reduced) If MV is a modulo axis, the following always applies: 0 <= $AA_LEAD_P[LW] <= $MA_MODULO_RANGE[LW]	RS		R			4
$AA_LEAD_V	REAL	$AA_LEAD_V[LW] Current master value – velocity	RS		R			4
$AA_SYNC	INT	$AA_SYNC [FA] Coupling status of following axis in master value coupling 0 => No synchronism 1 => Synchronization coarse (myVdiOut->getSynchCoarse() == TRUE) 2 => Synchronization fine (myVdiOut->getSynchFine() == TRUE) 3 => Coarse and fine Axes: Geometry axis, channel axis, machine axis	RS		R			4

15.2.50 Synchronized spindle

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_COUP_OFFS	REAL	$AA_COUP_OFFS[S2] Position offset for synchronized spindle as setpoint S2 is following spindle	RS		R			2
$VA_COUP_OFFS	REAL	$VA_COUP_OFFS[SPI(2)] Position offset for synchronized spindle as actual value SPI(2) is following spindle	RS		R			2

15.2.51 Safety Integrated 1

$VA_IS	REAL	$VA_IS[X] Safe actual position (SISITEC) Axes: Geometry axis, channel axis, machine axis	RS		R			3
$AA_SCTRACE	BOOL	$AA_SCTRACE[X] = 1 Write: Initiate IPO trigger for servo trace 0: No action !0: Initiate trigger Read: always 0, since the selftriggering bit is returned from the interface. 0: Current value (no status) Axes: Geometry axis, channel axis, machine axis	RS	W S	R	W		4
$AA_SCTRACE	BOOL	$AA_SCTRACE[X] = 1 Write: Initiate IPO trigger for servo trace 0: No action !0: Initiate trigger Read: always 0, since the selftriggering bit is returned from the interface. 0: Current value (no status) Axes: Geometry axis, channel axis, machine axis	RS	W S	R	W		4
$VA_DPE	BOOL	$VA_DPE[X1] Status of power enable of a machine axis Axes: Machine axis	RS		R			5
$AA_ACC	REAL	$AA_ACC Current acceleration value of axis with 1-axis interpolation. $AA_ACC = $MA_MAX_AX_ACCEL * progr. acceleration offset	RS		R			5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_MOTEND	INT	$AA_MOTEND Current motion end criterion at 1-axis interpolation 1 = Motion end at exact stop FINE 2 = Motion end at exact stop COARSE 3 = End of motion with exact stop, IPO stop Axes: Geometry axis, channel axis, machine axis	RS		R			5
$AA_SCPAR	INT	$AA_SCPAR Read current servo parameter set Axes: Geometry axis, channel axis, machine axis	RS		R			5

15.2.52 Extended stop and retract

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_ESR_STAT	INT	$AA_ESR_STAT[X] Status of "Extended stop and retract", bit-coded: BIT0: Generator operation triggered BIT1: Retraction triggered BIT2: Ext. stop triggered BIT3: DC link undervoltage BIT4: Generator minimum speed Axes: Geometry axis, channel axis, machine axis	RS		R			5
$AA_ESR_ENABLE	BOOL	$AA_ESR_ENABLE[X] = 1 Enable "Extended stop and retract" Axes: Geometry axis, channel axis, machine axis	RS	W S	R	W		5

15.2.53 Axis container

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AN_AXCTSWA	BOOL	EVERY $AN_AXCTSWA[n] == TRUE DO M99 Read: TRUE: An axis container rotation is currently being executed on the axis container with axis container name n. FALSE: No active axis container rotation is active			R			5
$AN_AXCTAS	INT	Read: Axis container rotation current rotation: The number of slot rotated for the current axis container is indicated for the axis container with the axis container name n. The value range is from 0 to the maximum number of assigned slots in axis container –1			R			5
$AC_AXCTSWA	BOOL	IF $AC_AXCTSWA[n] == TRUE GOTOB MARK1 Read: TRUE: The channel has enabled axis container rotation for the axis container name n and the rotation has not yet been completed. FALSE: The axis container rotation is terminated.			R			5

15.2.54 Electronic gear 2

$AA_EG_TYPE	INT	$AA_EG_TYPE[a,b] a: Following axis b: Leading axis Type of coupling for leading axis b 0: Actual-value coupling 1: Setpoint coupling Axes: Geometry axis, channel axis, machine axis	RS		R			5.2
$AA_EG_NUMERA	REAL	$AA_EG_NUMERA[a,b] a: Following axis b: Leading axis Numerator of coupling factor for leading axis b Axes: Geometry axis, channel axis, machine axis	RS		R			5.2
$AA_EG_DENOM	REAL	$AA_EG_DENOM[a,b] a: Following axis b: Leading axis Denominator of coupling factor for leading axis b Axes: Geometry axis, channel axis, machine axis	RS		R			5.2

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$AA_EG_SYN	REAL	$AA_EG_SYN[a,b] a: Following axis b: Leading axis Synchronized position of leading axis b Axes: Geometry axis, channel axis, machine axis	RS		R			5.2
$AA_EG_ACTIVE	BOOL	$AA_EG_ACTIVE[a,b] a: Following axis b: Leading axis Coupling for leading axis b is active, i.e. switched on Axes: Geometry axis, channel axis, machine axis	RS		R			5.2

15.2.55 Safety Integrated 2

$A_INSE	BOOL	$A_INSE[n] Image of a safety input signal (ext. NCI interface) n: Number of input 1 – ...	RS		R			4
$A_INSED	INT	$A_INSED[n] Image of a safety input signal (ext NCI interface) n: Number of input word 1 – ...	RS		R			5
$A_INSEP	BOOL	$A_INSEP[n] Image of a safety input signal (ext. PLC interface) n: Number of input 1 – ...	RS		R			4
$A_INSEPD	INT	$A_INSEPD[n] Image of a safety input signal (ext PLC interface) n: Number of input word 0 – ...	RS		R			5
$A_OUTSE	BOOL	$A_OUTSE[n] Image of a safety input signal (ext NCI interface) n: Number of output 1 – ...	RS	W S	R	W		4
$A_OUTSED	INT	$A_OUTSED[n] Image of a safety input signal (ext NCI interface) n: Number of output word 1 – ...	RS	W S	R	W		5
$A_OUTSEP	BOOL	$A_OUTSEP[n] Image of a safety input signal (ext PLC interface) n: Number of output 1 – ...	RS		R			4
$A_OUTSEPD	INT	$A_OUTSEPD[n] Image of a safety input signal (ext PLC interface) n: Number of output word 0 – ...	RS		R			5
$A_INSI	BOOL	$A_INSI[n] Image of a safety input signal (int. NCI interface) n: Number of input 1 – ...	RS		R			4
$A_INSID	INT	$A_INSID[n] Image of the safety input signals (int. NCI interface) n: Number of input word 1 – ...	RS		R			5

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$A_INSIP	BOOL	$A_INSIP[n] Image of a Safety input signal (int. PLC interface) n: Number of input word 1 – ...	RS		R			4
$A_INSIPD	INT	$A_INSIPD[n] Image of Safety input signals (int. PLC interface) n: Number of input word 1 – ...	RS		R			5
$A_OUTSI	BOOL	$A_OUTSI[n] Image of a Safety output signal (int. NCI interface) n: Number of output 1 – ...	RS	W S	R	W		4
$A_OUTSID	INT	$A_OUTSID[n] Image of Safety output signals (int. NCI interface) n: Number of output word 1 – ...	RS	W S	R	W		5
$A_OUTSIP	BOOL	$A_OUTSIP[n] Image of a Safety output signal (int. PLC interface) n: Number of output 1 – ...	RS		R			4
$A_OUTSIPD	INT	$A_OUTSIPD[n] Image of Safety output signals (int. PLC interface) n: Number of output word 1 – ...	RS		R			5
$A_MARKERSI	BOOL	$A_MARKERSI[n] Markers for Safety programming n: Number of marker 1 – ...	RS	W S	R	W	+	4
$A_MARKERSID	INT	$A_MARKERSID[n] Marker word (32 bits) for Safety programming n: Number of marker word 1 – ...	RS	W S	R	W	+	5 . 1
$A_MARKERSIP	BOOL	$A_MARKERSIP[n] Image of PLC Safety markers n: Number of marker 1 – ...	RS		R		+	4
$A_MARKERSIP D	INT	$A_MARKERSIPD[n] Image of PLC Safety marker words n: Number of the flag word 1 – ...	RS		R		+	5 . 1
$A_TIMERSI	REAL	$A_TIMERSI[n] Safety timer – unit in seconds Time is counted internally in multiples of interpolation cycle cycle; Counting for the timer variable is started by assigning the value $A_TIMERSI[n]=<starting value> To stop the counter variable, assign a negative value: $A_TIMERSI[n]=–1 The current time can be read while the counter is active or stopped. When the timer is stopped by assigning the value –1, the most up-to-date timer value is retained and can be read. n: Number of timer 1 – ...	RS	W S	R	W	+	4

Identifier	Type	Description: System variable/value range/index	Part pro		Sync		O	S
$A_STATSID	INT	$A_STATSID Safety: Status of cross-checking between NCK and PLC if value is unequal to zero, an error has occurred in cross-checking	RS		R			5
$A_CMDSI	BOOL	$A_CMDSI[n] Safety: Control word for cross-checking between NCK and PLC. Array index n = 1: Increase timer for signal change monitoring to 10 s n: Number of control signal for cross-checking NCK – PLC	RS	W S	R	W	+	5
$A_LEVELSID	INT	$A_LEVELSID Safety: Display of signal change monitoring level. Indicates the current number of signals marked for data cross-checking.	RS		R			5

Appendix

A Index

$

A

B

C

D

E

F

Q

R

S

T

U

V

W

Z

B Commands, Identifiers

C

D

E

U

V

W

X

To SIEMENS AG A&D MC IS P.O. Box 3180 D-91050 Erlangen Germany (Tel. 0180/525–8008/5009 [Hotline]) Fax +49(0)9131/98–1145 email: motioncontrol.docu@.siemens.de)	**Suggestions** **Corrections** for Publication/Manual: SINUMERIK 840D/840Di/810D/FM-NC Programming Guide Advanced User Documentation
From Name Company/Department Address: Telephone: / Telefax: /	Order No.: 6FC5298-5AB10-0BP2 Edition: 04.00 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

Siemens AG
Automation Group
Automation Systems
for Machine Tools, Robots
and Special-Purpose Machines
P: O. Box 3180, D-91050 Erlangen
Federal Republic of Germany

Siemens quality for software and training to
DIN ISO 9000, Reg. No. 2160-01.
This edition was printed on bleached paper using an
environmentally-friendly chlorine-free method.

Progress
in Automation.
Siemens

Siemens Aktiengesellschaft

Order No.: 6FC5298-5AB10-0BP2
Printed in the Federal Republic of Germany