OPEN*control*

Programming manual

DOCUMENT NUMBER: 45004607K
RELEASE REF.: **V 3.1**
EDITION: 04
AUTHOR: **Prima Electro S.p.A.**
DATE: **February 2013**

Edition	Date	Notes	Author
04	22/02/2013		Prima Electro S.p.A.

Contents

1. PREFACE

1.1 Introduction

This manual describes the procedures used for writing part programs with the OPEN*control* system. It provides programmers with all the information they need for creating machine control programs.

1.1.1 *References*

For additional information refer to:

- OPEN*control* Software characterization manual
- OPEN*control* User manual

The chapters in this manual are organised in sections. They describe the language elements (commands and functions) used for managing a specific task, e.g. axis programming, tool programming, probe management. Programming examples have been introduced in the command description.

1.1.2 *Summary*

1. Preface
This chapter contains the general introduction to the manual

2. Programming with OPEN*control* Systems
This chapter contains the general programming rules of the International Standards Organization (ISO) standard and the general programming rules for XML language. The chapter also provides an overview of the programming environment and a summary of the most used codes.

3. Programming the Axes
This chapter describes axis programming. The G codes and extended commands involved in this activity are provided with their characteristics. Several examples complete the command description and give suggestions for programming the major types of movements.

4. Programming Tools and tool offsets
This chapter describes tool programming and provides the functions and instructions used in tool operation.

5. Cutter Diameter Compensation
This chapter describes cutter compensation. T functions and G codes used in tool compensation are provided with characteristics and several examples.

6. Virtualisations and 3D Transformations
This chapter describes 3D transformations implemented by CNC referring to roto-translations of the tern and Tool Centre Point transformations.

7. Programming the Spindle
This chapter describes spindle programming. The G codes and extended commands involved in this activity are provided with their characteristics. Several examples complete the command description and give hints for solving the main cases of spindle programming.

8. Auxiliary M Functions
This chapter describes auxiliary functions and provides a list of M functions with their meaning and characteristics.

9. Parametric Programming
This chapter deals with special programming applications that use local and system variables.

10. Canned Cycles
This chapter provides a description of the canned cycles available with the control. The G codes and extended commands used in this activity are provided with their characteristics. Several examples complete the command description.

11. Paramacro
This chapter describes how paramacros can be used in programs.

12. Probing Cycles
This chapter provides a description of the probing cycles available with the control. The G codes and extended commands involved in probe management are provided complete with examples.

13. Managing the Infromation Exchange
This chapter discusses the commands used to handle the system screen from a part programs. Examples are given to complete the command description.

14. Modifying the Program Execution Sequence
This chapter contains the commands used for modifying the sequence of execution of a part program. It describes commands for branching, repeating blocks and executing subprograms, as well as commands for putting the part program on hold and releasing it.

15. Multiprocess management commands
This chapter shows OPEN*control* systems multi process potentials.

16. Filters
This chapter describes the various types of filters that can be configured and hence applied in OPEN*control* control units, designed to improve machine tool performances from the geometric and dynamic points of view and hence the finishing quality of the parts produced.

17. Technological variables
This chapter describes the different type of technological and I/O variables available on OPEN*control* systems. It describes how to access these variables, tool.

Appendices

18. Appendix A: Characters and Commands List
Appendix A provides a summary of all the characters allowed in the system and gives lists of G codes, mathematical functions and extended commands.

19. Appendix B: Error Messages
Appendix B provides a list of all the error messages that can occur during programming..

20. Appendix C: Error management from Part Program

1.1.3 *Commands*

Commands are dealt with in the chapters that describe the specific task. A common structure has been adopted in the command description. For each command, the following information is provided:

- Command name
- Command function
- Command syntax
- Parameters
- Characteristics and notes
- Examples

Where possible, examples consist of a portion of program and a diagram that shows how the commands in that portion work.

1.1.4 *Syntax conventions*

Use the following conventions with the commands:

SYMBOL	MEANING
[ ]	Brackets enclose optional entries. Do not enter the brackets.
{ }	Braces enclose entries which may be repeated more than once. This could also be described as a series of alternative entries, i.e. only one of these may be entered. Alternative entries are separated by a (|). Do not enter the braces in the command itself.
|	A vertical bar separates alternative entries. Do not enter the bar.

Key-words are written in **bold**. They must be entered exactly as they are represented in the syntax description.
Parameters that must be passed with commands are indicated by a mnemonic written in italics. Appropriate values must be entered in place of the mnemonic. Leading zeros can be omitted. For example, you can program G00 as G, G01 as G1.

Example:
(SCF,[*value*]**)**
SCF, the comma and parenthesis are key-words and must be written as described. *value* is a parameter name and must be replaced by an appropriate value. The brackets indicate that *value* is an optional value.

1.1.5 *Terminology*

Some terms appearing throughout the manual are explained below.

Control Refers to the OPEN numerical control unit comprising front panel unit and basic unit.

Front Panel Is the interface module between machine and operator; it has a monitor on which messages are output and a keyboard to input the data. It is connected to the basic unit.

Basic Unit Is the hardware-software unit handling all the machine functions. It is connected to the front panel and to the machine tool.

1.2 Reading keys and security instructions

ATTENTION! To use the system in a correct manner, follow the instructions provided in this manual and pay special attention to the symbols below.

WARNING! Dangerous Voltage: This symbol is associated with dangerous high voltages that could damage the system, the equipment and the operators.

WARNING! Danger: This symbol is associated with facts and circumstances that could damage the system, the equipment and the operators.

NOTES: This symbol is used for operations that have to be executed with great care in order to ensure their successful completion.

1.2.1 *Updates to present edition*

Chapter 2

pg. 29 G codes summary updated

Chapter 3

pg. 66 Feed value changed in rotary axes
pg. 152 SPL – Defintion of high speed axes (NEW)
pg. 158 SPL3D - Spline kinematic transformation (NEW)
pg. 166 Change of drive and axes parameters (NEW)

Chapter 5

pg. 215 Description updated

Chapter 6

pg. 227 Virtualizations available (NEW)
pg. 234 Description and characteristics updated
pg. 246 Description updated
pg. 256 New parameters added in machine kinematics table

Chapter 9

pg. 278 Desciptions updated
pg. 345 New paragraphs added (13.1.1, 13.1.2, 13.1.3)
pg. 347 Commands to access a file (NEW)

Chapter 15

pg. 394 Description updated

Appendix A

pg. 442 New codes added in the table

2. PROGRAMMING WITH OPEN*control* SYSTEMS

OPEN*control* part programs are written with a specific language defined by the ISO standard or by using XML commands associated to paramacros. This chapter describes the languages elements and discusses programming techniques and rules.

2.1 The program files

OPEN*control* part programs are stored in files which may be identified with a physical address, either through a logic DRIVE or directly by specifying their respective names.

- Physical names have a maximum length of 127 characters, including extension and path, as applicable, and they identify files stored in the system's physical directories.
- The names specified through a logic DRIVE identify the programs saved in the logic directories configured on the machine. The configuration of logic directories or DRIVES takes place at the machine characterisation stage in AMP. Given a file path containing a DRIVE, the system determines automatically the physical address associated with it, which must comply with the length requirement mentioned above.
- From the programs identified simply with a file name, the system works out the relevant physical path by searching for it among the default paths specified in AMP during machine characterisation.
- When using the Windows Editors, remember to give the "enter" command on the last line entered in the program. Quitting the program without this command gives a specific error message during program execution.

Program activation/deactivation may be executed by specifying the name of the program, either through a physical address or a DRIVE, or directly, by specifying its name, while the names of subroutines and those of all the ISO instructions for which a file has to be specified must be supplied directly or through a logic address by means of a DRIVE.

2.2 ISO Program Components

Address

An address is a letter that identifies the type of instruction. For example, these are addresses:

G, X, Y, F

Word

A word is an address followed by a numerical value. For example, these are words:

G1 X50.5 Z-3.15 F200 T1.1

When you assign a numeric value to a word, no zeroes must precede or follow the value. Insert decimal values after the decimal point.

Blocks

A program block comprises a set of words that identify an operation or a series of operations to be performed. When several fields are used in the same block, they must appear in the order shown in the following table:

block delete	label	sequence number	synchronisation asynchronisation	words codes
/	LABEL	NUMBER	# or &	ALL ALLOWED CHARACTERS

The maximum length of a block is 128 characters.

A technological program is a sequence of blocks that describe a machining operation.

Each block must end with: <CR> <LF>.

Comment blocks

It can be inserted in any position within the current block.

The following strings identify the comment start:

- **;**
- **//**
- **(***

Any character after ";" is considered as a comment.

Block delete

The block delete field is optional. It allows the operator to choose whether to execute program blocks that begin with the "/" character that are called slashed blocks.

Example:

/N100 G00 X100

The block shown in the example can be enabled or disabled using the user interface or by typing the three-letter code DSB on the keyboard.

Label

The label field is optional. It allows the programmer to assign a symbolic name to a block. A label can have up to six alphanumeric characters which must be between quotes. In case of a slashed block, the label must be inserted after the slash.

Example:

"START"

/"END"

When a label field is used in a 'GTO' command, the label defines the block that the control should jump to.

Sequence number

The "sequence number" field is optional. It allows the programmer to number each program block. A sequence number begins with the letter N and is followed by an integer number. Sequence number belongs to the following range: 0→1999999999.

The sequence number must appear in front of the first operand and after the label.

Example:

N125 X0

"START" N125 X0

"END" N125 X0

Synchronisation/asynchronisation

Characters # and & are used to override the default synchronisation/asynchronisation status. For further information on synchronisation, see "Synchronisation and Program Execution".

This field must appear in front of the first operand and after block delete, label and sequence number.

Example:

#(GTO,START, @PL1=1)

2.3 Block Types

Four types of blocks can be used in a part program:

- Comment blocks
- Motion blocks
- Assignment blocks
- Three-letter command blocks

2.3.1 *Comment blocks*

A comment block allows the programmer to insert free text in the program. This text may describe the function to be executed or provide other pieces of information that make the program more understandable and documented.
A comment block does not produce messages for the operator. The control ignores a comment block during execution of the program.

The first character of a comment block must be one of these three characters (;) or '//' or '(*'. The rest of the comment block is a sequence of alphanumeric characters. For example:
;THIS IS AN EXAMPLE OF COMMENT BLOCK
A comment can be inserted not only in a dedicated block, but also in other types of blocks after the character aforementioned. All characters after a ; are considered as a comment.

Example:
G1 X100 Y50 ; Motion block
E1=10 //; Local variable E
(ROT,45) (*;Rotation command

2.3.2 *Motion blocks*

Motion blocks conform to ISO and ASCII standards for programming blocks. There is no particular order for programming the components of a motion block.

Example:
G1 X500 Y20 F200

2.3.3 *Assignment blocks*

Assignment blocks are used to write variables' values directly from the program. Several types of assignments are possible as shown in the following table:

TYPE OF ASSIGNMENT	EXAMPLE
Simple assignment	E10=123.567
Multiple assignment	E1=10, 15.5, 123.467
	In multiple assignments values are loaded as follows:
	10 to E1 15.5 to E2 123.467 to E3
Math expression assignment	E20=(E10+125*SQR(E23))
System number	SN=1.5

2.3.4 *Three-letter command blocks*

Three-letter command blocks define an operation with a three-letter instruction in conformity with the RS-447 standard.

Example:

(ROT,45)
(DIS,"message text")

2.4 Programmable Functions

Programmed values for the functions can be preceded the symbol =.

Example:
The following blocks are equivalent

G1X100	G1X=100
G2X10Y10R200	G2X=10Y10R=200

2.4.1 *Axis coordinates*

Axis coordinates can be named with letters ABCUVWXYZPQD (according to the configuration set in AMP) and can be programmed in the following ranges:

-99999.99999	-0.00001	mm/inch
+0.00001	+99999.99999	mm/inch

NOTE:
It is impossible to program coordinates in the ±0.00001 range because 0.00001 is the minimum value accepted by the control.

2.4.2 *R coordinate*

In a circular interpolation (G02-G03- G12-G13) R represents the radius of the circle.
In a threading block (G33), the R coordinate represents the offset from the zero angular position of the spindle for multi-start threads.
In a standard canned cycle (G81-G89), the R coordinate defines the initial position value and retract value. This function is programmable in the following ranges:

-99999.99999	-0.00001	mm/inch
+0.00001	+9999 9.999 99	mm/inch

NOTE:
It is impossible to program values in the ±0.00001 range because 0.00001 is the minimum value accepted by the control.

2.4.3 *I J coordinates – K Function*

In circular interpolation (G02-G03), when bit 0 of ISOCOMP is not set (ISO commands are set as to comply with 10 Series in AMP) I and J values indicate abscissa and ordinate coordinates of the arc centre, no matter which interpolation plane is active.
In circular interpolation (G02-G03), when bit 1 of ISOCOMP is set (ISO commands are set as to comply with 10 Series in AMP) I and J values indicate the centre coordinates referred to the current point. Their use must be coherent with the interpolation plane active at that moment.

In the spatial circular interpolation programmed used the centre (G12-G13), I, J and K values indicate the centre active coordinates with reference to the abscissa, to the ordinate and to the third axis. In the circular interpolation for three points (G14) I, J and K indicate the coordinates of the third point of the circle.
I and J coordinates are also used in the deep hole drilling cycle (G83).
In a threading block (G33), the I address defines the pitch variation for variable pitch threads:

I+ Increasing pitch
I- Decreasing pitch

In the deep drilling cycle (G83) K defines the incremental value to apply in order to reduce to the minimum depth value (J) the starting pitch depth (I). In a threading block (G83) or in a tapping cycle (G84), K defines the thread pitch. In the helical interpolation (G02-G03), K defines the helix pitch.

This function is programmable in the following ranges:

-99999.99999	-0.00001	mm/inch
+0.00001	+99999.99999	mm/inch

NOTE:
It is impossible to program values in the ±0.00001 range because 0.00001 is the minimum value accepted by the control. When the values of the corresponding axis are expressed in diametrical units (according to the configuration set in AMP), the values of the centre coordinates (I and J) are also expressed in diametrical units.

2.4.4 *t function*

The t function defines the time needed to complete the move defined in the block.
The function is programmable in the following range:

+0.00001	+99999.99999	S

When a "t" function is programmed the block federate will be:

$$F = \frac{\text{total distance}}{\text{time}} * 60$$

A "t" value is valid only in the block in which it is programmed.

2.4.5 *F function and dynamic parameters*

The F code corresponds to two different function: to adjust work feed and to set modal or local dynamic parameters. In the first case the F function defines the axes feedrate. This function is programmable in the following range:

+0.00001	+99999.99999	mm/inch

In G94, F function defines the feedrate in millimetres per minute (G71) or inches per minute (G70).
In G93, the F function defines the inverse of the necessary time in minutes to complete the movement:

$$F = \frac{\text{speed}}{\text{total distance}} = 1/t \text{ (minutes)}$$

The F function is mandatory in the blocks when G93 is active and only affects that block.
In G95, F specifies the axes feedrate in millimetres per revolution (G71) or inches per revolution of the spindle (G70). The syntax needed when F code is used to set dynamic attributes is described in Chapter 2 of this manual.

2.4.6 *M function*

The M address can activate various machine operations. The programmable range goes from 0 to 999. See Chapter 6 for further information about these functions.

2.4.7 *S function*

The S function specifies the spindle rotation speed. It is programmable in the following range:

+0.001	999999.999	rpm/fpm

In G97, the S function defines spindle rotation speed expressed in revolutions per minute.
In G96, the S function defines the cutting surface speed expressed in metres per minute (G71) or feet per minute (G70). The above cutting speed remains constant on the surface.
Refer to Chapter 5 for further information about S function programming.

2.4.8 *T function*

The T function defines the tool and tool offset needed for machining. The tool code is programmable over 32 alphanumerical characters, and the offset can be programmed in a range from 0 to 300.
Chapter 4 provides a detailed description of T functions.

M, S and T functions vary according to their characterisation in AMP.
It is possible for the system to execute these functions inside a continuous move (G27-G28).

When planning an application the manufacturer must:

- configure the desired function as "ALLOWED IN CONTINUOUS" in AMP.
- write a machine logic to handle such a function.
- In turn, the programmer must remember that these functions produce different effects depending on how they are programmed:
- in **continuous mode** a function configured as "ALLOWED IN CONTINUOUS" will be executed in the sequence in which it has been programmed. In order not to lock the program the function will be executed in "NO WAIT" mode.
- in **point-to-point mode** a function configured as "ALLOWED IN CONTINUOUS" will be executed in standard mode.

2.4.9 *h functions*

h functions permit to alter an offset during both continuous and point to point moves.
An h function must be programmed by itself in a block. Its value may range from 0 through 300 and may be either an integer or an E variable.

2.4.10 *G functions*

G codes program machining preparatory functions for machining. The following section deal with this codes.

2.5 G Codes

This section shows how to write preparatory G codes in part program blocks. A preparatory G code is identified by the G address followed by one or two digits (G00-G99). At present, only some of the 100 possible G codes are available.

Paramacro subroutines can be called with a three-digit G code. This class of G codes is described in Chapter 10. Three-digit G codes are classified as follows:

G100 - G299	Reserved
G300 - G699	Non modal paramacro range
G700 - G998	Modal paramacro range
G999	Reset modal paramacro

The G code must be programmed after the sequence number (if defined) and before any other operand in the block.

For example:

N100 G01 X0 - operand

It is possible to program several G codes in the same block, provided they are compatible with each other. Table below defines compatibility between G codes. Zero indicates that the G codes are compatible and can be programmed in the same block; 1 means that the G codes are not compatible and cannot be programmed in the same block without generating an error.

Compatible G Codes

G	00	01 10	02 03	12 13 14	33	53 54	60 67	81 89	80	72 73 74	93 94 95	96 97	41 42	40	27 28	29	04	09	90 91	79	70 71	16 17 18 19	92 99 98
G00	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
G01	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
G02	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
G03	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
G04	0	0	0	0	1	1	1	1	0	1	0	0	0	0	1	0	1	1	0	0	0	1	1
G09	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	1
G10	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
G12	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	1
G13	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	1
G14	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	1
G16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G27	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	1	1	0	0	0	0	1	1
G28	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	1	1	0	0	0	0	1	1
G29	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1	1
G33	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	1
G40	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1
G41	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1
G42	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1
G53	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

G54	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1
G60	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
G61	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G62	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G63	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G66	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G67	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G70	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1
G71	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1
G72	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G73	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G74	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G79	0	0	0	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0	1	1	0	1	1
G80	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1
G81	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	0	1	1
G82	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1
G83	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1
G84	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1
G85	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1
G86	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1
G89	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1
G90	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1
G91	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1
G92	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G93	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
G94	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
G95	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
G96	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1
G97	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1
G98	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
G99	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

NOTE:

0 means compatible G codes

1 means incompatible G codes

The following table gives a summary of the G codes available in the control. This default configuration can be modified through the AMP utility.

G codes summary

CODE	GROUP	MODAL	DESCRIPTION	POWER UP
G00	a	YES	Rapid axes positioning	YES
G01	a	YES	Linear interpolation	NO
G10	a	YES	Linear interpolation with rapid dynamic parameters.	NO
G02	a	YES	Circular interpolation CW	NO
G03	a	YES	Circular interpolation CCW	NO
G12	a	YES	3D circular interpolation CW	NO
G13	a	YES	3D circular interpolation CCW	NO
G14	a	YES	3D circular interpolation for 3 points	NO
G33	a	YES	Constant or variable pitch thread	NO
G16	b	YES	Circular interpolation and cutter diameter compensation on a defined plane	NO
G17	b	YES	Circular interpolation and cutter diameter compensation on 1st-2nd axes plane	NO
G18	b	YES	Circular interpolation and cutter diameter compensation on 3rd-1st axes plane	YES
G19	b	YES	Circular interpolation and cutter diameter compensation on 2nd-3rd axes plane	NO
G27	c	YES	Continuous sequence operation with	YES
G28	c	YES	automatic speed reduction on corners	NO
G29	c	YES	Continuous sequence operation without speed reduction on corners	NO
G99	d	NO	Point-to-point operation	NO
G98	d	NO	Axis pre-setting without mirror	NO
G92	d	YES	Axis pre-setting with mirror	YES
G40	e	YES	Delete G92	YES
G41	e	YES	Cutter diameter compensation disable	NO
G42	e	YES	Cutter diameter compensation-tool left	NO
G70	f	YES	Cutter diameter compensation-tool right	NO
G71	f	YES	Programming in millimetres	YES
G80	g	YES	Disable canned cycles	YES
G81	g	YES	Drilling cycle	NO
G82	g	YES	Spot-facing cycle	NO
G83	g	YES	Deep hole drilling cycle	NO
G84	g	YES	Tapping cycle	NO
G85	g	YES	Reaming cycle	NO
G86	g	YES	Boring cycle	NO
G89	g	YES	Boring cycle with dwell	NO
G90	h	YES	Absolute programming	YES
G91	h	YES	Incremental programming	NO
G79	i	NO	Programming referred to axis home switch	NO
G04	j	NO	Dwell at end of block	NO

G09	j	NO	Deceleration at end of block	NO
G72	k	NO	Point probing with probe tip radius compensation	NO
G73	k	NO	Hole probing with probe tip radius compensation	NO
G74	k	NO	Probing for theoretical deviation from a point without probe tip radius compensation	NO
G93	l	YES	Inverse time (V/D) feed rate programming mode	NO
G94	l	YES	Feedrate programming in ipm or mmpm	NO
G95	l	YES	Feedrate programming in ipr or mmpr	YES
G96	m	YES	Constant surface speed (feet per minute or metres per minute)	YES
G97	m	YES	Spindle speed programming in rpm	NO
G53		YES	Absolute roto-translation of a linear axes tern	NO
G54		YES	Roto-translation on a linear axes tern towards the active origin.	NO
G60		YES	Closes spline profile (HSM)	YES
G61		YES	Opens spline profile (HSM)	NO
G62		NO	Divides the spline profile in two continuous units	NO
G63		NO	Divides spline profile in two units with connector	NO
G66		NO	Divides spline profile in two units with edge	NO
G67		NO	Divides spline profile in two units with speed rate reduced on the edge.	NO

2.6 XML programming

OPEN*control* executes XML commands recalling paramacros which names correspond to XML command. Any parameters on the XML command are assigned to H variables defined within the control.

A mapping file can be defined to associate a H variable to each XML parameter. XML parameters are mapped on consecutive H variables and, for H variables, the starting index can be defined.
Only one string parameter, mapped on the HS variable, can be defined for an XML command. Concerning the other parameters indexing, string parameter is transparent and doesn't affect the calculation of other index parameters.
An XML command appearing in the file mapping can't have any parameters in addition to the ones defined by the file. An XML command doesn't have to use all the parameters defined in the file mapping.
If a parameter is used in an XML call, HF flag corresponding to the H variable referred to the parameter, will be set at 1.

If there are no corresponding files or XML command is not in the file, the parameters will be assigned to H variables according to their position within the XML command. If the position of parameters between two calls of a command changes, they will be assigned to different H variables. In this case, string parameters cannot be used.

Examples:
Mapping File:
XML_MOVE 5 double X double Y double FEED
XML_MODE integer ABSOLUTE

Part Program:

XML command	ISO paramacro	H and HF values
<XML_MOVE FEED="1000" X="42" />	XML_MOVE	H5 = 42 H6 = ?? H7 = 1000 HF5 = 1 HF6 = 0 HF7 = 1
<XML_MODE,10 ABSOLUTE="23" />	XML_MODE	H10 = 23 HF10 = 1
<CIRCLE X="15" Y="16" R="8" />	CIRCLE	H0 = 15 H1 = 16 H2 = 8 HF0 = 1 HF1 = 1 HF2 = 1

XML command	ISO paramacro	H and HF values
<CIRCLE Y="16" R="8" X="15" />	CIRCLE	H0 = 16 H1 = 8 H2 = 15 HF0 = 1 HF1 = 1 HF2 = 1
<XML_TOOL X="10" NAME="T" NUM="2" />	XML_TOOL	H0 = 10 H1 = ?? H2 = 2 HS = "T" HF0 = 1 HF1 = 0 HF2 = 1 HFS = 1

2.6.1 *XML parameters equivalence file (Mapping File)*

The mapping file defines the parameters related to an XML command and the corresponding H variables. File name is always **XMLrules.txt**.

Each line of the mapping file defines XML parameters and, in some circumstances, defines also the index of the first H variable corresponding to these parameters. If index is not specified, the first parameter will correspond to the H0 variable.

Each definition can be divided in two parts: the XML section and the parameters section.

2.6.2 *XML section*

The XML section defines the name of the XML command.

XML command name can be defined by a string of alphanumeric characters and the '_' and '-' characters surrounded by the '<' and '>' characters.

The first character of the name of the XML command must be a capital letter.

The following are example of XML command:

<ABSOLUTE_PROGRAMMING> ←valid command name
<Incremental_prog> ← valid command name
<1ABSOLUTE> ← invalid command name, it does not start with a capital letter
<INCREMENTAL@PROG> ←invalid command name, it contains an invalid character

2.6.3 *Parameters definition*

If the command uses parameters, they will be listed after the command name. They use the following syntax:

---

index $type_1$ $param_1$ $type_2$ $param_2$ …… $type_n$ $param_n$

---

where:

index is an integer number indicating the total number of parameters of the XML command

type is a string of characters specifying the type of the parameter. There are three allowed values: *integer*, *double* and *string*.

param is the name used to identify the parameter into the XML file. It is a string of alphanumeric characters and the '-' and '_'.characters. Parameter position indicates to which variable is referred, and the variable index is calculated according to this formula

indexH = *param* position + *index*

Index value doesn't apply in case of string parameter and string parameter **does not** affect the position calculation in the formula above.

An XML command appearing in the file mapping cannot use undefined parameters

2.6.4 *Components of the XML program*

Parameter
A parameter is a string composed of the alphanumeric characters and the '-' and '_' characters. It identifies a parameter of an XML command.

The following are valid XML parameters:
Speed
ID
X
FLOW_ID

XML block
XML blocks are the blocks used to execute an XML command with its parameters.
An XML block can occupy more than one line in the part program.

The following table shows the syntax used by XML blocks:

Block start	XML command	parameter assignment	end of block
<	COMMAND BLOCK	PARAMETER ASSIGNMENT	/>

- ***Block start***

The field start block is compulsory. It is used by interpreter to switch to XML mode.

- ***XML Command***

The field XML command is compulsory and it specifies the XML command that will be executed.
It is a string of alphanumeric characters and the '-' and '_' characters.

The following could be valid XML command:
ABSOLUTE
CIRCLE
CUT_FAST
GET-TOOL

- ***Parameter assignment***

The field parameter assignment is optional and allows values to be assigned to the parameters of an XML command. A block parameter assignment consists of a group of assignments respecting the following syntax:

parameter = "value"

There is a predefined parameter that allows the specification of the line number of an XML block. This parameter is **LINENUMBER**.

Example:
<ABSOLUTE_PROGRAMMING LINENUMBER = "13" X="12.3" NAME = "TEST" \>

2.7 ISO program synchronization and execution

The terms "synchronised" and "asynchronous" apply only to part program blocks not implying a movement, that is, assignment or calculation blocks. A motion block is any block containing axes motion together with other actions:

- Axis moves
- M codes
- S codes
- T codes

A synchronisation block is taken into consideration and executed only after the motion block that precedes it in the program is completed, that is after the axis move has been executed.

On the other hand, a non-synchronised block is executed as soon as it is read by the part program interpreter, i.e. when perhaps the previous move is still in progress.

The advantage of asynchronous block execution is that variable assignments and complex calculations can be made between moves. This allows reduction of waiting time between two motion blocks caused by calculations.

2.7.1 *Default Synchronisation*

At power up, the following commands and codes are automatically synchronised:

- G16, G17, G18, G19, G72, G73, G74, G53, G54, G80
- AXF, AXS, AXT, CPA, CPD, DLY, FLT, GTA, GTS, PAD, PAE, REL, RQO, SCF, SDA, SND, TCP, TTC, UDA, UPR, WAI, WOS, XDA

No other commands are synchronised.

This default assignment can be changed. This means that the commands that are synchronised by default at power-up can become asynchronous and that the commands that are not synchronised by default at power-up can become synchronous. The next section explains how to override default synchronisation.

NOTE:
Default synchronisation cannot be modified for three-letter codes (AXF, AXS, AXT, FLT, GTA, GTS, TCP, TTC, UPR, XDA) and for G53 and G54 codes

2.7.2 *Overriding Default Synchronisation*

Under certain circumstances, the part program may request modification of the default synchronisation.

If the command is synchronised by default and the programmer wants it to be executed by the interpreter as soon as it is read (asynchronous operation), an "&" must be programmed in the first position of the block, immediately after the "n" number.

If the command is asynchronous and you wish to activate synchronous operation, the first character in the block must be #.

Both # and & are active only in the block where they are programmed.

To avoid possible damage to the workpiece, note that programming synchronised blocks between contouring blocks clears the motion buffer at each synchronised block. This will result in dwells while the buffer is reloaded and all the calculations are performed.

2.7.3 *Part Program Interpreter*

When the system reads a part program block it executes various activities, depending on the type of block:

- A motion block will be loaded in the motion buffer queue. If the move is defined by a variable, the stored move values stored are those of the variable. The buffer size is configurable from 2 to 255 blocks through AMP.

- An asynchronous assign or calculation block will be executed.

Three factors cause the part program interpreter to stop reading blocks:

- The motion buffer is full. When the active motion block is completed, the interpreter will read another motion block and load it in the buffer queue.

- A non-motion block that contains a synchronised command or a code that forces synchronisation is read. The interpreter does not start again until the last loaded motion block is completed. At this point the block calling for synchronisation is executed and the interpreter starts reading the following blocks.

- Error conditions

2.7.4 *ISO commands execution sequence*

1. Diameter axes
2. Scale factor (SCF)
3. Measuring units (G70 G71)
4. Paraxial compensation (u v w)
5. Programming mode (G90 G91)
6. Mirror machining (MIR)
7. Plane rotation (ROT)
8. Origins (UAO UTO UIO G92)

2.7.5 *Double numbers programming restrictions*

Double numbers are represented according to the IEEE numerical standard through 32 bit floating-point numbers.
The precision of these numbers is therefore limited to 17 digits in all, with a maximum of 17 digits for whole numbers and 16 digits for decimal numbers.
If more numbers are programmed, the system does not display any error; it expresses the value in the scientific format (number with exponent) truncating the programmed number at the 17th digit, thereby losing precision.

3. PROGRAMMING THE AXES

3.1 Axis motion codes

3.1.1 *Defining Axis Motion*

In this manual axes motion directions are defined in compliance with EIA standard RS-267. By convention, we always assume that the tool moves towards the part, no matter whether the tool moves towards the part or the part moves towards the tool in the actual process.

The control has four levels of definition of dynamic parameters (acceleration, deceleration, jerk, negative jerk ramp time in acceleration and ramp time in deceleration) for the movements: default parameters, on the profile (modal) parameters, local (related to the current block) parameters and axis related parameters.

Default and axis related parameters are set in AMP, on profile and local parameters are set using **F** function. Work feed can be programmed in different ways.

The G code in the following table activates the available ways:

G CODE	FUNCTION
G93	Feed is defined as the inverse time needed to complete the movement
G94	Feed is defined in millimetres (G71) per minute or inches (G70) per minute
G95	Feed is defined in millimetres (G71) or inches (G70) per revolution of the spindle

Basic movements can be defined with the motion G codes listed in the following table:

G CODE	FUNCTION
G00	Rapid axes positioning
G01	Linear interpolation
G10	Linear interpolation with rapid dynamic parameters
G02	Circular interpolation clockwise
G03	Circular interpolation anticlockwise
G12	Circular interpolation clockwise in the space
G13	Circular interpolation anticlockwise in the space
G14	Circular interpolation for three points in the space
G33	Constant or variable pitch threading

3.1.2 *ACCUR – Calculation accuracy*

ACCUR code defines the internal calculation accuracy.

Syntax

ACCUR = *value*

where:

value indicates the calculation accuracy value using the unit of measurement active (mm or inches. It can be programmed by a real number or by an E parameter.

Characteristics:

Accuracy value indicates the lowest numerical value after which all real numbers are =0. Usually, the calculation accuracy has an internal use identifying edges, lengths and distances that can be considered as null.

3.1.3 *ISOCOMP – Compatibilities mask*

ISOCOMP defines a read only variable containing in one byte the compatibility information.

At the moment only bit 0 and bit 1 are used having the following meaning:

- **Bit 0:** if bit is set, G2 and G3 work in compatible ISO mode. If bit is not set, G2 and G3 work as on the OSAI 10/Series controls.
- **Bit 1:** if bit is set, H variables type are local, therefore each execution level has its' own H variables group. If bit is not set, all the paramacro levels have only one group of H variables. Bit not set correspond to OSAI 10 series systems behave.

ODM sets compatibility information in the active AMP.
ISOCOMP bit 0 corresponds to the *ISO commands compatibility* parameter within the ODM configuration process. .
ISOCOMP bit 0 corresponds to the *Local H variables* parameter within the ODM configuration process.

3.1.4 *G93 G94 G95 – Feed programming modes*

The G codes G93, G94 and G95 define three different modes for feed programming.

Syntax:

G93[*G-codes*]
G94[*G-codes*]
G95[*G-codes*]

where:

G-codes Other G codes that are compatible with G93, G94 and G95 (See "Compatible G codes" table in Chapter 2).

Characteristics:

In G93, the F function defines the inverse of the time in minutes necessary to complete the movement:

$$F = \frac{\text{speed}}{\text{total distance}} = 1/t\,(\text{minutes})$$

The work feed is mandatory in the blocks when G93 is active and only affects that block.

In G94, F function defines the feed rate in millimetres per minute (G71) or inches per minute (G70).

In G95, F specifies the axes feed rate in millimetres per revolution (G71) or inches per revolution of the spindle (G70).

3.1.5 *F – Dynamic parameter and work Feed setting*

Depending on the syntax, the F code allows the setting of the value of work feed on the profile only or the setting of the values of the dynamic parameters. The dynamic parameters, that contain the work feed, can also be defined as parameters valid for the whole profile or for the current block only.

Syntax:

F*value*

---

where:

F identify work feed on profile

value is a positive real number that can be programmed directly or indirectly with an E parameter. It specifies the work feed value on profile. Measuring unit depends the characterised measuring unit (G70/G71) and on the programmed feed type (G93/G94/G95).

Syntax:

{**F**|**f**}[**%**]**[[F***valF*][**,A***valA*] [**,D***valD*] [**,J***valJ*] [**,K***valK*] [**,T***valT*] [**,B***valB*]**]**

---

where:

{**F**|**f**} Flag identifying when the dynamic parameters are applied. **F** specifies that the parameters refer to the whole profile, while **f** specifies parameters referring to the current block only.

% optional parameter. If present then values for dynamic parameters are a percentage of the corresponding default parameters defined in the current AMP.

F id for parameter: work feed

A id for parameter: acceleration

D id for parameter: deceleration

J id for parameter: jerk

K id for parameter: negative jerk

T id for parameter: ramp time (acceleration)

B id for parameter: ramp time (deceleration)

valF
valA
valD
valJ
valK
valT
valB are positive real numbers that can be programmed directly or indirectly with E parameters. They specify the value for the corresponding parameter.

Characteristics:

The syntax for assigning the dynamic parameters allows four types of values to be specified:

- Absolute value on the profile (modal values). The parameter will acquire the value specified in the command. The value will be valid until a RESET command or until the next assignment.
- Percentage on the profile (modal values). The parameters specified in the command correspond to a percentage of the default value as defined in AMP. The value will be valid until a RESET command or until the next assignment.
- Absolute value for current block. The parameter will acquire the values specified in the command. This value will be applied only to the block to which the command belongs.
- Percentage for current block. The parameters specified in the command correspond to a percentage of the default value as defined in AMP. This value will be applied only to the block to which the command belongs.

For each dynamic parameter (e.g. acceleration) up to four levels of definition can exist:

1. Local value for the block (identified by the syntax **f[A***value***]**)
2. Value on the profile, or modal value (specified by the syntax **F[A***value***]**)
3. Default value (specified in AMP, in process configuration)
4. Physical value for the axis (specified in AMP, in axis configuration)

The priority between levels corresponds to the order of the levels: local value for the block overrides all other values.

RESET command resets all values to default values as defined in AMP.

3.1.6 *G00 – Rapid axes positioning*

G00 defines a linear movement at rapid feedrate that is simultaneous and coordinated for all the axes programmed in the block.

Syntax:

G00 [G-codes] [axes] [offset] **[F..] [a] [d]** [auxiliary]

where:

G-codes Other G codes that are compatible with G00 (See "Compatible G codes" table in Chapter 2).

axes Axis name followed by a numerical value. The numerical value can be programmed directly with a decimal value or indirectly with an E parameter. Up to twelve axes can be written in a block.

offset Offset factors on the profile. For three axes these factors are entered with u, v, and w respectively. See "Paraxial compensation" in Chapter 4 for further information.

F Feedrate for coordinated moves. It is given with the F address followed by the feedrate value. This parameter does not affect the move of the axes programmed in the G00 block, but is retained for subsequent feedrate moves. The rapid feedrate forced by G00 is a velocity along the vector of the axes programmed in the block. The maximum rapid feedrate is defined during characterisation with ODM utility.

auxiliary Programmable M, S, and T auxiliary functions. Up to four M functions, one S (spindle speed) and one T (tool selection) can be programmed in the block.

3.1.7 *G01 – Linear interpolation*

G01 defines a linear move at machining feedrate that is simultaneous and coordinated on all the axes programmed in the block.

Syntax

G01 [*G-codes*] [*axes*] [*offset*] [**F**..] [*auxiliary*]

---

where:

G-codes Other G codes compatible with G01 (See "Compatible G codes" table in Chapter 2).

axes Axis name followed by a numerical value. The numerical value can be programmed directly with a decimal value or indirectly with an E parameter. Up to twelve axes can be written in a block.

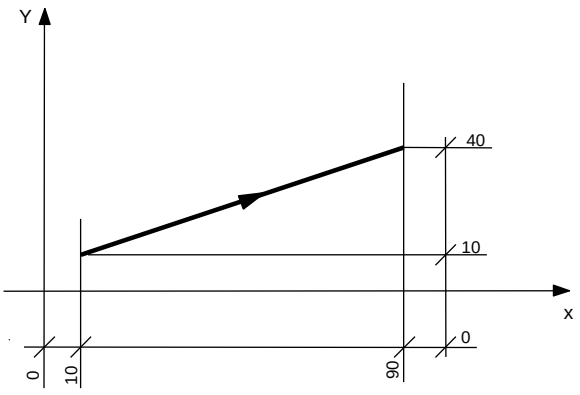
offset Offset factors on the profile. These factors are entered for three axes with the characters u, v, w respectively. See "Paraxial compensation" in Chapter 5 for further information.

F Dynamic parameters used for the move. It is given with the **F** (or **f** in case of parameters referring to the current block only) address followed by the feed rate value.

auxiliary Programmable M, S, T auxiliary functions. Up to four M functions, one S (spindle speed) and one T (tool selection) can be programmed in the block.

Example:

This example shows how to program a G01 code.



Program:

N70 G0 X10 Y10
N80 G01 X90 Y40 F200

3.1.8 *G10 – Linear interpolation with rapid dynamic parameters*

G10 defines, on all axes programmed within the block, a linear, concurrent and coordinated motion using rapid dynamic parameters (feed, acceleration, jerk…) for the calculation.

Syntax

G10 [*G-codes*] [*axes*] [*offset*] [**F**|**f**..] [*auxiliary]*

where:

G-codes G codes compatible with G10 (see table "Compatible G codes" Chapter 2)

axes Character corresponding to an axis name followed by a numerical value. The numerical value can be programmed using a decimal value or an E parameter. Up to 12 axes can be defined.

offset Offset factors on profile. These factors are introduced for three axes namely with u, v, w characters. Additional information can be found on the "Paraxial compensation" section in Chapter 5.

F

f Dynamic parameters coordinate movements: defines in the **F** address complying with their syntax. This parameter doesn't affect the motion of the axes programmed with G10 active, defining instead the dynamic parameters for following motions at working feed rate. Rapid dynamic parameters forced by G10 represent the vector composition of the axes rapid dynamic parameters programmed in the bock. Maximum values for each axis are defined in AMP during the ODM characterization.

auxiliary Programmable auxiliary functions M, S, T. Up to four M functions can be programmed in the block, one S (spindle rotation speed) and one T (tool selection)

3.1.9 *G02 G03 – Circular interpolation*

These codes define the following circular movements:

G02 Circular interpolation clockwise (CW)
G03 Circular interpolation counter-clockwise (CCW)

The circular move is performed at machining feed rate and is coordinated and simultaneous with all the axes programmed in the block.

Syntax

G02 [G-codes] [axes] **I**.. **J**.. **K.**. [**F**|**f**..] [auxiliary]
or
G02 [G-codes] [axes] **R**.. [[**F**|**f**..] [auxiliary]

G03 [G-codes] [axes] **I**.. **J**.. **K.**. [**F**|**f**..] [auxiliary]
or
G03 [G-codes] [axes] **R**.. [**F**|**f**..] [auxiliary]

---

where:

G-codes Other G codes that are compatible with G02 and G03 (See "Compatible G codes" table in Chapter 2).

axes Axis name followed by a numerical value programmed directly with a decimal value or indirectly with an E parameter.
If no axes are programmed in the block, the move is a complete circle in the active interpolation plane.

I Parameter meaning changes according to the ISO compatibility variable (ISOCOMP):

- OSAI 10 Series compatibility: parameter corresponds to the abscissa of the circle centre. Symbol used is always **I** no matter which interpolation plane is active.
- Standard ISO compatibility: parameter corresponds to the coordinate of the circle centre towards the first axis of the process. Value corresponds to the incremental shifting from the current point. It can be omitted if the first axis does not belong to the interpolation plan.

This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. The abscissa is expressed as a diameter unit when the corresponding axis is a diameter axis.

J Parameter meaning changes according to the ISO compatibility variable (ISOCOMP):

- OSAI 10 Series compatibility: parameter corresponds to the ordinate of the circle centre. Symbol used is always **J** no matter which interpolation plane is active.
- Standard ISO compatibility: parameter corresponds to the coordinate of the circle centre towards the second axis of the process. Value corresponds to the incremental shifting from the current point. It can be omitted if the first axis does not belong to the interpolation plan.

This is a value in millimetres or inches that can be programmed directly or indirectly with

an E parameter. The ordinate is expressed as a diameter unit when the corresponding axis is a diameter axis.

K Parameter is used only if the ISO variable (ISOCOMP) is set as ISO standard compatible. It corresponds to the coordinate of the circle centre towards the third axis of the process and can be omitted if the third axis does not belong to the interpolation plane. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. The ordinate is expressed as a diameter unit when the corresponding axis is a diameter axis.

R Circle radius alternative to the I and J coordinates. If the arc of a circle is less than or equal to 180 degrees, the radius must be programmed with positive sign; if the arc of a circle is greater than 180 degrees the radius must be programmed with negative sign.

NOTE: **R** is not allowed with arc of 360 degrees.

F

f Feed rate used for the move. It is given with the **F** address (or **f** in case of parameters referring to the current block only)followed by the feed rate value.

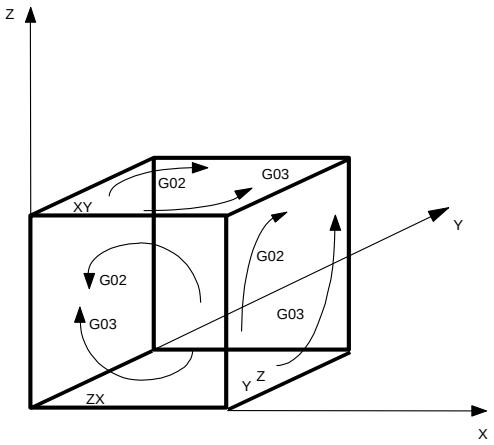
auxiliary Programmable auxiliary functions M, S, T. Up to four M functions, one S (spindle speed rotation) and one T (tool selection) can be programmed in the block.

Characteristics:
The maximum programmable arc is 360 degrees, i.e. a full circle. Before programming a circular interpolation block, the interpolation plane must be defined with G16, G17, G18, G19. The interpolation plane after reboot/reset can be configured in AMP using ODM.

The coordinates of the start point (determined from the previous block), the end point and the centre of the circle must be calculated so that the difference between start and end radius is less than the default value (0.01 mm or 0.00039 inches). If this difference is equal or greater than the default value, the control displays an error message and the circular move is not performed.
Incremental programming (G91) can be used in conjunction with circular interpolation. With G91 the end point and the centre point of the circular move are referenced to the start point programmed in the previous block.

The direction (CW or CCW) of a circular interpolation is defined by looking in the positive direction of the axis that is perpendicular to the active interpolation plane.

The following examples show the directions for circular interpolation on the active planes.



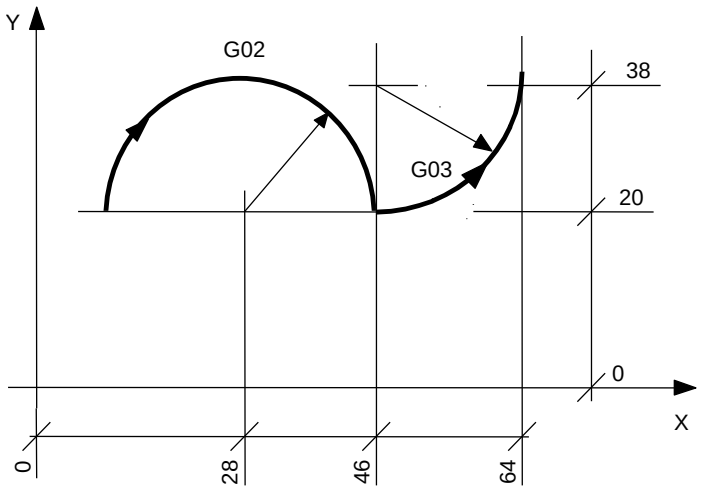
Directions of a circular interpolation

Circular interpolation in absolute programming with the I and J coordinates of the centre of the circle.

N14 X10 Y20

N15 G2 X46 Y20 I28 J20 F200

N16 G3 X64 Y38 I46 J38



Circular interpolation in absolute programming with the value R of the radius of the circle.

N14 X10 Y20

N15 G2 X46 Y20 R18 F200

N16 G3 X64 Y38 R18

Circular interpolation in incremental programming with the coordinates J and I.

N14 X10 Y20

N15 G2 G91 X36 I18 J0 F200

N16 G3 X18 Y18 I0 J18

Circular interpolation in incremental programming with the value of the radius R.

N14 X10 Y20

N15 G2 G91 X36 R18 F200

N16 G3 X18 Y18 R18

3.1.10 *G12 G13 – Circular interpolation clockwise and anticlockwise in space*

Circular movements in space with a definite direction of rotation are defined by the following codes:

G12 Circular interpolation clockwise in space
G13 Circular interpolation anticlockwise in space

Circular movement is executed with a programmed working velocity and is co-ordinated and concurrent on all axes programmed in the block.

Syntax

G12 [G-codes] [axes] **I.. J.. K..** [**x**.. **y**.. **z**..] [**F**|**f**..] [*auxiliary*]
G13 [G-codes] [axes] **I.. J.. K..** [**x**.. **y**.. **z**..] [**F**|**f**..] [*auxiliary*]

---

where:

G-codes Other G-codes that are compatible with G12 and G13 (see “Compatible G codes” table in Chapter 2)

axes Axis name followed by a numerical value. The numerical value can be programmed directly with a decimal value or indirectly with an E parameter. If there are no programmed axes within the block, the following movement is a complete circle in the active interpolation plan.

I Abscissa of the circle centre. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. The abscissa is expressed as a diameter unit when the corresponding axis is a diameter axis. No matter what interpolation plane you are using, the symbol for the abscissa is always **I**.

J Ordinate of the circle centre. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. No matter what interpolation plane you are using, the symbol for the ordinate is always **J**.

K Third axis coordinate of the circle centre. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. The ordinate is expressed as a diameter unit when the corresponding axis is a diameter axis. No matter what interpolation plan you are using, the symbol for the third axis coordinate is always **K**.

x y z Identify three components of a versor orthogonal to the plane to which the programmed circle movement belongs. Only if the movement is a circumference or a semi-circle these parameters are required.

F
f Feed rate used for the move. It is given with the F address followed by the feed rate value. If omitted, the system will use the programmed value. If no feed rate has been programmed an error will occur.

auxiliary Programmable auxiliary functions M, S, T. Up to four M functions, one S (spindle speed) and one T (tool selection) can be programmed in the block.

Characteristics:
The maximum programmable arc is 360 degrees, i.e. a full circle.

Before programming a circular interpolation block, the interpolation plane must be defined with G16, G17, G18, G19. The interpolation plane after reboot/reset can be configured in AMP using ODM.

The coordinates of the start point (determined from the previous block), the end point and the centre of the circle must be calculated so that the difference between start and end radius is less than the default value (0.01 mm or 0.00039 inches). If this difference is equal or greater than the default value, the control displays an error message and the circular move is not performed.

When the final point coordinates and the coordinates of the circumference centre refer to those of the starting point in the previous block, circular interpolation can be programmed incrementally (G91).

The programmed arc of the plane matches the plane obtained as following:

- rotating the Cartesian plane as to superimpose the abscissa axis on the segment connecting the centre of the arc with the interpolation starting point
- fixing the final point as part of that plane

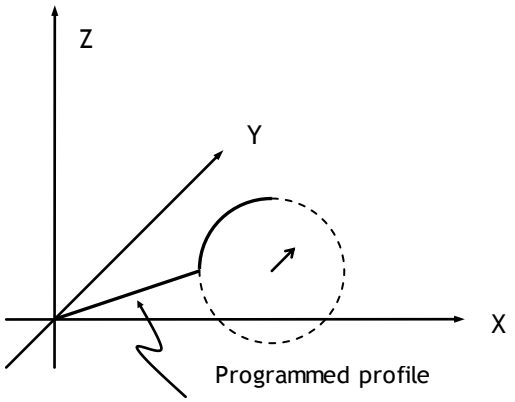
As in case of complete or half circles the three points of the circumference arc (namely starting point, final point and centre) are aligned, the components of a versor orthogonal to the interpolation plane must be defined using **x, y, z** addresses.

The direction (CW or CCW) of a circular interpolation is defined by looking in the positive direction of the axis that is perpendicular to the active interpolation plane. The following examples show the directions for circular interpolation on the active planes.

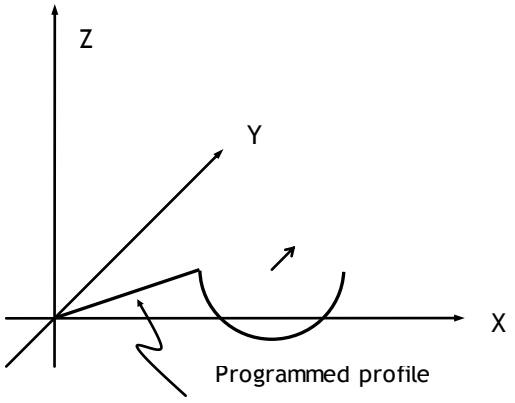
If the tangential control (corresponding to three-letter code AXT) is active, the circular movement in the space will be considered as a rectilinear movement in the plane pre-set for the control.
If there are any protected areas active, circular movement in the space is approximated by two rectilinear movements in the protected area plane. Movements will correspond to the movement between the starting point and the point in the semi-break of the arc and the movement between the point in the semi-break of the arc and the circle movement final point.
Cutter diameter compensation (G41-G42) is **incompatible** with circular moves in space.

Examples:

G17
G1X10Y5Z0 F[F2000, A3000,D3000]
G13X15Y10Z5I15J10K0



G17
G1X10Y5Z0 F[F2000, A3000,D3000]
G12X20I15J10K0y1



3.1.11 *G14 – Circular interpolation for three points in space*

This code defines a circular movement in space identifying a third point in addition to the final one. As these three points are not aligned, code G doesn't perform a complete circle.

Syntax

G14 [*G-codes*] [*axes*] **I.. J.. K..** [**F**|**f..**] [*auxiliaries*]

where:

G-codes Other G-codes compatible with G14 (see "Compatible G codes" table in Chapter 2)

axes Axis name followed by a numerical value. The numerical value can be programmed directly with a decimal value or indirectly with an E parameter. If there are no programmed axes within the block, the following movement is a complete circle in the active interpolation plan.

I Abscissa of the third point of the circle. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. The abscissa is expressed as a diameter unit when the corresponding axis is a diameter axis. No matter what interpolation plane you are using, the symbol for the abscissa is always **I**.

J Ordinate of the third point of the circle. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. When the corresponding axis is a diameter, ordinate has a diametrical value. No matter what interpolation plane you are using, the symbol of the ordinate is always **J.**

K Third axis coordinate of the circle centre. This is a value in millimetres or inches that can be programmed directly or indirectly with an E parameter. The ordinate is expressed as a diameter unit when the corresponding axis is a diameter axis. No matter what interpolation plan you are using, the symbol for the third axis coordinate is always **K**.

F
f Feed rate used for the move. It is given with the F address followed by the feed rate value. If omitted, the system will use the programmed value. If no feed rate has been programmed an error will occur.

auxiliary Programmable auxiliary functions M, S, T. Up to four M functions, one S (spindle speed) and one T (tool selection) can be programmed in the block.

Characteristics:

The Cartesian plane is defined using the G16, G17, G18, G19 codes. The starting Cartesian tern is defined by AMP using ODM.

The third point used to define the circumference, must be between the start and the final point of the circle. Its position defines the direction of the circumference as well.

The coordinates of the start point (determined from the previous block), the end point and the third point of the circle must be calculated so that the difference between start and end radius is less than the default value (0.01 mm or 0.00039 inches). If this difference is equal or greater than the default value, the control displays an error message and the circular move is not performed.

When the final point coordinates and the coordinates of the circumference centre refer to those of the starting point in the previous block, circular interpolation can be programmed incrementally (G91).

The direction (CW or CCW) of a circular interpolation is defined by looking in the positive direction of the axis that is normal to it (see example).

If the tangential control (corresponding to triliteral AXT) is active, the circular movement in the space will be considered as a rectilinear movement in the plane pre-set for the control.

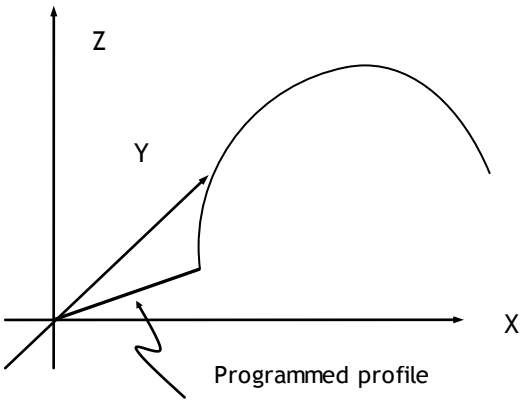
If there are any protected areas active, circular movement in the space is approximated by two rectilinear movements in the protected area plane.

Movements will correspond to the movement between the starting point and the point in the semi-break of the arc and the movement between the point in the semi-break of the arc and the circle movement final point.

Cutter diameter compensation (G41-G42) is **incompatible** with circular moves in space.

Example:

```
G17
G1X10Y5Z0F[F2000, A3000,D3000]
G14 X20Y15Z0 I15J12.5K5
```



3.1.12 *CET – Circular Endpoint Tolerance*

In circular interpolations, CET defines the tolerance for the variance between the starting and final radiuses of the circle arc.

Syntax

CET=*value*

---

where:
value Tolerance expressed in millimetres (inches). The default value is 0.01 mm (0.000039 in).

Characteristics:
If the difference between starting and final radius is smaller than the tolerance but not zero, the system normalises the circle data according to the values specified in CET and ARM.
If the difference is equal to or greater than the value assigned to CET, an error occurs and the programmed final points will not be executed. In this case, you must either modify the program or increase the CET tolerance.

The value assigned to CET can be modified as follows:

- In the AMP configuration using ODM tool
- By means of a specific data entry
- By writing a new CET in the part program.

CET tolerance is always expressed in the characterised measuring unit (G70/G71 apply).

If the variance between programmed start and final radius is higher than the CET value, the circle arc can be executed as follows:

- By making the CET value greater than current variance
- By programming the arc with the circle radius rather than with the centre using the format: G2/G3, final point and R radius

A RESET instruction re-establishes the default tolerance.

Example:
G71
CET=0.02 defines a 0.02 mm tolerance

3.1.13 *FCT – Full Circle Threshold*

In a circular interpolation, the FCT instruction defines a threshold for the distance between the first and the last point in an arc. Within this distance the arc is considered a full circle.

Syntax:

FCT=*value*

---

where:
value Threshold expressed in millimetres (inches). The default value is 0.001 mm (0.000039 in).

Characteristics:
The FCT command allows dealing with inaccurate program data that would otherwise prevent the system from forcing a complete circle. In other words, if the distance from the first to the last point is less than FCT, the system uses the points as if they were overlapping and forces a full circle.

FCT thresholds can be modified as follows:

- In the AMP configuration using ODM tool
- By user interface
- By writing a new CET value in the part program.

The FCT threshold is always expressed in the characterised measuring unit (G70/G71 apply).

A RESET instruction re-establishes the default threshold.

Example:
G71
FCT=0.005
In this example, FCT defines a threshold 0.005 millimetres.

3.1.14 *ARM – Defining Arc Normalisation Mode*

The ARM code defines the method the system uses to normalise an arc (programmed with the centre coordinates I and J and a final point) in order to render it geometrically congruent.

An arc is normalised when the difference between initial and final radius is less than the characterised accuracy tolerance or than the tolerance programmed with the CET command.

Before executing an arc, the system calculates the difference between initial and final radiuses.

- If the difference is zero, the control will execute the programmed arc without normalising it.
- If the difference is greater than the CET value, the control will stop without executing the move, and display a profile error message.
- If the difference is less than the CET value, the control will execute the move normalising the arc with the method specified by ARM.
- If the distance between the starting point and the final point is less than the FCT threshold, the system will force the complete circle.

Syntax

ARM=*arc mode*

where:

arc mode Is the numerical value that defines the arc normalisation mode.
Accepted values are:
0 displaced centre within the CET tolerance (default mode)
1 displaced starting point displaced by the CET tolerance
2 displaced centre independent of the CET tolerance
3 centre outside the CET tolerance range
Default value is **2**.

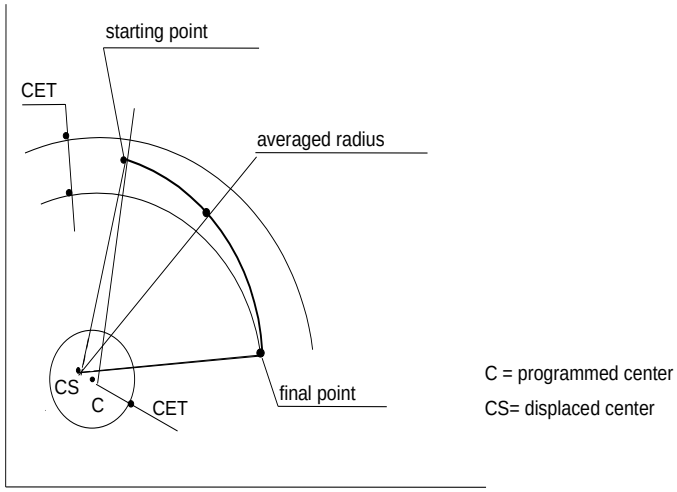
Characteristics:
The arc normalisation mode can be modified as follows:

- In the AMP configuration using ODM tool
- By user interface
- By writing a new CET in the part program.

Examples below illustrate normalisation modes.

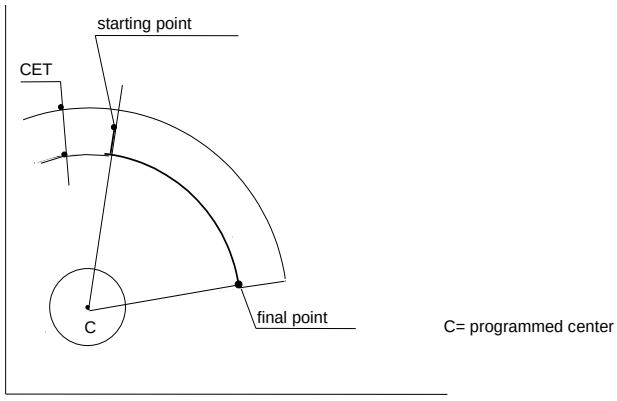
ARM=0

This is an arc through the initial and final programmed points whose centre is displaced within the tolerance defined by CET. The arc is executed with averaged radius.



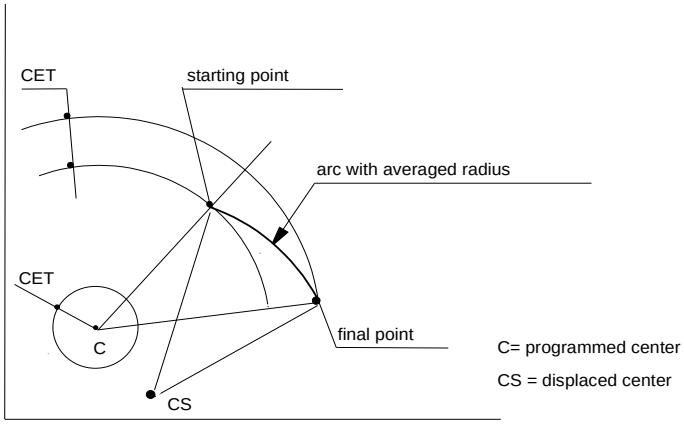
ARM=1

This is an arc through the programmed final point and the starting point displaced within the CET tolerance. The arc is executed with final radius.



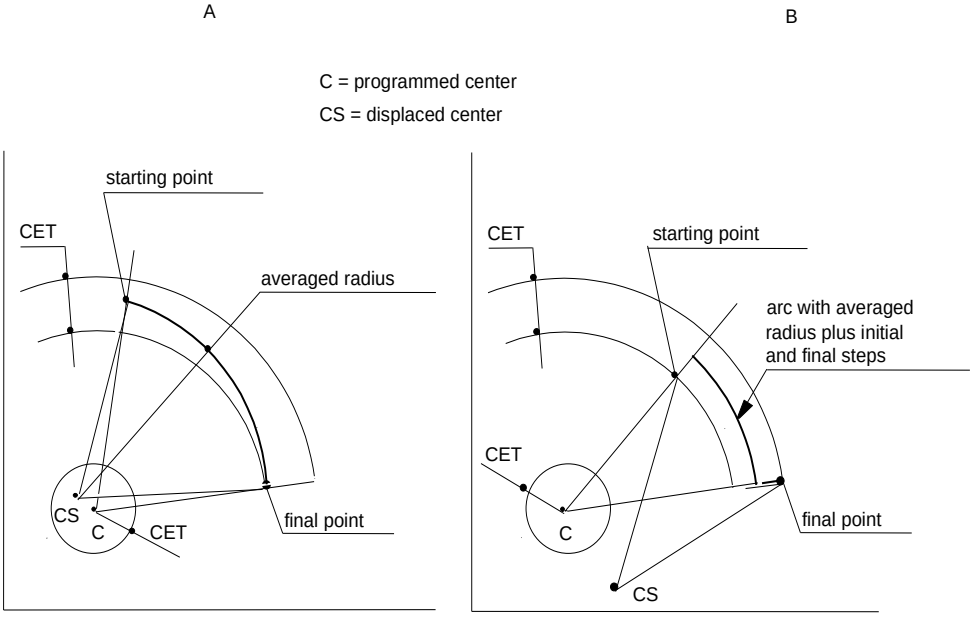
ARM=2

This is an arc whose centre is displaced irrespective of the tolerance defined with CET. In this case the arc is executed with averaged radius, while starting and final points are respected.



ARM=3

If the displacement of the centre arc is within the tolerance defined with CET, the arc centre will be displaced and the arc will pass through the programmed starting and final points. If the displacement of the centre is not within the CET tolerance, the arc will use the programmed centre and pass through the displaced starting and final points (both points are displaced within the CET/2 tolerance).
In this case the arc is executed with averaged radius.



When ARM = 1 or ARM = 3 the resultant profile can show inaccuracies ("steps"), as shown in the examples.
When ARM = 1 there will be a step at circle start equal to the difference between starting and final radiuses.
When ARM = 3 there will be a step both at circle arc start and end.
To prevent these steps from causing a servo error, program a CET value smaller than the characterised servo error threshold.

3.1.15 *SHAPERR – Circular interpolation speed reduction threshold*

KVGAIN – Circular interpolation speed reduction constant

The variables SHAPERR (Circle Reduction Threshold (CRT) in OSAI 10 Series) and KVGAIN (Circle Reduction K-Constant (CRK) in OSAI 10 Series) are used to reduce the speed on circular elements by applying an algorithm which is a function of the radius and the admissible servo error.

Syntax

SHAPERR = *value*

---

where:

value It is the maximum departure in mm (inches) desired between the programmed and the actual path. A value of 0 (zero), which is the default value, cancels this operation.

KVGAIN = *value*

---

where:

value Is the axis position interlocking gain (Kv). The value to be specified must be the same as that configured in AMP in the corresponding axis field.

If both SHAPERR and KVGAIN are nil, then no reduction speed is applied to the circular elements.

Characteristics:

If SHAPERR is not zero, the speed is reduced on all circular elements as a function of the radius of curvature of the path in order to make sure the error does not exceed the specified limit. If KVGAIN is zero, the system uses the value configured in AMP.

Hence, maximum admissible speed will be a function of the radius, the error specified by SHAPERR and the interlocking gain KVGAIN, according to the following formula:

$$V_{\max} = KVGAIN * \sqrt{2 * R * SHAPERR} * f(VFF, FLT\_2)$$

Where $f(VFF, FLT\_2)$ is a function that depends on the Velocity Feed Forward (VFF) set and the type 2 filter, if any, activated (centripetal acceleration compensation filter). For further details, see the Chapter on filters in this manual.

NOTE:

Since speed on circles is limited as a function of the percentage of VFF, it is necessary, when working with VFF, to have enabled the process variable VFF and configured the same percentage value both on the drives and on the axes.

Values assigned to variables SHAPERR and KVGAIN may be modified as follows:

- by means of ODM during AMP configuration
- from the part program with the specified syntax.

The values assigned to SHAPERR are always expressed in the current unit of measurement of the process (programmed with G70/G71).

RESET command restores the characterization values.

3.1.16 *Helical interpolation*

G02 and G03 program a helical path in only one block. The system performs the helical path by moving the plane axes in a circular interpolation while the axis that is perpendicular to the interpolation plane moves linearly.
To program a helical path, simply add a depth coordinate and the helix pitch (K) to the parameters specified in the circular interpolation block.

Syntax

G02 [*G-codes*] [*axes*] **I**.. **J**.. **K**.. [**F**..] [*auxiliary*]
or
G02 [*G-codes*] [*axes*] **R**.. **K**.. [**F**..] [*auxiliary*]

G03 [*G-codes*] [*axes*] **I**.. **J**.. **K**.. [**F**..] [*auxiliary*]
or
G03 [G-codes] [axes] R.. K.. [F..] [auxiliary]

---

where:

G-codes Other G codes compatible with G02 and G03 (See "Compatible G codes" table in Chap. 2).

axes An axis letter followed by a numerical value programmed (either decimal value or E parameter). If no axes are programmed in the block, the move will generate a full circle on the active interpolation plane.

I Abscissa of the circle centre. This is a value in millimetres or inches (decimal number or E parameter). The abscissa is expressed as a diameter unit when the corresponding axis is a diameter axis. No matter what the interpolation plane, the symbol for the abscissa is always **I**.

J Ordinate of the circle centre. This is a value in millimetres or inches (decimal number or E parameter). The ordinate is expressed as a diameter unit when the corresponding axis is a diameter axis. No matter what the interpolation plane, the symbol for the ordinate is always **J**.

R Circle radius. It is specified with the R address followed by a length value, and is alternative to the I and J coordinates.

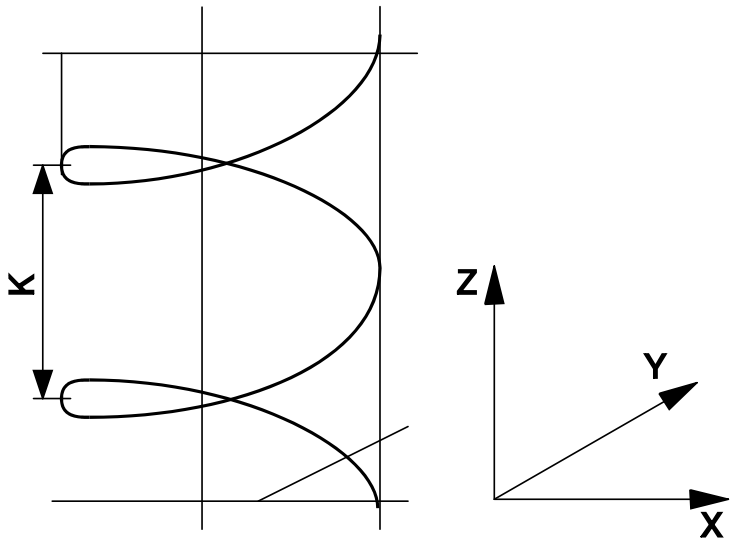
K Helix pitch. This parameter is specified with the K address followed by the pitch value. It can be omitted if the helix depth is less than one pitch.

F Feed rate. It is specified by the F address followed by a value. If it is omitted, the system will use the previously programmed feed rate. If no feed rate has been programmed, the system will signal an error.

auxiliary Programmable M, S, and T functions. Up to four M functions, one S (spindle speed) and one T (tool selection) can be programmed in the block.

Characteristics:
If helix length is a multiple of K, it is not necessary to program the final point.
If the depth is not an integer number of pitches, i.e. if the movement on the third axis is not equal to n * K), the length of the circle arc must be calculated with the decimal remainder of the pitch number. For example, if Z = 2.7 * K, then the arc that must be programmed is 360 * (2.7 - 2) = 252 degrees.

Example:
G2 X . . Y. . Z . . I . . J . . K . . F. .
In this example, addresses X, Y, I, and J refer to circle programming; addresses Z and K refer to helix programming and are respectively the depth and the helix pitch. The figure below shows the typical dimensions of a helical interpolation.



Dimensions Helix

3.1.17 *G33 – Constant or variable pitch threading*

G33 defines a cylindrical, taper, or face threading movement with constant or variable pitch. The threading move is synchronised to spindle rotation. The parameters programmed in the block identify the type of thread.

Syntax

G33 [*axes*] **K**.. [**I**..] [**R**..]

where:

axes An axis letter followed by a numerical value. It can be programmed directly or indirectly using an E parameter.

K Thread pitch (mandatory). For variable pitch threads, K is the initial pitch. It is programmed using **K** address followed by increment value.

I Pitch variation for variable pitch threading. For increasing pitch threading, **I** must be positive; for decreasing pitch threading **I** must be negative. It is programmed using **I** address followed by increment value.

R Deviation from the zero spindle angular position in degrees. R is used in multi-start threading to avoid displacing the starting point. It is programmed using **R** address followed by increment value.

Characteristics:
All these numerical values can be programmed directly with decimal numbers or indirectly with E parameters. In decreasing pitch threads, the initial pitch, the pitch variation, and the thread length must be calculated so that the pitch is greater than zero before reaching the final coordinate. Use the following formula:

$$I \leq \frac{K^2}{2\,(Zf - Zi)}$$

where:

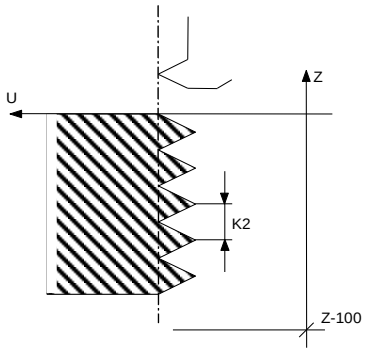
I Is the maximum pitch variation
K Is the initial pitch
(Zf - Zi) Is the thread length.

During the threading cycle the control ignores the CYCLE STOP button and the FEEDRATE OVERRIDE selector/softkey, but the SPINDLE SPEED OVERRIDE selector must be disabled by the PLC. VFF may be disabled using user interface or with a VFF command.

Constant Pitch Threading

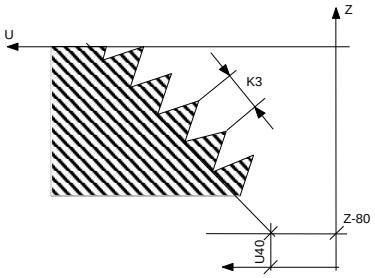
The figures that follow illustrate examples of constant pitch threading. Note that the U axis is a diameter axis.

Cylindrical threading



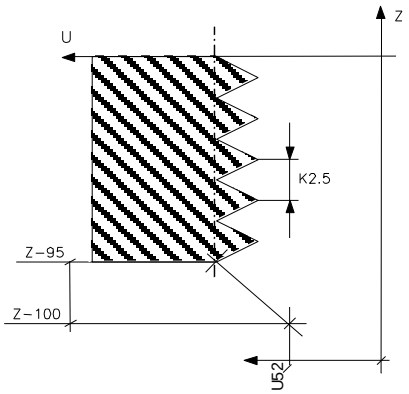
Part program block: G33 Z-100 K2

Conical threading



Part program block: G33 U40 Z-80 K3

Cylindrical-conical threading

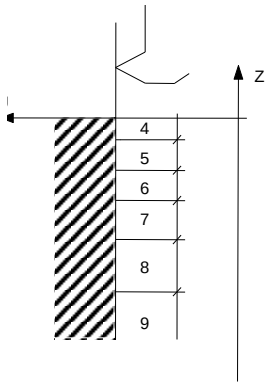


Part program blocks: G33 Z-95 K2 .5
Z-100 U52 K2.5

Variable Pitch Threading

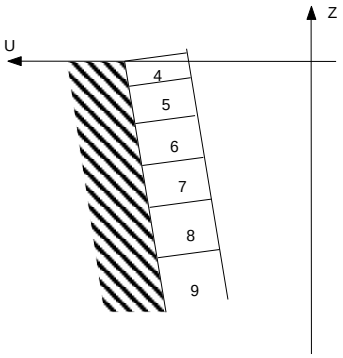
The figures that follow illustrate variable pitch threading. Note that the U axis is a diameter axis.

Cylindrical threading with increasing pitch



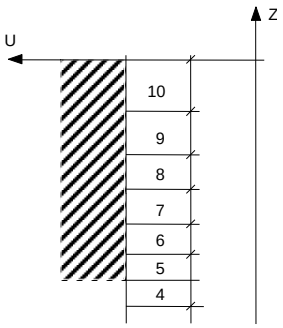
Part program block: G33 Z-50 K4 I1

Conical threading with increasing pitch



Part program block: G33 U50 Z-40 K4 I1

Cylindrical thread with decreasing pitch



Part program block: G33 Z-50 K10 I-1

Multi-start threading

An R word in a G33 block makes the control start moving the axes from an angular position that varies according to the programmed R value.

This allows the same start point for all threads to be programmed, rather than moving the start point of each thread by a distance equal to the pitch divided by the number of starts.

Example:

Three-start threading

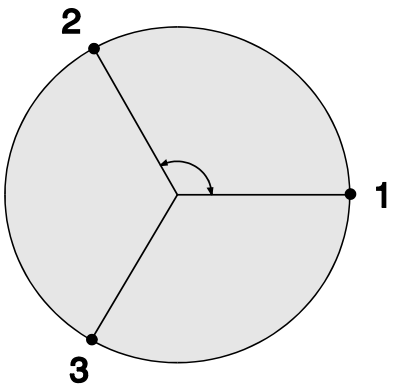
N37 G33 Z3 K6 **1st thread**

.
.
.

N41 G33 Z3 K6 R120 **2nd thread**

.
.
.

N45 G33 Z3 K6 R240 **3rd thread**



3.1.18 *Rotary axes*

In the system characterisation, axes can be configured as rotary axes, i.e. a rotary table.
To program rotary axis moves simultaneously and coordinated with the axes programmed in the same block:

- Always use decimal degrees (from +0.00001 to +99999.99999 degrees) starting from a pre-selected origin.

- Select either the rapid rate (G00) or the feed rate (G10). In a rotary move rates are always expressed in degrees per minute (dpm, F5.5 format). For example, with F75.5 the axis moves at 75.5 dpm.

To perform milling operations on a circle with a rotary table, calculate the rotary rate with the following formula.

$$F=360/pi*(A/D) = 114.64*(A/D)$$

where:

F Is the rotary rate in dpm

A Is the linear rate on the arc in millimetres or inches per minute

D Is the diameter on which the milling operation is performed (in mm or inches).

To move rotary and linear axes simultaneously in the same block, you may calculate the feed rate with one of the following formulas.

With G94:

$$F = A * \frac{\sqrt{X^2+Y^2+Z^2+B^2+C^2}}{L}$$

where:

F Is the feed rate

A Is the feed rate on the part (in mm/min or inches/min)

X Y Z B C Is the actual travel performed by each axis (in mm or inches for linear axes, in degrees for rotary axes)

L Is the resultant path length (in mm or inches).

With G93:

$$F = \frac{A}{\sqrt{X^2 + Y^2 + B^2}}$$

where:

F Is the feed rate

A Is the desired feed rate (in mm/min or inches/min) on the part

X Is the X axis incremental distance

Y Is the Y axis incremental distance

B Is the B axis incremental distance

The control cannot calculate the desired tool feed rate directly because the radius is not programmed. In these cases, the feed rate can be specified as inverse time with G93.
A block moving only the rotary axes generates an arc. If rotary and linear moves are combined, the resulting path may be an Archimedean spiral, a cylindrical helix or more complex curves, depending on the programmed number of linear axes.

3.1.19 *Axes with Rollover*

Axes with rollover are rotary or linear axes whose position is controlled between zero and a positive value configured in the *rollover pitch* parameter.
In the following description the axis with rollover is rotary and has a 360 degree rollover pitch. We assume that the axis position is controlled in the 0 to 359.9999 degree range. That is, when the axis reaches 360 degrees, the displayed position rolls over to zero degrees.

An axis with rollover can be programmed in a block or in a MDI in two different modes:

absolute mode (G90) programs the move in degrees.

incremental mode (G91) programs the move as increments in degrees from current axis position.

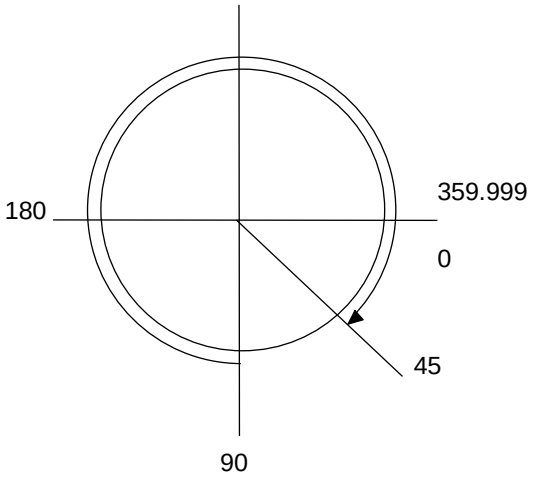
3.1.20 *G90 - Absolute mode*

In this mode:

- Displayed position is from 0 to +359.99999 degrees
- Programmed range is from 0 to ± 359.99999 degrees
- Direction of axis rotation depends on the sign of the programmed move. By convention, a positive move is clockwise and a negative move is counter clockwise.

For example, let's assume that rotary axis B is positioned at 90 degrees and the following block is written in the part program or in an MDI:

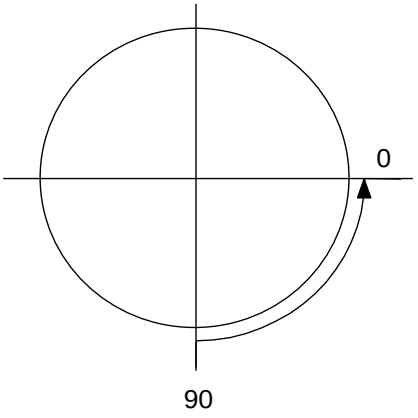
G90 B45



Clockwise Rotation

The B axis rotates by 315 degrees clockwise from the 90 degree position to reach the absolute position of 45 degrees (the sign of the move is positive).
Now let's assume that the B rotary axis is at 90 degrees and, the following block is written in the part program or in MDI:

G90 B-0



Counter clockwise Rotation

The B axis rotates by 90 degrees counter clockwise to absolute position 0 degrees because the sign of the move is negative.

3.1.21 *G91 - Incremental mode*

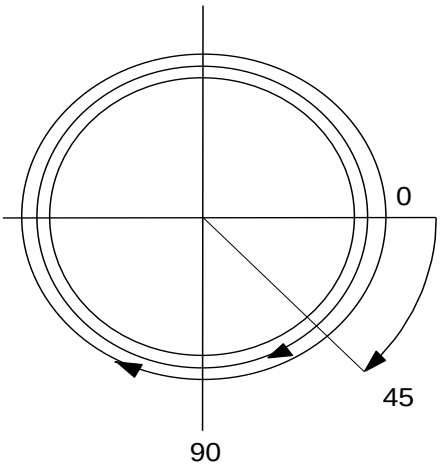
When an axis with rollover is programmed in incremental (G91) mode, the following conditions apply:

- Displayed position is from 0 to +359.99999 degrees.
- Program range is from +/-0.00001 to +/-99999.99999 degrees.
- Direction of axis rotation depends upon the sign of the programmed move. By convention, a positive move is clockwise and a negative move is counter clockwise.

The displayed position is beyond the programmed range when the programmed range is greater than +359.999 degrees.

For example, if the absolute zero position of the B rotary axis is 0 degrees and the following block is written in the part program or in MDI:

G91 B765



Incremental clockwise revolution

The B axis makes two complete clockwise revolutions plus 45 degrees (360 + 360 + 45 = 765).

3.1.22 *Pseudo axes*

A pseudo axis is an auxiliary function that may be addressed as an axis and is handled by the PLC. The pseudo axis name can be any allowed axis name (X,Y,Z,A,B,C,U,V,W,P,Q,D). In a part program block it is only possible to program up to 3 pseudo axes even though in the AMP it is possible to configure up to 6 pseudo axes.

3.1.23 *Diameter axes*

A reaming/facing head can be mounted on the spindle and controlled simultaneously with other axes. By programming such an axis (typically a U axis) as a diameter, the following can be obtained:

- Boring operations on cylindrical or conical holes
- Circular radiuses (concave or convex)
- Chamfers
- Grooves
- Facing operations
- Threads

Programming a U (diameter) axis is similar to programming other linear axes; however, its coordinates must be expressed in diameters. The measuring units can be inches or millimetres according to the current mode (G70/G71).

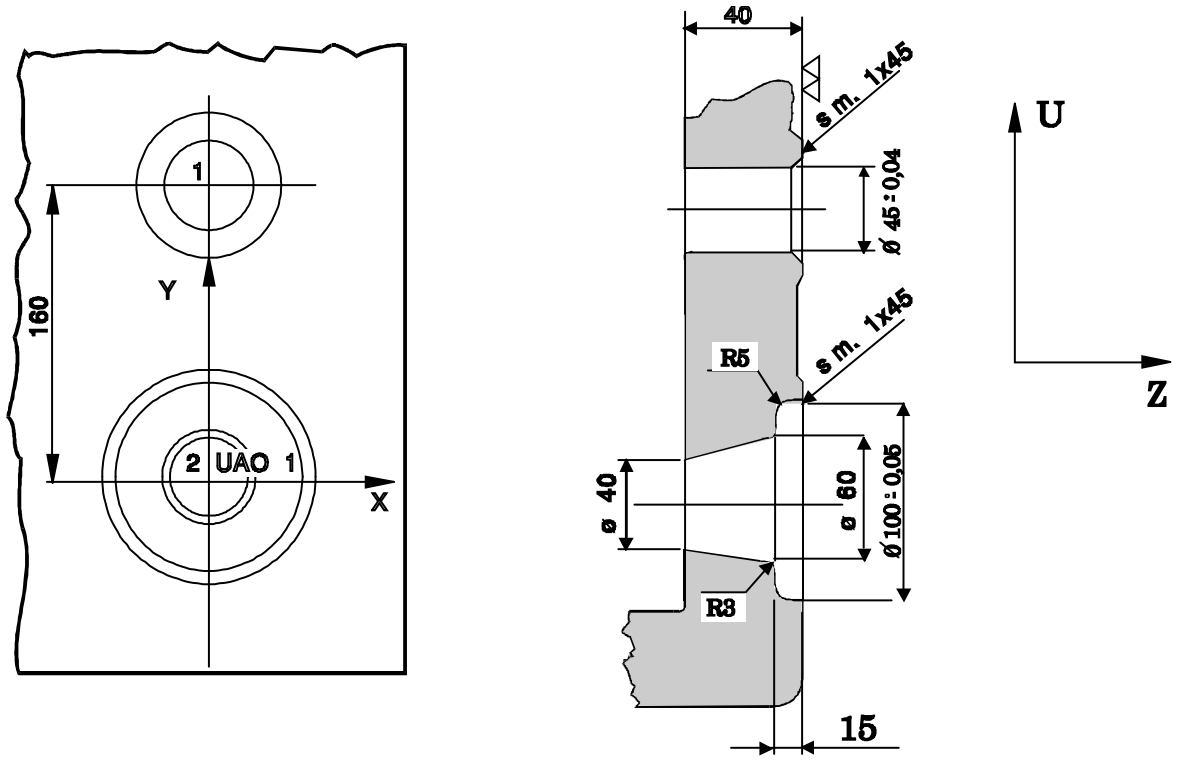
When the U axis is programmed in the same block as an X, Y or Z move, it is simultaneous with and coordinated to the other axes. U axis moves can be performed at rapid rate (G00) or feed rate (G01) with F in ipm or mmpm.
Before executing a profile with the U axis, the interpolation plane must be defined with the following command:

G16 Z U

The order of Z and U in this command is critical, i.e. G16 UZ and G16 ZU define two different interpolation planes.
Cutter diameter compensation (G41 or G42) and a machining allowance (MSA) can be applied to profiles programmed with U.

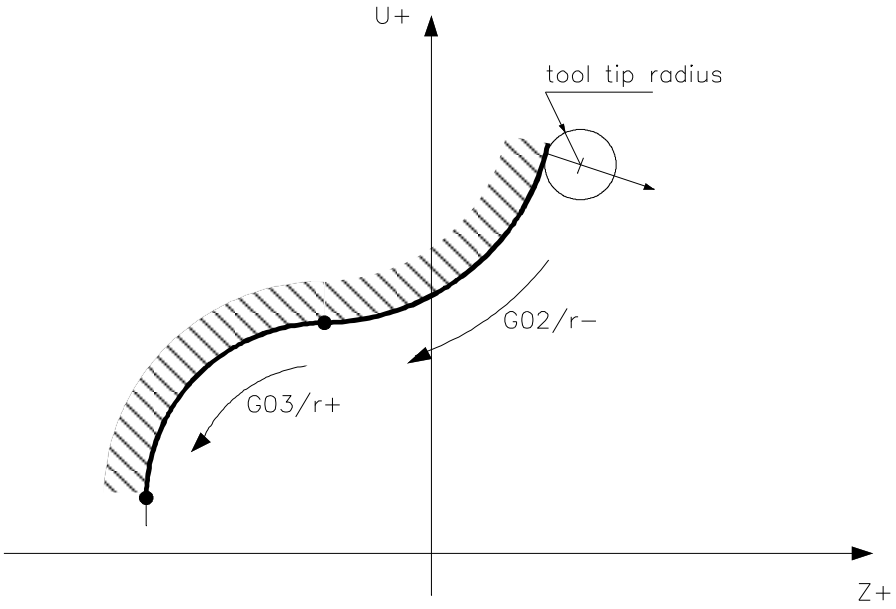
Example:

This is an example reaming/facing head used in a finishing operation.




```
N116 (DIS, "FINISHING WITH R/F HEAD")
N117 F60 S630 T9 .9 M6
N118 G16 Z U                ;Defines interpolation plane
N119 (UAO, 2)               ;Calls absolute origin for the head
N120 (UTO, 1, Z-200)        ;Temporary origin for Z (skimming the part)
N121 X Y160 M3              ;Position to hole 1
N122 G41 Z2 U51
N123 G1 Z-1 U44 .98         ;Executes the chamfer
N124 Z-44                   ;Executes hole diameter 45
N125 G G40 U40
N126 Z2 F40 S380
N127 G41 Y U106             ;Positions to hole 2
N128 G1 Z-1 U99 .975        ;Executes the chamfer
N129 Z-15                   ;Executes hole diameter 100
N130 r5                     ;Executes radius R = 5
N131 U60                    ;Executes counter boring
N132 r-3                    ;Executes radius R = 3
N133 Z-40 U40               ;Executes taper
N134 G40 Z-44               ;Continues Z axis travel
N135 G U35
N136 Z100 M5
N137 G16 X Y
N138 (UAO,1)
```

The direction of the arcs programmed with G02/G03 or with the r address and the direction for cutter diameter compensation (G41/G42) can be determined by looking at the profile on the Z-U plane. Since negative diameters are usually not programmed, you must consider only the first two quadrants of the plane.



3.1.24 *UDA – Dual axes*

It is possible to treat one or more axes as slaves or subordinate to another defined as the master. In this way only the movements of the Master need be programmed as the movement of the slaves is determined by those of the Master with which they are associated and by whether or not reverse mirror movement has been applied.

Syntax

(UDA,*master1*/*slave1*[*slave2..slaven*][,*master2*/[*slave2..slaven*],…,*master6*/[*slave2..slaven*]])
(UDA)

where:

master1. . . master6 Are the master axis names (one ASCII character per axis). Up to 6 master axes are allowed.

slave1. . . slave17 Are the slave axis and they can be:

- one ASCII character if axis is programmed by name
- a '/' character if axis is programmed by ID

Slave axes programmed by ID can be axes that do not belong to the current process. They can be axes belonging to the PLC or axes belonging to other processes and shared with PLC. Up to 17 slaves are allowed, 6 of them can be external to the process.

If slave axis is preceded by a '-', then the axis will move opposite to its master.

no parameters (UDA) without parameters disables the dual axes mode (UDA)

Characteristics:
Dual axes management does not require any special setting in the system using ODM.
After an (UDA...) command, the positive operating limit is the minimum between the positive limit of the master axis and the current position of the master plus the distance that may be covered by the slave axis. In short:

PositiveLim = min(Master PositiveLim, MasterPosition + Slave PositiveLim - SlavePosition)

In the case of a "mirror", the distance that may be covered by the slave refers to its negative limit, so it will be:

PositiveLim = min(Master PositiveLim, MasterPosition - Slave NegativeLim + SlavePosition)

The considerations made for the positive limit also apply to the negative limit:

NegativeLim = max(Master NegativeLim, MasterPosition + Slave NegativeLim - SlavePosition)

In the case of a mirror, it will be:

NegativeLim = max(Master NegativeLim, MasterPosition - Slave PositiveLim + SlavePosition)

When the (UDA...) command is executed, made both to the master axis and the slave axes have to be referenced (homing cycle already performed).
Programming a UDA command will disable any previously programmed UDA or SDA command.
The RESET command does not cancel the association between the master and slave axes.

NOTE:
The names of masters and slaves must be separated by a / (slash).

To mirror the slave axis movement, you must program the - operator before the axis name. This rule does not apply to master axes.

Example:

(UDA,X/-U)	U is slaved and mirrored to X
(UDA,Z//12)	Axis with ID 12 is slaved to a Z.
(UDA, A/B - CD)	B, C and D are slaved to A and C is mirrored to A
(UDA,X/AB/-14)	A and B are slaved to X, moreover axis with id 14 is slaved and specular to X.

3.1.25 *SDA – Special Dual Axes*

It is possible to treat one or more axes as slaves or subordinate to another defined as the master. In this way only the movements of the Master need be programmed as the movement of the slaves is determined by those of the Master to which they are associated and by whether or not reverse mirror movement has been applied. The movement of the master and slave axes can occur even if the axes are not referenced.

Syntax

(SDA, *master1/slave1*[*slave1..slaven*][,*master2/*[*slave1..slaven*],…,*master6/*[*slave1..slaven*]]**)**
(SDA)

where:

master1. . . master6 Are the master axis names (one ASCII character per axis). Up to 6 master axes are allowed.

slave1. . . slave17 Are the slave axis and they can be:

- one ASCII character if axis is programmed by name
- a '/' character if axis is programmed by ID

Slave axes programmed by ID can be axes that do not belong to the current process. They can be axes belonging to the PLC or axes belonging to other processes and shared with the PLC.
Up to 17 slaves are allowed, 6 of them can be external to the process.
If slave axis is preceded by a '-', then the axis will move opposite to its master.

no parameters (SDA) without parameters disables the special dual axes mode (SDA)

Characteristics:
Dual axes do not require any special setting during the system characterization with ODM.
After an (SDA...) command, the positive operating limit is the minimum between the positive limit of the master axis and the current position of the master plus the distance that may be covered by the slave axis. In short:

PositiveLim = min(Master PositiveLim, MasterPosition + Slave PositiveLim - SlavePosition)

In the case of a "mirror", the distance that may be covered by the slave refers to its negative limit, so it will be:

PositiveLim = min(Master PositiveLim, MasterPosition - Slave NegativeLim + SlavePosition)

The considerations made for the positive limit also apply to the negative limit:

NegativeLim = max(Master NegativeLim, MasterPosition + Slave NegativeLim - SlavePosition)

In the case of a mirror, it will be:

NegativeLim = max(Master NegativeLim, MasterPosition - Slave PositiveLim + SlavePosition)
Upon activating the (SDA,...) command the master and the slaves do not need to be referenced.

Programming an SDA command will disable any previously programmed UDA or SDA command
The RESET command does not remove the master/slave association.
This allows the zero point micro-search cycle to be performed with dual movement active. In this case when performing the zero point micro-search cycle, the system simultaneously moves the associated slaves. However at the end of the search the master axis is referenced whereas the slave axes are not. In order to reference the slave axes they should be exchanged, one by one, with the master axis by new SDA programming and the zero point micro-search cycle repeated for each one of them, redefined as the master. If this process is not completed the slave axis may have problems with software over travels. The initialisation of the slaves is not however necessary if one intends to program only the master with SDA active.
The use of the SDA function is recommended in cases where a zero point micro-search cycle is required after a shutdown of the system with the work still on the work-table.

NOTE:
The names of masters and slaves must be separated by a / (slash).


To mirror the slave axis movement, you must program the - operator before the axis name. This rule does not apply to master axes.

Example:

(SDA,X/-U)	U is slaved and mirrored to X
(SDA, A/B - CD)	B, C and D are slaved to A and C is mirrored to A
(SDA,X/AB/-14)	A,B are slaved to X, moreover axis with id 14 is slaved and mirrored to X

3.1.26 *XDA – Master/Slave axes*

One or more axes can be used as "slave" axes, i.e. subordinate to another axis, defined as "master". In this way, you only need to program the movements of the master, since the movements of the slaves are determined by the master they are associated with and by a following factor, which is defined specifically for each individual slave. Commands of different types can be imparted with this feature, each of them having a specific syntax.

Master/Slave Association

This instruction defines the association between a master axis and its slaves (up to 17 axes). This instruction **does NOT activate the "following" function**, whose activation is done by means of a specific command. This means that after this instruction a movement of the master does not bring about a movement of the slave(s). **After this instruction and until the slave is released from the master, slave movements cannot be programmed.**

Syntax

(XDA, **1**, *master/slave1*[*slave2*[..]], *mode*, *ratio*, *space***)**

---

where:

master Is the name of the master axis and is denoted by a single ASCII character.

slave1…slave17 Are the slave axis and they can be:
one ASCII character if axis is programmed by name
a '/' character if axis is programmed by ID
Slave axis programmed by ID can be axis that do not belong to current process. They can belong to PLC or axis belonging to other processes and shared with PLC.

Up to 17 slaves are allowed, 6 of them can be external to the process.
If slave axis is preceded by a '-', then the axis will move opposite to its master.

mode Defines the master axis following mode used by the slave(s). It can be:

- **0** The slave follows the master point by point
- **1** The slave follows the master in terms of speed
- **2** The slave follows the master in terms of position
- **3** The slave follows the master in terms of position, and the synchronisation distance is taken up

ratio This is the specified master / slave(s) following ratio . It must be viewed as a multiplication factor for the feed rate of the master or the distance covered by it. If the value of this ratio is 1.0, the motion of the master is reproduced exactly by the slave; if it is smaller than 1.0, feed rate/distance are reduced, if it is greater than 1.0 they are increased. This value can be preceded by a sign.

distance This is the distance to be covered by the slave to synchronise with the motion of the master.

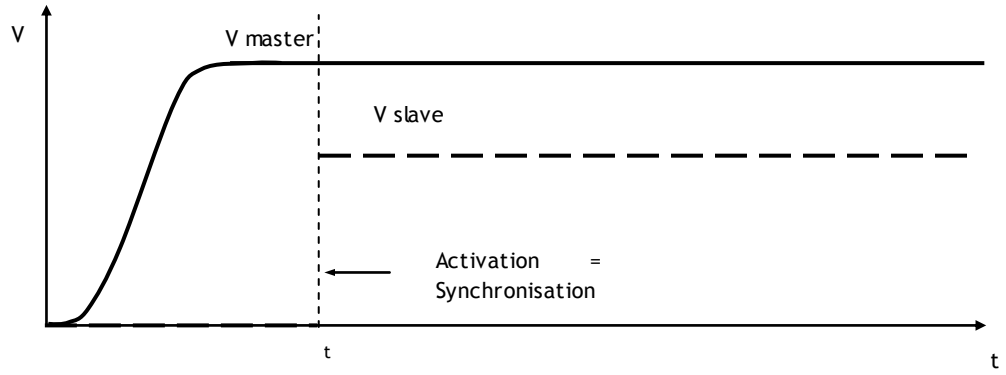
Characteristics:
A slave axis can be a shared axis, i.e. an axis shared with the PLC environment. In this case, the axis may continue to be moved by PLC even after the association with the master, however it cannot be moved while it is following the master.
Master axis identifies an axis in the process where XDA is activated and a non-existing axis as well. In the latter case the process creates a "virtual axis" having the name specified.
The axis has poor dynamic characteristics as, for example, its maximum feed will be equal to the lowest feed of the slave axes. This axis could be part of TCP/virtualizations and can't be homed.

RESET command does not remove master / slave association

The various following modes available are:

Mode 0
In this mode, the slave axis follows the master proportionately to the value of the ratio (if the ratio = 1, the slave reproduces the movement of the master axis), synchronisation is instantaneous and the variation in the feed rate of the slave is "*in steps*". Slave position and feed rate values are calculated, instant by instant, according to the following formulas:

$$\mathbf{Vslave = Vmaster * FollowRate}$$
$$\mathbf{PosSlave = PosSlave_{t0} + (PosMaster - PosMaster_{t0}) * FollowRate}$$

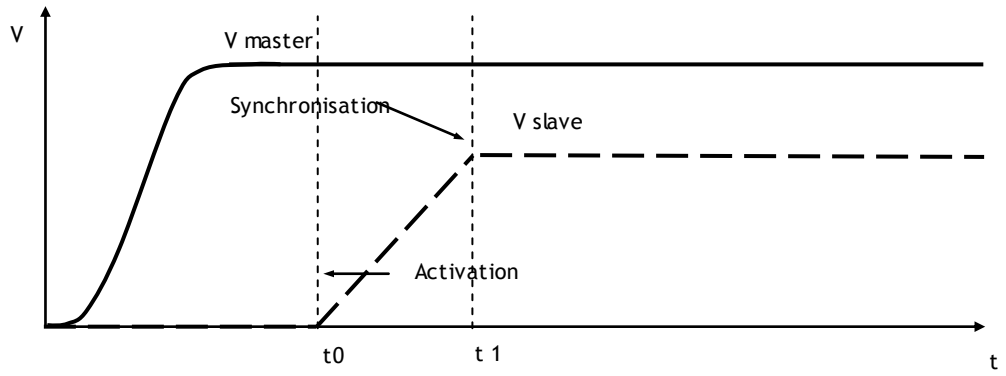


If the speed specified for the slave axis as a result of the following command exceeds the maximum admissible value for this axis, the system will reduce the feed rate requested accordingly and will give out an emergency (servo error) message, since the slave is unable to follow the required position.

Mode 1

In this mode, the slave follows the feed rate of the master proportionately to the value of the ratio (if the ratio = 1, the slave copies the movement of the master axis exactly); the synchronisation depends on the dynamic characteristics of the slave axis and/or the distance parameter which defines the synchronisation distance.

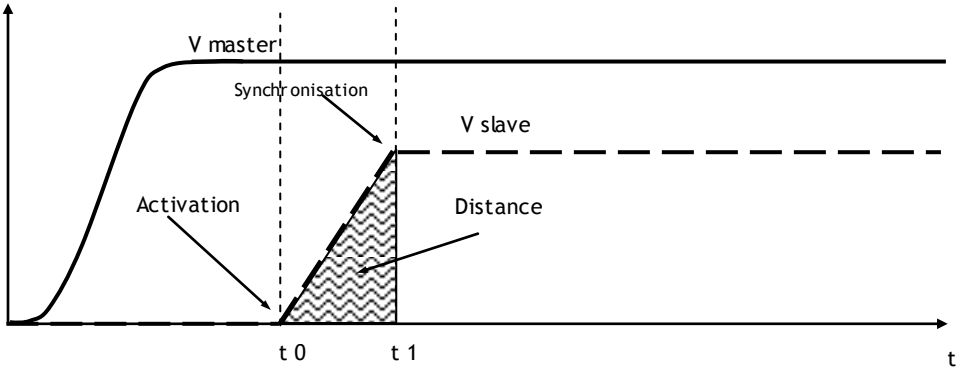
If the distance value = 0, the slave will synchronise with the master based on its maximum acceleration and using linear ramps **only**.



If the value is not 0, the slave will synchronise with the master based on an acceleration calculated as a function of the synchronisation distance and using **linear ramps only**. The acceleration will be calculated again with each sampling process based on the following formula:

$$\mathbf{Aslave = ((Vmaster * FollowRate)^2 + Vslave^2) / 2 * Distance}$$

where the value of the distance is gradually reduced based on the distance covered during the synchronisation stage. No check is made on the ensuing acceleration value, and therefore servo errors may arise if the acceleration exceeds the maximum value that can be withstood by the axis.



Once the synchronisation with the master has taken place, the slave will move according to this formula:

$$\mathbf{Vslave = Vmaster * FollowRate}$$

The feed rate (**Vslave**) determined in this manner is "theoretical", since it is necessary to determine whether this request is compatible with the dynamic characteristics of the axis (maximum feed rate and maximum acceleration). The moment the feed rate of the master varies, the slave will follow this variation based on its acceleration value. If the feed rate requested of the slave exceeds its maximum admissible feed rate, the system will reduce the feed rate requested accordingly. Hence, the feed rate and acceleration values with which the slave has to be moved, $Vslave_i$ and $Aslave_i$, will be determined

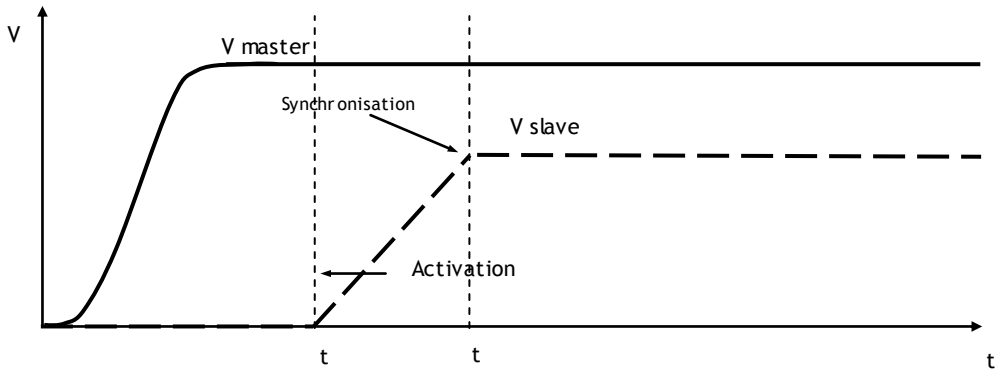
instant by instant. The position of the slave will therefore be calculated on the basis of these values:

$$\mathbf{PosSlave}_{tn+1} = \mathbf{PosSlave}_{tn} + \mathbf{Vslave}_{i} + \mathbf{Aslave}_{i}$$

Mode 2

In this mode, the slave follows the position of the master proportionately to the value of the ratio (if the ratio = 1 the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave axis and/or the distance parameter which defines the synchronisation distance.

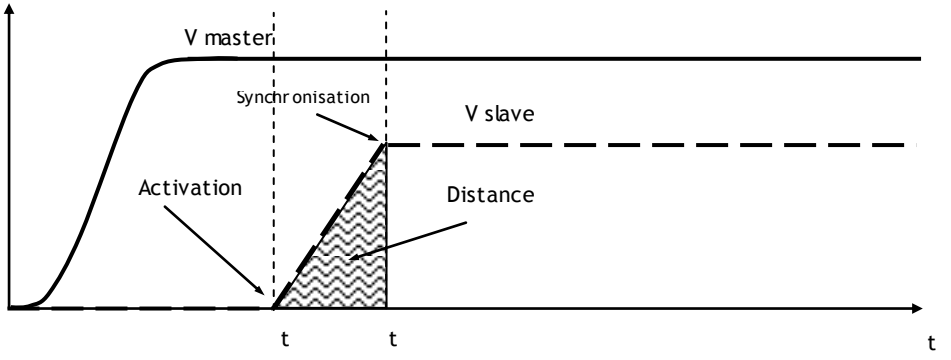
If the distance value is 0, the slave will synchronise with the master based on its maximum acceleration and using linear ramps **only**.



If the value of distance is not 0, the slave axis will synchronise with the master axis based on an acceleration calculated as a function of the synchronisation distance and using linear ramps **only**. The acceleration value will be calculated again with each sampling step according to this formula

$$\mathbf{Aslave} = ((\mathbf{Vmaster} * \mathbf{FollowRate})^2 + \mathbf{Vslave}^2) / 2 * \mathbf{Distance}$$

where the value of the distance is gradually reduced based on the distance covered during the synchronisation stage. No check is made on the ensuing acceleration value and therefore servo error messages may be generated the moment the acceleration exceeds the maximum value that the axis can withstand.



Once the synchronisation with the master axis has occurred, the slave will move according to the following formulas:

$$\mathbf{PosSlave} = \mathbf{PosSlave}_{t1} + (\mathbf{PosMaster} - \mathbf{PosMaster}_{t1}) * \mathbf{FollowRate}$$
$$\mathbf{Vslave} = \mathbf{Vmaster} * \mathbf{FollowRate}$$

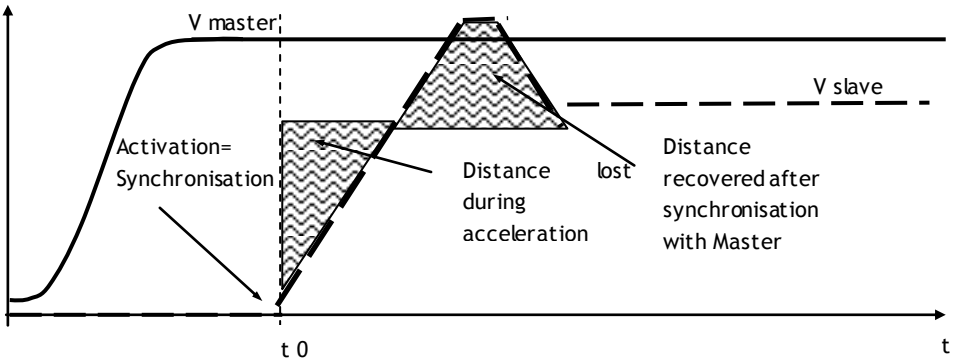
The position, **PosSlave**, and the feed rate, **Vslave**, determined in this manner should be rated as "theoretical" values, since it is necessary to determine whether the values requested are compatible with the dynamic characteristics of the axis (Maximum feed rate and maximum acceleration). The moment the feed rate of the master varies, the slave will follow this variation according to its own acceleration value. If the feed rate requested for the slave exceeds the maximum value admissible for this axis, the system will reduce the feed rate accordingly. To this end, the two values with which to move the slave, $\mathbf{Vslave}_i$ and $\mathbf{Aslave}_i$ will be calculated instant by instant. The actual position of the slave axis will therefore be calculated on the basis of these values:

$$\mathbf{PosSlave}_{tn+1} = \mathbf{PosSlave}_{tn} + \mathbf{Vslave}_i + \mathbf{Aslave}_i$$

The difference between the actual and the theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than the theoretical value (**Vslave**).

Mode 3

In this mode, the slave follows the position and feed rate of the master proportionately to the value of the ratio (if the ratio = 1, the slave reproduces exactly the movement of the master); synchronisation depends on the dynamic characteristics of the slave.



During the entire movement of the slave (i.e. both during and after the synchronisation stage), the motion of the axis is according to the following formulas (always using linear ramps):

$$\mathbf{PosSlave = PosSlave_{to} + (PosMaster - PosMaster_{t0}) * FollowRate}$$

$$\mathbf{Vslave = Vmaster * FollowRate}$$

The position (**PosSlave**) and the feed rate (**Vslave**) determined in this manner should be rated as "theoretical" values, in that it is necessary to determine whether these requests are compatible with the dynamic characteristics of the axis (max admissible feed rate and max admissible acceleration). The moment the feed rate of the master varies, the slave follows the variation according to its own acceleration value. If the feed rate requested of the slave is higher than its maximum admissible feed rate, the system reduces the feed rate requested accordingly. To this end, the two values with which the axis is to be moved (**Vslave** $_i$ and **Aslave** $_i$) will be calculated instant by instant. The actual position of the slave will therefore be calculated on the basis of these values:

$$\mathbf{PosSlave_{tn+1} = PosSlave_{tn} + Vslave_i + Aslave_i}$$

The difference between the actual and the theoretical position of the axis is taken up by the slave during its motion (even when the master has stopped moving) by moving, to the extent feasible, at a rate higher than the theoretical value (**Vslave**).

3.1.27 *Releasing the Slave(s) from the Master*

This instruction removes the association between the master and the slave(s). **Following this instruction it will be possible to program any movement of the slave axis.**

Syntax

(XDA)

3.1.28 *Defining/Changing the following ratio*

This instruction defines/changes the parameter that determines the ratio according to which the master is followed by the slave(s) concerned.

Syntax

(XDA, 2, *slave1*[*slave2*[..]]**,** *ratio***)**
(xda, 2, *slave1*[*slave2*[..]]**,** *ratio***)**

where:

slave1…slave17 Are the slave axis and they can be:

- one ASCII character if axis is programmed by name
- a '/' character if axis is programmed by ID

Slave axes programmed by ID can be axes that do not belong to the current process. They can be axes belonging to the PLC or axes belonging to other processes and shared with the PLC.
Up to 17 slaves are allowed, 6 of them can be external to the process.
If slave axis is preceded by a '-', then the axis will move opposite to its master.

ratio This is the specified master/slave(s) following ratio. It must be viewed as a multiplication factor for the feed rate of the master or the distance covered by it. If the value of this ratio is 1.0, the motion of the master is reproduced exactly by the slave; if it is smaller than 1.0, feed rate/distance are reduced, if it is greater than 1.0 they are increased. This value can be preceded by a sign.

Characteristics:
The command can be used both when a slave is already following the master axis (it then brings about the release of the slave from the master and activates a new synchronisation stage using the new following parameter) and when the following function is not active (the command activates the following value to be used in the next movement stage).
If the uppercase syntax is used, the movement is stopped and the continuous command underway, if any, is terminated. If the lowercase syntax is used, instead, a continuous mode command is given out; at any rate, the axes are stopped at zero speed and after that are restarted immediately. If you do not want the movement to stop, this can be accomplished by having the PLC execute a similar command.

3.1.29 *Activating the following function*

The following of the Master axis by the slave(s) is immediately activated. The following mode is defined by the "mode" parameter contained in the master/slave association command.

Syntax

(XDA, 3, *slave1*[*slave2*[..]]**)**
(xda, 3, *slave1*[*slave2*[..]]**)**

---

where:

slave1…slave17 Are the slave axis and they can be:

- one ASCII character if axis is programmed by name
- a '/' character if axis is programmed by ID

Slave axis programmed by ID can be axis that do not belong to current process. They can be axis belonging to PLC or axis belonging to other processes and shared with PLC.

Up to 17 slaves are allowed, 6 of them can be external to the process.

If slave axis is preceded by a '-', then the axis will move opposite to its master.

Characteristics:

If the uppercase syntax is used, the movement is stopped and the continuous command underway, if any, is terminated. If the lowercase syntax is used, instead, a continuous mode command is given out; at any rate, the axes are stopped at zero speed and are restarted immediately. If you do not want the movement to stop, this can be accomplished by having the PLC execute a similar command.

3.1.30 *Disabling the following function*

The following of the Master axis by the slave(s) is immediately deactivated. The release modality is defined by the "mode" parameter contained in the master/slave association command.

Syntax

(XDA, 4, *slave1*[*slave2*[..]]**)**
(xda, 4, *slave1*[*slave2*[..]]**)**

---

where:

slave1…slave17 Are the slave axis and they can be:

- one ASCII character if axis is programmed by name
- a '/' character if axis is programmed by ID

Slave axes programmed by ID can be axes that do not belong to the current process. They can be axes belonging to the PLC or axes belonging to other processes and shared with the PLC.
Up to 17 slaves are allowed, 6 of them can be external to the process.
If the slave axis is preceded by a '-', then the axis will move opposite to its master.

Characteristics:
The slave axis remains associated with the master, it just does not follow it any longer. Depending on the "*mode*" parameter defined in the master/slave association command, either of the following will occur:

0 The slave changes abruptly from the current feed rate to zero.
others The slave comes to a halt according to its deceleration ramp.

If uppercase syntax is used, the movement is stopped and the continuous command underway, if any, is terminated. If lowercase syntax is used, instead, a continuous mode command is given out; at any rate, the axes are stopped at zero speed and are then restarted immediately. If you do not want the movement to stop, this can be accomplished by having the PLC execute a similar command.

Example:

```
N10 (XDA,1,X/ZA,3,0.8,0.0)   Activates master X and slaves Z and A
N20 (XDA,3,ZA)               Activates following mode by A and Z
N30 G1X100F2000
N40 X300
N50 (xda,4,Z)                Deactivates following mode by Z in continuous mode
N60 X400
N70 X500
N80 (xda,4,A)                Deactivates following mode by A in continuous mode
N90 X660
N100 X700
N110 (xda,3,ZA)              Reactivates following mode by A and Z in continuous mode
N120 GX0
N130 (XDA)                   Removes association of slaves Z and A with master X
N140 GX
```

Example 2:

Let axes XYZAB be configured with id 1 2 3 4 5 respectively, and the axes belonging to the PLC (a and b) with IDs 14 and 20, respectively).

N10 (XDA,1,X/14A/20,3,0.8,0.0)	Activates master X and slave A from process and a & b from PLC
N20 (XDA,3,A/14/20)	Activates the following mode by A, a, and b
N30 G1X100F2000	
N40 X300	
N50 (xda,4,/14)	Deactivates following mode by a in continuous mode
N60 X400	
N70 X500	
N80 (xda,4,A)	Deactivates following mode by A in continuous mode
N90 X660	
N100 X700	
N110 (xda,3,/14A)	Reactivates following mode by A and a in continuous mode
N120 GX0	
N130 (XDA)	Removes association of slaves A, a, and b with master X
N140 GX	

3.1.31 *AXT – Tangential control*

This command activates the tangential control on a rotary axis.
The tangential control automatically introduces a movement on the rotary axes so its inclination will not change along the profile.
This command replaces TCP of type 2 of the OSAI S10 controls.

Syntax

(AXT, *axp1axp2axr*, *direction*, *min angle* [,*null angle*][, *rotary feed* [, *mode*]]]*)*
(AXT)

where:

axp1axp2axr — These are the names of the two axes of the plane where the profile is defined (abscissa and ordinate respectively) and the rotary axis that will be controlled.

direction — Indicates the rotary axis direction for positive motions. It has a positive value (+) if the rotation is CW, while the value is negative (-) during CCW rotation.

min angle — It is the amplitude of the threshold angle between two elements that make it necessary to introduce a move of the rotary axis at the end of the first element.

null angle — It is the threshold for the minimum angle between two elements. If an angle between elements is smaller than *null angle*, then the tangential control does not activate and the rotary axis does not move. If this value is not provided than it is assumed to be the same as the value for null angle configured in AMP.

Rotary feed — Feed when the rotary axis moves between two elements (angle between elements bigger than *min* angle). If not provided than programmed feed is used

Mode — Indicates if an approaching element is needed before function is enabled.
Valid values are:
0 an approaching element has to be defined
1 **it is not** necessary to define an approaching element
Parameter default value is 0.

(no parameter) — The command without parameters disables tangential control

Characteristics:
This function introduces a movement of the specified rotary axis, so that its inclination will not change along the profile. The profile is defined in the plane identified by the two linear axis specified in command.

The movement is introduced:

- *between two elements* to rotate between edges:
  If the angle between the current element and the following one is bigger than *min angle,* the rotation movement is added between the current element and the beginning of the following one. If the angle is smaller than *min angle* the rotation is executed during the execution of the following element

- *on a circular element* to maintain the same inclination on the element.
  In this case the rotation movement runs simultaneously with the circular element.

If mode value is 0, all previous considerations applies to all elements of the profile following the command excluding the first one. In this case, the first element is considered as the in feed element, so even if it is a circular element, no movement is applied to the rotary axis. System will start to generate movements for the rotary axis starting from the second element after the AXT command.

If a movement block contains an explicit movement of the rotary axis, then the tangential control will be disabled to let the axis move as programmed. It will be re-enabled on the first movement block with no explicit movement for the rotary axis.

3.1.32 *AXS – Axes shared with PLC*

This command allows the PLC to share one or more axes belonging to the process.

Syntax

(AXS, *type***,** *axis1* [[**,** *axis2*] … [**,** *axis12*]]***)***
(AXS)

where:

type It is a parameter specifying the operation requested on the axes.
There two allowed values:
- **S:** The axes, belonging to the process, will be shared with PLC.
- **R:** The axes, belonging to the process, currently shared with PLC will be released.

axis1…axis12 parameters ASCII characters identifying an axis name belonging to the process. Up to 12 can be shared with PLC.

(no parameters) If the command is executed with no parameter, then all axes currently shared will be released.

Characteristics:
Part program cannot move axes shared with PLC.
Any attempt to move an axis shared with PLC from part program will trigger an error.
The RESET does not delete any axes sharing

3.1.33 *AXF – Dynamic tracking axes definition*

From a dynamic point of view, this command allows the definition of some axes to be managed separately from others. The axes in that triliteral delineate the programmed geometry, but their movement is independent from other axes in the process.

Syntax

(AXF, *axes_names***)**
(AXF)

where:

axes_names	Is the list of the axes names to which the tracking algorithm will be applied
(no parameter)	If the command is executed with no parameters, all the axes will be interpolated together

Characteristics:
The axes on which the dynamic tracking algorithm is applied (tracking axes) are interpolated separately from the others as this triliteral creates an interpolator for each axis to be followed. As a result, profile velocity won't reach zero where the axes begin or finish their movement, making all the processing more fluid (see example).

Tracking axes have to be programmed anyway.

Only if there is at least one axis on which the algorithm is applied this command is active.
All the new triliteral programming deactivate a previous tracking axes configuration.

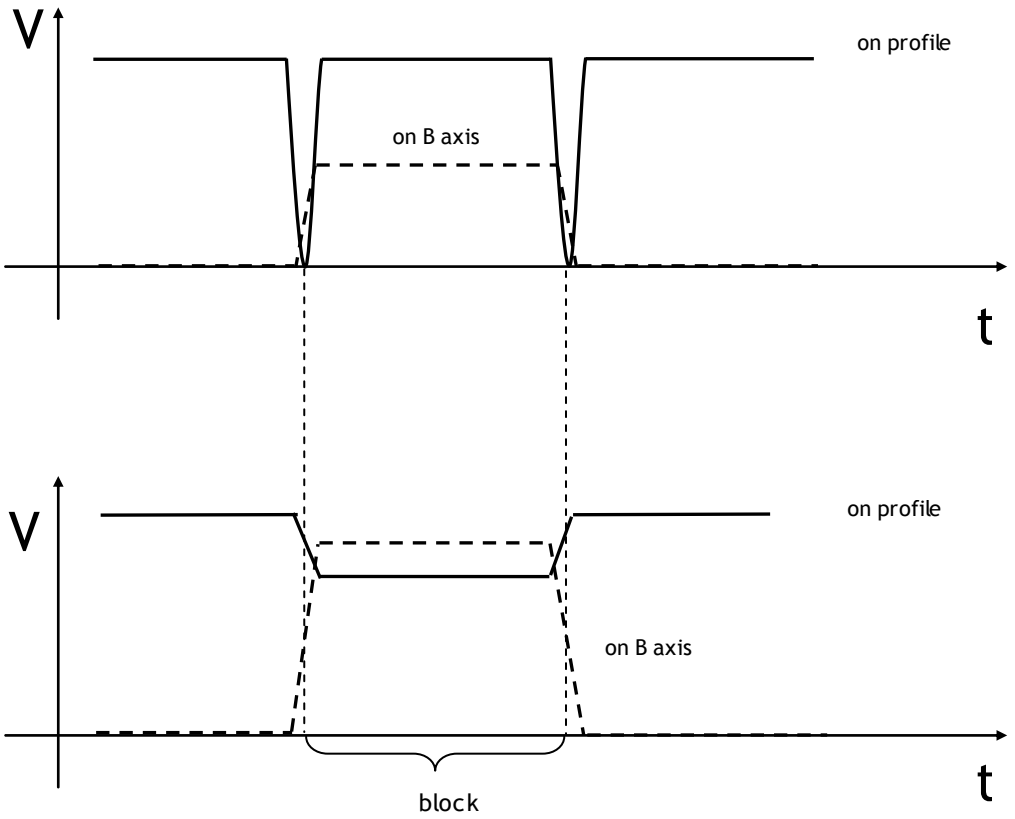
Tracking axes are moved according to the setup dynamic parameters. If the programmed velocity value is higher than the maximum velocity, the velocity value will be automatically reduced to observe the dynamic parameters of the tracking axes.

Example:

Given a process with 3 linear axes (XYZ) and 1 rotary axis (B), programmed as follows:

(AXF, B)	Following this command, B axis is enabled to dynamically follow all the axes in the process.
G17 G1 X100 YZB F5000 G3 X150 Y50 I100 J50 B90 G1 Y150	Following this command, B axis is enabled to dynamically follow all the axes in the process. A rounded edge is programmed on which a rotary movement is inserted: › If the AXF command is **not used**, linear axes at the beginning and at the end of the block stop as B axis is added to the interpolation. At the end of the block, B axis will have finished the movement as well. › **with the** AXF command, the B movement doesn't limit the linear axis dynamic and so the velocity on the profile doesn't reach 0.
(AXF)	After this command, rotary B is interpolated with other axes of the process.

The graph below shows trend velocity on the profile and on the B axis, for both the aforementioned cases.



3.2 ORIGINS AND COORDINATES CONTROL CODES

The functions in this class perform the following operations:

G CODE	FUNCTION
G04	Dwell at end of step
G09	Deceleration at end of step
G16	Define interpolation plane and Cartesian tern for circular interpolation in space.
G17	Circular interpolation and cutter diameter compensation on the plane defined by first and second axes defined in AMP. Cartesian tern for circles in space formed by three axes in AMP.
G18	Circular interpolation and cutter diameter compensation on plane defined by third and first axes defined in AMP. Cartesian tern for circles in space formed by three axes in AMP.
G19	Circular interpolation and cutter diameter compensation on plane defined by second and third axes defined in AMP. Cartesian tern for circles in space formed by three axes in AMP.
G27	Continuous movement with automatic velocity reduction on bevel
G28	Continuous movement without automatic velocity reduction on bevel
G29	Point-to-point movements
G70	Programming in inches
G71	Programming in millimetres
G79	Programming referred to axes home switch
G90	Absolute programming
G91	Incremental programming
G92	Axis pre-setting
G93	Inverse time (V/D) feed rate programming mode
G94	Feed rate programming in ipm or mmpm
G95	Feed rate programming in ipr or mmpr
G98	Axis pre-setting It functions like G92 but also applies when mirror is active.
G99	Cancel Axis pre-set

NOTE:
The planes specified in G17, G18, G19 are valid if at least three axes have been configured in AMP.

3.2.1 *G17 G18 G19 – Selecting the interpolation plane*

These G codes are used for defining the interpolation plane as and the Cartesian tern described below:

G17 Active interpolation plane formed by axes 1 and 2 (usually XY) and third tern axis (axis 3, usually Z).

G18 Active interpolation plane formed by axes 3 and 1 (usually ZX) and third tern axis (axis 2, usually Y).

G19 Active interpolation plane formed by axes 2 and 3 (usually YZ) and third tern axis (axis 1, usually X).

Axes 1 (X), 2 (Y), and 3 (Z) are the first three axes declared in AMP configuration.

Syntax

G17
G18
G19

The syntax for each function is simply the G code by itself in one block without parameters or other pieces of information.

Characteristics:
G17, G18, G19 cannot be used if the following G codes are active:

- Cutter compensation (G41-G42)
- Standard canned cycles (G81-G89)
- High- speed programming (G61)

3.2.2 *G16 – Defining the interpolation plan*

Like G17, G18, and G19, G16 defines the abscissa and the ordinate of the interpolation plane but is not linked to the first and second configured axes.

Syntax

G16 *axis1 axis2* [*axis3*]

---

where:

axis1 Is the name of the abscissa of the interpolation plane (typically X). It must be one of the configured axes in the system.

axis2 Is the name of the ordinate of the interpolation plane (typically Y). It must be one of the configured axes in the system.

axis3 Is the name of the third axis in the Cartesian tern (typically Z). It is used to program the circular interpolation in space. It must be one of the configured axes in the system.

Characteristics:

G1 cannot be used if the following G codes are active:

- Cutter compensation (G41-G42)
- Standard canned cycles (G81-G89)
- High- speed programming (G61)

Example:

G16 X A specifies the interpolation plane formed by axes X and A .

3.2.3 *G27 G28 G29 – Defining the dynamic mode*

The G functions in this class define how the axis moves on the profile and positions at profile end. These codes are always accepted by the control.

G27 Specifies a continuous move with automatic velocity reduction on bevels. At the end of each element velocity is automatically calculated by the control and optimised according to the profile shape. This calculation is based on DLA and MDA values and the maximum velocity step configured on the axes.

G28 Specifies a continuous move without automatic velocity reduction on bevels. At the end of each element the velocity on the profile is equal to the programmed feed rate.

G29 Specifies a point-to-point move that is independent from the programmed path function (G01-G02-G03– G12 – G13). At the end of each element the velocity on the profile is 0.

Syntax

G27 [*G-codes*] [*operands*]
G28 [*G-codes*] [*operands*]
G29 [*G-codes*] [*operands*]

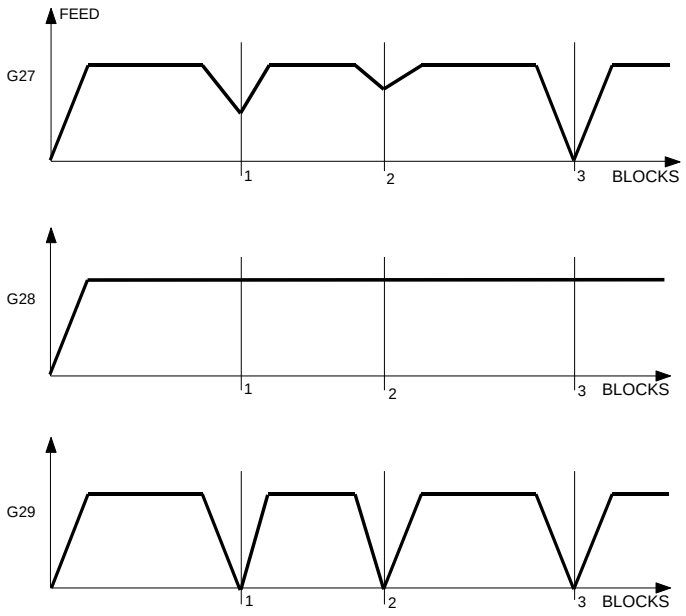
---

where:

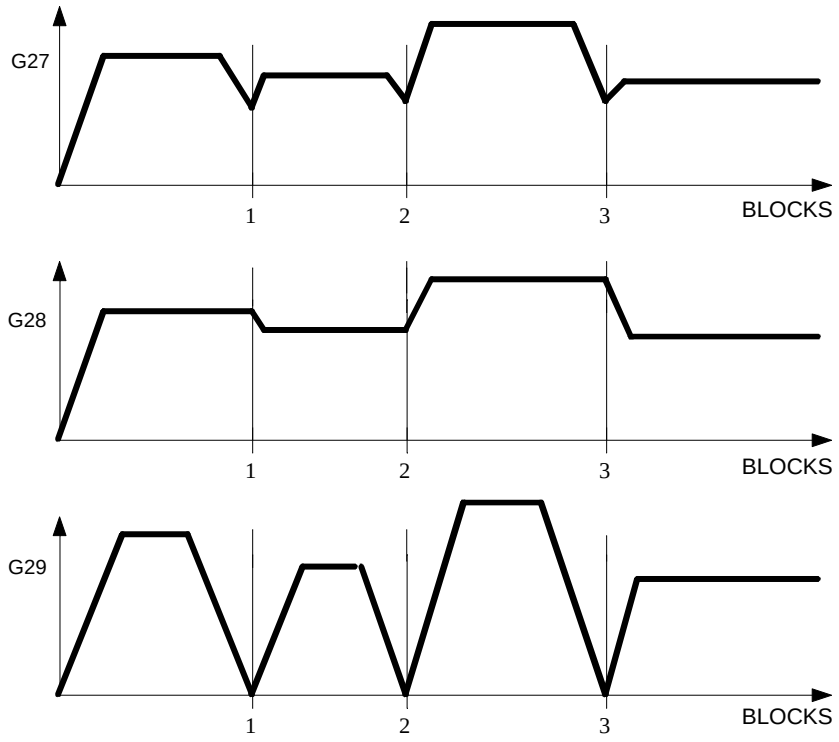
G-codes Other G codes that are compatible with G27, G28 and G29 (See "Compatible G codes" table in Chapter 2).

operands Any operand or code that can be used in a G function block.

Characteristics:
The following diagram shows how G27, G28 and G29 operate when the programmed feed rate is constant throughout the profile.



The following diagram shows how G27, G28 and G29 operate when the programmed feed rate varies through the profile.



In each block the move is divided into three steps:

1) Acceleration
2) Uniform move at programmed feed rate
3) Decelerated motion

G27 and G28 differ only in the last step as shown in previous diagram.
Positioning at the machining rate (G1, G2, G3) is available in continuous mode (G27, G28 and G29) whereas rapid positioning (G0) is always point to point, i.e. with deceleration down to null velocity and accurate positioning regardless of the system status. With G27-G28 (continuous mode) the control explores and executes the profile as if it were a single block.

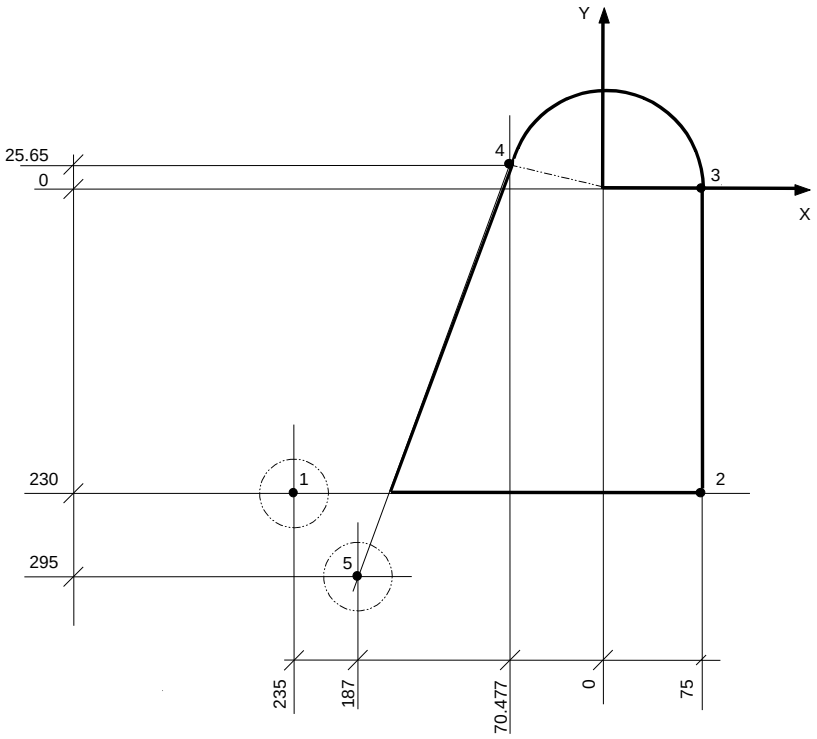
For this reason, auxiliary functions S and T are not allowed within the profile executed in G27-G28. M functions can be executed if in AMP the parameter "allowed in continuous" is set.

Continuous mode can be temporarily closed by a G00 move that is still part of the profile. The allowed M, S and T functions may therefore be programmed in a block following G00.

NOTE:
The G code that has been configured in AMP (typically G27) is automatically selected at power-up or after a reset.

Example:

This is a contouring example in continuous and point-to-point mode.



Program 1 (continuous mode):

```
      N9 (DIS,"MILL DIA. 16")
      N10 T4.4 M6 S800
1     N11 G X-235 Y-230 M13
      N12 Z-10
2     N13 G27 G1 X75 F500          ;Continuous mode starts (G27)
3     N14 Y .
4     N15 G3 X-70.477 Y25.651 I J
5     N16 G1 X-187 Y-295
      N17 G Z5 M5                  ;Temporary shift to point to point mode (G00) when reaching
                                    position 25
      N18 (DIS,"MILL DIA. 28")     ;for spindle stop, tool change and S functions
      N19 T5.5 M6 S1200
      N20 X.. Y.. M13
      N21 Z-..
      N22 G1 X.. Y..               ;Continuous mode restarts
```

If G29 were programmed in block N17, continuous mode would stop when movement finishes (Z=5) and subsequent moves in G1-G2-G3 would be performed in point-to-point mode.

Program 2 (point-to-point mode):

```
      N9 (DIS, "MILL DIA. 16")
      N10 T4.4 M6 S800
1     N11 G29 G X-235 Y-230 M13     ;Point-to-point operation starts
      N12 Z-10
2     N13 G1 X75 F500 M5            ;Spindle stop
3     N14 Y S1200 M13               ;Spindle CW with coolant
4     N15 G3 X-70.477 Y25.651 I J
      N16 DWT=2
5     N17 G1 G4 X-187 Y-295         ;Dwell at the end of the element
      N18 G Z5 M5
      N19 (DIS,"MILL DIA. 28")
      N20 T5.5 M6 S1200
      N21 G X.. Y.. M13
      N22 Z-..
      N23 G1 X.. Y..
```

By programming point-to-point with G29 in block N11, M and S functions have been included in the profile (blocks N13 and N14). The dwell at the end of the element (block N17), however, can also be programmed in continuous mode.

3.2.4 *Automatic deceleration on bevels in G27 mode*

When G27 mode is active, the control automatically calculates the vector velocity on the bevels (i.e. between two subsequent moves) using a two-step algorithm.
During the first step the vector velocity is calculated with a formula based on profile variations. The variation of the profile is associated with the angle formed by two subsequent moves.
The control compares the actual angle with the MDA value; if the angle is greater, the vector velocity is put to zero as in G29 mode; otherwise, the control calculates a velocity for this bevel that is based on the angle, MDA variable and the maximum velocity step configured on the axes.
The second step of the algorithm, called "***look ahead****",* is optional. It can be enabled or disabled according to the value of the DLA variable.

The "***look ahead***" step is an optimisation of the first step. In fact, in order to provide a correct stop at the profile end, the calculated vector velocity is re-processed taking into account the total distance to be covered in G27 mode and the acceleration configured for each axis.

The ***look ahead*** feature (G1 G27) does not handle feed rate override. In fact, at this stage a 100% feed rate is assumed. Higher feed rates may generate SERVO ERRORS.

3.2.5 *DLA – Deceleration Look Ahead*

The DLA code enables/disables look ahead calculation in G27 dynamic mode. The control reads the motion blocks that m ake up the profile and those that follow the block in execution in order to recalculate the exit feed rate for the various blocks. It also calculates the deceleration on the bevels according to the profile. If the profile includes sudden trajectory variations and there are not enough block lengths to ensure appropriate deceleration, it is critical for the system to anticipate these events so that velocities can be adjusted. The number of motion blocks the system can look ahead after the current block can be specified in AMP characterisation. It ranges from 2 to 255 blocks.

Syntax

DLA=*value*

---

where:

value	can be:	**0**	disables look ahead
		1	enables look ahead

NOTE:
If DLA=1 system block time increases because the control must execute a greater number of calculations for each instruction. This results in greater accuracy.

It is advisable to set DLA=0 when it is clear that the programmed feed rate and the total distance to be covered in continuous mode are such as to provide a good stop at the end of the profile.

With DLA=0 the control will consider only the deviations from the theoretical profile on bevels.

Characteristics:
The default value of this variable is configurable in ODM.

3.2.6 *CRV – Feed optimization parameter*

CRV variable is used to optimise feed value for curves described using a sequence of small lines or when noise affects smooth trajectories.

Syntax:

CRV = *value*

---

where:

value Is a positive real number. It fixes the time (in ms) used for curve computation. If 0 no optimisation will be performed.

Characteristics:

The numerical control computes feed on profile depending on the angle between elements and it does not take into account the length of the elements.

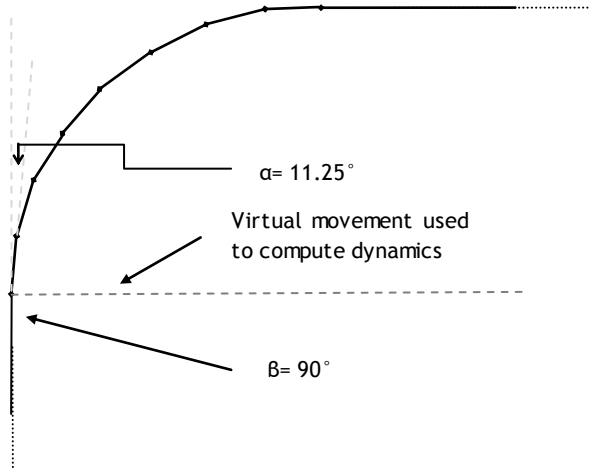
Sometimes, when a part program is generated by points, curves are described by a relevant number of small segments with small angles between them. In this case computed speed is high and the segments are travelled in a very small time, or can even be skipped; as a consequence the final profile could be inaccurate.

When CRV variable is set, the control takes into account the length of segments and their shared angles during feed computation, in this way it can control the feed value on curves. To define the space used for curve computation it is compulsory to define an "observation" time. Defining an "observation" makes it possible to take into account ratios between speed and the lengths of the element defining the trajectory. In this way it is possible to control both curves made up by very short segments travelled at reduced feed and curves made up by longer segments travelled at high feed.

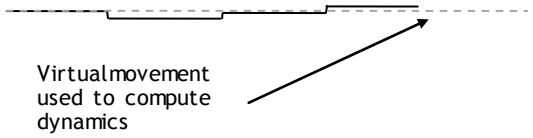
Variable CRV can also be used to remove hard braking where "calculation noise" appears on smooth profiles. Usually a value for CRV between 4.0 and 6.0 assures good results.

Higher values could generate problems of missing elements, since many elements would need to be analysed before a movement could start.

This function is available in continuous mode (G27 / G28), only.
Default value for CRV is configured in AMP.



This profile shows the case where a curve is described using multiple segments. The amplitude of the angle they share is 11.25°, while the amplitude of the global angle is 90°. If optimization is not enabled (CRV =0), computed feed could be too high, since the numerical control could not properly identify the profile. When optimization is enabled the control identifies the curve and it can subsequently reduce speed.



This profile shows the other use of the feed optimization function. In this case the desired profile is linear, but generated points are affected by “noise” due to computation errors.

When optimization is not enabled the control reduces feed when it finds the “steps” in the profile. If the function is enabled feed is not reduced.

3.2.7 *MDA – Maximum Deceleration Angle*

The MDA code defines the maximum angular axis departure in which G27 is active. The selected value (from 0 to 180 degrees) defines an angle that is the limit of G27 operation.

Syntax

MDA=*value*

---

where:
value is an angle between 0° and 180°

Characteristics:
In order to alter the default value (MDA = 90 degrees) you may assign MDA in the configuration or enter it through a specific data entry or a part program block.

The system forces the axis to decelerate to zero velocity when the direction is greater than the angle defined by the MDA value. The system calculates a deceleration ramp for the programmed axis if the direction is less than or equal to the angle defined by the MDA value. Since the system calculates deceleration on bevels from the actual angle and the MDA value, it is possible to alter velocity reduction by changing the MDA value and the maximum velocity step configured on the axes. Small values of MDA generate dramatic deceleration on bevels. The system RESET restores the configured MDA value.

Examples:
MDA=90°
MDA=180°

3.2.8 *MOM – Manual Operating Modality*

MOM variable is used to configure the modality for the execution of manual movements (JOG). Different operating modes correspond to different modes of axis control during manual movements.

Syntax

MOM = *value*

where:

value is an integer that identifies the mode for movement execution. It can be programmed directly with a decimal number or indirectly with E parameters. The available modes are defined in the following table:

Mode	Description
0	In this mode the resulting movement depends whether one or multiple axes move; in both cases axes can move in their operating range (if activated) For manual movements concerning a *single real axis* (meaning a non-virtual axis or tool direction axis), the resulting movement is set into the range of the operating limits, meaning that the limits are included in movement parameters and they can be checked before the actual movement starts. The axis, when and if it reaches the limit, stops in a controlled way, stops using its value for deceleration in manual. This mode is activated after the axis has been selected using the human interface (e.g. WinNBI). For manual movements concerning *multiple real axes* or for movements along a virtual axis and/or tool direction axis, the resulting movement has no limit for axes movement, a specific look-ahead algorithm checks if the axis exceeds the operating limits. When the algorithm estimates that the limit will be exceeded the system generates an emergency and each axis stops using its rapid deceleration. this implies that if a virtual axis (for instance W axis in tool direction) was moving, the corresponding stop cannot happen along the (virtual) axis direction, since each axis stops using its deceleration value, so the original trajectory can't be respected When the operating limits are reached the axis uses "S" ramps and ignores the value of the JRK variable; as a consequence the accelerations can exceed, during the central phase of stopping, rapid acceleration value. The mean acceleration for stopping is, however, equal to rapid acceleration.
1	For a single real axis operation is the same as mode 0. In all the other cases, movement involving more than one real axis or along a virtual axis and/or tool direction, the system uses new algorithms to forecast and to brake when reaching operating limits. When there is a move along the tool direction axis or along UPR virtual axes, a forecast of when the operating limits will be reached is made considering the direction of movement along the virtual axes in use.; as soon as the forecast position is reached, controlled deceleration is made along the virtual axes using the virtual axes manual acceleration. In that case, the original trajectory defined by the virtual axes is maintained.., . Axes are stopped using a controlled stop (using manual acceleration) on the axis that

	reaches its limit and not by generating an emergency. All other axes can continue in their movement. For example, moving simultaneously the UPR virtualization of a P axis and the A axis, reaching the P operational limit (i.e. the limits of the XYZ real axes) will cause only the P axis to stop (XYZ) and not A axes. If TCP is active, operational limit handling (axes or tool direction) is checked in relation to the axes position. It is possible to activate the control for the tool tip as well; this activation can be made axis by axis. In the TPC configuration (L367) "dynamic" table, the user has to select the variable "operational limits mode" to execute the aforementioned operation. Bits 1,2 and 3 (starting from 0 bit) are used to activate operational limit control referred to the tool tip for the first, second and the third TCP axis. For example, allocating variable L367 with the value 7, the real-time operational limit control will be activated. This activation refers to both tool tip and physical positions of the X and Y axes, keeping active only that referred to the physical position for Z. The algorithm for forecasting when operational limits will be reached for each physical axis is kept active in order to check the limits when the axis move is performed by different components (for example, in the TCP case, the compensation made by the movement of a rotary axis is added to the compensation of the linear axis). Algorithm behaviour is modified in order to: Stop only the moves leading the axis movement to the operational limits; for example, in the TCP case, if the limit is on the linear X axis, the linear axis itself and the rotary axes are blocked as leading to a move of the X axis, but not to a move of Y and Z axes. Similarly, in the UPR case, only the virtual axis that moves the X axis is blocked, it means P in the UPR,0,XYZ,PQD,0,0,0 or all the virtual axes for UPR,0,XYZ,PQD,10.0,10.0,10.0. In the TCP, 5 case, the complete movement along W will be stopped when the operational limit for one of the linear TCP axes is reached. Stop with no emergency; axes selected for the manual movement (to be stopped as described above) will stop with their rapid acceleration. If TCP 5 is active, in this mode the tool axis direction (W axis) can be moved simultaneously with other axes, no matter if related or not to the TCP. All stopping generated when reaching operating limits uses "S" ramps and ignores the value of the JRK variable; as a consequence the accelerations can exceed, during the central phase of stopping, the value of rapid acceleration. The mean acceleration for the stopping is, however, equal to rapid acceleration.
2	It behaves like mode 1, in this case all axes are stopped when any one of them reaches its operating limits.
16 17 18	Similar to 0, 1, 2 value to which value 16 is added. During the manual movement, this value is used in relation to the presence of UPR, 8. When a manual movement is executed with UPR 8, horizontal rotary axis movement leads to rotation plane recalculation in order to move the virtual axes that matches the head position. At the end of the manual movement the UPR plane at the start is usually restored. At the end of the movement, by using the MOM variable set to 16, 17, 18, the plane remains oriented in relation to the rotary axes position.
32 33 34	Similar to 0, 1, 2 value to which value 32 is added. During the manual movement, this value is used in relation to the presence of UPR, 8. When a manual movement is executed with UPR 8, PQ axes (the first two virtual axes activated by the UPR) are moved with the dynamic characteristics of

	the two real axes (XY); for example If the characteristics of Y are less than those of X then the Y parameters will be used.. With the MOM variable set to 32,33 or 34, axes will move using the lower dynamic features but not the lower velocity; P and Q axes will move at the same velocity as Q and Y axes.

3.2.9 *MOA – Motion Auxiliary*

MOA variable is used to activate some options related to axis movement.

Syntax

MOA = *value*

where:

value Can be 0 or 1, it can be programmed directly or indirectly with E parameters. It enables or disables the handling of null movements on axes.

Characteristics:
Allows the dynamic filtering of small movements programmed on the axis, in order to avoid decelerations guaranteeing to reach the programmed position. If MOA is active, the feed rate is not reduced even if (according to the MDA values and the maximum jump in feed on the edge) required in case of angular deviation of two adjoining movements.

The performance does not apply between two movements having a continuous change of the rotary planes (lowercase upr).

3.2.10 *ERF – Error Form*

ERF variable checks the value of shape error during the execution of a profile while a floating average on the command is running.

Syntax

ERF = *value*

where:

value is a positive real number that can be programmed directly or indirectly with E parameters. It defines the maximum shape error with respect to the programmed profile.

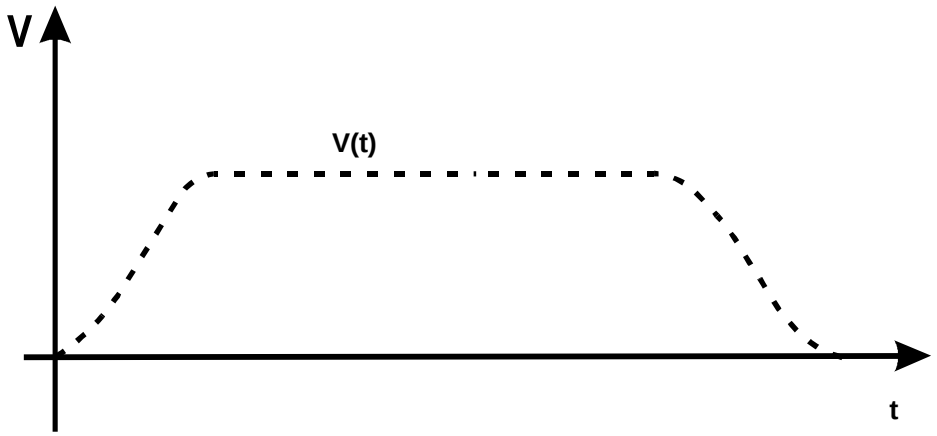
Characteristics:
If the points sent to the axis are filtered (FLT,1) this introduces an error on the profile: the floating average, computed to smooth the path, applied to the computed points produces a distortion to the path geometry. This distortion can be controlled using the ERF variable, that modulates the feed (mainly on corners) in order to limit the shape error, generated by the floating average algorithm, under the requested tolerance.

3.2.11 *Jerk Limitation*

The speed diagrams shown in the previous sections show the continuity of the speed function **V(t)**, while the acceleration function **a(t)** has a step pattern. Depending on the characteristics of the machine and the type of machining process, this may cause defects in the finish of the part.

This problem may be solved using an acceleration function **a(t)** with a continuous pattern.

The purpose of the "Jerk Limitation" function is to limit variations in acceleration, so as to control its maximum value, resulting in smoother movement and, consequently, a better surface finish.



3.2.12 *RAMP – Movement mode*

The RAMP code is used to define some characteristics of the movement management.

Syntax:

RAMP = *value*

where:

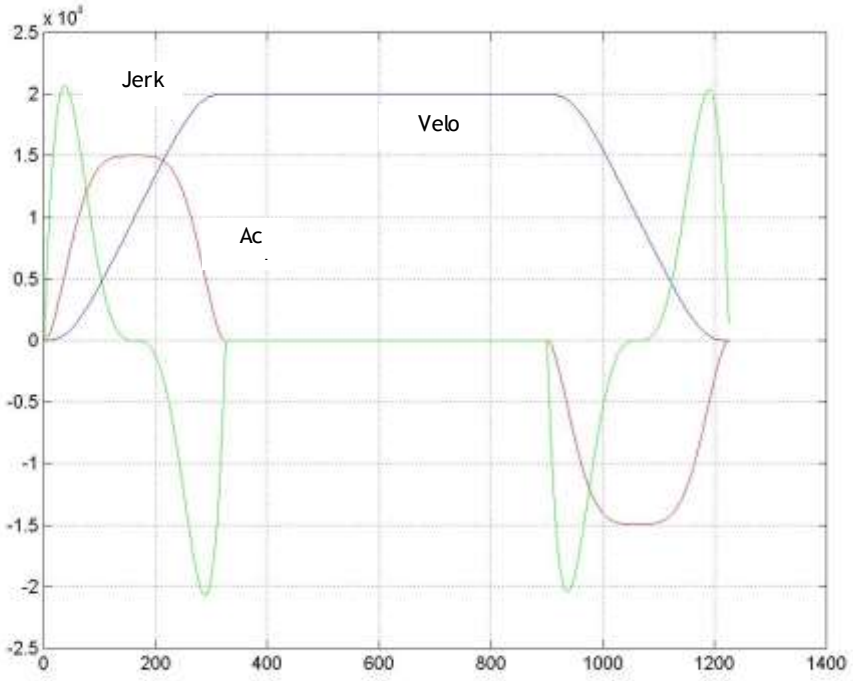
value movement behaviour to be enabled. The value to be specified is obtained from the sum of the values corresponding to each of the features desired.
Uses the ramp configured on the axes
0 Linear ramp
1 S non -linear ramp
2 Trapezoidal ramp
3 Ramp with Jerk limitation

Characteristics:
The RAMP value can be configured in AMP.
RESET restores default value.

Non-linear ramps work both on point to point movements (G29) and continuous movements (G27 and G28). The activation or deactivation of non-linear ramps in a continuous movement is not perceived, it is only perceived in point by point movement mode.

It is recommended to use value 4 for machines with high dynamic levels (high jerks), while value 3 should be used in machines where dynamic levels have to be reduced (low jerks). Configured jerk being the same, value 3 performs better (in terms of execution times) than value 4, but engenders more "abrupt" movements.

S non-linear ramp:
Enables the non-linear ramps (**S ramps**) where the execution mode is characterised by the value of the JRK variable. The system continuously modulates the acceleration value between 0 and a maximum whose value is given by A/JRK, so that, with JRK=1 the nominal acceleration value is reached, with JRK=2 half the nominal acceleration value is reached, with JRK=0.5 twice the nominal acceleration value is reached. The ensuing jerk values on the movement will NOT be controlled and will range from 0 to an undetermined value.



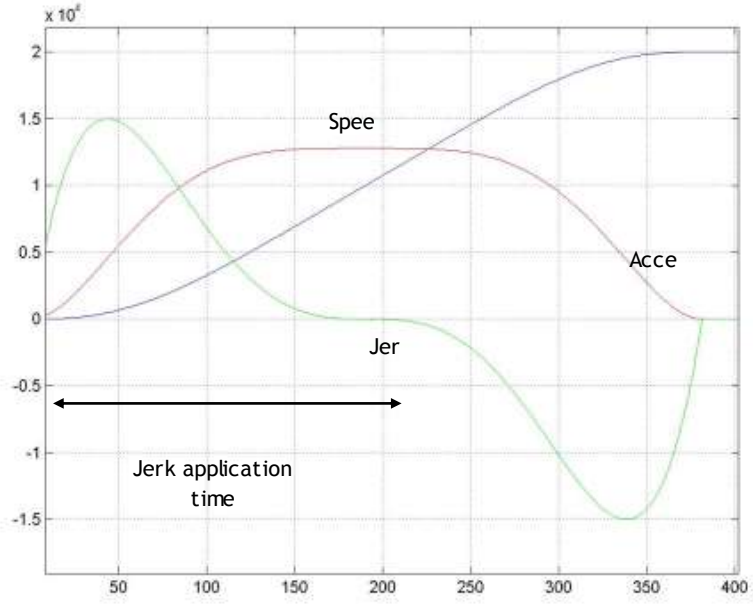
In the example above, a movement at a speed of 20000 mm/min with an acceleration of 1500 mm/sec^2 has been programmed. The maximum acceleration value is reached only at a time halfway between the acceleration stage and the deceleration stage. For JRK=1 the acceleration value reached corresponds to the value requested.

The execution time of the acceleration stage corresponds to the time it would have taken using a linear acceleration ramp, multiplied by the value of JRK and constant 1.485.

$$T_{ramp} = T_{linear\ ramp} * JRK * 1.485.$$

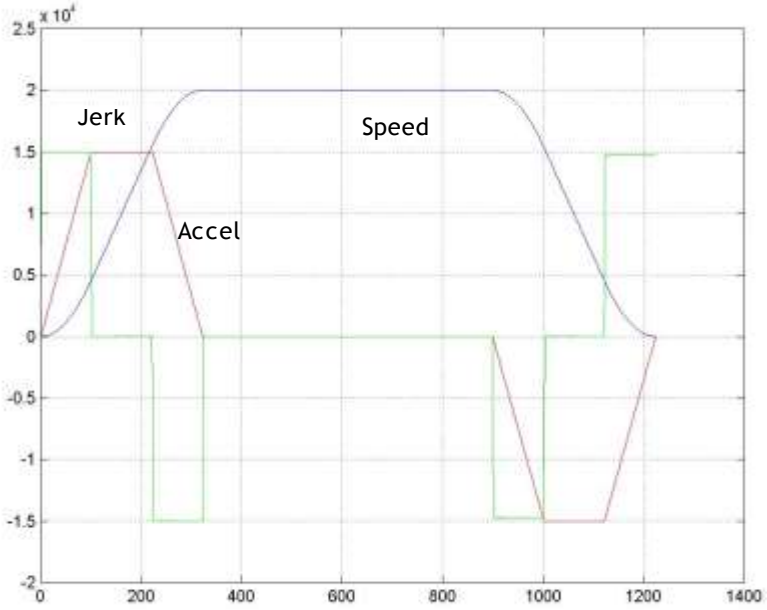
Ramp time is also affected by the “minimum ramp time” parameter configured in AMP for each axis. If the time calculated for the ramp does not meet the minimum ramp time limit, the ramp will be recalculated to comply with this requirement.

N.B. The minimum ramp time is time it takes to reach the maximum acceleration value (Jerk application time) so that, as can be seen from the chart below, the total ramp time corresponds to twice the minimum time.



Trapezoidal ramp

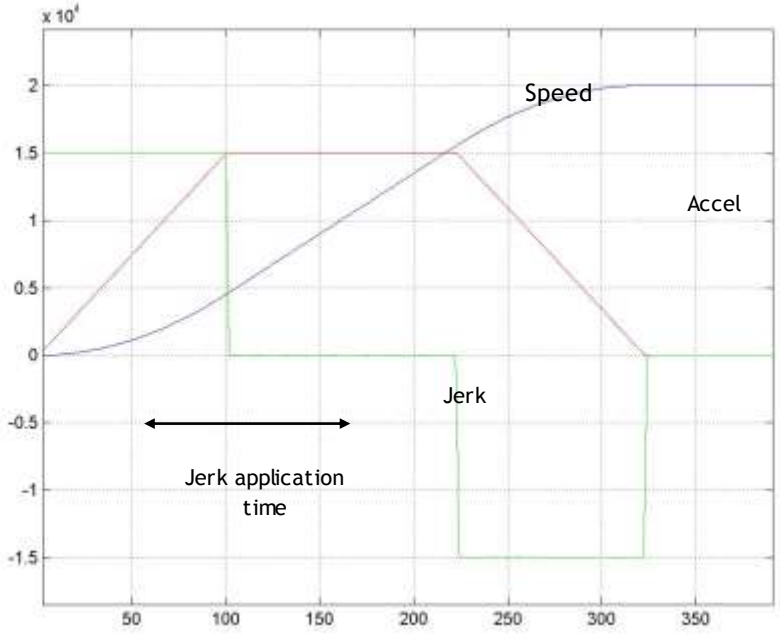
Acceleration ramps are calculated on the basis of the jerk parameters configured in AMP. The jerk value used within a movement is calculated by the system so as to ensure that the movement complies with the jerk characteristics specified for each axis. The jerk value configured for each axis must be construed as the value that can be used during the acceleration stage. The system keeps the jerk value continuously during the acceleration stage.



In the example above, a movement at a speed of 20000 mm/min with an acceleration of 1500 mm/sec^2 and with jerk of 15000 mm/sec^3 has been programmed. The maximum acceleration value is reached and maintained for a certain time in the central part of the acceleration or deceleration stage; it may or may not reach the maximum value requested as a function of speed step to be made.

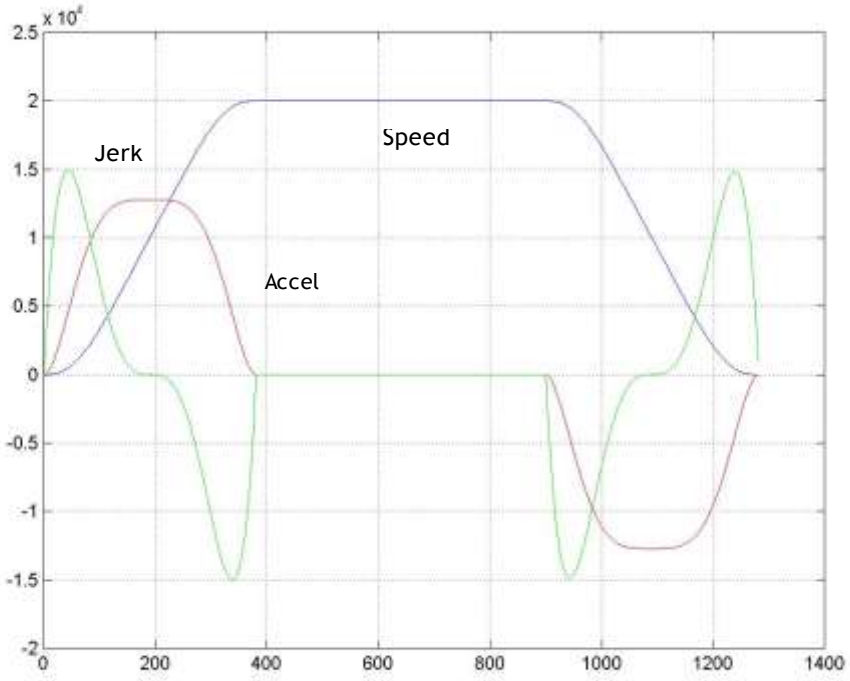
Ramp time is affected by the jerk parameter and also by the "minimum ramp time" parameter configured in AMP for each axis. If the time calculated for the ramp does not meet the minimum ramp time limit, the ramp will be recalculated to comply with this requirement.

N.B. The minimum ramp time is time it takes to reach the maximum acceleration value (Jerk application time).



Ramp with jerk limitation

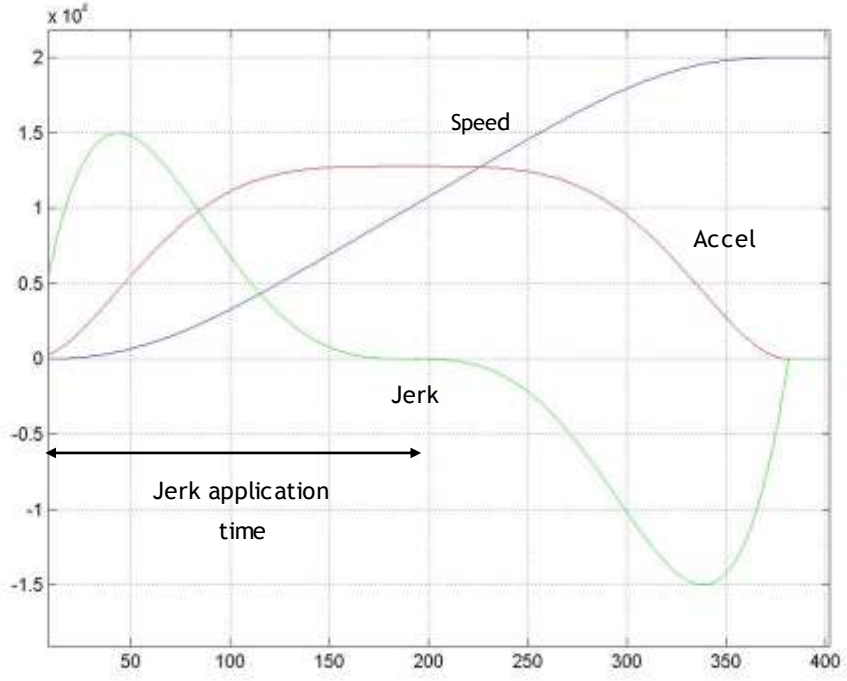
Acceleration ramps are calculated on the basis of the jerk parameters configured in AMP. The jerk value used within a movement is calculated by the system so as to ensure that the movement complies with the jerk characteristics specified for each axis. The jerk value configured for each axis must be construed as the maximum value that can be reached during the acceleration stage. The system modulates the jerk value continuously between 0 and its maximum value, and it will modulate in the same manner the acceleration value, which will not necessarily reach the maximum value configured.



In the example above, a movement at a speed of 20000 mm/min with an acceleration of 1500 mm/sec^2 and with jerk of 15000 mm/sec^3 has been programmed. The maximum acceleration value is reached only at a time halfway between the acceleration stage or the deceleration stage and it is below the maximum value that is requested. Only with a higher jerk will it be possible to reach the maximum acceleration.

Ramp time is affected by the jerk parameter and by the “minimum ramp time” parameter configured in AMP for each axis. If the time calculated for the ramp does not meet the minimum ramp time limit, the ramp will be recalculated to comply with this requirement.

N.B. The minimum ramp time is time it takes to reach the maximum acceleration value (Jerk application time) so that, as can be seen from the chart below, the total ramp time corresponds to twice the minimum time.



3.2.13 *JRK Jerk Constant*

The JRK code defines the acceleration management mode during the execution of movements with S ramps (RAMP=1).

Syntax:

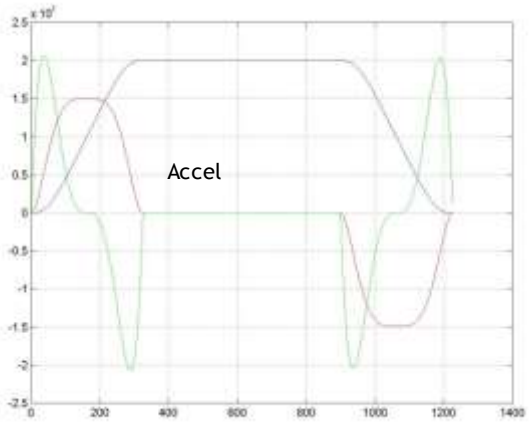
JRK = *value*

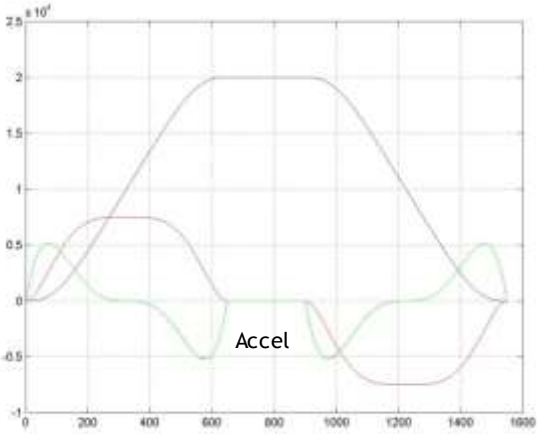
where:

value a numeric value greater than 0.5, which is used to define the acceleration management mode in a non-linear ramp

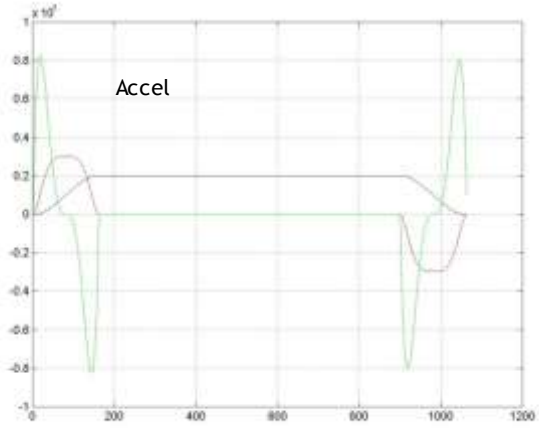
Characteristics:

The default value is 1. The JRK value can be configured in AMP. RESET restores the default value. By setting JRK = 1, the acceleration ramp retains the values configured for axis accelerations. Acceleration increases with a value lower than 1, and decreases with a value higher than 1.





If an acceleration of 1500 mm/sec² is necessary in both movements. In the first case JRK=1 has been used so that the nominal acceleration value is reached, in the second case JRK=2 has been used so that half the nominal acceleration value is reached. In the next example JRK=0.5 has been used so that twice the nominal acceleration value is reached.



3.2.14 *FEEDROT – Feed rate programmed on linear axes only*

This system variable enables the programmed feed rate to be applied for linear axes only. It is the equivalent of bit 6 in the Series 10 MOV variable.

Syntax:

FEEDROT = *value*

---

where:

value **0:** Feed rate programmed on all movement axes

1: Feed rate programmed on linear axes, only

Characteristics:

If this flag is set to 1, the feed rate programmed in a movement block applies to the linear axes. If a rotary axis is programmed in addition to the linear axes, feed rate is recalculated automatically so as to ensure that it remains the same on the linear axes. The newly calculated speed is applied only if a movement includes both linear and rotary axes. It is not applied to the circular motions (G2/G3/G12/G13/G14), in which a rotary axis is included in the interpolation plan.

3.2.15 *RAPSTOP – Rapid stop activation*

Enables the rapid stop mode in case of RESTE, HOLD, FeedHold, CycleStop, Override=0. It's like the bit 1 of the MOV variable in OSAI 10 Series controls.

Syntax

RAPSTOP = *value*

---

where:

value: **0**: rapid stop disabled

1: rapid stop enabled

Characteristics:

If the rapid stop is active, movement is always blocked using S ramps considering only the acceleration value on the profile. If there are any Jerk or minimum ramp times, they are not taken into account

3.2.16 *ODH – Online Debug Help*

This system variable is used to check whether the part program includes continuous moves (G28) in which the velocities programmed on the profile are too high and may generate path errors.

Syntax

ODH = ***value***

Characteristics:
To obtain reliable results from the use of ODH, the variable itself must be reset before it is used in the part program: that is, ODH must be forced to zero before the blocks to be tested. This variable is normally used when the Part Program is debugged: the system modifies the ODH variable when it detects critical conditions.

This information is coded in bits with the following meanings:

Bit 0: Indicates that the queue of the elements processed continuously is too short. It occurs when, in G28 (and in some cases in G27), the machining speed is too high with respect to the value set in AMP as the number of blocks to be processed during continuous motion. The possible solutions are to decrease the speed or increase the size of the queue of pre-calculated elements.

Bit 1: Indicates an excessively high speed and occurs when elements in the profile are too short, and are skipped due to the excessively high speed. In this case, the machined profile is deformed. The possible solutions are to reduce the speed, or to recalculate the points in the program.

Bit 3: Occurs when the system has forced deceleration to avoid deforming the profile when it cannot completely calculate the acceleration/deceleration ramps for the movement elements analysed, during machining with non-linear ramps enabled. The possible solutions are to reduce the speed or to increase the size of the queue of pre-calculation elements.

Example:

```
. . . . . .
. . . . . .
ODH=0                                ;the variable is set to zero for velocity verification
G1G28F1000                           ;continuous mode starts
. . . . . .
. . . . . .
. . . . . .
. . . . . .                          ;continuous mode blocks
. . . . . .
. . . . . .
. . . . . .
G29M5                                ;continuous mode end
(GTO,KO,ODH=1)                       ;branches if ODH=1
(DIS;"VEL.WITHIN SYSTEM LIMITS")
(GTO,CONT)
. . . . . .
"KO"
(DIS,"VELOCITY TOO HIGH")
. . . . . .
. . . . . .
"CONT"
```

3.2.17 *MBA – Multi-Block retrace Auxiliary functions*

The MBA (Boolean) variable makes it possible to enable/disable the auxiliary functions (M functions) during Retrace operations:

Syntax:

MBA = *value*

---

where:

value **0** emission of auxiliary functions disabled
1 emission of auxiliary functions enabled

Characteristics:
The emission of auxiliary functions is performed only during forward retrace
(see user manual)

After a reset, the variable is reset with the value configured in AMP.

3.2.18 *REM – Automatic return to profile at end of move*

The REM (Boolean) variable determines the Jog return mode.

If the variable is set to 0 the return point is the point at which the hold stage begins (standard execution).
If the variable is set on 1 the return point is the final point of the movement in which the hold stage begins.

The return movement is performed automatically, without manual movements (JOG).

Syntax:

REM = *value*

Where

value **0** Return to input point in hold
1 Return to end point of movement

RESET does not modify the value of the variable.

3.2.19 *G70 G71 – Measuring units*

The G70 and G71 codes define the measuring unit used by the control.

G70 specifies inch programming
G71 specifies millimetre programming

Syntax

G70 [*G-codes*] [*operands*]
G71 [*G-codes*] [*operands*]

where:

G-codes Other G codes that are compatible with G70 and G71 (See "Compatible G codes" table in Chapter 2).

operands Any operand or code allowed in a G function block.

Characteristics:
If neither G70 nor G71 is programmed, the system will assume the default measuring unit stored in the system configuration (typically G71).

When the system switches from G71 to G70 or from G70 to G71 it also converts all position and feed rate information into the relevant unit.

- Offsets and origins tables are not automatically converted into the alternative unit when the system switches between G71 and G70.
- When the system switches from G71 to G70 or vice versa, the value for machining stock allowance (MSA variable) is set to zero.

3.2.20 *G90 G91 G79 – Absolute, incremental and zero programming*

G90 Sets absolute programming, i.e. moves referred to the current origin (position to move).

G91 Sets programming in the incremental system, i.e. moves referred to the position reached with the previous move.

G79 Sets programming in absolute reference to home position. It is valid only in the block in which it is programmed (distance to move).

Syntax

G90 [*G-codes*] [*operands*]
G91 [*G-codes*] [*operands*]
G79 [*G-codes*] [*operands*]

---

where:

G-codes Other G codes that are compatible with G90, G91 and G79 (See "Compatible G codes" table in Chapter 2).

operands Any operand or code that can be used in G function blocks.

Characteristics:
If none of these codes is programmed, the default programming mode is absolute or G90, i.e. coordinates referred to the programmed origin.

G90 and G91 are modal whereas G79 is not. After programming a block with G79, the control restores the programming mode of the previous block.

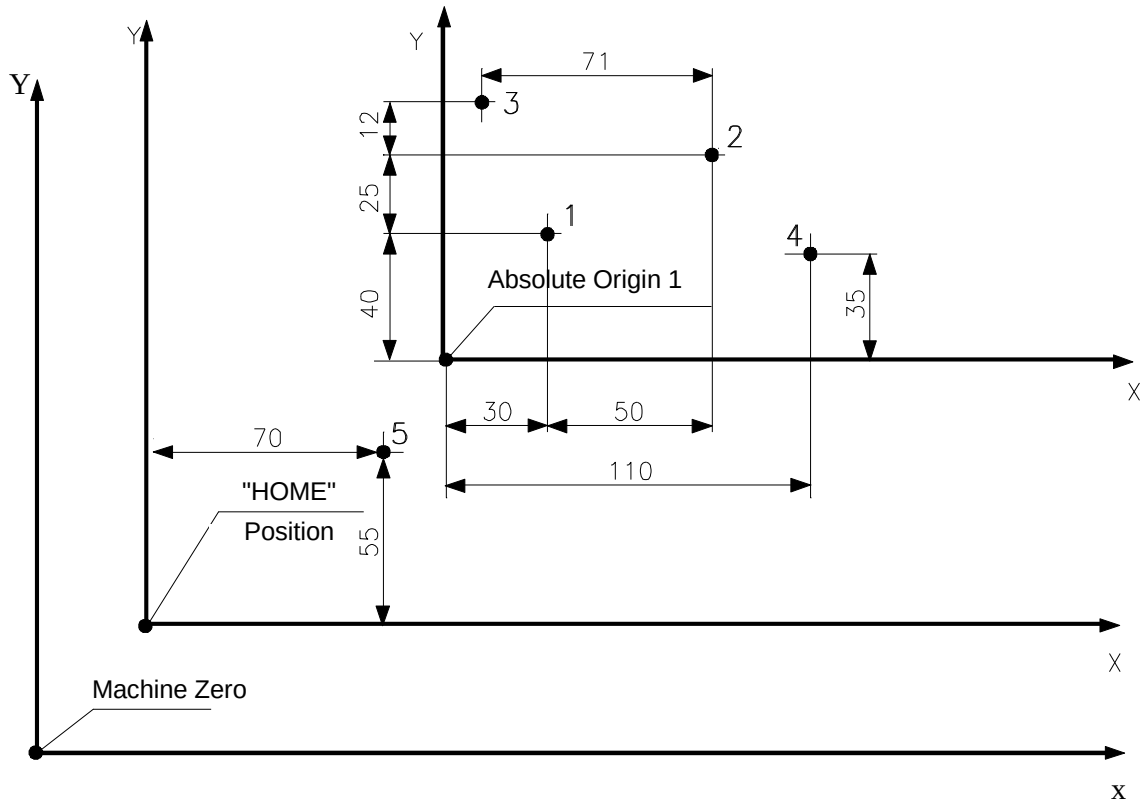
Using characters >> allow mixed incremental/absolute programming in the same block.

Characters >> positioned before the numeric value of an operand, indicate that it must be considered as an incremental value and that it is valid for that operand only. Characters >> have a meaning only if G90 absolute programming is active. They may be used for all operands on which it is possible to use G91 function.

Example:
G90 G1 X + 80 Y >> + 35 Z-70
The value associated with Y is considered as incremental.

Example:

This example shows how to use the different reference systems: absolute, incremental and HOME position.



Program:

(UAO, 1)	;Enables absolute origin 1
N1 G X Y	;X and Y positioned to absolute origin 1 (assuming default mode G90)
N2 X30 Y40	;X and Y positioned to point 1
N3 G91 X50 Y25	;Incremental mode positioning to point 2 (X+50, Y+25 from point 1)
N4 X-71 Y12	;Incremental mode positioning to point 3 (X-71, Y+12 from point 2)
N5 G90 X110 Y35	;Absolute mode positioning to point 4 (X+110, Y+35 from the origin)
N6 G79 X70 Y55	;Positioning referred to home position on point 5 (X+70, Y+55 from home position)

3.2.21 *G92 G98 G99 – Axis pre-setting*

G92 and G98 represent an alternative method for introducing axis offsets. Since G92/G98 define the reference position, they are used in a part program block by themselves. The difference between current and new position is stored in a separate G92/G98 offset register. In this way, other active offsets such as tool offset and origins, will not be destroyed when a G92/G98 offset is introduced. The G99 code resets the G92/G98 code.

Syntax

G92 *axes*
G98 *axes*

---

where:
axes Are fields containing axes name-value. Up to 12 couples axes-value can be specified.

Characteristics:

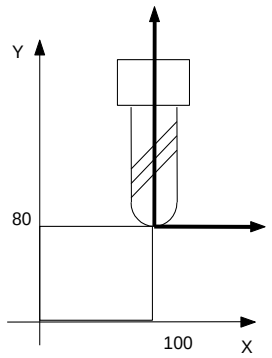
G98 works the same way as G92, except that G92 does not take into account the MIRROR applied to the programmed axes and G98 does.

Codes G92 and G98 are cancelled by the following functions:

- G99
- M30
- System reset
- Machine logic (PLC)

The G92 or G98 offset shifts the origin for a part program but does not cause any axis motion. When the axis value is entered in a G92 or G98 block it becomes the current axis position.

Example:




```
. . . . . .
. . . . . .
G0 X100 Y80
G92 X0 Y0
. . . . . .
. . . . . .
```

3.2.22 *G04 G09 – Dynamic mode attributes*

Two G codes belong to this class:

G04 Dwell at end of block

G09 Deceleration at end of block

Syntax

G04 [*G-codes*] [*operands*]
G09 [*G-codes*] [*operands*]

---

where:

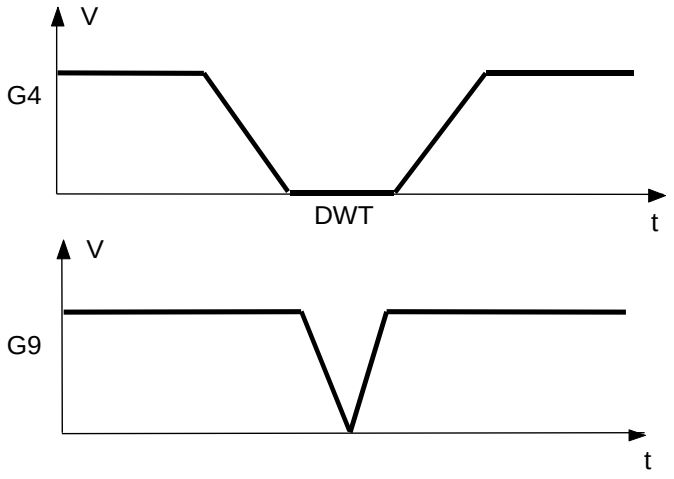
G-codes Other G codes that are compatible with G4, G9 (See "Compatible G codes" table in Ch. 2)

operands Any operand or codes allowed in a G function block.

Characteristics:

G04 causes a dwell at motion end in a block. The DWT command must be programmed in a block that precedes G04. If no DWT is be programmed, the system assumes the characterised dwell.
G04 is allowed only when the control is in point-to-point (G29) and is valid only in the block in which it is programmed. The value set in DWT may be expressed in seconds (G94 or G95 with G0 active) or revolutions (G95).

G09 forces a feed rate equal to zero at the end of the block in which it is programmed, but does not vary the machining status of the profile in progress. If control is in point-to-point mode (G29) or is programmed at the end of a block, G09 does not cause any change in the control status and is valid only in the block in which it is programmed.



3.2.23 *t- Block execution time*

Both in G93 and G94 it is possible to establish the block execution time by programming the **t** function at the end of the block.

Example:
G1 X10 Y1 t6

Characteristics:
The **t** function is valid only for the block in which it is programmed. The time is expressed in seconds and the control automatically calculates the feed rate at which the moves of the axes programmed in the block will be executed.

3.2.24 *DWT –Dwell Time*

The DWT command lets you assign a dwell time at block end. This dwell time is used by G04 and canned cycle blocks.
Although the DWT command can be programmed anywhere in the part program, but must come before any G04 or fixed cycle linked to it.

Syntax

DWT = *value*

where:

value Is a time in seconds or revolutions. It can be programmed directly with a decimal number or indirectly with E parameters.

Example:

DWT = 12.5	assigns a dwell time of 12.5 seconds
E32 = 13.4	assigns the value 13.4 to the E32 variable
DWT = E32	assigns a dwell time of 13.4 seconds

3.2.25 *G93 – V/D Feed rate*

G93 defines the axes feed rate (F) as the inverse time (in minutes)required to execute the entity:

$$\boldsymbol{F} = \frac{1}{T}$$

Linear interpolation:

$$\boldsymbol{F} = \frac{Feedrate}{Distance}$$

Circular interpolation:

$$\boldsymbol{F} = \frac{Feedrate}{Arc}$$

where:

Feed rate linear or circular speed expressed in mmpm (G71) or ipm (G70)

Distance vector distance of linear motion expressed in inches or mm

Arc arc length expressed in inches or mm.

In blocks where G93 is active, specified F value is considered as local and does not affect following blocks.

Example:

```
G93     X. . Y. . F. .
G1
        X. . Y. . F. .
```

3.2.26 *VFF – Velocity Feed Forward*

This command enables/disables VFF. VFF has an effect on the interlocking of the axes allowing the speed to be controlled as well as the position.

Syntax

VFF=*values*

where:
value Can be:

0: disables VFF: (only the position of the axes is controlled, that is taking into consideration the error margin between the tracking and its theoretical position)

1: enables VFF: (the speed of the axes is also controlled). The control is made only for movements in AUTO or MDI

2: similar to mode 1, but only for manual movements

3: similar to mode 1, but for all movements

NOTE:
The VFF default value is 0 and can be configured in AMP using ODM tool.

3.3 CODES THAT MODIFY THE AXES REFERENCE SYSTEM

The commands in this class allows the change of the Cartesian reference system in which the profile is programmed.

COMMAND	FUNCTION
SCF	Scale factor
MIR	Mirror machining
ROT	Active plane rotation
UAO	Absolute origin
UTO	Temporary origin
UIO	Incremental origin
RQO	Origin requalification

When active, the following functions are executed in this sequence:
SCF - G70/G71 - MIR - ROT – MODIFY ORIGINS.

3.3.1 *SCF – Scale Factors*

The SCF command assigns a scale factor to programmed axis dimensions. The control applies the scale factors to the axes specified in the SCF command.

Syntax

(SCF [*axis1value1*, . . . , *axis12value12*]**)**
Or
(SCF [,*value*]**)**

---

where:

value1 . . .*value12* Programs the scale factor. Scale factor can be programmed directly with a decimal number or indirectly with an E parameter.

axis1 . . . *axis12* Addresses of axes configured in the system and scale factor.

Characteristics:
You can specify up to twelve axes in the SCF command. The control cancels scaling for axes that are not specified in the command. A SCF command programmed without a scale factor or axes cancels scaling for all axes.

Example:

.
.
.
(SCF, 3) Applies a 3 scale factor to the programmed dimensions for all configured axes.
.
.
(SCF, X2.5, Y3) Applies a scale factor of 2.5 to the programmed dimensions of the X axis and 3 to the Y axis and deactivates scaling for all other axes.
(SCF) Deactivates scaling for all axes.

The system RESET disables the scale factor for all axes

3.3.2 *MIR – Using Mirror Machining*

The MIR command reverses (mirrors) the programmed direction of motion for the axes you specify in the command referred to the current origin.

Syntax

(MIR [**,** *axis1***, . . . ,** *axis*12])

---

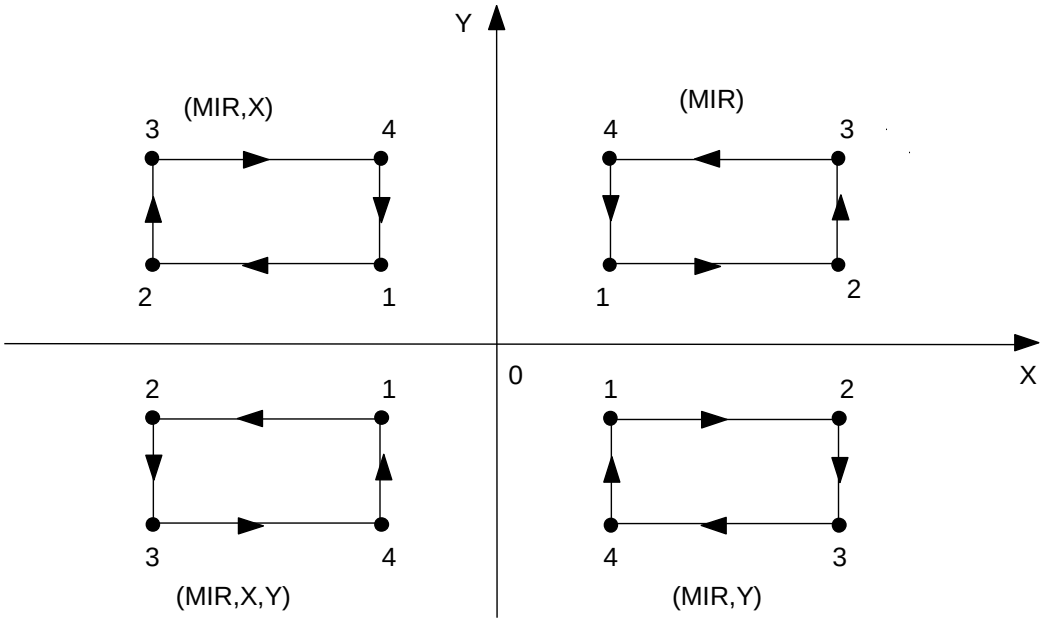
where:
axis1. . .axis12 Is the name of the axis to be moved in mirror.

Characteristics:
You can program up to twelve axes in the MIR command. Program an axis only once per each MIR command. If a given axis is not programmed in MIR, any mirror function for that axis will be turned off.
If no axis is programmed in the MIR command, the mirror function will be reset for all configured axes.
The control mirrors the programmed axis move with respect to the current origin, applying the mirror function from the first motion block including that axis after the MIR command.
If you use plane rotation (ROT) in conjunction with the mirror (MIR) command, the control processes them in the following order: MIR then ROT.

The system RESET disables the MIR command for all axes. It is equivalent to programming (MIR) without parameters.

Example 1:

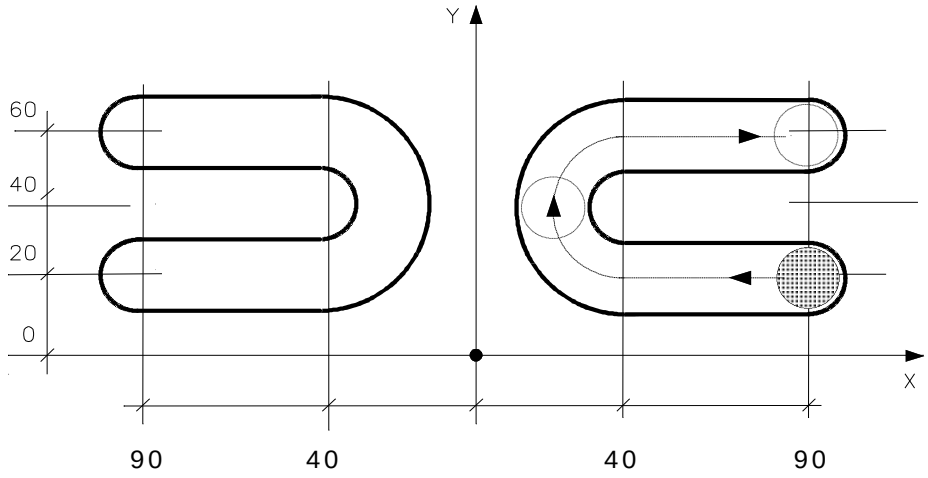
The following example shows how mirror machining works.



.
. Mirroring not active. Moves occurring the first quadrant. Moves referred to the current origin.
.
N24 (MIR,X) Mirroring active for the X axis only. Programmed +X moves generate a move shown in the 2nd quadrant.
.
.
.
N42 (MIR, X, Y) Mirroring active for the X and Y axes. Result of programmed moves
. shown in the 3rd quadrant.
.
.
N84 (MIR, Y) Mirroring active for the Y axis and inactive for the X axis. Moves in the 4th quadrant.
.
.
.
N99 (MIR) Mirroring inactive for all axes. Moves in the 1st quadrant.

Example 2:

This example shows how to use the MIR command. Also note the use of RPT and ERP.



Program:

```
N199 (DIS."MILLING CUTTER D =16")
N200 S1500 T8.8 M6
N201 (RPT, 2)
N202 G X90 Y20 M3
N203 Z-100
N204 G1 Z-105 F150
N205 X40 F200
N206 G2 Y60 I40 J40
N207 G1 X90
N208 G Z-100
N209 (MIR, X)
N210 (ERP)
N211 (MIR)
N212 Z
```

3.3.3 *ROT – Interpolation plane rotation*

ROT is a part program command that rotates the active interpolation plane by a programmed angular value. The centre of rotation is the current origin. ROT can be activated through MDI or as a part program code.

Syntax

(ROT, *angle* [**,** *mode*]**)**

where:

angle Represents the value of an angle expressed in decimal degrees. You can program the angle directly with a numerical value or indirectly with an E parameter. Positive angles are measured CCW from the abscissa of the current interpolation plane. Negative angles are CW. If the angle is zero plane rotation is disabled.

mode If *mode* is set to 0 or not specified the rotation does not interfere with temporary (UTO) and incremental (UIO) origins applied after the rotation. If it is set to 1 the rotation does interfere with temporary (UTO) and incremental (UIO) origins applied after the rotation, meaning that the origin rotates, too.

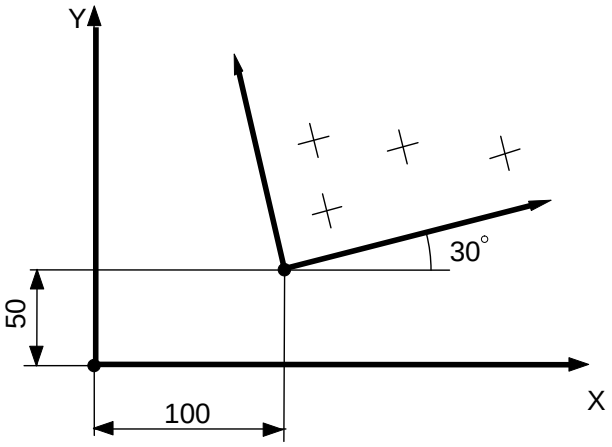
Characteristics:
The control rotates programmed coordinates beginning with the first block after the ROT command. Coordinates referred to the machine zero (G79) are not rotated.
If you use axes rotation (ROT) in conjunction with mirroring (MIR), the control performs them in the following order: MIR first then ROT.

The system RESET disables the plane rotation. It is equivalent to programming (ROT,0).

Example using Mode 0 (or no mode)

(UAO, 1)
(ROT, 30)
(UIO, X100, Y50)
…

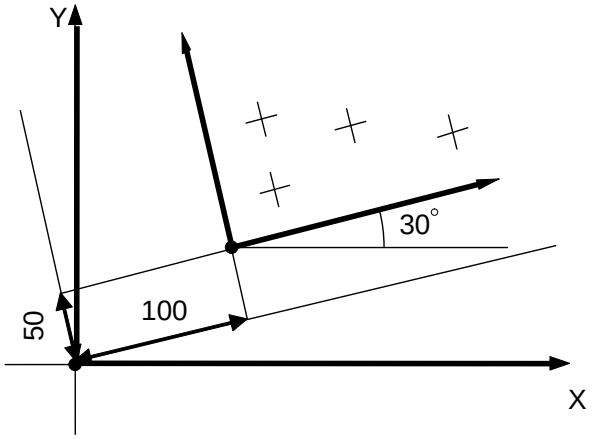
Origin is applied with respect to point 100,50 referred to origin 1, the rotation is applied on this new origin.



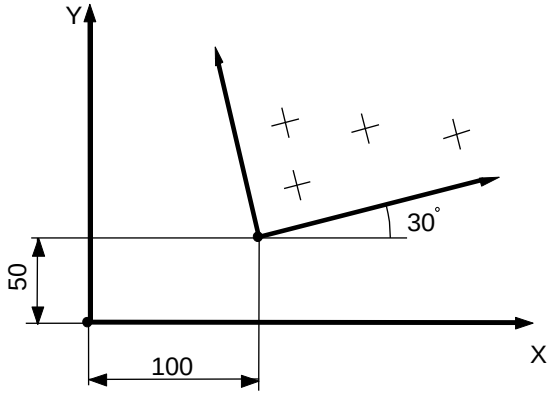
Example using Mode 1

(UAO, 1)
(ROT, 30, 1)
(UIO, X100, Y50)
…

Origin is applied with respect to point 100,50 **rotated with** respect origin 1, the rotation is applied on this new origin .



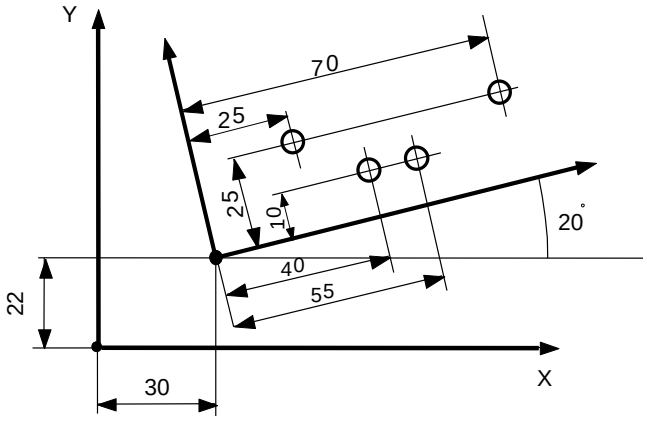
Example 1:



Program:

(UTO, 1, X100, Y50)	Activates absolute origin 1 with temporary origin (absolute offset) X100 and Y50
(ROT, 30)	Specifies 30 degree rotation CCW referred to temporary origin
.	
.	Moves in this portion of the program are referred to the temporary origin and rotated 30 degrees CCW
(UAO, 1)	Reactivates absolute origin 1
(ROT, 0)	Deactivates rotation by specifying a 0 degree rotation around origin 1

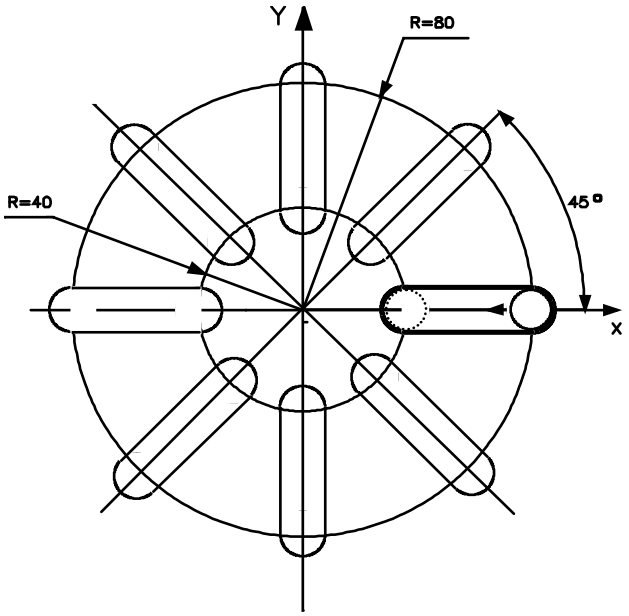
Example 2:



Program:

```
N99 (DIS, "DRILLING D=6")
N100 S2000 F200 T3. 03 M6
N101 (UTO , 1, X30, Y22)
N102 (ROT, 20)
N103 G81 R-90 Z-110 M3
N104 X25 Y25
N105 X40 Y10
N106 X55
N107 X70 Y25
N108 G80 Z
N109 (UAO, 1)
N110 (ROT, 0)
N111 S1000 T4.4 M6
```

Example 3:
This example shows how to use Plane Rotation (ROT) with Repeat (RPT) and Parametric Programming.



Program:

```
N148 (DIS, " ...")
N149 S1500 T5.5 M6
N150 E25 =0
N151 (RPT, 8)
N152 ( ROT, E25)
N153 G X40 Y M3
N154 Z0
N155 G29 G1 Z-10 F150
N156 X80 F200
N157 Z-18 F150
N158 X40
N159 G Z0
N160 E25 =E25 + 45
N161 (ERP)
N162 (ROT, 0)
```

3.3.4 *UAO – Using Absolute Origins*

The UAO command lets you activate and use one of the absolute origins stored in memory.

Syntax

(UAO, *n* [**,***axis1***, . . . ,** *axis12*])

where:

n Defines the absolute origin number. It may be an integer between 0 and 100 or an E parameter. 0 enables the Home Position.

axis1,...,axis12 Is the name of the axis referred to absolute origin *n*.

Characteristics:

The UAO command allows up to twelve axes. Only one absolute origin can be active at a time for a specific axis.
When an axis is not specified in the UAO command, it continues to use its currently active absolute origin. A UAO command programmed without axes (UAO, n) activates origin n for all axes. At power-up, after a control reset, or with n=0 and no axes, all axes are referred to the home position.
If the program requires different origins for different axes, program a separate UAO command for each origin required.

The origin values are automatically converted into and displayed in the unit of the current active mode (G70/G71).
The origins are referred to the HOME position that has been characterised in AMP.

Example:

(UAO,1)	Activates origin 1 for all axes.
.	
.	This portion of the program uses origin 1 for all axes.
.	
(UAO, 2, X, Y)	Activates absolute origin 2 for axes X and Y only.
(UAO, 3, Z)	Activates origin 3 for the Z axis.
.	
.	This portion of the program uses origin 2 for X and Y, origin 3 for Z,
.	and origin 1 for all other axes.
(UAO, 1)	Reactivates origin 1 for all axes.
.	
.	
.	
(UAO, 0)	Reactivates Home position for all axes.

3.3.5 *UTO – Using Temporary Origins*

The UTO command temporarily increments the position of the specified absolute origin by a programmed amount for each declared axis.

Syntax

(UTO, *n* **,***axis1* **[,***axis2* **, . . . ,** *axis12***])**

---

where:

n Defines the number of the absolute origin to be temporarily modified. It is an integer from 0 to 100. If n=0, the programmed offset is added to the home position value.

axis1,...axis12 Is an axis and a dimension. The control treats the dimension as an absolute offset and adds it to the value of the absolute origin for that axis.

Characteristics:
You must declare at least one axis with a dimension in the UTO block but may declare up to twelve axes with dimensions. An axis can only be declared once in each UTO command.

The axis dimension in the UTO command must be programmed in the currently active measuring unit (G70/ G71).

If an axis is omitted from in the UTO block, the current absolute origin for that axis stays active.

Once you activate a temporary origin it stays active until you:

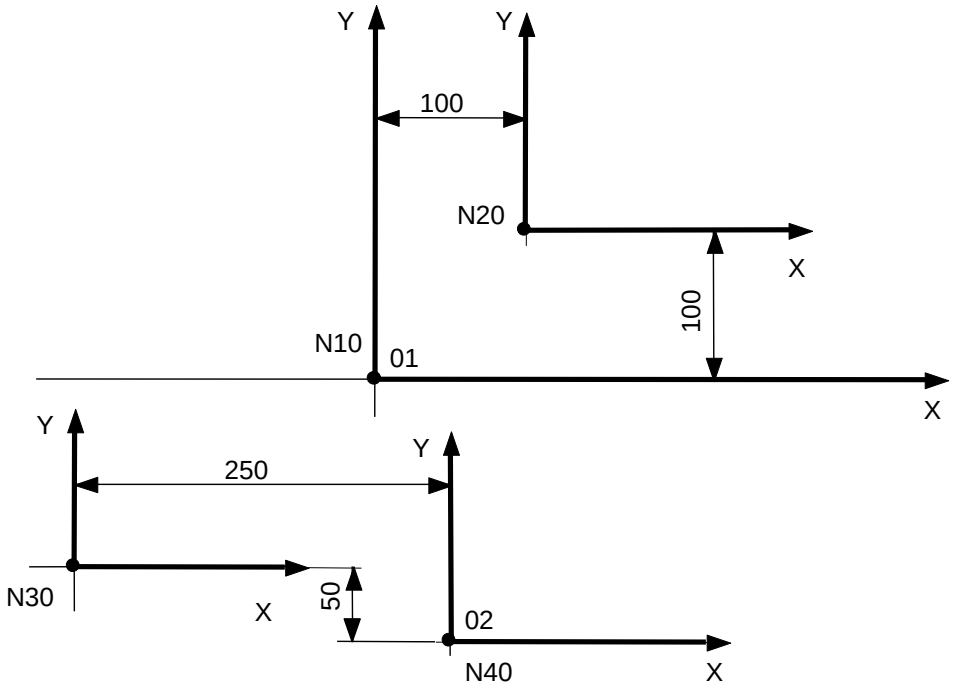
- activate a new temporary origin with the UTO command
- activate a new absolute origin with the UAO command
- perform a control reset.

If a scale factor (SCF) is set, the control applies it to the UTO temporary origin.

Example 1:

Dimensions can be E parameters as shown in the block below:
(UTO,1,XE100)

Example 2:



Program:

N10 (UAO,1)	Activates absolute origin 1 for all axes.
.	
.	In this portion of the program all axes use origin 1.
.	
N20 (UTO, 1, X100, Y100)	Applies a temporary origin (absolute offset) to origin 1, X100 and Y100.
.	
.	In this portion of the program axes use origin 1, with X100 as temporary origin.
N30 (UTO, 2, X-250, Y50)	Activates a temporary origin (absolute offset) to origin 2, X-250 and Y50.
.	
.	This portion of the program uses absolute origin 2, with X- 250
.	and Y50 as temporary origin and origin 1 for the other axes.
N40 (UAO, 2)	Re-establishes absolute origin 2 for all axes.

3.3.6 *UIO – Using Incremental Origins*

The UIO command causes an incremental shift of the currently active origin for each axis specified in the command.

Syntax

(UIO, *axis1* **[,***axis2***, . . . ,** *axis12***])**

---

where:

axis1,...axis12 Is an axis and a dimension. The control treats the dimension as an incremental offset and adds it to the value of the current origin for that axis.

Characteristics:
You must declare at least one axis in the UIO command, but you may declare up to twelve axes.

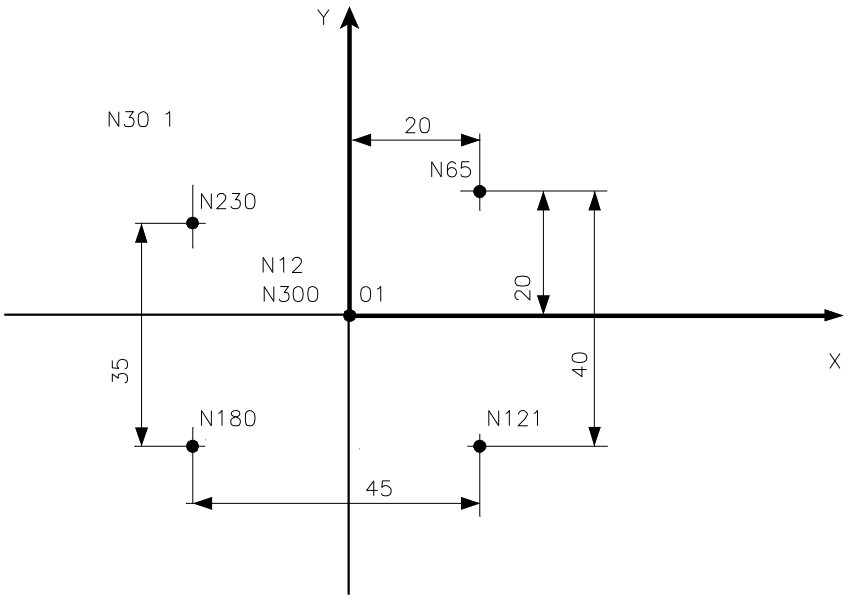
The axis dimension in the UIO command must be programmed with the current measuring unit (G70/ G71).If an axis is omitted from in the UIO block, the current absolute origin for that axis stays active.

Once you activate an incremental origin for an axis it stays active until you:

- activate a new incremental origin for the axis with the UIO command
- activate an absolute origin with the UAO command
- perform a control reset.
- close the plane roto-translation using UPR or UPR, 99 instructions.

If a scale factor (SCF) is set, the control applies it to a UIO incremental origin.

Example:



Program:

N12 (UAO,1)	Activates absolute origin 1 for all axes.
N65 (UIO, X20, Y20)	Applies an incremental offset of X20 and Y20 from origin 1. Absolute origin 1 for other axes remains in effect.
N121 (UIO, Y-40)	Applies a Y-40 increment to the last origin. The X20 incremental origin remains in effect.
N180 (UIO, X-45)	Applies an X-45 increment to the last origin. The Y-40 incremental origin remains in effect.
N230 (UIO, Y35)	Applies a Y35 increment to the last origin. The X-45 increment remains in effect.
N300 (UAO,1)	Restores absolute origin 1 for all axes.

3.3.7 *RQO – Requalifying origins*

The RQO command lets you requalify, i.e. update and modify, an absolute origin from program. The RQO command modifies the specified origin by the specified amount for each axis you declare in the block. The origin must be stored in the origins table.

Syntax

(RQO, *n* **,***axis1* [**,***axis2***, . . . ,** *axis12*])

where:

n Defines the absolute origin number (1 to 100). The absolute origin must have been stored via user interface. n can be programmed directly with a positive integer number, or indirectly with an E parameter.

axis1,...axis12 Is an axis address and a dimension. The dimension is the increment added to the specified absolute origin of the specified axis and can be programmed directly with a decimal number, or indirectly with an E parameter.

Characteristics:

You must specify at least one axis and its dimension in the RQO command. Up to twelve axes with their dimensions can be programmed. Program a specific axis only once for each RQO command.

The dimensions specified in the RQO command must be in the default measuring unit. No scale factor (SCF) must be applied to dimensions programmed in an RQO command.

The origin is requalified both in the origins file (so that the requalification becomes permanent) and in memory (if the origin is active when the requalification is applied).

In the table of the origins the requalification values are applied in the unit of measure in which the selected origin is expressed.

In the case of a diametrical axis, the requalification dimension must be programmed in radial terms, since the dimension is an increment to be added to the value already present in the origin, which is a radial value.

Example:

(RQO, 3, X (E31)) Modifies absolute origin n^{o} 3 for axis X of the value contained in E31.

3.4 OVERTRAVELS AND PROTECTED AREAS

The commands in this class define the over travel limits and the protected areas as described below.

COMMAND	FUNCTION
SOL	Defines software over travels
DPA	Defines protected area
PAE	Enables protected area
PAD	Disable protected area

3.4.1 *SOL – Software over travel*

The SOL command defines axis travel limits measured from the current origin.

Syntax

(SOL,*axis-name*,*lower-limit*,*upper-limit***)**

where:

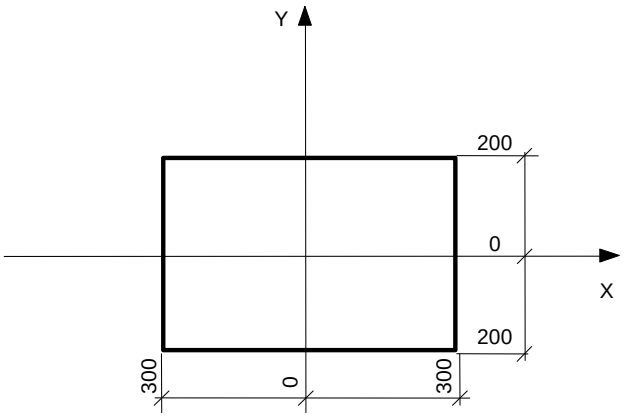
axis-name Is the name of the axis whose travel limits must be defined.

lower-limit Is a dimension for the lower limit.

upper-limit Is a dimension for the upper limit. It must be greater than the lower limit.

Characteristics:
If the programmed software travel ends exceed the limits specified in AMP, the control signals an error. Software over travels must be programmed in the measuring unit (G70/G71) that is active when you program the SOL command. Active scale factors (SCF) are applied to the travel limits.

Example:



Program:
(SOL, X, -300, 300)
.
.
.
(SOL, Y, -200, 200)

3.4.2 *DPA – Define Protected Area*

DPA command defines a protected area, that is to say it defines an area inside which (or outside which) no movement can be programmed. At activation time is possible to choose whether the area extends into the third dimension or not.

Syntax

(DPA, *n*, *name*, *value***)**

---

where:

n identification number for the area (1-4)

name name of the file containing area description.
Name is specified directly using a string of characters, or indirectly using a string variable preceded by the key '**?**'

value specifies if the tool is confined inside the protected area or outside of it.
0: the tool cannot exit from protected area
1: the tool cannot enter the protected area

Characteristics:
Up to four protected areas can be defined either by part program or by the PLC.
Each area is defined with respect to an absolute reference system. At activation time the area will be linked to the current reference system.

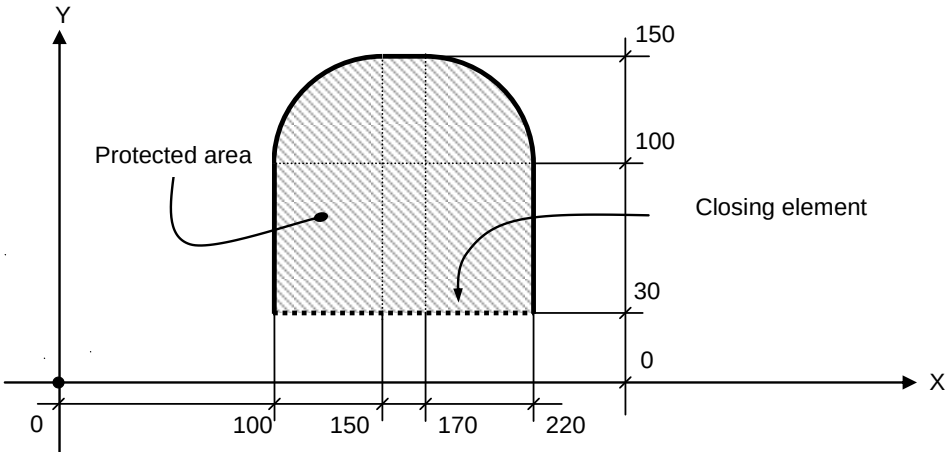
The part program describing the protected area has to respect the following rules:

- The X axis describes the abscissas and the Y axis describes the ordinates. It is not necessary that X and Y axes are defined into the process.
- The first block of the part program defines the starting point of the protected area. This block should contain both abscissa and ordinate of the point. No G2 or G3 code is allowed in this block.
- G0, G1, G2 and G3 are the G codes allowed in the part program. No other G codes can be used
- No three-literal or PLC commands are allowed
- If no G code has been specified, then all blocks are assumed to describe linear elements

The surface describing the protected area has to:

- be convex
- be closed. If the surface described in the part program is not closed, the control will add a movement between the first and last point to close it
- be defined by no more than 10 circular or linear elements. When the described profile is open the number of allowed elements is reduced to 9, since the control will add a tenth "closing" element

Example:



Program:

```
....
N10(DPA,1,PROGRAMS\PUNCH.TXT,1)
....
```

PUNCH.TXT

```
N1 X100Y30
N2 Y100
N3 G3X150Y150R50
N4 G1X170
N5 G3X220Y100R50
N6 Y30
```

3.4.3 *PAE – Protected Area Enable*

PAE command activates checks for violation of a protected area.
When check of a protected area is enabled the control verifies, before starting a programmed linear or circular movement, that it will not interfere with the enable protected areas.

Syntax

(PAE, *n* [,*comp*] [,*origin*] [,*axis3 inf*, *axis3 limit sup*] [,*ax1 transl*] [,*ax2 transl*] [,*ax3 transl*]**)**

where:

n identification number for the area already defined using DPA command (1-4)

comp Specifies if the protected area should take into account current tool radius (or machining allowance):
0: tool radius and machining allowance are ignored
1: protected area should be "adjusted" to take into account tool radius and machining allowance
By default protected areas are not adjusted.

origin Specifies the reference origin of the protected area.
If the parameter is not specified, the control assumes that the area is defined with respect to the current origin.

axis3 inf
axis3 sup Name of the axis used to extend the area in the third dimension and upper and lower bounds of the area. If this pair of parameters is specified then the area has bounds in the third dimension and violation checks are performed in space.
By default the violation check is performed by projecting the movement onto the interpolation plane.

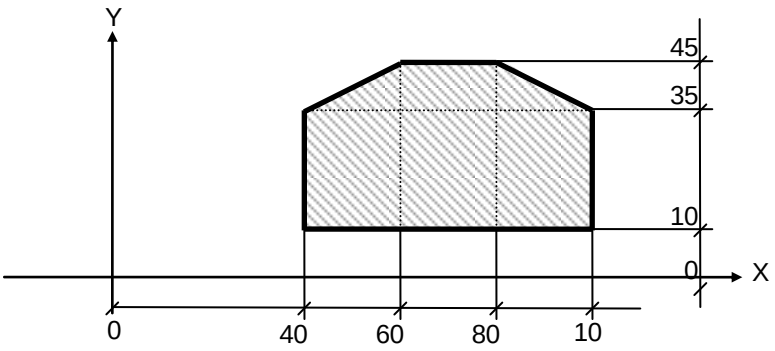
ax1 transl
ax2 transl
ax3 transl Axis name and translation value. It specifies the value of a translation of the protected area. The axes names have to correspond to current abscissa and ordinate and to the axis optionally specified in the 3D extension parameter.

Characteristics:
At activation time the protected area is referred to the active interpolation plane and to the origin and translation optionally specified. For instance if the active plane is YX, then the system will associate the abscissas of the protected area to Y axis and the ordinates to the X axis.

If the parameter *compens* is enabled (set to 1) then the extension of the protected area will change depending on tool radius and/or machining allowance.
If the parameters *axis3 inf* and *axis3 sup* are not specified the system will check for violation of the protected area by projecting every movement on the current interpolation plane.

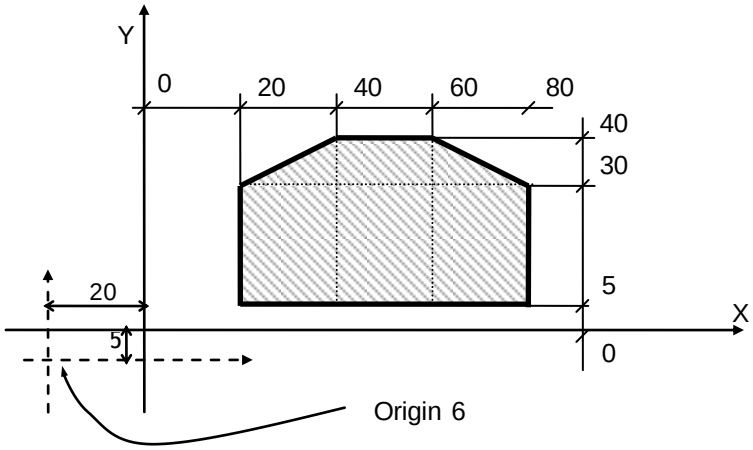
Examples:

(PAE,2,,,C-10,C20) it enables a check for violations on protected area number 2. Tool radius will not be taken into account, area is referred to the current origin, there is no translation of the area. The area is considered as a 3D area, its limits are -10 and 20 along C axis.



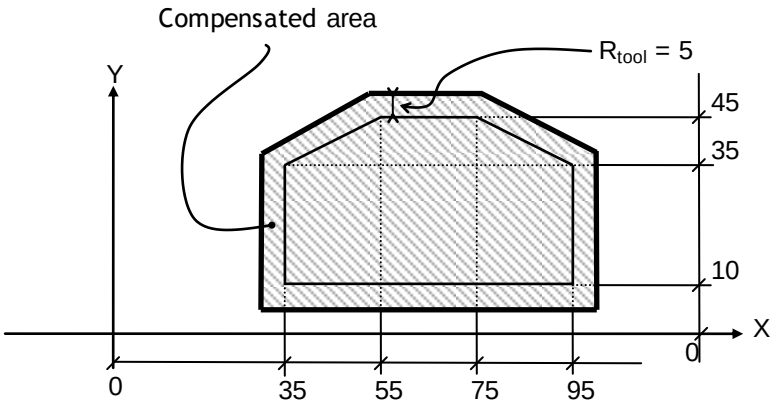

```
AREA.TXT
N1              X40             Y10
N2                              X100
N3                              Y35
N4              X80             Y45
N5                              X60
N6              X40             Y35
N7      Y10
```

(PAE,4,,6) it enables a check for violations of protected area number 4. Tool radius will not be taken into account, area is referred to origin number 6, there is no translation of the area.




```
AREA.TXT
N1              X40             Y10
N2                              X100
N3                              Y35
N4              X80             Y45
N5                              X60
N6              X40             Y35
N7      Y10
```

(PAE,1,1,,,,X-5) it enables a check for violations of protected area number 1. Tool radius will be taken into account, area is referred to current origin, the area is translated by 5 (mm or inches depending on current configuration).



AREA.TXT
N1X40Y10
N2X100
N3Y35
N4X80Y45
N5X60
N6X40Y35
N7Y10

3.4.4 *PAD – Protected area disable*

PAD command disables check for violation on one or all the protected areas.

Syntax

(PAD[**,** *n*]**)**

where:

n Is the identification number of an area already defined by a DPA command and enabled via a PAE command.

Characteristics:
This command disables the check for violation of an active protected area, if no parameter is given all enabled areas are disabled.

The RESET command **does not** disable the enabled protected area. Protected areas have to be disabled explicitly.

Example:
(PAD,3)
This command disables checks on protected area number 3.

3.5 High-speed machining (SPLINES)

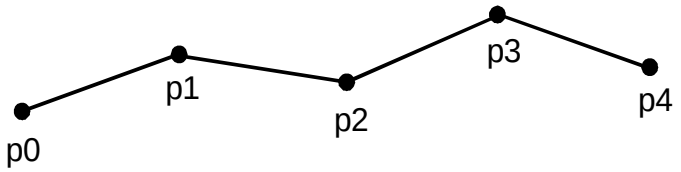
On machine tool having up to 12 axes, High Speed Machining (HSM) allows the machining of surfaces (profiles) defined by points.
In order to use HSM, the AMP "High-speed interpolation" parameter must be set on **yes** in the process configuration that is about to use it.

3.5.1 *Points programming and profile features*

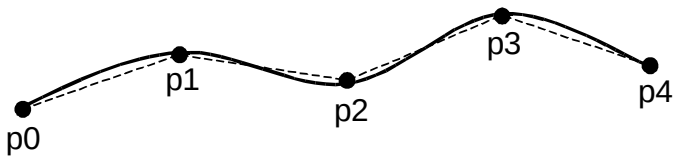
A profile is a set of points defining the ISO part-program of the working surface. It is generated by CAD/CAM, respecting the characteristics mentioned below.
A polynomial curve will be built on the programmed points, describing the trajectory to follow. This trajectory will pass through the programmed points unless there is a tolerance threshold to be set. Codes G0, G1, G10, G2 and G3 can be programmed with the points, defining the points conjunction modes.

The best way to calculate polynomial curves is the points programming using at least 5 numbers after the decimal (ex. 10,37854); if the machining is not precise enough, the profile could result irregular.

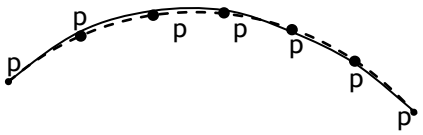
Distances travelled in **G00** will be considered as single positioning; each point will be linked to the following by a "linear movement" having the same dynamic features as the rapid positioning (each G00 line will start and stop at null speed): this is why G00 mode won't calculate the polynomial curve. When a line in G00 finishes, there is no stop (end motion synchronisms, entry in tolerance, etc.), but the following motion will restart immediately. This behaviour is similar to the programming of G01 and G09 codes in the block.



In **G01** and **G10** each point will be linked to the others using a polynomial curve, therefore the trajectory can be considered as **"continuous".** This connection will be interrupted either from a G00 programming or from specific G codes programming described below.

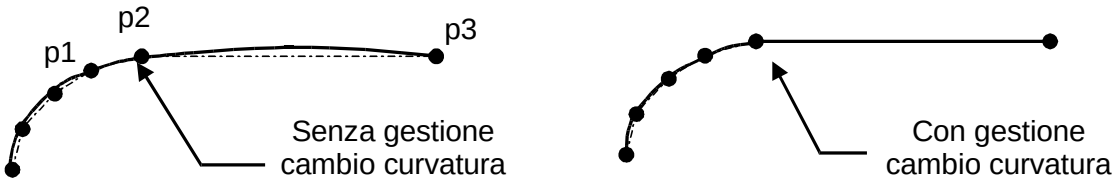


As in G2 or G3 the polynomial curve substitutes the circular interpolation, the trajectory traced has approximately the same shape of a circumference arc. The error introduced is lower than tolerance range programmed for the splines (TBS variables).



3.5.2 *Curve change management*

In high speed machining it's important to identify the curve changes as spline behaviour could be unsatisfactory. For example, let's assume to have a curve followed by a line: the spline on the line could oscillate or create a "hump", therefore is important to identify this motion and reduce it as much as possible.



In the "curve change" point, system will automatically introduce G62, or G63 or it will insert some points within the longer line. The number of points inserted depends from the relationship between the lengths of the long and short lines.
In this context, parameters are managed by **CURLEN**, **CURRAP** and **CURMODE** variables.

3.5.3 *Angles management*

If angles are not identified, oscillations on the splines could be wrong.

The best approach to work with an angle is to stop at feed rate = 0 and then restart on the next line. Sometimes this stop could damage the machining as the tool keeps on removing materials creating some "notches" on the machined part.

Defining how to stop in the angle helps to prevent this problem.

Configuration parameters for this function are: **DAC**, **DAV** and **DPMODE.**

3.5.4 *SPL – Definition of high speed axes*

The SPL command defines on which axes the high speed algorithm will be applied and which axes are involved in the speed calculation on the profile.

Syntax

(**SPL,** *[axesnames1][,axesnames2])*

where:

axesnames1 Is the list of axis names for which the Feed is programmed and where the parameter is calculated. Other axes (rotary and/or additional) are not involved in the feed calculation but follow the profile execution.
axesnames1 is an axesnames2 subset, if not programmed, the parameter will be calculated on the Cartesian axes of the process

axesnames2 Is the list of names for which the Splines will be calculated: it is a subset of the axes configured in the process. The Splines will be calculated on all the axes of the process if not programmed.

Example:
Process 1 configured with XYZABC axes and U

(SPL, XA, XYZAC) The Spline profile will be made by the XYZAC axes where length will be calculated considering only the XA axes

(SPL, , XYAC) The parameter of the XYAC axes Spline will be calculated on XY axes only

(SPL, XY) Z axis will not be involved in the speed calculation of the XYAC axes Spline and U (parameter calculated on XY)

3.5.5 *G61 G60 – High speed machining start and end*

Codes G61 and G60 determines when high speed machining starts and ends.

Syntax

G61
G60

Syntax:

G61 It's the code indicating high speed machining start. All the subsequent motion blocks will be processed by the high speed machining algorithm.

G60 It's the block indicating the end of high speed machining.

Characteristics:

Both codes must be programmed alone in the block.

When high speed machining is active, following operations are not allowed:

- To enable the TCP and tcp transformation (TCP change in continuous)
- To enable the UPR and upr transformation (UPR change in continuous)
- To change the active interpolation plane (codes G16, G17, G18 and G19)
- To execute axes shift (GTA)
- To associate a new name to an axis (DAN)
- To share an axis with the PLC (AXS)

3.5.6 *G62 – G63 – G66 - G67 – Profile interruption*

Codes G62 – G63 – G66 and G67 introduce a profile interruption to divide it in two different curves.

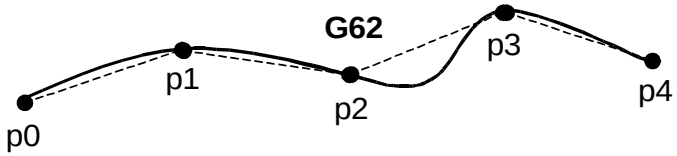
Syntax

G62
G63
G66
G67

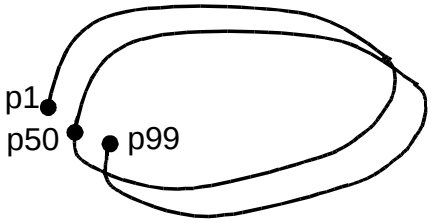
Characteristics:
These G codes are used to divide up the profiles to force the use of two different curves: one is generated using the points defined **before** the G code, the other is generated using the points defined **after** the G code.

G codes have different continuity and different dynamic characteristics on the point of the profile shared by the two curves.
Below are listed the aforementioned behaviours:

- **G62:** the two curves are continuous and connected. The starting inclination of the second curve is equal to the final inclination of the previous one. In G62, no deceleration and acceleration ramp is created to connect the two curves.



G62 can be useful to define "similar" or repetitive curves. Let's assume a 100 point profile where the first 50 points represent the finished profile (from **p1** to **p50**) and the remaining points (from **p50** to **p99**)are the same profile slightly shifted.



As the points between **p1** and **p50** are very similar to the those between p50 and p99, the calculation of the two polynomial curves is similar: therefore, two curves almost similar and "parallel" are created.

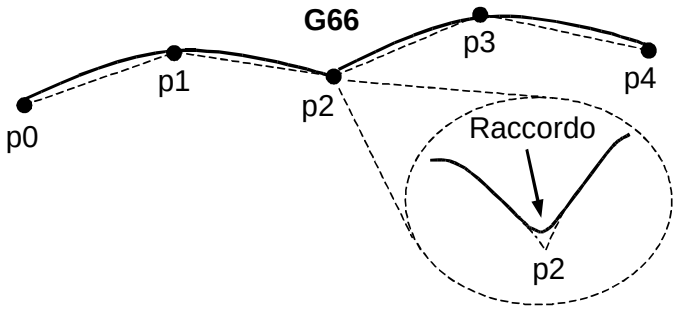
If G62 is not programmed on p50, curves that are not really parallel could be generated due to the fact that the polynomial calculation does not consider the previous "history" of the trajectories calculated.

It is obvious that the "history" of p1 is different from p50: p1, has no history whereas in p50 the the preceding 49 points are known.

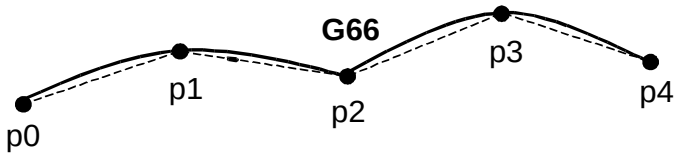
With G62, the previous "history" is deleted and the geometric movement is almost like the one calculated from p1.

If G62 is not programmed in multi-pass machining, there could be different levels of machining.

- **G63**: the first curve inclination does not affect the second curve. In order to keep the continuity between the two curves, to keep the curves continuous, a conjunction has been inserted depending on the accuracy requested by the splines calculation. This could lead to a slowdown at the conjunction of the two curves.

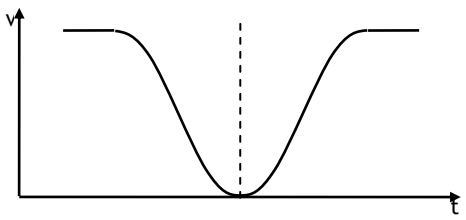


- **G66**: the inclination of the first curve does not affect the second one. G66 will correspond to an angle introducing a discontinuity between the two curves.

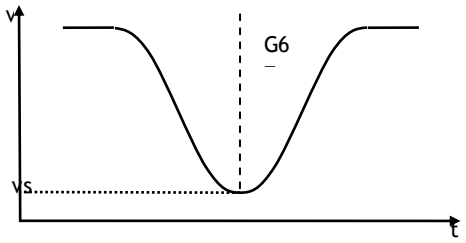


The discontinuity will have a null speed rate. The first curve will finish with a deceleration ramp until it reaches 0 while the second will start using an acceleration ramp to reach the required work feed.

G6



- **G67**: as for G66, the two curves are discontinuous. The discontinuity will have a feed rate (vs) allowing the axes to work on the angle without any dynamic problem. Feed rate value is calculated according to the accelerations applied to each axis.



3.5.7 *MAXLEN – Maximum spline length*

This variable defines the maximum length of the polynomial created. It is advisable to use a value higher than 5 mm (or 0.2 inches) as lower values could slow down machining.

Syntax

MAXLEN= *value*

---

where:

value It's a real positive number indicating the maximum length of the polynomials. It can be programmed directly or indirectly using an E parameter. The unit of measurement is the same as that of the system.

3.5.8 *TBS – Tolerance on axis work feed rate (spline)*

TBSGO – Tolerance on axis rapid feed rate (spline)

These variables define, for each axes, the tolerance range to use in generating polynomial curves. It is possible to define different ranges for work feed rate and rapid federate movements.

Syntax

TBS (*axis*)= *value*
TBSGO (*axis*)= *value*

---

where:

axis parameter indicates to which axis the tolerance value is referring.
It can be specified in two ways:

- Using ASCII character identifying axis name
- Using a number between 1 and 64 specifying axis ID

value It's the tolerance value expressed in the active unit of measurement. It can be expressed directly by a real number or indirectly by an E parameter.

Characteristics:
The polynomial curves generated could pass through or next to the programmed points. This variable defines for each axis the distance allowed between the curve generated and the polygonal one corresponding to the programmed profile.

If the motion programmed involves several axes, the tolerance range applied on the curve created will be the lowest among all the axes ranges involved in the motion.

3.5.9 *SNMT – Null movement threshold*

It is possible to define a null movement threshold for the curves generated.

Syntax

SNMT (*axis*)= *value*

where:

axis Parameter indicates the axis to which the null movement threshold refers.
It can be specified in two ways:
- Using ASCII character identifying the axis name
- Using a number between 1 and 64 specifying axis ID

value Is the threshold value expressed in the active unit of measurement

Characteristics:
If the distance between two points is less than the null movement threshold, the second point is not taken into account.

3.5.10 *TBV – Precision on work feed rate versors*

TBVGO - Precision on rapid feed rate versors

TBV and TBVGO define the tolerance band to use in generating polynomial curves calculated on versors, both with work feed (**TBV**) and rapid feed rate (**TBVGO**) movements.

Syntax

TBV= *value*
TBVGO= *value*

---

where:

value tolerance value is proportional to the tolerance band value of the linear axes and it's a value between 0 and 1. Value 0 disables the tolerance management on versors.

Characteristics:

As the precise positioning of the rotary axes depends from the precision in calculating curves on versors, the tolerance band value used to generate polynomial curves calculated on versors is very important.
It can be advisable to set a value 0.1 times higher than the tolerance value defined for linear axes.

3.5.11 *DAC – Discontinuity angle threshold (linear axes)*

DAV – Discontinuity angle threshold (rotary axes)

These variables define a threshold angle on the HSM profile elements; once the profile is exceeded, the system automatically introduces a transition code.
It is possible to define a threshold angle for both linear (DAC) and rotary (DAV) axes.

Syntax

DAC = *value*
DAV = *value*

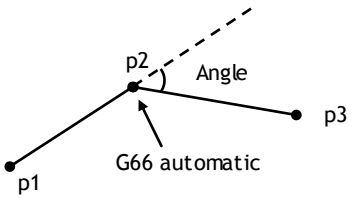
---

where:

value Indicates angle threshold value. It can be programmed directly using a real number between 0 and 180 or indirectly using a E parameter.

Characteristics:
Angle threshold parameter for discontinuities allows to identify any angle on the HSM profiles programmed. If no angles are found, wrong oscillations could be performed on the splines. The best approach to work on an angle is to stop at null speed and restart the movement from the following line.

In some circumstances stopping could be dangerous as the tool keeps its rotation removing material and it could create some "notches" on the workpiece.



The transition code depends on the AMP configuration and can be modified using the **DPMODE** variable.

3.5.12 *DPMODE – Discontinuity management mode*

This variable allows to define the transition code inserted by the CN when the angle between two HSM elements exceeds the discontinuity threshold angle.

Syntax

DPMODE= *value*

---

where:

value it is one of the values allowed for the variable and is:

1. to insert a G66 code
2. to insert a G67 code
3. to insert a G63 code

Characteristics:

The variable defines the transition code inserted by the CNC when the angle between HSM elements exceed the discontinuity threshold angle.

The threshold angle value depends on the AMP configuration and can be modified using both the DAC variable (for linear axes) and the DAV variable (for rotary axes).

3.5.13 *CURLEN – Minimum long line length to change the curve*

The variable defines the minimum length of the “long line” in order to manage the curve change. CURMODE variable defines the curve change managing mode.

Syntax

CURLEN= *value*

---

where:

value It is the minimum length value expressed in the active measurement units for the long line. It can be programmed directly using a real number or indirectly using an E parameter.

3.5.14 *CURRAP – Long/short line length ratio to change the curve*

The variable defines the minimum value of the ratio between short and long line in order to identify the curve change.

Syntax

CURRAP= *value*

---

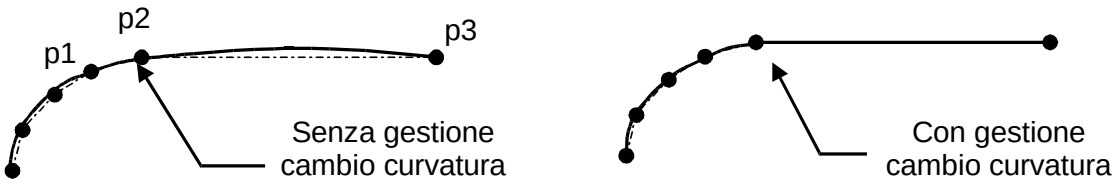
where:

value it is the value of the ratio between long and short lines of the profile expressed by the unit of measurement active for the long line. It can be programmed directly using a real number or indirectly using an E parameter.

Characteristics:

This variable allows defines the threshold ratio between two profile lines to enable the curve change management.

For instance, if CURRAP=6 the distance between p1 and p2 must be 6 times higher than the distance between p2 and p3 in order to enable the curve change.



Curve change management mode is defined by CURMODE variable.

3.5.15 *CURMODE – Curve change management mode*

The variable defines control behaviour when curve changes.

Syntax

CURMODE= *value*

---

where:

value it is one of the values allowed for the variable and can be:

3. to insert a G63 code
4. to insert a G62 code
5. to intensify the points using middle points on the long line

3.5.16 *IDMODE – Internal Discontinuities management mode*

The variable defines the transition code managed by the control when working on a discontinuity due to the high-speed machining algorithm.
Angles on the original profile are managed by DAC, DAV and DPMODE variables, but, if the tolerance thresholds are very low, the optimization algorithm can generate additional angles.

Syntax

CURMODE = *value*

where:
value it is one of the values allowed for the variable and can be:

1. to insert a G66 code
2. to insert a G67 code
3. to insert a G63 code

3.5.17 *CEND – Continuity type for final conditions*

The variable defines the calculation conditions to be set both at the beginning and at the end of each line of the curve (in relation to the programmed points).
The programming of that value can affect the "smoothness" of the curve created.

Syntax

CEND = *value*

where:
value it is one of the values allowed for the variable and can be:

1. programmed point
2. programmed point and first derivative
3. programmed point and second derivative

Characteristics:
The continuity on the point generated at the beginning and at the end of the curve, changes according to the variable value.

At the beginning of the curve the effect of these values is:

1. The forcing of the passage through the first point
2. The forcing of both the passage through the first point and the first derivative, it's similar to forcing the direction joining the two points.
3. The forcing of the passage through the first point, the first and the second derivative, it's similar to imposing the curve generated by the first three points.

At the end of the curve the effect of these values is:

1. The forcing of the passage through the last point
2. The forcing of both the passage through the last point and the first derivative, it's similar to imposing the direction joining the two last points.
3. The forcing of the passage through the last point, the first and the second derivative, it's similar to imposing the curve generated by the last three points.

3.5.18 *SPL3D – Spline kinematic transformation*

Enables/disables the algorithm of the Spline kinematic transformation from part positions to those of the joint.

Syntax

SPL3D = *value*

where:

value Can be 0 (default) or 1. If 0, the transformation is made by the TCP algorithm while if the value is 1 the kinematic transformation is made by the high speed algorithm.

Characteristics:

When the machine kinematics are enabled (that is when the TCP is programmed) and the Splines kinematic transformation (SPL3D = 1) is also enabled, the Spline in work units is transformed (using the inverted kinematics) and then regenerated in terms of the joint. In doing so the dynamics can be calculated with precision using the maximum dynamics offered by the axes, while maintaining the programmed feed on the workpiece.

3.6 Changing drive and axes parameters

The commands in this class allow the CNC read/write parameters associated with the axis configuration and the drive associated with the axis itself.

COMMAND	FUNCTION
CPA	Reads/writes a CNC parameter associated with an axis
CPD	Reads/writes a parameter associated with the axis drive

3.6.1 *CPA – Reading/writing of an axis parameter*

The CPA triliteral allows reading/writing of an axis parameter. The available parameters are both those configured in the AMP and additional parameters not configured in AMP.

Syntax

(CPA, *mode*, *axis*, *parameter*, *value***)**

---

where:

mode is the mode used to access the parameter, the valid values are:
- **R**: read
- **W**: write

axis It can be the numerical axis identifier or the axis name. When *axis* is specified by name (alphabetical character), the axis must belong to the current process, otherwise the system displays a format error. If the axis is specified using the identifier, it can be specified directly with an using an integer value or indirectly with a variable.

parameter Is the number identifying the axis parameter. The list of numbers available and the corresponding parameters are specified below.

value Depending on the required access mode, it is the variable where the value read from the specified axis parameter (read access) will be stored or the value to be written to the selected axis parameter (write access).
For read access, *value* must be a variable, but for write access it can be either a number or a variable.

Characteristics:

The parameter change made by part program is enabled starting from the following line.

The parameter change is temporary as the default values defined in AMP are restored.when the control is rebooted

The Reset command does **NOT** restore the configured value of a parameter changed with the CPA command.

Some parameters are read only, the table below specifies which parameters are read and write or read only.

Parameter code	Mnemonic PLC	Parameter description	Access type
1	fn_AXTYPE	Axis type	R
2	fn_AXCLOCK	Axis clock time	R
3	fn_CLOCKRAT	Ratio between the frequency of the interpolator clock associated with the axis and the axis clock frequency	R
4	fn_DIGITAL_AXIS	Indicates if the axis is digital (value 1) or analog (value 0).	R
5	fn_LINEAR_OPTICAL_ENCODER	Linear scale present	R / W
6	fn_CHANNEL_A_POLARITY_INV	Indicates if the channel A polarity of the encoder is inverted or not.	R / W
7	fn_CHANNEL_B_POLARITY_INV	Indicates if the channel B polarity of the encoder is inverted or not.	R / W
8	fn_CHANNEL_Z_POLARITY_INV	Indicates if the channel Z polarity of the encoder is inverted or not.	R / W
9	fn_DIRECTION_COUNT	Indicates if the direction count is inverted or not.	R / W
10	fn_MARKER_DETECTION	Method for detectng the zero marker. It can be a level (value 0) or an edge (value 1)	R / W
11	fn_RAPID_TRAVERSE_FEED	Rapid traverse feed	R / W
12	fn_RAPID_ACCELERATION	Rapid acceleration	R / W
13	fn_MANUAL_FEED	Manual or G1, G2, G3 Feedrate	R / W
14	fn_MANUAL_ACCELERATION	Manual acceleration	R / W
15	fn_RAPID_JERK	Rapid jerk	R / W
16	fn_WORKING_JERK	Working jerk	R / W
17	fn_ELECTRICAL_PITCH	Electrical pitch value	R / W
18	fn_MECHANICAL_PITCH	Mechanical pitch value	R / W
19	fn_ROLLOVER_PITCH	Rollover pitch	R / W
20	fn_RAPID_TRAVERSE_VOLTAGE	Maximum voltage for rapid traverse (maximum axis feed)	R / W
21	fn_MAXIMUM_FEED	Maximum axis feed	R / W
22	fn_HOME_POSITION_FEED	Feedrate for marker search after limit switch	R / W
23	fn_NULL_OFFSET_VALUE	Null Offset value	R / W
24	fn_HOME_POSITION_VALUE	Home position value	R / W
25	fn_HOMING_DIRECTION	Homing direction search	R / W
26	fn_PERCENT_OF_VFF	VFF percentage	R / W
27	fn_UPPER_SW_OVERTRAVEL	Upper software overtravel	R / W
28	fn_LOWER_SW_OVERTRAVEL	Lower software overtravel	R / W
29	fn_SERVO_LOOP_GAIN	Servo loop gain	R / W
30	fn_STAND_STILL_LOOP_GAIN	Loop gain position at standstill	R / W
31	fn_POS_ERROR_STAND_STILL	Position error at standstill	R / W
32	fn_POS_ERROR_WITH_VFF	Error position with VFF	R / W
33	fn_POS_ERROR_WITHOUT_VFF	Error position without VFF	R / W
34	fn_IN_POSITION_BAND	Position tolerance	R / W
35	fn_IN_POSITION_WAIT	Input threshold tolerance	R / W

Parameter code	Mnemonic PLC	Parameter description	Access type
36	fn_IN_POSITION_WINDOW	Maximum waiting time for input tolerance	R / W
37	fn_AXIS_BACKLASH	Backlash value	R / W
38	fn_DEAD_ZONE	Dead zone value	R / W
39	fn_DRIVER_ADDRESS	Drive address	R
40	fn_HOMING_TYPE	Homing cycle type selected	R / W
41	fn_ADDITIONAL_SERVICE	Additional services	R / W
42	fn_MOTOR_TRANSDUCER	Motor transducer	R
43	fn_PROBING_CONFIGURATION	Probing configuration	R / W
44	fn_SPINDLE_ZERO_OFFSET	Determines the spindle position	R / W
45	fn_SPINDLE_WITH_RAMPS	Determines spindle with ramp	R / W
46	fn_SPINDLE_CSS_MASTER	Determines the axis whose position will control the constant speed	R / W
47	fn_SPINDLE_REV_TIME_GEAR1	Maximum time for spindle reversal (Gear 1)	R / W
48	fn_SPINDLE_REV_TIME_GEAR2	Maximum time for spindle reversal (Gear 2)	R / W
49	fn_SPINDLE_REV_TIME_GEAR3	Maximum time for spindle reversal (Gear 3)	R / W
50	fn_SPINDLE_REV_TIME_GEAR4	Maximum time for spindle reversal (Gear 4)	R / W
51	fn_SPIN_POSERR_STANDSTILL	Position error at spindle zero speed	R / W
52	fn_SPINDLE_SPEED_GEAR1	Spindle speed (Gear 1)	R / W
53	fn_SPINDLE_SPEED_GEAR2	Spindle speed (Gear 2)	R / W
54	fn_SPINDLE_SPEED_GEAR3	Spindle speed (Gear 3)	R / W
55	fn_SPINDLE_SPEED_GEAR4	Spindle speed (Gear 4)	R / W
56	fn_SPINDLE_VOLTAGE_GEAR1	Voltage for maximum spindle speed (Gear 1)	R / W
57	fn_SPINDLE_VOLTAGE_GEAR2	Voltage for maximum spindle speed (Gear 2)	R / W
58	fn_SPINDLE_VOLTAGE_GEAR3	Voltage for maximum spindle speed (Gear 3)	R / W
59	fn_SPINDLE_VOLTAGE_GEAR4	Voltage for maximum spindle speed (Gear 4)	R / W
60	fn_SPINDLE_GAIN_GEAR1	Spindle servo loop gain (Gear 1)	R / W
61	fn_SPINDLE_GAIN_GEAR2	Spindle servo loop gain (Gear 2)	R / W
62	fn_SPINDLE_GAIN_GEAR3	Spindle servo loop gain (Gear 3)	R / W
63	fn_SPINDLE_GAIN_GEAR4	Spindle servo loop gain (Gear 4)	R / W
64	fn_OFFSET_BETWEEN_MARKERS	Offset between Markers	R / W
65	fn_SKEW_ERROR_MIN	Skew error	R / W
66	fn_SKEW_ERROR_MAX	Maximum skew error	R / W
67	fn_SKEW_GAIN	Skew gain	R / W
68	fn_DRIVE_STATUSW	Drive status word	R
69	fn_OS3_DRIVE_WARNINGS	Drive warning	R
70	fn_AXIS_IOSTAT	Status of the I/O associated with the drive	R / W
71	fn_ACT_MICRO_MARKER_DIST	Distance between markers and zero switch	R
72	fn_MAX_MICRO_MARKER_DIST	Maximum allowable distance between zero switch and marker	R / W
73	fn_PROCESS_NUMBER	Identifier of the process to which the axis belongs	R
74	fn_AXIS_SHARING	Axis shared or exclusive	R
75	fn_TRANSDUCER_CONFIG	Determines transducer configuration	R / W
76	fn_ACTIVE_GEAR_NUMBER	Active spindle gear range	R
77	fn_ZEROSHIFT1	Zeroshift 1 value	R / W

Parameter code	Mnemonic PLC	Parameter description	Access type
78	fn_ZEROSHIFT2	ZeroShift 2 value	R / W
79	fn_AXIS_BACKLASH_TIME	Application time for backlash	R / W
80	fn_TRANSDUCER_ID	Axis transducer identifier	R
81	fn_CONVERTER_ID	Axis converter identifier	R
82	fn_SETRESRIF	Reported Axis status	R / W
83	fn_RAPID_TIME_MIN_RAMP	Ramp time rapid acceleration	R / W
84	fn_WORKING_TIME_MIN_RAMP	Ramp time working acceleration	R / W
85	fn_RAPID_DECELERATION	Rapid deceleration value	R / W
86	fn_MANUAL_DECELERATION	Manual deceleration value	R/ W
87	fn_RAMP_MODALITY	Axis ramp type	R / W
88	fn_GET_MASTER_SLAVE	Master axis identifier associated with slave	R
89	fn_AXIS_INTERFACE_TYPE	Axis interface type	R
90	fn_BW_ENABLE	Broken wire enabled or disabled	R / W
91	fn_BW_TIME	Broken wire application time	R / W
92	fn_EDGE_LEAP_FEED	Maximum velocity jump on corner value	R / W
93	fn_GET_LAST_ERROR	real-time axis error value	R
94	fn_SPINDLE_SS_GAIN_GEAR1	Stand still servo loop gain (Gear 1)	R / W
95	fn_SPINDLE_SS_GAIN_GEAR2	Stand still servo loop gain (Gear 2)	R / W
96	fn_SPINDLE_SS_GAIN_GEAR3	Stand still servo loop gain (Gear 3)	R / W
97	fn_SPINDLE_SS_GAIN_GEAR4	Stand still servo loop gain (Gear 4)	R / W
98	fn_SPIN_ORIENT_SPEED_GEAR1	Speed for spindle orientation (Gear 1)	R / W
99	fn_SPIN_ORIENT_SPEED_GEAR2	Speed for spindle orientation (Gear 2)	R / W
100	fn_SPIN_ORIENT_SPEED_GEAR3	Speed for spindle orientation (Gear 3)	R / W
101	fn_SPIN_ORIENT_SPEED_GEAR4	Speed for spindle orientation (Gear 4)	R / W
102	fn_SPIN_ORIENT_ACCEL_GEAR1	Acceleration for spindle orientation (Gear 1)	R / W
103	fn_SPIN_ORIENT_ACCEL_GEAR2	Acceleration for spindle orientation (Gear 2)	R / W
104	fn_SPIN_ORIENT_ACCEL_GEAR3	Acceleration for spindle orientation (Gear 3)	R / W
105	fn_SPIN_ORIENT_ACCEL_GEAR4	Acceleration for spindle orientation (Gear 4)	R / W
106	fn_SPIN_STOP_THRESH_GEAR1	Speed threshold for spindle stop (Gear 1)	R / W
107	fn_SPIN_STOP_THRESH_GEAR2	Speed threshold for spindle stop (Gear 2)	R / W
108	fn_SPIN_STOP_THRESH_GEAR3	Speed threshold for spindle stop (Gear 3)	R / W
109	fn_SPIN_STOP_THRESH_GEAR4	Speed threshold for spindle stop (Gear 4)	R / W
110	fn_SPIN_IN_POS_BAND	Spindle in position band value	R / W
111	fn_SPIN_IN_POS_WAIT	Spindle in position wait time	R / W
112	fn_SPIN_IN_POS_WINDOW	Spindle in position window wait time	R / W
113	fn_ ENABLE_SERVO_MODE	Servo loop type for axis enabling	R / W
114	fn_SKEW_RECOVERY_VEL	Split axis skew recovery velocity	R / W
115	fn_DRIVE_CONTROLW	Drive Control Word	R / W
116	fn_MIN_AX_VELOCITY	Minimum axis velocity	R / W
117	fn_MIN_AX_POSITION	Minimum axis position threshold	R / W
118	fn_DEN_MEC_PITCH	Mechanical pitch denominator	R / W
119	fn_ZEROSHIFT1_SETUP	Zeroshift 1 value setup	R / W
120	fn_ZEROSHIFT2_SETUP	Zeroshift 2 value setup	R / W

Parameter code	Mnemonic PLC	Parameter description	Access type
121	fn_RAPID_DJERK	Rapid DeJerk value	R / W
122	fn_WORKING_DJERK	Working DeJerk value	R / W
123	fn_RAPID_TIME_MIN_DRAMP	Rapid deceleration ramp time	R / W
124	fn_WORKING_TIME_MIN_DRAMP	Work deceleration ramp time	R / W
125	fn_REFERENCE_OFFSET	Converter offset	R / W
126	fn_EMERG_DECELERATION	Emergency deceleration value	R / W
127	fn_UPPER_SW_OVERT_TTIP	Upper tool tip software overtravel	R / W
128	fn_LOWER_SW_OVERT_TTIP	Lower tool tip software overtravel	R / W
129	fn_DRIVER_SUBADDRESS	Drive sub-address	R
130	fn_INTERP_POINT	Interpolated point (for simulation)	R / W
131	fn_LEAP_ACCE	Maximum acceleration leap on tangent	R / W

Example 1:
To change the *rapid acceleration* (RAPID_ACCELERATION) configured in AMP by writing 2500 mm/sec^2 for the Y axis identified as 2 in the current process. The following commands are equivalent:

(CPA, W, E0, E1, E2) where E0 = 2, E1 = 12 and E2 = 2500
(CPA, W, 2, 12 , E2)
(CPA, W, Y, 12 , 2500)

3.6.2 *CPD – Reading/writing of a drive parameter*

The CPD command allows reading or writing of a drive parameter associated with a digital axis.

Syntax

(CPD, *mode*, *axis*, *parameter*, *value*, *format* , *mask*]**)**

where:

mode represents the access mode parameter, the valid values are:
- **R**: read access
- **W**: write access

axis It can be the numerical axis identifier or the axis name. where *axis* is specified by name (alphabetical character), this must belong to the current process, otherwise the system displays a format error. If the axis is specified using the identifier, it can be specified directly with an integer or indirectly with a variable.

parameter Is the number identifying the axis parameter. The list of numbers available and the corresponding parameters are specified below.

value For read access, *value* must be a variable, but in for write access it can be either a number or a variable.

format For drives having OS-Wire or D.S.I interface, field is not important, set value =0.
For drives having Mechatrolink™ interface, set as follow:
- bit 0 (parameter dimension): **0** parameter uses 2 bytes, **1** parameter uses 4 bytes
- bit 1 (access type): **0** access in RAM, **1** access in EEPROM

mask It can be used for YASKAWA drives having a Mechatrolink interface. This is the mask for writing the drive parameter. It is an optional field, if programmed it must be an hexadecimal code of 8 characters maximum, preceded with 0x. If the mask is specified, the parameter will be written only where the corresponding bits are set to 1; if all the mask bits are 0, the parameter will be completely reset; if the mask value is 0xFFFFFFFF, the entire parameter will be written.
The default value is 0xFFFFFFFF, therefore if a mask is not specified, all parameter bits will be written. For other drive types, the mask has no effect.

Example 1:

e.g. to read the speed loop proportional loop gain parameter (cod. param. = 5004) configured for the Z axis (id = 3) on an OS-Wire drive. The following commands are equivalent:

(CPD, R, E0, E1, E2, E3) where E0 = 3, E1 = 5004 and E3 = 0
(CPD, R, 3, 5004, E2, 0)

With this command:
(CPD, R, 3, 5004, 2.5, 0) -> the system displays a **format error** because there is no variable specified for the destination.

Example 2

e.g. to write the Pn50A parameter “Input Signal Selection 1” of the Sigma-V drive connected to the Z axis:

(CPD, W, Z, 1290, 32768, 0, 0XF000)

The “P-OT Signal Mapping” field is set to 8 (fourth nibble corresponding to 32768) of the Pn50A parameter (1290 in decimal) without changing other fields, writing is to RAM and the parameter is 2 bytes long (format = 0).

4. PROGRAMMING TOOLS AND TOOL OFFSETS

This chapter explains how to program tools and tool offsets. All the functions described in this chapter must be handled by the PLC.

The system integrator develops the interface between the control and the machine tool, as well as the application-specific T and M functions and also the code of the function used for requalifying and pre-setting: RQT and RQP.
For more information about these T functions you must refer to the machine tool documentation provided by the machine manufacturer.

You can also find further information about M functions in Chapter 7.

Neither the RESET nor system power off can disable the offset values applied using the "h" or T functions.
The PLC decides whether the offset values are disabled by RESET.

4.1 T address for programming tools

The T address defines the tool and the tool offset used in a given machining process.

Syntax

T [*tool*] [**.***tool offset*]

where:

tool is the ASCII code of the tool, it is maximum of 32 characters long. This can be programmed explicitly (with an integer) or implicitly with a local or system variable. If you program it as a whole number, the system converts it automatically into a string.

tool offset is the tool offset number and ranges from 0 to 300.
It can be an integer or an E parameter

Characteristics:
For the sake of compatibility with OSAI Series 10, the tool code can be programmed in numerical format of up to 12 digits. The numerical format may be programmed implicitly.

The following examples show different ways of programming tools and offsets.

Examples:

T1	Selects tool 1 and the tool offset defined in the tool table.
T "U01".1	Selects tool U01 and tool offset 1.
T(SC.5).0	Selects the tool whose name is specified in the five SC variables, starting from address 0, and without offset.
T.0	Removes the tool offset from the active tool.
T0	Disables the active tool and tool offset.
T.1	Enables tool offset 1 with the active tool.
T1234567890123	Shows a syntax error: tool code programmed in numerical format with more than 12 digits.
T"1234567890123"	Selects tool 1234567890123 and the offset given in the tool table.
T(!VbUser).1	Selects the tool identified by user variable !VbUser, string type, and an offset of 1.
T(LS).2	Selects the tool identified by user variable LS and the offset of 2.
TE0.1	Selects the tool identified by user variable E0 and the offset of 1.

4.2 T address for multi-tool programming

The T address defines the tools to be used simultaneously for a given machining process.

Syntax

T [*master*] [**.***tool offset*] [*/*{*slave* l] [first_*slave, last slave*]l {*variable, num_variables*}}]

where:

master	is the ASCII code of the tool, it is 32 characters maximum long. This can be programmed explicitly (with an integer) or implicitly with a local or system variable. If you program it as a whole number, the system converts it automatically into a string.
tool offset	is the tool offset number. It can be an integer or an E parameter, it ranges from 0 to 300.
slave	is the number of the tool. This can be an integer or a local or system variable.
first_slave	is a tool number representing the first of a series of tools. This can be an integer or a local or system variable.
last_slave	is a tool number representing the last of a series of tools. This can be an integer or a local or system variable.
variable	is a local or system variable containing the first of a series of tools.
num_variable	is an integer or a local or system variable representing the number of variables to take into consideration starting from "variable".

Characteristics:
Multi-tool programming is used on perforating machines.
The management of tools associated with the T code is handled by PLC that receives the values of the programmed tools.
Accepted values for the slave tool codes vary from 0 to 65535. Maximum number of slave tools that can be simultaneously programmed is 60.

As showed above, the list of tools to be used can be specified in three different formats:

1.Single Format

Examples:

T1.2/ 50	tools	1, 50
	offset	2
T1.2 /20,33,45,46	tools	1,20,33,45,46
	offset	2

2. Numerical Sequence Format

This simplifies programming of multiple tools by using sequential code.

Examples:

T1.3 /[30, 35]	equivalent to T1.3 /30,31,32,33,34,35
T1.3 /[56, 51]	equivalent to T1.3 /56,55,54,53,52,51
T1.3 /[50, 52], [10,13]	equivalent to T1.3 /50,51,52,10,11,12,13

As you can see, the starting tool number can be > or < the final tool number. In the first instance the tools codes are considered in ascending order and in the second instance they are considered in descending order.

3. Variable Sequence Format

This is based on an array of variables from which to select the tool numbers.

Example:

E0 = 1, 30, 45	load in E0, E1, E2 the values 1, 30, 45
T1.2 /{ E0, 3 }	equivalent to T1.2 /E0, E1, E2 & to T1.2 /1, 30, 45
E0 = 1, 30, 45	load in E0, E1, E2, E3 the values 1, 30, 45
SN0 = 4, 77	load in SN0, SN1 the values 4, 77
SN4 = 3	load in SN4 the value 3
T1.2 /{ E0, SN4 }, { SN0,2 }	equivalent to T1.2 /E0, E1, E2, SN0, SN1 & to T1.2 /1,30,45,4,77

The three formats shown above may be mixed.

Example:

E0 = 29, 56	load in E0, E1 the values 29, 56
SN6 = 2	load in SN6 the value 2
T1.3 /[7, 10], 15, { E0, SN6 }	equivalent to T1.3 /7,8,9,10,15,29,56

4.3 h address

The h address permits the application of a tool offset both in point-to-point (G29) and in continuous (G27-G28) modes.

Syntax

h [*tool offset*]

where:

tool offset is the tool offset number. It can be an integer or a local or system variable between 0 and 300.

Characteristics:

The "h" address must be programmed by itself in the part program block.

An "h" address disables the active offsets programmed with a "T" command.

The axes to which *tool offset* is applied are those programmed in a "T *tool.offset*" command.

The *offset* values are applied to the axes when the system reads the "h" address in the part program.

The "h" address does not need synchronisation either with the logic or with the axes moves.

If “h” is not synchronized, it is displayed when it is read by the part program and not when it is implemented.
The same occurs in the case of synchronous uses of “h”, as in G96. In this case, the application of “h” must be synchronized with “#” for the spindle revolutions to function correctly.

Programming "h" without *tool offset* or with *tool offset=0* will disable the tool offset value programmed with previous "h *tool offset*" command.

RESET does not disable the correction values applied with "h" address.

The PLC decides whether or not to apply offset values after a RESET.

The offset values programmed with h address are usually displayed in the field reserved for the T address.

The offset table fields from which offset values are read are as follows:

Offset value	Offset table fields
Length 1	Length 1 + Current requalification length 1
Length 2	Length 2 + Current requalification length 2
Length 3	Length 3 + Current requalification length 3
Length 4	Length 4 + Current requalification length 4
Length 5	Length 5 + Current requalification length 5
Diameter 1	Diameter 1 + Current requalification diameter 1
Diameter 2	Diameter 2 + Current requalification diameter 2

Example 1:

.....
.....
.....
T[n].x programs a tool and a tool offset
.....
.....
hy cancels the **x** tool offset and enables the **y** tool offset
.....
.....

Example 2:

.....
.....
.....
hy enables **y** tool offset
.....
.....
T[n].x cancels the **y** tool offset enabled by **h**
.....
.....

Example 3:

.....
.....
.....
hx
.....
.....
hy cancels the programmed **x** tool offset and enables **y**
.....

4.4 AXO – Axis Offset Definition

This command associates the length offset values contained in the offset table with the machine axes.

Syntax

(AXO, [[-] [*axis1*]] [**,** [[-] [*axis2*]] [**,** [[-] [*axis3*]] [**,**[[-] [*axis4*]] [**,**[[-] [*axis5*]]]]] **)**

where:

axis1 name of the axis associated with length offset 1 in the offsets table. If the "-" sign is written before the name, the offset value is used with the inverse sign. If omitted, length 1 of the offset is not associated with any axis.

axis2 name of the axis associated with length offset 2 in the offsets table. If the "-" sign is written before the name, the offset value is used with the inverse sign. If omitted, length 2 of the offset is not associated with any axis.

axis3 name of the axis associated with length offset 3 in the offsets table. If the "-" sign is written before the name, the offset value is used with the inverse sign. If omitted, length 3 of the offset is not associated with any axis.

axis4 name of the axis associated with length offset 4 in the offsets table. If the "-" sign is written before the name, the offset value is used with the inverse sign. If omitted, length 4 of the offset is not associated with any axis.

axis5 name of the axis associated with length offset 5 in the offsets table. If the "-" sign is written before the name, the offset value is used with the inverse sign. If omitted, length 5 of the offset is not associated with any axis.

Characteristics:

The default association between length offsets and characterised axes is made in the AXIS GENERAL INFO page of the AMP configuration.

This association can be changed with the AXO command.

The pre-set values introduced in the offset tables always have a positive sign. The AXO command can associate these values with axes bearing negative signs.

The following are two examples of length offset values applied with and without AXO commands:

Example 1:

.
.
.

N1 T1.4 M6 activates length 1 of offset 4 on the axis associated to length 1 in the offset characterisation phase.

.
.

N100 T0 M6 disables the length offset value applied to the axis.

Example 2:

.
.
.

N1 (AXO,-X,Z) associates X to length offset 1 with negative sign and Z to length offset 2 with positive sign

.
.

N50 T1.4 M6 applies length offset values 1 and 2 from offset 4 to axes X and Z as defined in the AXO command. Length offset 1 will be applied to the X axis with negative sign.

.
.

N100 T0 M6 disables the length offset applied to axes X and Z.

Example 3:

.
.

N1 (AXO,,,,Z) associates the length 4 of the offset to axis Z.

The system RESET command or the selection of a new part program re-establishes the axis/offset default association characterised in AMP.

4.5 RQT – Requalifying Tool Offset

The RQT command requalifies the length and/or diameter dimensions stored in the offset tables.

Syntax

(RQT,*tool*,*offset* [,**L***val1*][,**l***val2*][,**d***val3*] [,**L(***n***)***valn*] [,**d(***m***)***valm*])

where:

tool is the tool ASCII code

offset is the number of the offset to be requalified. The offset number is a value between 1 and 300.

val1 Requalification value for the associate length/radius.

val2 The values can be either numbers or specified as variables.
val3
valn
valm

L value to be added to offset length 1.
L(1)

l value to be added to offset length 2.
L(2)

n Ranges from 1 to 5 and is one of the five possible offset lengths.

m Ranges from 1 to 2 and is one of the two possible offset diameters.

L(*n***)** value to be added to the n-th offset length.

d offset increment to be added to diameter 1.
d(1)

d(*m***)** offset increment to be added to the m-th diameter.

In the event of overlap between “L” and “L(1)”, “l” and “L(2)” or “d” and “d(1)”, the values are assigned in the order in which they are written (from left to right) without any error message being displayed.

Characteristics:
You must specify the length and diameter increments in the RQT command with the currently active measuring unit (inches or millimetres, G70/G71). These values cannot be associated with a scale factor (SCF).

Examples:

(RQT,10,1,L E40,d E41)	This block requalifies tool 10, offset 1. The length 1 increment is contained in E40 and the diameter increment is contained in E41.
(RQT,10,1,L E50,L(4) E51)	This block requalifies tool 10, offset 1. The length 2 increment is contained in E50 and the length 4 increment is contained in E51.

4.6 RQP – Requalifying Tool Offset (pre-setting values)

The RQP command requalifies and pre-sets a specific tool offset according to programmed length and diameter dimensions. When the control executes this command, it writes the corresponding dimensions in the tool offset table by adding the specified length and diameter values. All the offset values are reset.

Syntax

(RQP,*tool*,*offset* [,**L***val1*][,**l***val2*][,**d***val3*] [,**L(***n***)***valn*] [,**d(***m***)***valm*]**)**

where:

tool is the tool ASCII code

offset number of the offset to pre-set. It is a value between 1 and 300. The upper limit of the offset number depends on the number of records in the tool offset file.

L value to be added to offset length 1.
L(1)

l value to be added to offset length 2.
L(2)

n ranges from 1 to 5 and is one of the five possible lengths of the offset.

m ranges from 1 to 2 and is one of the two possible diameters of the offset.

L(n) value to be added to the n-th length of the offset.

d offset increment to be added to the diameter 1.
d(1)

d(m) offset increment to be added to the m-th diameter.

Characteristics:
You must specify the values of length and diameter increments in the RQP command with the currently active measuring units (inches or millimetres, with G70/G71). These values cannot be associated with a scale factor (SCF).

Examples:
(RQP,10,1,L E40, L(3) E41, d E42)

This block pre-sets offset 1 of tool 10. The value of length 1 is contained in E40, the value of length 3 is contained in E41 and the value of diameter is contained in E42.
The command requires PLC logic in order to function.

4.7 TOU – Tool Expiry Declaration

The TOU command is used to declare the specified tool's life has expired.

Syntax

(TOU,*tool***)**

where:
tool is the ASCII code of the tool to be declared expired. It can be a local or a system variable

Example:
(TOU,5) ;Declares tool "5" in the tool table expired
E1=10;
(TOU,E1) ;Declares tool "10" in the tool table expired

5. CUTTER DIAMETER COMPENSATION

Cutter compensation is based on the tool geometry and is applied to a programmed profile defined by segments and arcs.

There are two kinds of compensation available

- **Cutter diameter compensation:** it is perpendicular to a programmed profile (fig 1). Since the typical tool section is a circle, the correction is applied to the tool diameter.

- **Cutter thickness compensation:** it is tangential to a programmed profile (fig 2). Usually the tool direction is in the same plane as the profile

The following figures show how tool compensation acts in the previously described cases-

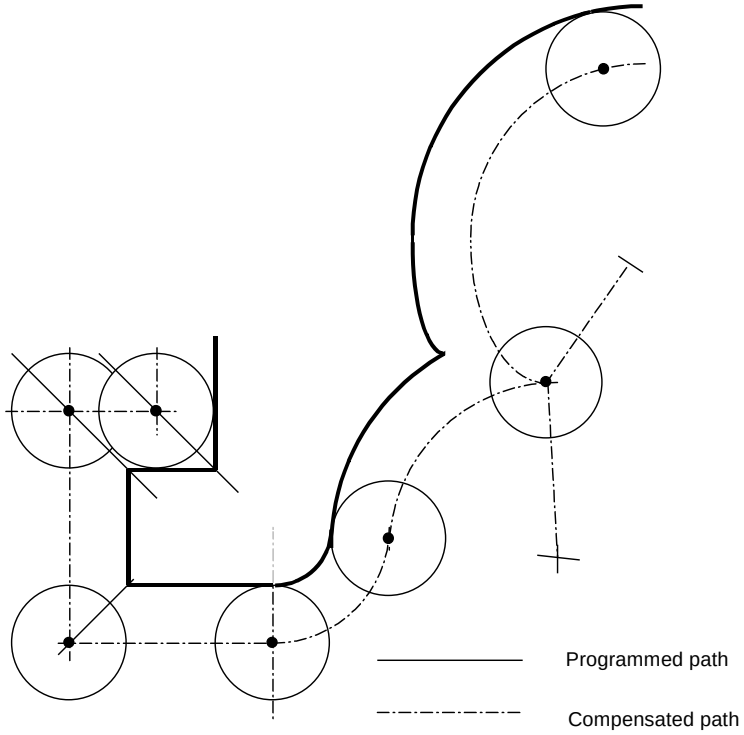


Fig.1 Tool Positioning for Cutter Diameter Compensation

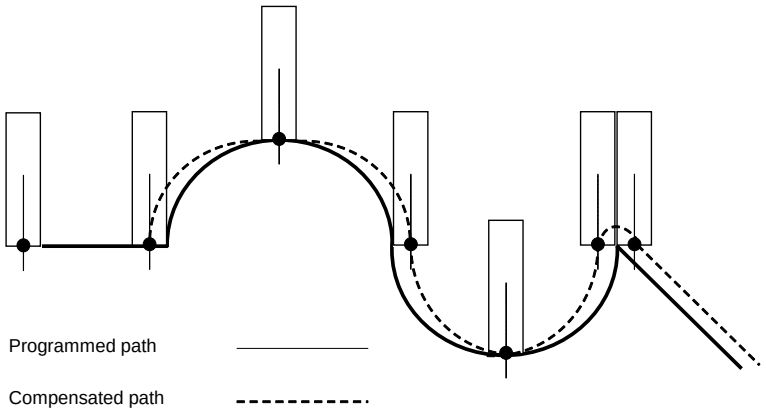


Fig.2 Tool Positioning for Cutter Thickness Compensation

5.1 TTC – Tool compensation type

TTC command configures the type of cutter compensation required.

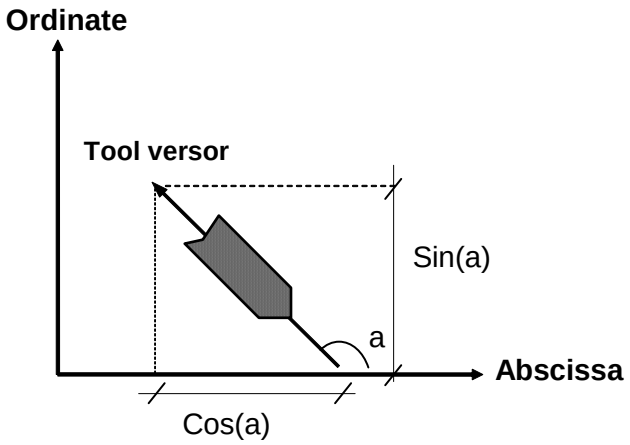
Syntax

(TTC [, *type* [,*cos(a)*,*sin(a)*]]**)**

where:

type type of cutter compensation that will be applied, it can assume the following values:
0. Enables cutter diameter compensation. It is the default value.
1. Enables cutter thickness compensation.
The default value for this value can be configured in AMP.

cos(a),sin(a) versor indicating tool direction. It consist of the sine and cosine of the angle between the tool and the abscissa axis of the interpolation plane.
Default values are 0.0 and 1.0 meaning that the tool is "vertical" and its point is down.



no parameters Disables cutter thickness compensation and enables default compensation (diameter).

A RESET command restores default values.

5.2 G40 G41 G42 – Cutter compensation

These G codes disable or enable cutter compensation:

G40 Disables cutter compensation

G41 Enables cutter compensation when the tool travels left of the profile

G42 Enables cutter compensation with the tool travels right of the profile

Syntax

G40 [*G-codes*] [*operands*]
G41 [*G-codes*] [*operands*]
G42 [*G-codes*] [*operands*]

---

where:

G-codes Other G codes that are compatible with G41, G42 and G40 (see "Compatible G codes" table in Chapter 2).

operands Any operand or code allowed in a G function block.

When cutter compensation is active, the tool is positioned on the intersection or tangency point of the two geometrical elements translated by a value depending on the requested compensation type.

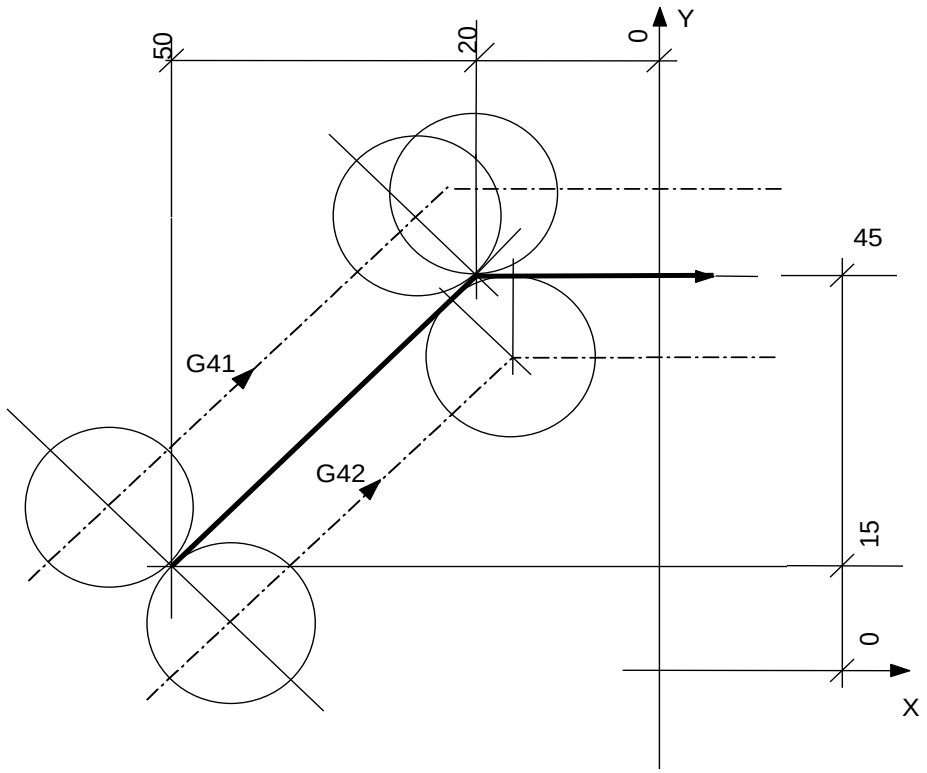
5.2.1 *Enabling Cutter Compensation*

Cutter compensation is enabled by codes G41 or G42, compensation type (diameter or thickness) depends on the TTC command. The movement to the first point in the profile must be linear (G00-G01).

5.2.2 *Cutter diameter compensation (TTC, 0)*

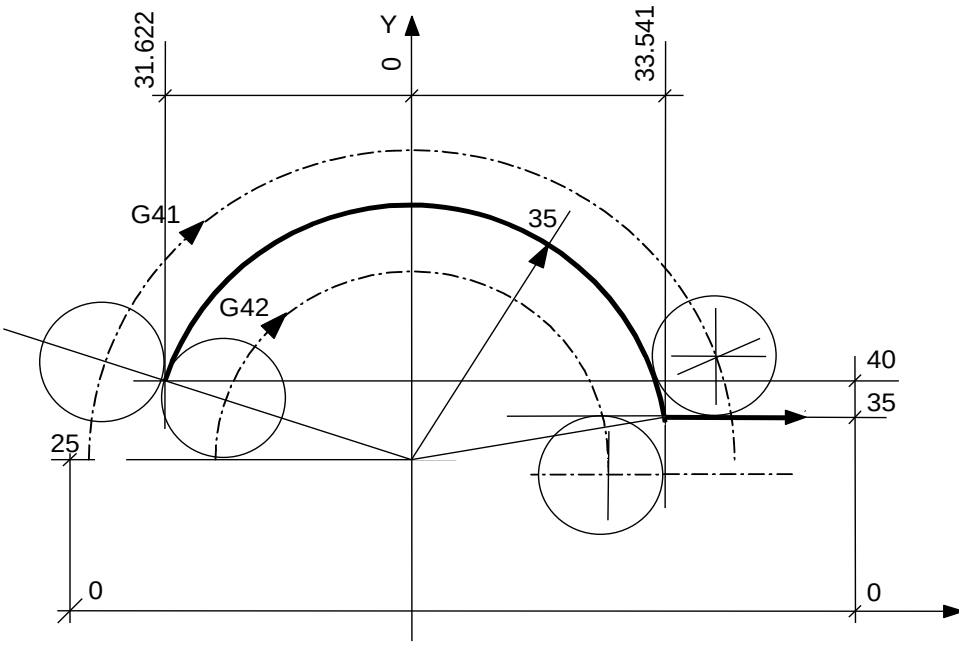
On the first point in the profile, cutter diameter compensation is perpendicular to the first linear or circular move programmed after G41 or G42 on the active plane.
The examples below show how cutter diameter compensation is applied.

- **Linear move first in the profile**



Program:
Cutter diameter compensation on the right of the profile:
G1 G42 X-50 Y15 F200
X-20 Y45
Cutter diameter compensation on the left of the profile:
G1 G41 X-50 Y15 F200
X-20 Y45

- **Circular move first in the profile**



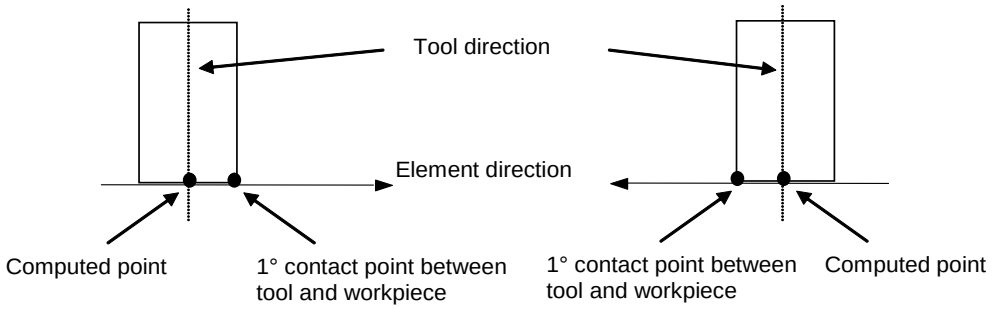
Program:
Cutter diameter compensation on the right of the profile:
G1 G42 X-31.622 Y40 F200
G2 X33.541 Y35 I J25
G1 X.....

Cutter diameter compensation on the left of the profile:
G1 G41 X-31.622 Y40 F200
G2 X33.541 Y35 I J25
G1 X....

5.2.3 *Cutter thickness compensation (TTC, 1)*

When compensation is enabled using G41 and G42 functions the system automatically computes the component of the tool direction on the working plane and, depending on its value, it compensates the programmed profile.
If no versor is specified in the TTC command, then the system assumes the tool is vertical.
The interpolation plane is specified with G code (G16, G17, G18, G19), the system computes the component, lying in that plane, normal to the tool direction. On this component it will apply the offset due to tool thickness.
If the tool is vertical, tool centre will move to the left, when G41 is the current programmed code, or to the right, when G42 is the current programmed code, of an amount equal to half the thickness of the tool. When the tool is not vertical the behaviour will be similar, but the offsetting movement will happen along the tool direction.

In the case of a linear movement normal to the tool direction tool compensation is applied in this direction. Offset direction does not depend on G41/G42, it is assumed to be opposite to the element direction:

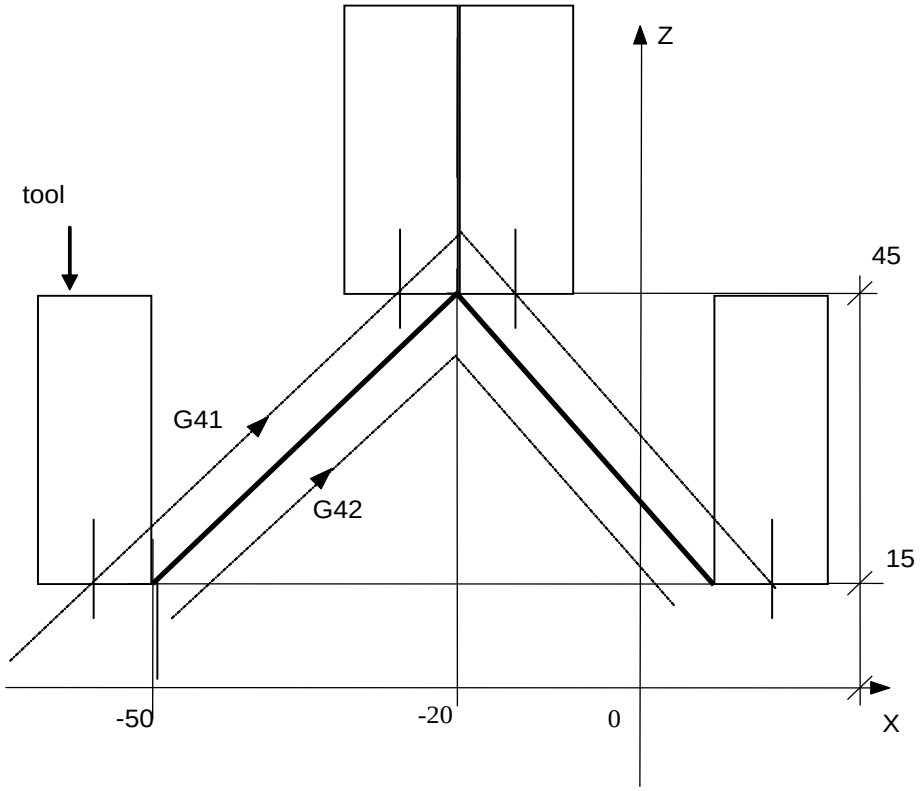


Compensation is performed in the plane and assumes a fixed tool inclination. So while compensation is working (G41 / G42 tool inclination must not be changed. If tool inclination has to change, the compensation must be stopped (G40).

The following examples show the compensation with a circular and with a linear starting point.

- **First movement on the profile is linear**

A linear element, in compensation, is transformed into another linear movement. This movement is translated depending on its direction, on tool direction and in the presence of mirroring.



Program:
Cutter thickness compensation on the right of the profile:
(TTC, 1)
G1 G42 X-50 Z15 F200
X-20 Z45
X10 Z15

Cutter thickness compensation on the left of the profile
(TTC, 1)
G1 G41 X-50 Z15 F200
X-20 Z45
X10 Z15

- **First movement on profile is circular**

A circular element, in cutter thickness compensation, can be divided in one or more elements depending if the tangential point (P_T) position between tool and circle is external or internal to the movement limits P_s and P_e (Fig. 3).

- ***P_T external***: the system creates the first circle, only. It is centred in c', identified translating c with a value equal to tool thickness in the correct direction and the circle radius is the same radius as the programmed movement.

- ***P_T internal***: the system creates the first cycle as in the case of external P_T, then it creates the line through P_T tangential to the circle (when compensation is external to the circle) and a second circle, centred in c'' translated in the opposite direction with respect to the first one and with the same radius.

When **PT corresponds to the limits of the arc** (P_s or P_e) the system creates an arc and the line tangential to P_T.

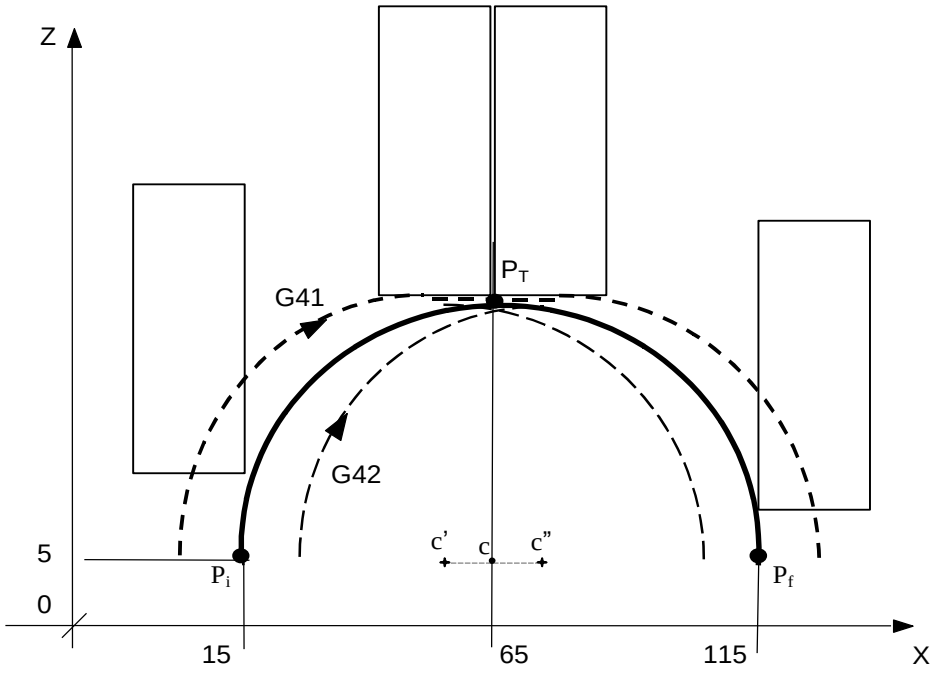


Fig. 3. Cutter thickness compensation on a circular profile

Program:

Cutter thickness compensation on the right of the profile:

(TTC, 1)
G1 G42 X15 Z5 F200
G2 X115 I65 J5
***G1 X*.....**

Cutter thickness compensation on the left of the profile:

(TTC, 1)
G1 G41 X15 Z5 F200
G2 X115 I65 J5
***G1 X*.....**

5.2.4 *Notes on using cutter compensation*

Once enabled, cutter compensation is applied to all the moves programmed at machining or rapid feed rate. Once cutter compensation has been enabled by G41 or G42, the following G functions cannot be programmed:

- G81 - G89 (canned cycles)
- G70 - G71 (mm/inch programming)
- G79 (programming referred to machine zero)
- G33 (threading)
- G72 - G73 - G74 (measuring cycles)
- G16 - G17 - G18 - G19 (change of interpolation plane)
- G12 – G13 – G14 (circular interpolation in space)

When cutter compensation is active, the control displays an error message if:

- the programmed internal radius is shorter than the tool radius.
- the execution of a compensated linear move would reverse the tool direction with respect to the original profile.

Inside a profile compensated with G41 or G42 you can program movements of axes that are not in the interpolation plane. Up to six consecutive movements can be programmed; however the use of the interference correction feature reduces this number.

5.2.5 *Tool Path Optimisation*

Within the cutter diameter compensation programming, the Tool Path Optimization can be activated either by MDI or by part program. This function is enabled and configured using TPO variable. Further configuration can be done with the TPT system variable.

TPO can define two optimisation modes:

- automatic “reduction” of the tool path on corners between two linear or circular blocks
- infeed/exit tangent to the profile (on a circle arc).

TPO and TPT are discussed in detail later in this chapter.

5.2.6 *Disabling Cutter Diameter Compensation*

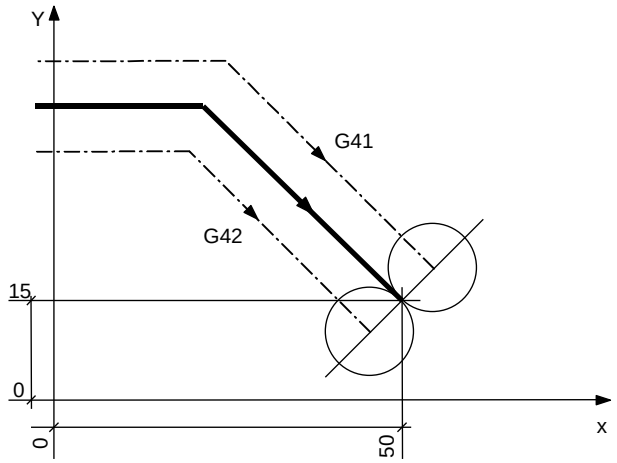
Cutter diameter compensation is disabled by the G40 code. Then the cutter may exit from the part as follows:

- if TPO is not active (standard offset mode), the move programmed by G40 is still considered as an offset;

- if TPO is active, the move programmed by G40 is considered as the exit move from the offset.

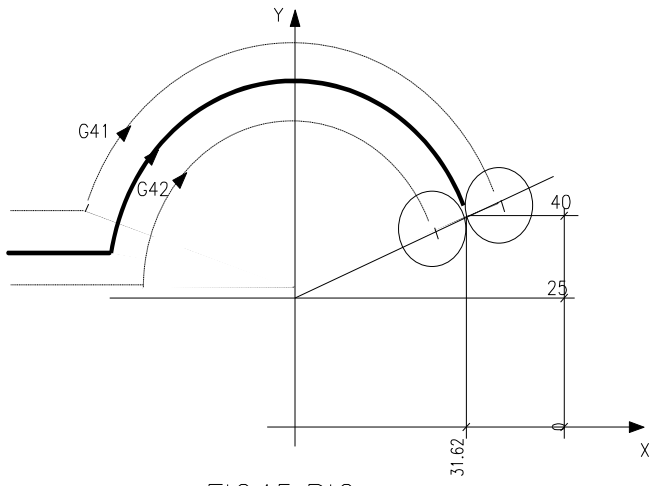
These options are illustrated in the examples below.

- ***The last move in the profile is linear (TPO=0)***



Program:
N88 G1 G40 X50 Y15
N100 G X.. Y..

- **The last move in the profile is circular (TPO=0)**



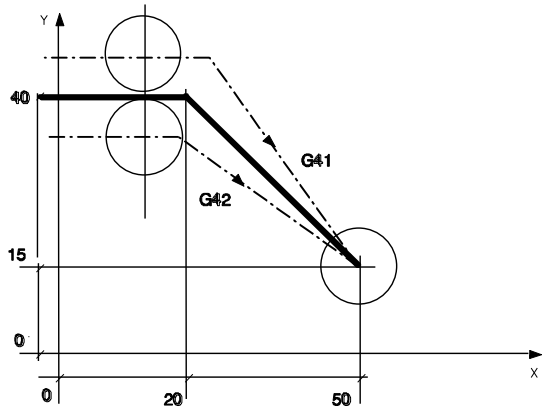
Program:
N99 G2 G40 X36.62 Y40 I J25
N100 G X.. Y..

5.2.7 *Disabling Compensation with TPO active*

With TPO active, the cutter diameter is always compensated through G40; the only difference with respect to the standard procedure is how the last block is executed.

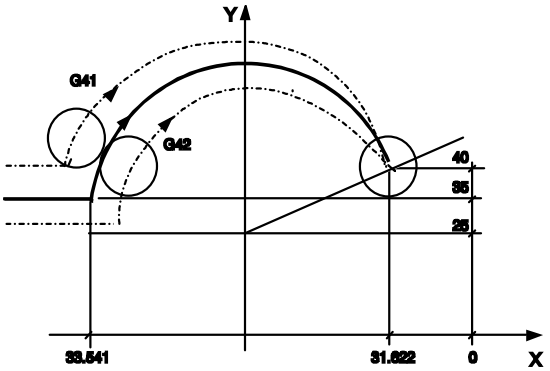
These options are illustrated in the examples below.

- **The last move in the profile is linear (TPO active)**



Program:
N87 G1 X20 Y40
N88 G1 G40 X50 Y15
N100 G X.. Y..

- **The last move in the profile is circular (TPO active)**



Program:
N98 G1X-33.541 Y35
N99 G2 G40 X36.62 Y40 I J25
N100 G X.. Y..

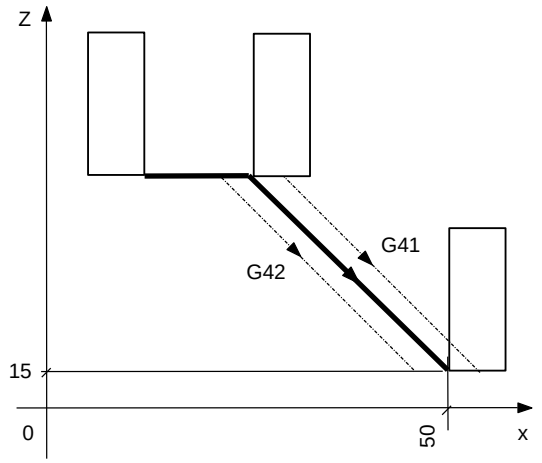
If a block is programmed including G40 only (i.e. not associated with any final point), the same result would be obtained as with standard diameter compensation (TPO=0). Prior to writing a program read this section and the descriptions of TPO and TPT carefully.

5.2.8 *Disabling Disk Thickness Compensation*

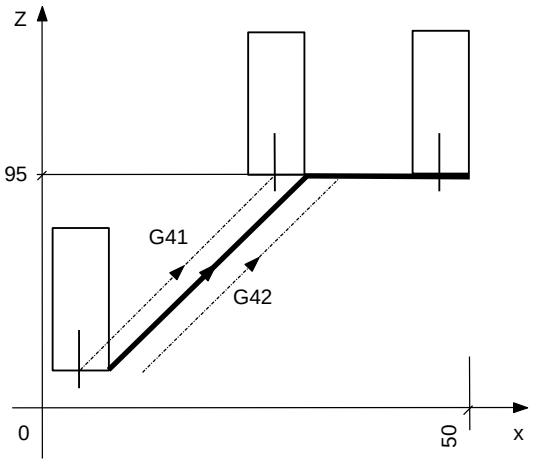
Disk thickness compensation is disabled by the G40 code
The cutter exits always in standard offset mode, it is not affected by TPO values, this mean that the block where G40 is programmed, is considered as compensated.

The following examples show different cases:

- **Last movement on the profile is linear**

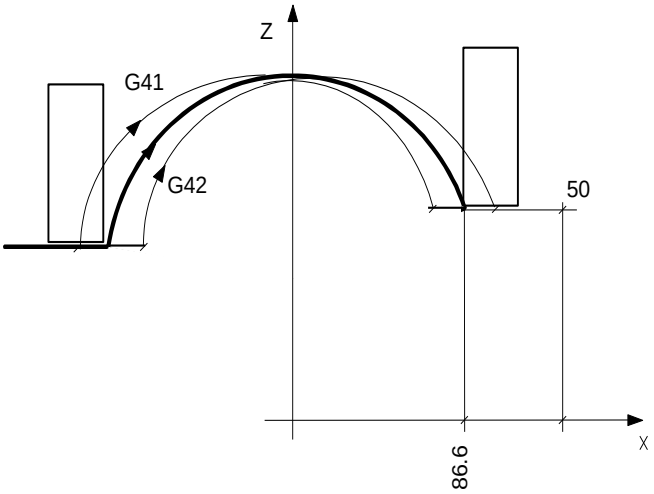


Program:
N88 G1 G40 X50 Z15
N100 G X.. Z..



Program:
N88 G1 G40 X50 Z95
N100 G X.. Z..

- **Last movement on profile is circular**



Program:
N99 G2 G40 X86.6 Z50 R100
N100 G X.. Z..

5.2.9 *Tool compensation change (during machining)*

This section describes how changes of compensation type (G41 → G42 and vice versa) are handled during offset profile machining. Compensation type changes may occur at the point of intersection of the programmed paths (with left/right or right/left compensation) or by the automatic addition of a new movement block by the system.

Type of compensation change (on the intersection or with an additional connection block) depends on type of the previous movements and of the subsequent movement. The different possibilities are discussed in the following pages:

- Linear/Linear with tangential movement blocks
- Linear/Linear with inversion of direction
- Linear/Linear with automatic generation of a new movement block
- Linear/Linear without automatic generation of a new movement block
- Linear/Circular - Circular/Linear
- Circular/Circular

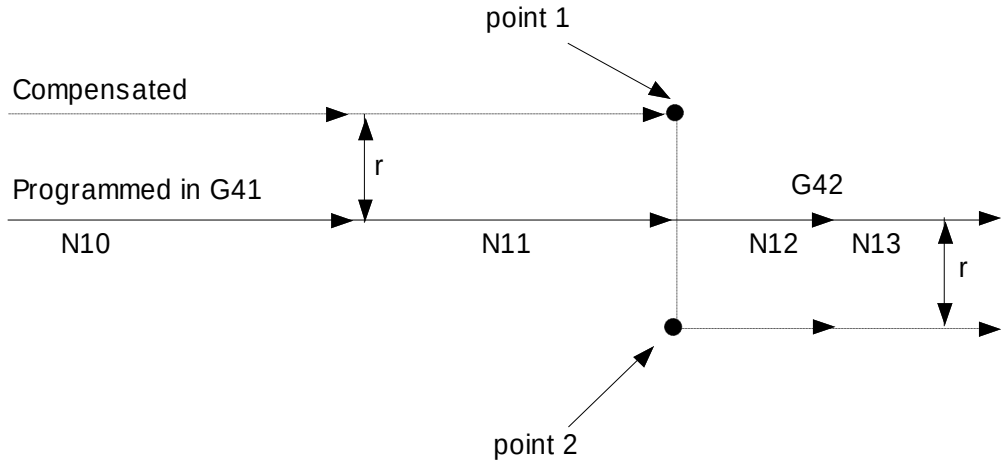
5.2.10 *Linear/Linear tool path*

The following figure illustrates the tool path when compensation changes from G41 to G42 during execution of two linear type movements.

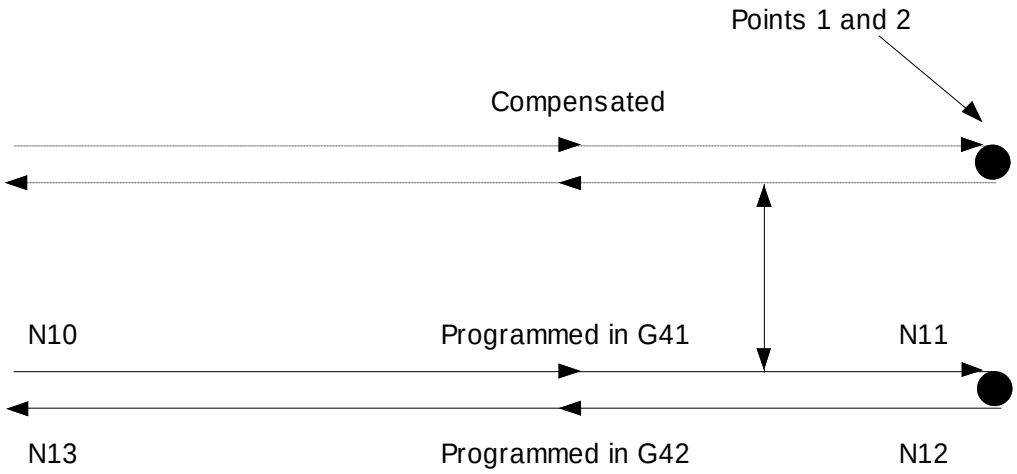
In changing from G41 to G42, the control generates two points, namely *point 1* and *point 2*.

- *Point 1* is the final position of the tool before compensation type change.
- *Point 2* is the desired starting position for the first block using the changed compensation direction.

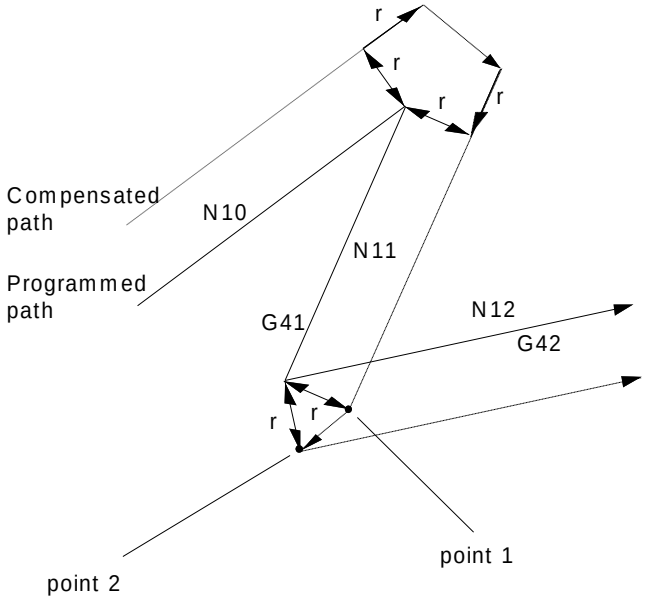
The control automatically generates the movement block connecting point 1 with point 2:



Linear/Linear change with tangential movement blocks



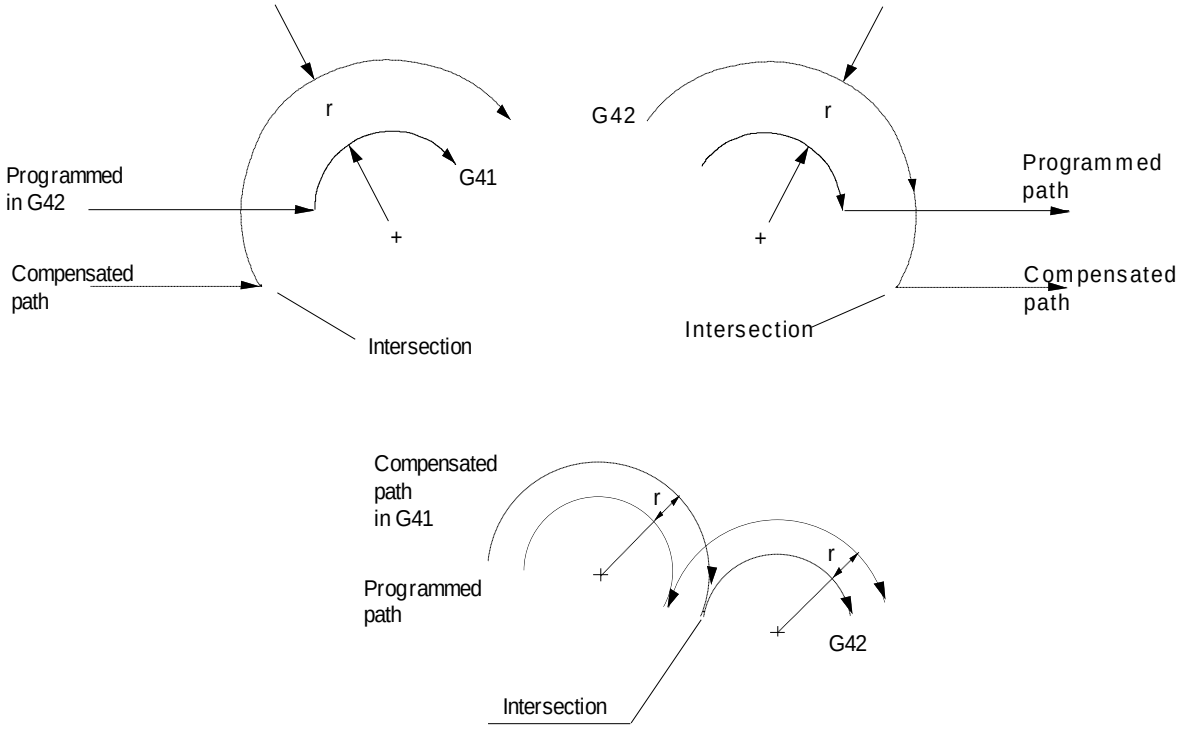
Linear/Linear change with inversion of direction



Linear/Linear change with generation of a new block

5.2.11 *Linear/Circular, Circular/Linear, Circular/Circular tool paths*

For each of the following types of tool path, in which a change of compensation direction occurs, the OPEN*control* system will try to find a point of intersection between the path programmed in G41 and that programmed in G42 (or vice versa).
If OPEN*control* finds a point of intersection, it modifies the *final point* of the original compensated tool path whereas the *starting point* of the new compensated tool path will coincide with the intersection (see figure below).

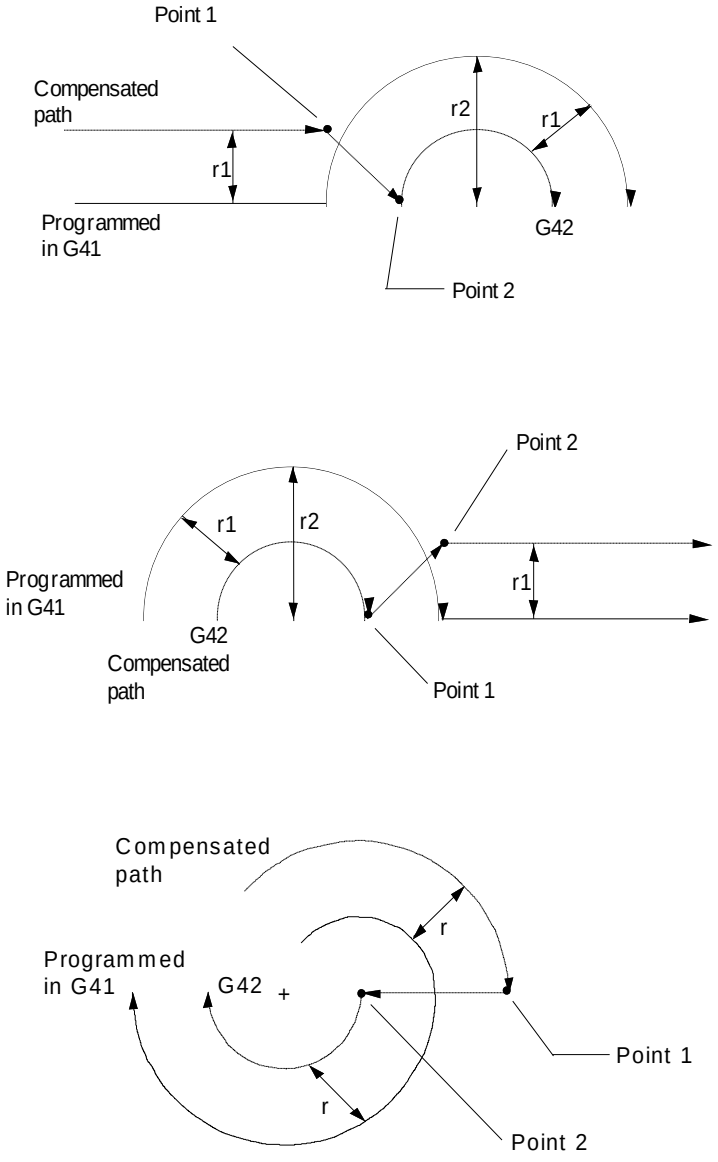


Change of compensation with intersection of current path

However, cases may arise in which there is no intersection between the tool paths; in these cases, when changing from G41 to G42 (or vice versa) the system behaves as illustrated in the figures that follow.

- *Point 1* is the final position of the tool before the change of compensation type
- *Point 2* is the desired position for start of the first block using the changed compensation direction.

The control automatically generates the movement block connecting point 1 with point 2:



Change of compensation without possibility of intersection between the tool paths

5.3 r – Radiuses in compensated profiles

When machining convex profiles, you may want a circular radius between geometric elements.

Syntax

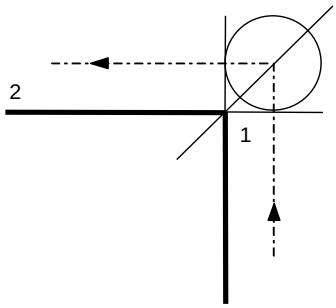
r *value*

where:

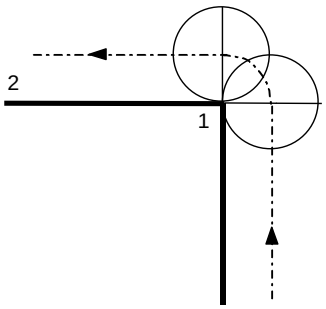
value The radius to be programmed. For clockwise moves program a negative radius; for counter clockwise moves program a positive value.

Programming r0 (radius equal to zero) causes the tool centre to follow a circular arc whose centre is on the profile corner.

Example:



Without radius
1) N20 G1 X100 Y100
2) N21 X-100



With radius
1) N20 G1 X100 Y100
N21 r0
2) N22 X-100

5.4 b – Bevels in compensated profiles

By programming the b address you can insert a bevel between two linear or circular motion blocks that generate intersecting paths.

Syntax

***b** value*

where:

value bevel length measured from the intersection point.

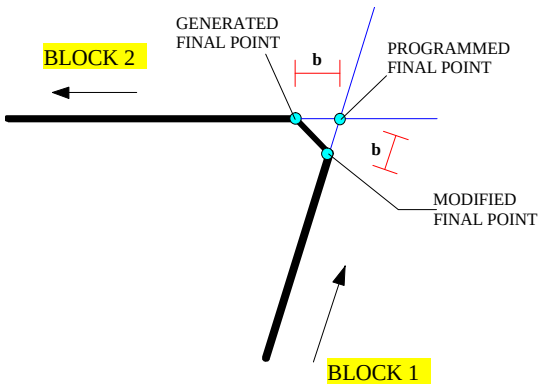
This value **b** may be interpreted as follows:

- **Bevel between two linear profiles:** ***b*** is the distance from the generated final point to the theoretical intersection point between the extended profile segments.

- **Bevel between a linear and circular profile:** ***b*** is the distance from the theoretical intersection point between the extension of the linear profile to the tangent of the circular profile.

- **Bevel between two circular profiles:** ***b*** is the distance to the theoretical intersection point between the extensions of the tangents to the circular profiles.

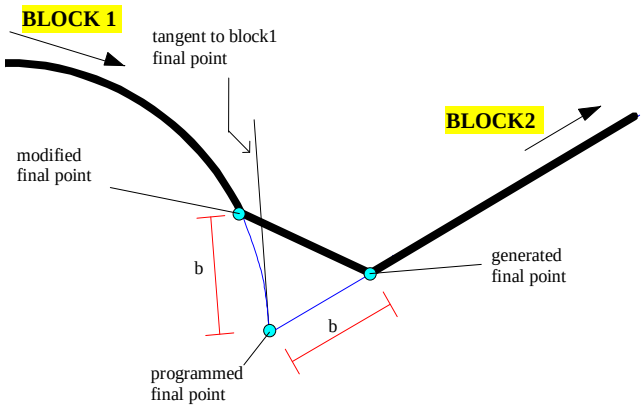
Examples:



Bevel between two linear blocks

Programming example:

```
        .
        .
N10 G1 X10 Y100     ;block 1
N11 b3
N12 X-100           ;block 2
        .
        .
```

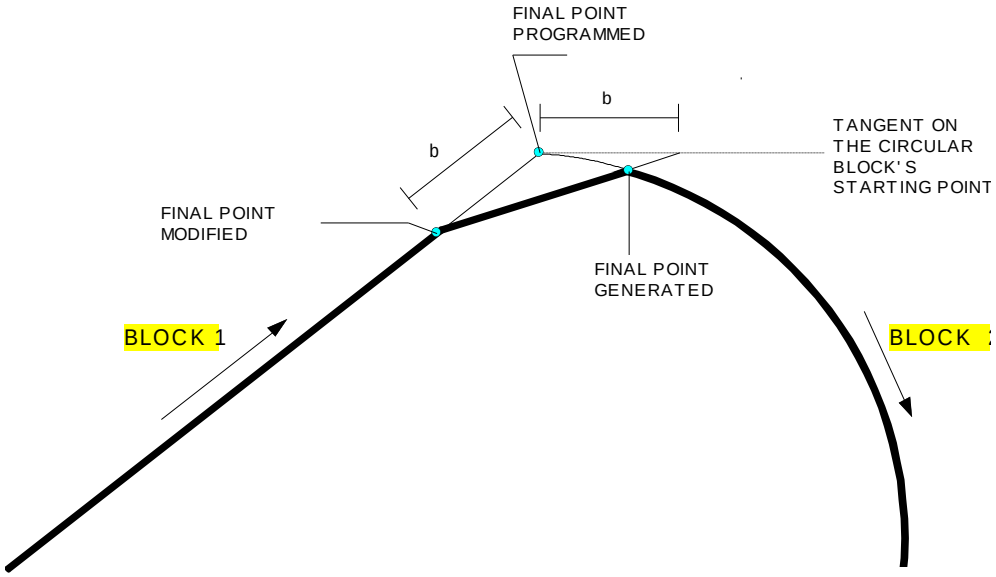


Bevel between circular and linear motion blocks

Programming example:

```
.
N10 GXY
N11 G42
N12 G1X10Y70F1000
N14 G2X70Y20R50      ;block 1
N16 b10
N18 G1X150Y90        ;block 2
N20 G40
.
.
.
```

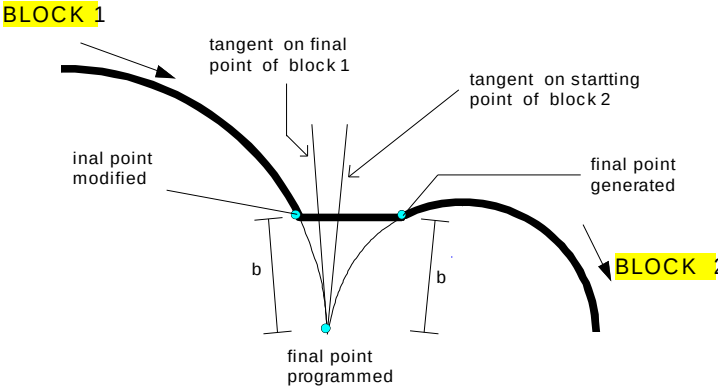
Bevel between linear and circular motion blocks



Programming example:

```
.
N20   G42
N22   GXY
N24   G1X40Y40F1000        ;block 1
N26   b10
N28   G2X80Y5R70           ;block 2
N30   G40
.
```

Bevel between two circular motion blocks



Programming example

```
.
.
N10   G42
N20   GX10Y60
N30   G2X50Y40R50F1000      ;block 1
N40   b5
N50   G2X100Y5050R30        ;block 2
.
.
```

5.5 IBSIZE – Interference correction in G41/G42

The process variable IBSIZE (Interference Buffer Size) activates interference correction when cutter compensation is working.
When interference correction is enabled, the algorithm will try to correct the error of wrong direction on profile.
The profile, after correction, will differ from the programmed one, but the system will continue part program execution and the work piece will not be damaged.

Syntax

IBSIZE = *n*

where:

n Dimension of the buffer used for correction.
The following table shows the accepted values for n. If *n* is 0 the correction algorithm is disabled.
Default value for IBSIZE is 0, this value can be configured in AMP, using the ODM tool.

The RESET command restores default value.

n	Function
0	Correction algorithm is disabled
3	Correction algorithm is enabled. Interferences requiring removal of no more than one movement can be corrected
4	Correction algorithm is enabled. Interferences requiring removal of no more than two movements can be corrected
5	Correction algorithm is enabled. Interferences requiring removal of no more than three movements can be corrected

Characteristics:
Programming a block containing the instruction IBSIZE=n enables the algorithm of interference correction.
An interference is defined as a compensated movement with tool path direction reversal with respect to the programmed direction.
The interference correction increases execution time of a part program, since it is necessary to pre-compute some movements in order to identify and to correct interference.
The increased time is not fixed, it depends on the IBSIZE value and on the tool path optimisation value.

During the disabling of compensation, interference correction is always disabled. So if there interference appears in this phase, then part programs execution stops with an error.

The algorithm disables TPO when trying to solve the interference. As a consequence it is possible that more elements are removed than are strictly necessary to solve the interference.
Once the interference is solved, TPO, if present, is restored and the correction movement is computed with TPO enabled.

The interference correction feature modifies the number of axis movements that do not belong to the profile, that can be programmed when compensation is working.
Moreover programmed bevels (and radiuses), when interference correction is running, are considered to be movement of axes that do not belong to the profile.

The number of allowed movements will be:

IBSIZE=3

Four axis movements that do not belong to the profile between four compensated movements.

IBSIZE=4

Two axis movements that do not belong to the profile between six compensated movements.

IBSIZE=5

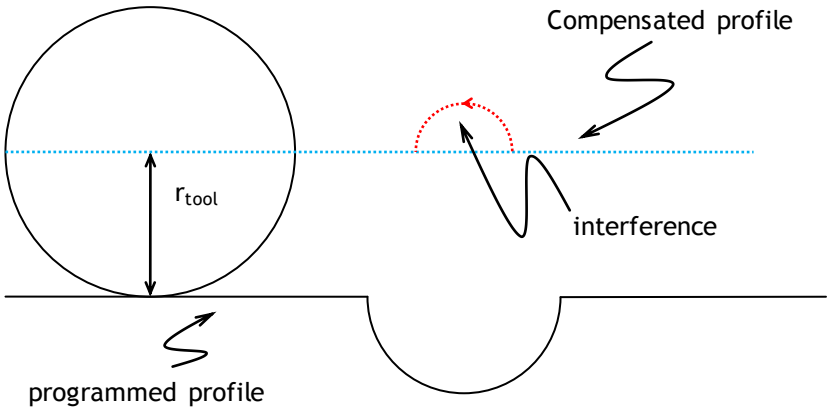
No axis movements that do not belong to the profile are allowed

Interferences belong to two different classes:

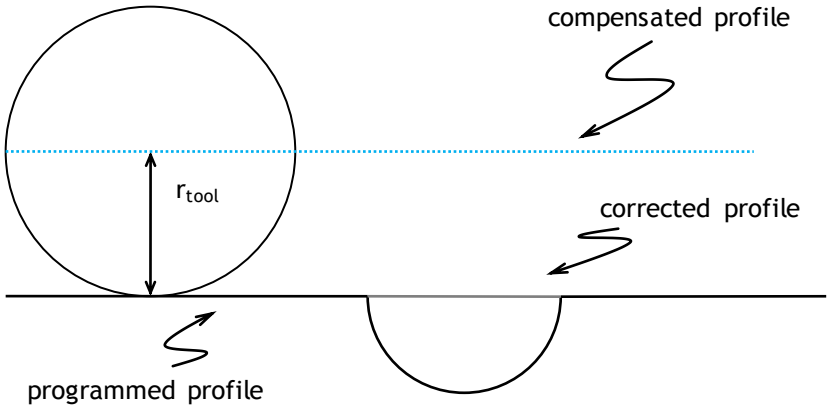
- Interference generated by a single circular movement
- Interference generated by a combination of linear and/or circular movements

5.5.1 *Interference generated by a single circular movement*

This kind of interference appears when circular movement is travelled internally (path is concave) and its radius is smaller than tool diameter.
In this case the standard compensation algorithm would generate a profile inconsistent with the one programmed.



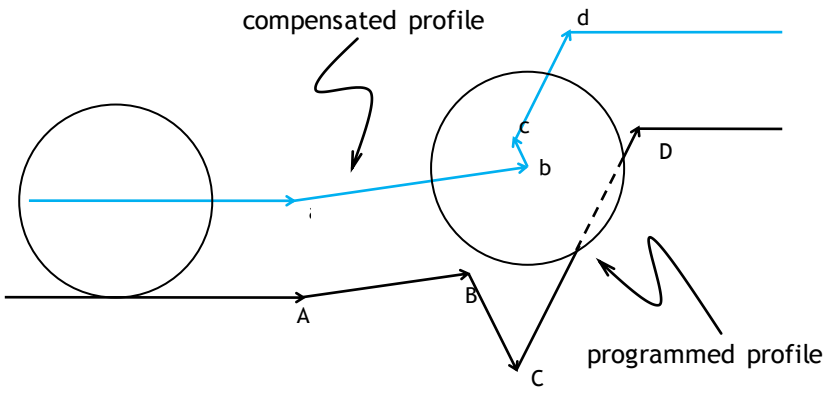
When interference correction is enabled (IBSIZE different to 0), the correcting algorithm replaces the circular movement with a linear movement having the same start and end point of the programmed (circular) movement.



The resulting profile is different from the programmed one, but the system can continue the execution of the part program and the work piece is not damaged.

5.5.2 *Interference generated by a combination of linear and/or circular movements*

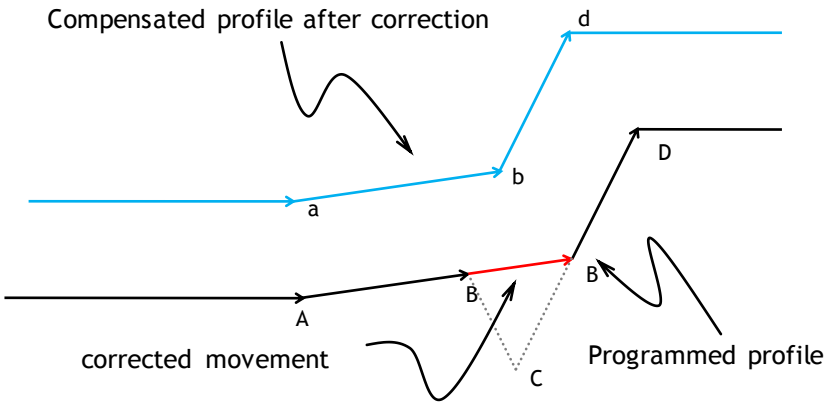
Interference can happen due to a combination of linear and circular movements whose length is near to the tool radius value.



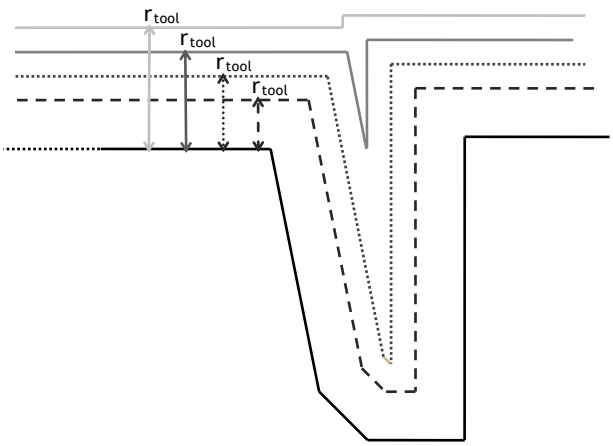
In this case the direction of bc movement (compensated) is inverted with respect to the direction of movement BC (programmed), moreover the tool, moving to b, damages the work piece.
When the interference correction is enabled, the algorithm identifies the interference and tries to solve it before the tool arrives at b
The algorithm looks for an intersection between the movement being analysed and the preceding ones.
Referring to the previous image, the interference is identified during the analysis of the movement CD.

The algorithm assumes that the movement that caused the interference is the previous one (BC) and it removes it from the profile, then looks for an intersection between the movement being analysed (CD) and the movement preceding movement BC (AB). If it can find an intersection, then interference is solved and part program execution can continue.

If there is no intersection, then the algorithm looks for other movements in the interference buffer (whose dimension corresponds to the IBSIZE value), if it finds a movement it continues to look for an intersection, if the buffer is empty then the algorithm was unable to solve the interference, the system stops the execution of the part program and it emits an error of inverted direction.
The algorithm will solve the interference shown in the previous figure as follows:



In this case removing one movement will solve the interference.
There are other cases where it will be necessary to remove more than one movement to solve the interference.
For instance in the following profile as the tool dimension increases one, two or three movements shall need to be removed.



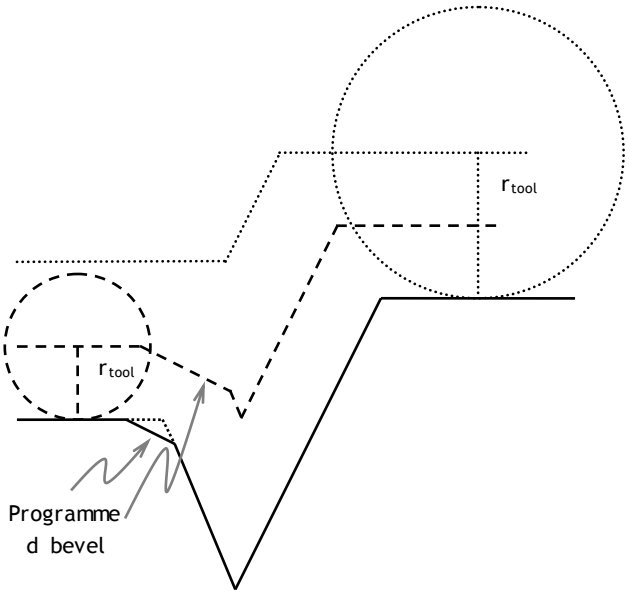
The first compensated profile (for r_{tool1}) does not require correction since there is no interference.
Starting from the second value of tool radius (r_{tool2}) interference correction has to be enabled.
In this case the intersection between element D and element E reverses the direction of the movement.
Removing element D and creating the interference between element C and element E solves the interference.

Starting from r_{tool3} just removing D will not solve the interference since the interference between C and E generates interference, too. IBSIZE value has to be bigger than 3 since at least two elements will be removed; in fact the interference between B and E is the first that does not generate interference.

Starting from r_{tool4} IBSIZE value has to be 5: in this event the intersection between B and E generates an interference and, to solve the interference, the entire "hole" corresponding to B,C,D will be skipped and A will be connected directly with E.

In all the examples the final profile differs from the programmed one, but the system can continue program execution and the work piece is not damaged.
If a programmed bevel (or a radius) is connected to a movement that has been removed during the correction phase, then the bevel (radius) will be removed, too.

This is a consequence of the removal of one of the movements the bevel (radius) refers to.



During the creation of the corrected movement the features of TPO=64 (circular radius movements on internal bevels) is disabled.

TPO is disabled if the corrected movement implies the intersection between two compensated elements that do no intersect on the programmed profile.
This happens because it is impossible to define a starting point for the second movement of the intersection. If this point does not exist optimisation behaviour is unpredictable.

During correction only movements on the profile are removed. If a block containing a removed movement contains PLC commands (e.g. M functions), these functions will be executed before or after the correction movement. M functions of "expedite" type will not be executed since the movement they refer to has been removed.

5.6 TPO – Tool Path Optimization with G41/G42

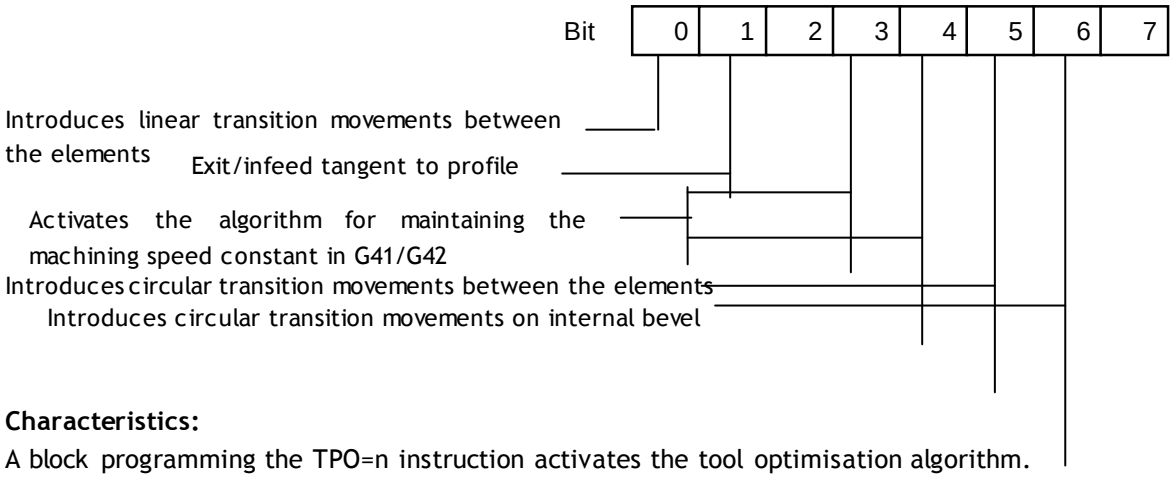
TPO (Tool Path Optimisation) allows optimisation of the tool path when G41 or G42 are active.
The algorithm automatically introduces circular interpolation at profile start and end, and linear or circular radiuses on the profile bevels.

Syntax

TPO = *n*

where:

n Optimisation mode. Allowed values are:
The value to be specified for *n* in the instruction is obtained from the sum of the decimal weights corresponding to each of the desired features.
If *n* = 0 optimisation algorithm is disabled.



Characteristics:
A block programming the TPO=n instruction activates the tool optimisation algorithm.
This algorithm can automatically introduce movements at the beginning/at the end of the profile and it can optimise the path on corners. Depending on the profile, the algorithm could automatically introduce from 1 to 3 optimisation moves.

At power up the system is enabled with the TPO mode configured in AMP. This optimisation mode may be altered from the program.
RESET restores the TPO mode configured in AMP.

The activation of bit 5 and/or of bit 6 works only if bit 0 is also enabled, i.e. when you want to introduce a circular transition in lieu of a linear one, set **TPO = 33**.

If you want to introduce a circular transition on the inner corners of a profile, set **TPO = 65**

5.6.1 *TPO=1, TPO=33 and TPO=65 modes*

Mode TPO=1 activates the tool path optimisation algorithm by introducing linear transition movements between the profile elements as a function of the angular deviation between the elements themselves. Furthermore, if bit 5 has also been set (TPO=33), then, provided that the conditions for corner optimisation are met, through the introduction of linear movements a circular transition is introduced, having its centre at the intersection of the two non-compensated elements and a radius corresponding to tool radius.
If bit 6 is active too (TPO= 1+32+64=97), the above considerations are applied to the inner corners, too.

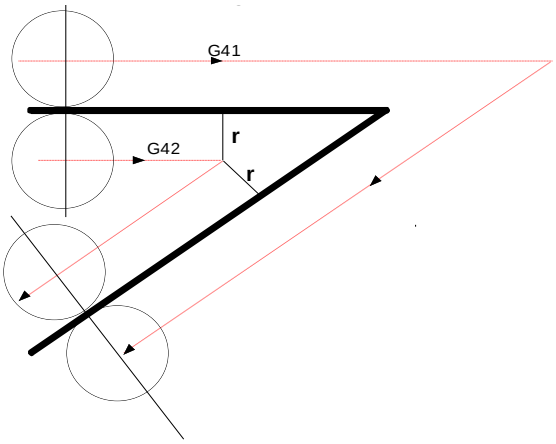
The cases of application of the algorithm are as follows:

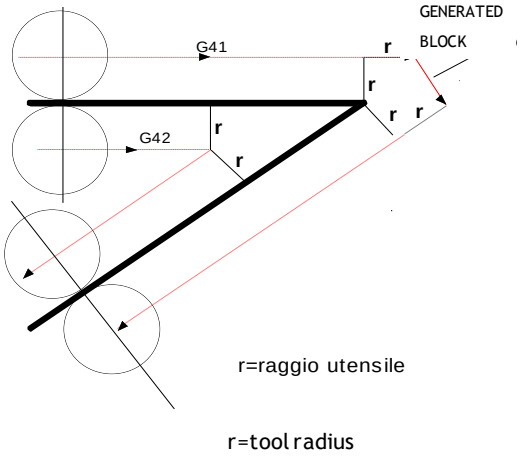
A) Right angle-right angle with angle beta of deviation between 0° and TPA (0° < β < TPA°), with variable TPA, which is set by default to 90°

B) Circle-right angle, right angle-circle, circle-circle with angle of deviation between 0° and 180° (0° < β < 180°).

If either a bevel (b) or a radius (r) has been programmed between the elements described in A) and B), the algorithm is not applied.

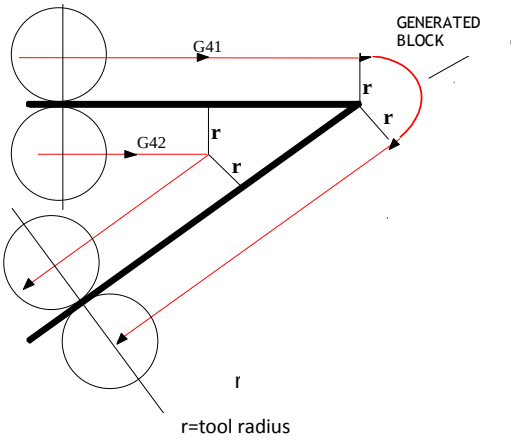
Example:
In this example, β is smaller than 90°

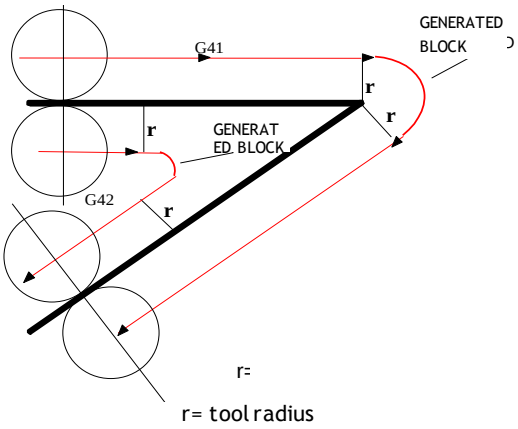




Without optimization (TPO =0)

With optimization (TPO=1)



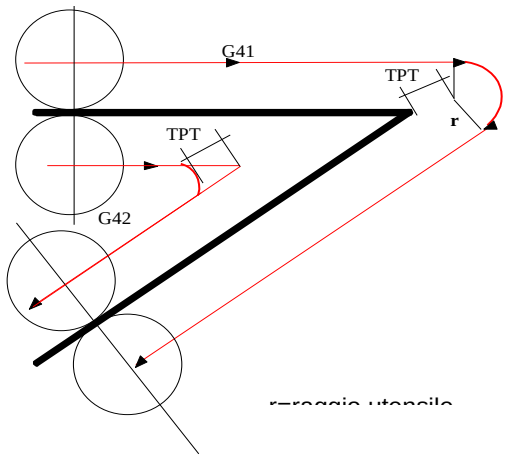


With optimization (TPO=33) *With optimization (TPO=97)*

Note that in the above examples TPT threshold has been ignored, as if TPT=0.
The figure below illustrates the strategy of TPO when TPT is not zero, i.e., a threshold deviation from the corner is programmed:

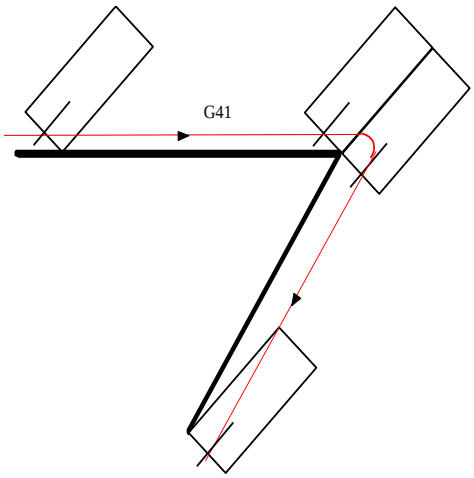
- in the case of an **outer corner**, the afore-mentioned circular radius will have its centre translated according to the same calculation criteria as are used for linear radiuses, and the radius will be reduced accordingly.

- in the case of an **inner corner**, the radius of the circular radius is calculated so as to have the tool path deviate from TPT by the amount that would be obtained without the introduction of the radius



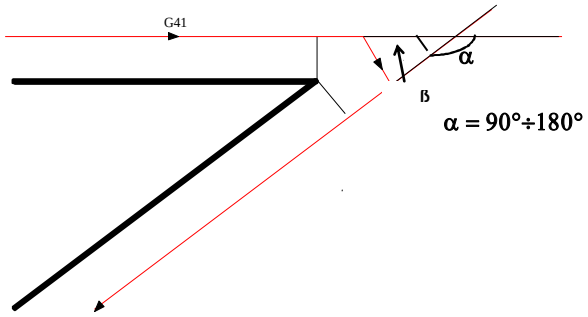
r=tool radius

Using cutter thickness compensation (TTC,1), the circular transition inserted when TPO = 33 does not depend on TPT.

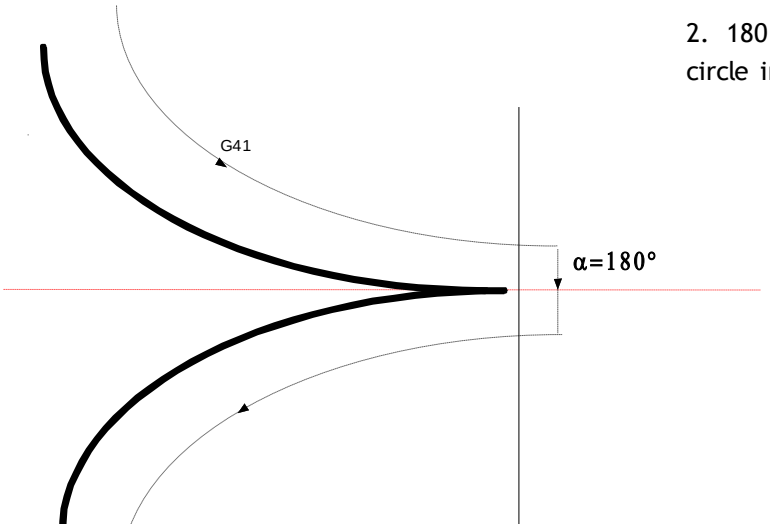


5.6.2 *Examples of profile optimisation with TPO=1*

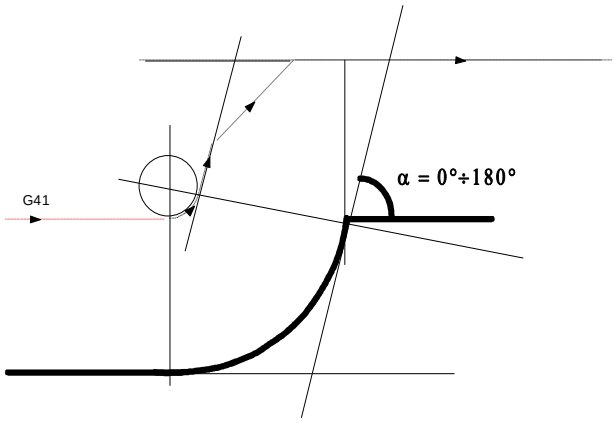
In these profiles the angle deviation requires an optimisation algorithm.



1. Angles from 0° to 90° formed by two straight lines



2. 180° angle with circle-circle intersection

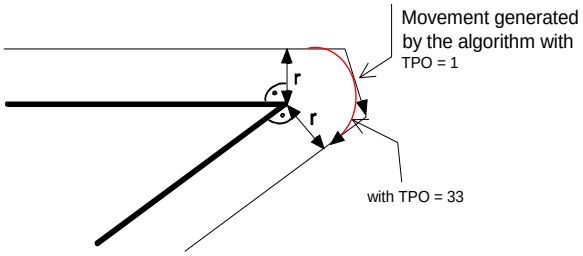


3. Angle from 0° to 180° formed by a circle-straight line intersection

5.6.3 *Examples of profiles where algorithm introduces optimization moves*

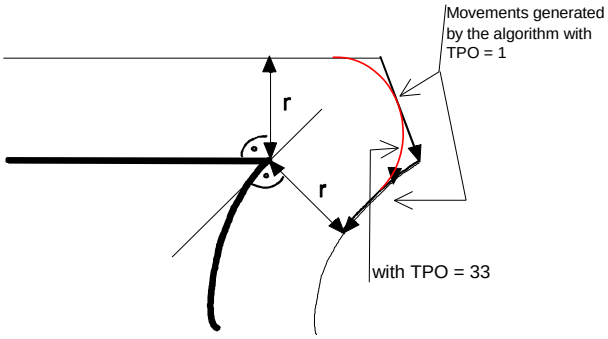
In these examples the algorithm recalculates the profile and introduces from 1 to 3 optimisation moves between bevels.

1) Line - line



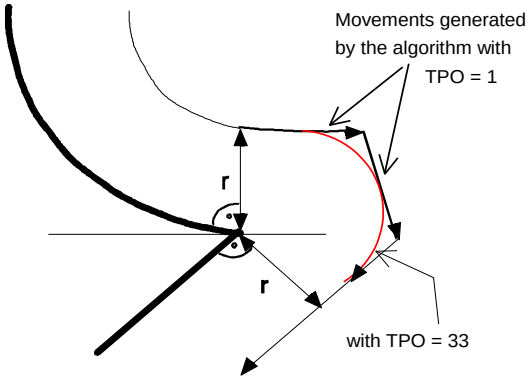
The algorithm generates 1 move

2) Line - circle



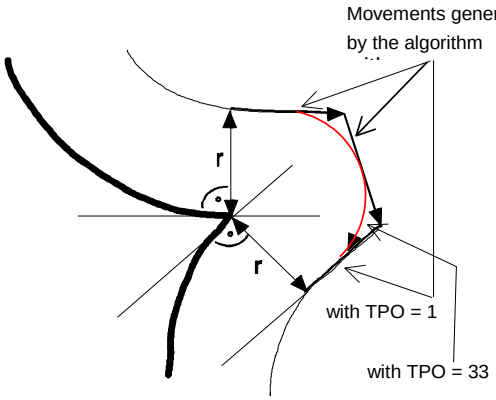
The algorithm generates 2 moves

3) Circle - line



The algorithm generates 2 moves

4) Circle - circle



The algorithm generates 3 moves

5.6.4 *Examples of profile optimisation with TPO=2*

TPO=2 enables an infeed/exit algorithm that keeps the tool tangent to the profile by introducing circular elements at profile start and end.

A) At profile infeed the algorithm generates a circle between the first point of the preceding point (P0) and the offset profile (P1). The first point is the one programmed in the G41 or G42 block.

```
.
X80                 ;P0
G41 X100 Y100       ;P1
X140                ;P2
.                   ;etc.
.
```

B) At profile exit the algorithm generates a circle between the last point of the offset profile (P99) and the last point of the exit element in the profile (P100). The exit element is the one programmed in the G40 block.

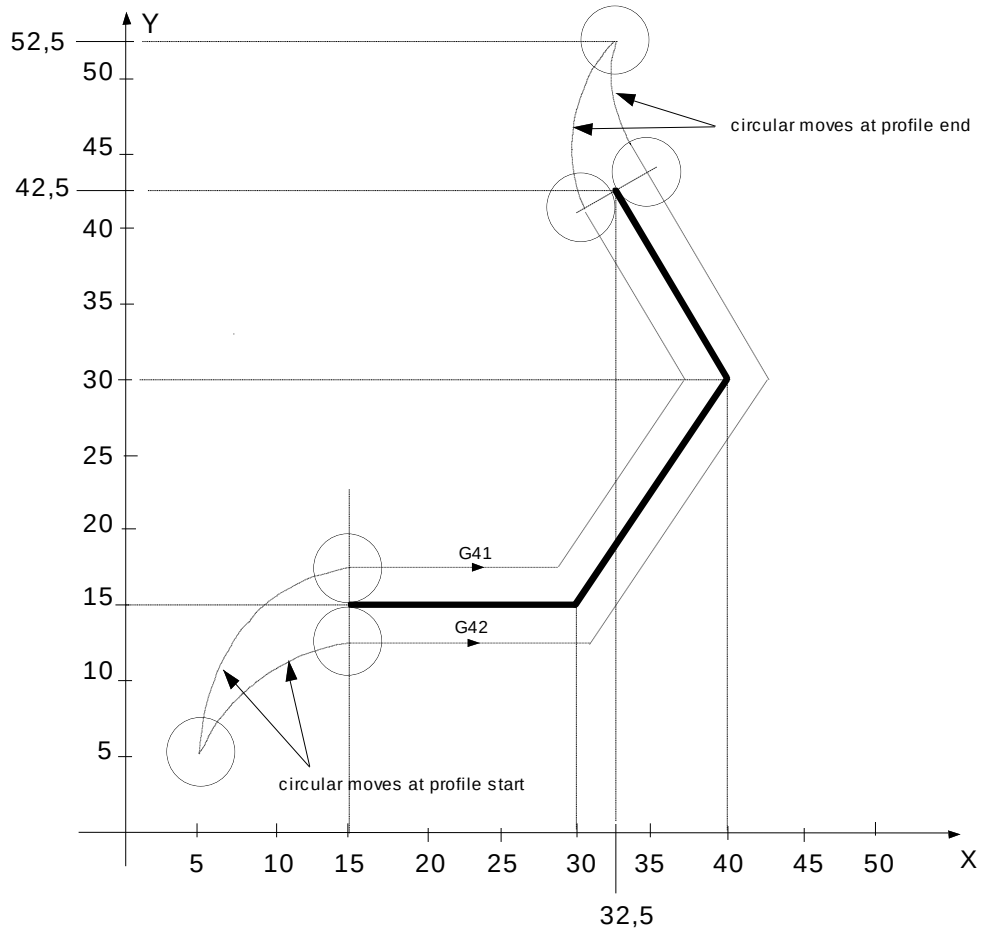
```
.

G41 X100 Y100       ;P1
.
.
X170 Y160           ;P99
G40 X180 Y185       ;P100
.                   ;etc.
  .
```

5.6.5 *TPO=3 mode*

TPO=3 simultaneously enables TPO=1 and TPO=2 algorithms, which introduces radiuses both between elements and circular moves at profile start and end.
The rules defined for TPO=1 and TPO=2 also apply to TPO=3.

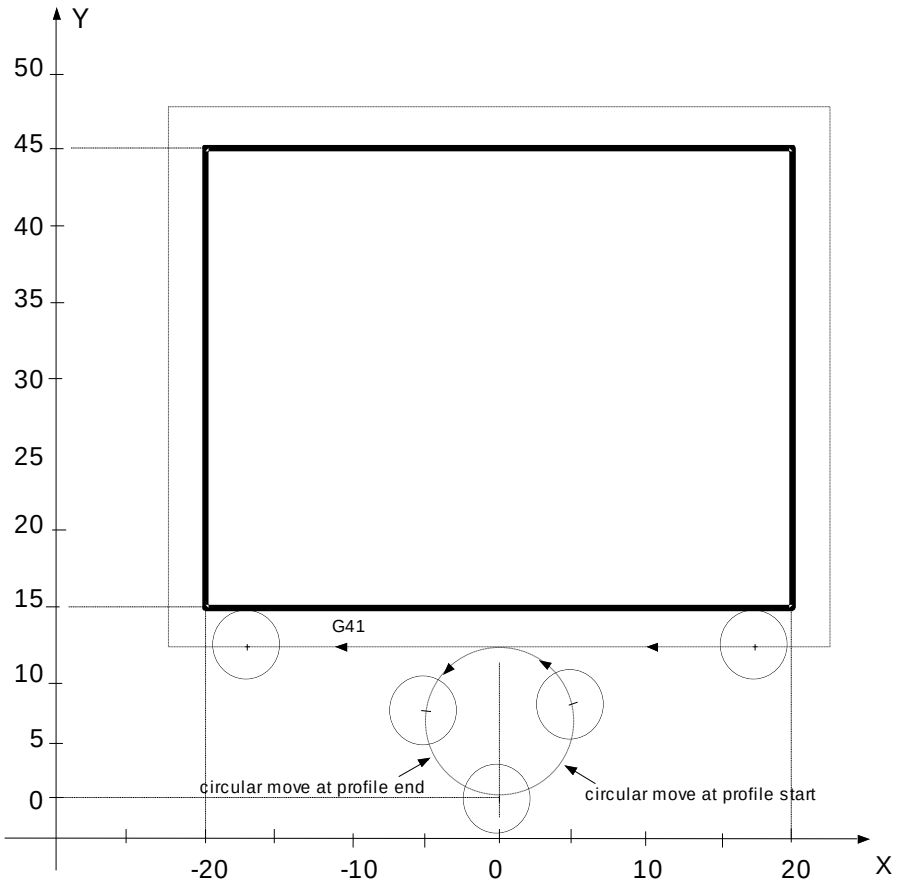
5.6.6 *Examples of TPO=2 mode*

Example 1:




```
N1    S1000 M3 T1.1M6          ;tool radius = 2.5
N2    X5 Y5
N3    G1 G42 X15 Y15 F500      ;first point in the
                               profile
N4    X30
N5    X40 Y30
N6    X32.5 Y42.5              ;last point in the
                               profile
N7    G40 X32.5 Y52.5
N8    GX100
```

Example 2:



N 1	S1000 M3 T1.1 M6	;tool radius = 2.5
N2	X0 Y0	
N3	G1 G41 Y15 F800	;first point in the profile
N4	X -20	
N5	Y45	
N6	X40	
N7	Y15	
N8	X0	;last point in the profile
N9	G40 Y0	

5.6.7 *TPO=8 and TPO=16 modes*

TPO=8 and TPO=16 modes activate the algorithm that keeps the machining speed in G41/G42 constant. In particular, the speed of contact between the tool and the part is kept constant, varying the feed rate with reference to the centre of the tool.

This variation is a function of the tool radius and is only applied to circular movements. It will produce an increase or decrease in the feed rate with reference to the centre of the tool depending on whether the radius of the circle moved by the centre of the tool is greater or less than the radius of the programmed circle. The feed rate is increased by setting bit 3 of variable TPO (TPO=8) while it is decreased by setting bit 4 (TPO=16).

Bits 3 and 4 of TPO may be set together and even at the same time as all the other bits of the same variable to enable the corresponding features.

5.6.8 *TPO=32 mode and 64 mode*

Is the same as having programmed TPO = 0.
This is so because the activation of external and inner bevel optimisation through the introduction of a circular radius requires the prior programming of TPO = 1. If bit 0 = 0 and bit 5 and/or bit 6 = 1 (TPO=32 and/or TPO=64), the optimisation algorithm by means of circular radiuses is not enabled.

5.7 TPA – Threshold angle for TPO

System variable that consists of the threshold angle within which two linear elements can be radiused in tool compensation mode (G41/G42).

Syntax

TPA = *value*

where:

value Threshold angle expressed in degrees. If the angle between two adjacent elements is smaller than the value of TPA, a circular or linear radius is introduced in accordance with the values set in TPO.

The default value of TPA is = 90°; this value can also be configured in AMP.

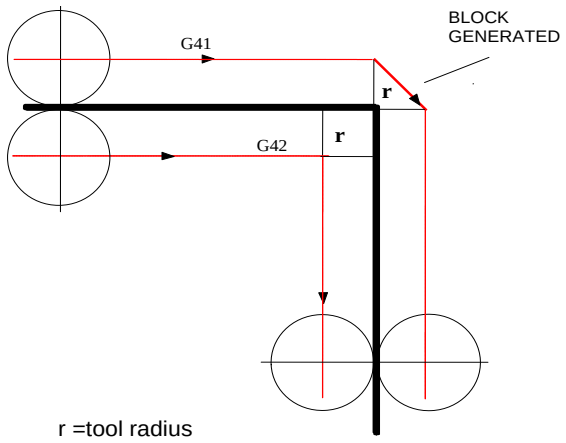
Characteristics

This variable has been introduced to allow configuration of the threshold angle used by the Tool Path Optimisation (TPO) algorithm to radius two linear elements in tool compensation mode. If at least one element is circular the optimisation algorithm remains unchanged, i.e., a radius (either linear or circular, depending on the value of TPO) is introduced regardless of the angle between the two elements.

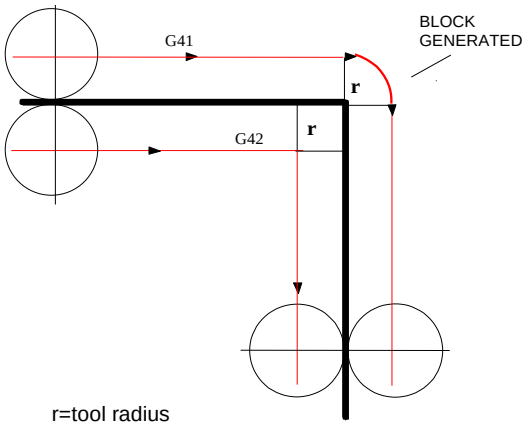
RESET restores default value.

Example:

With TPA > 90° and TPO = 1



With TPA > 90° and TPO = 33



5.8 TPT – Tool Path Threshold

This instruction specifies a threshold for bevels during tool path optimisation when TPO=1.

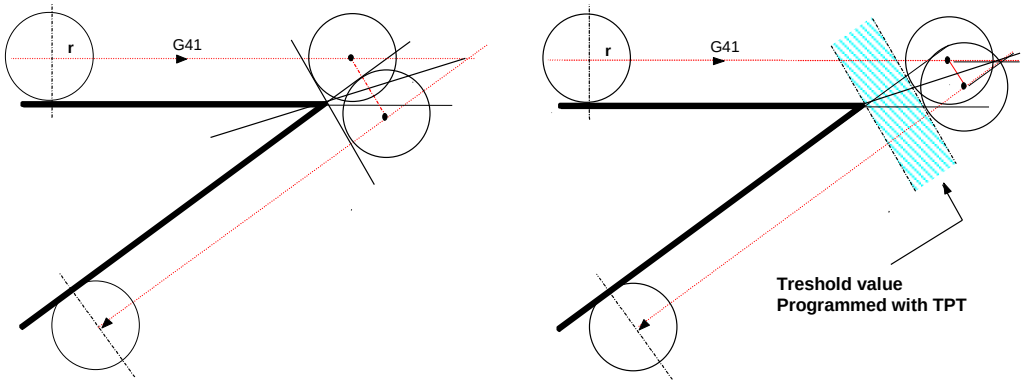
Syntax

TPT = *value*

where:

value Threshold expressed in the active unit of measure (mm/inch). It represents the distance between the tool cutter and the bevel generated by the programmed profile. It can also be configured in AMP.

Example:



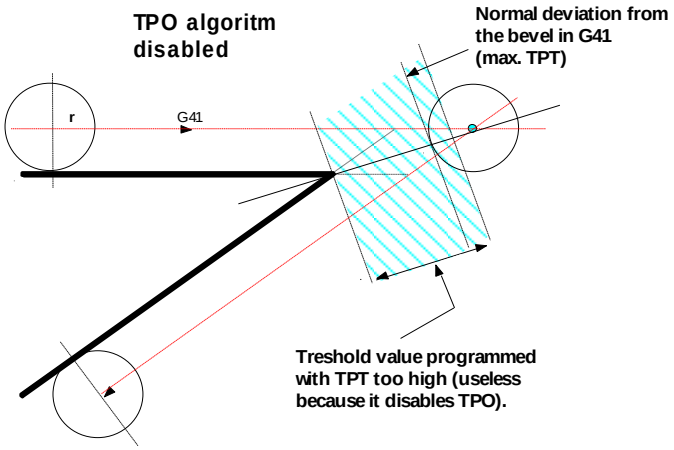
With TPT=0 *With TPT programming*

Characteristics:

The threshold programmed with TPT will be ignored by the system if it takes the mill centre beyond the theoretical point obtained through standard tool diameter compensation (G41/G42 without TPO programming).

In this case the TPO algorithm will be temporarily disabled, G41/G42 will be applied without TPO and there will be no error signal.

Example:



5.9 u v w – Paraxial compensation

When compensation factors u,v,w are programmed in a block, the control positions the axes to a point whose coordinates are equal to the programmed coordinates value minus a value equal to the compensated radius multiplied by the compensation factor, assuming the compensation factors are related to X, Y and Z axes.

X position = programmed X + (cutter radius * u)

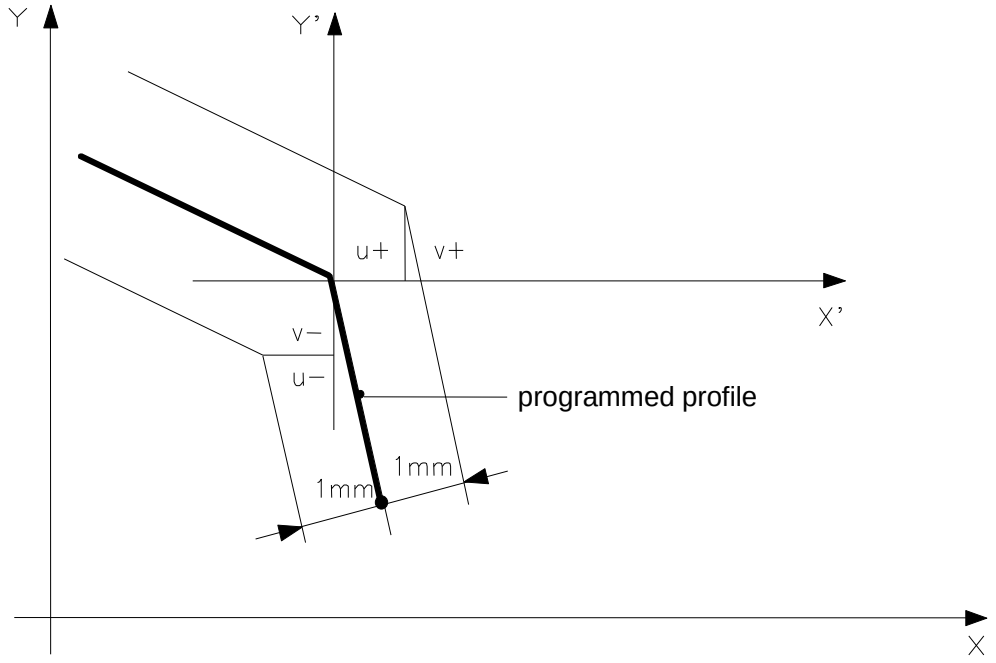
Y position = programmed Y + (cutter radius * v)

Z position = programmed Z + (cutter radius * w)

These compensation factors are used both for extremely simplified profiles (contouring parallel to axes) and for three-dimensional milling surfaces.

You cannot use the u,v,w factors when the control is in cutter compensation mode (G41-G42).
When factors u,v,w are negative, they must be followed by a minus sign. If they are positive, the plus sign can be omitted.

In order to determine the value and the sign you must consider paraxial factors u,v,w respectively as the X, Y, Z coordinates of the profile vertex corrected by a unit. X, Y and Z are referred to a system of Cartesian axes that are parallel to the axes of a machine whose origin is represented by the point to be compensated.



Determining the sign of u, v, w compensation factors

The unit vector represents a cutter radius offset that is one unit long; the control, the increments of translation are the product of the u,v,w values multiplied by the cutter radius.

You can compensate profiles that consist of:

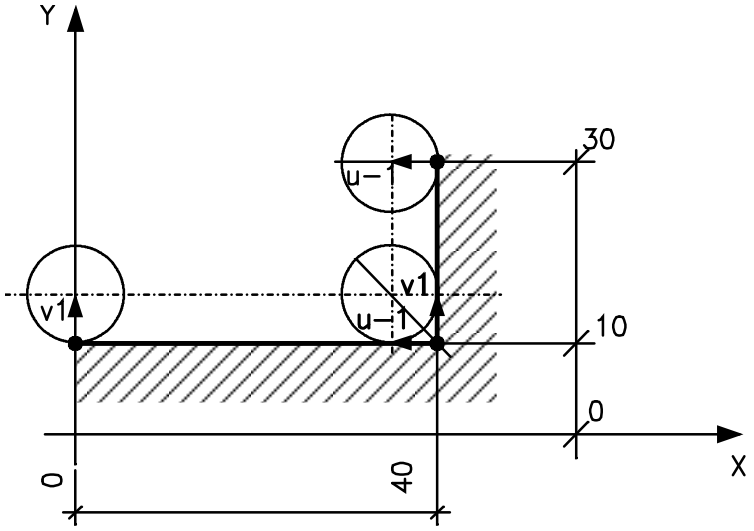
- straight line segments parallel to the axes or forming an angle with the axes
- straight line segments and arcs tangent to the straight lines
- tangent arcs (provided they are still tangent to one another after they have been translated on a parallel path).

The u,v,w factors are only valid in the block in which they are programmed. They are used by the control only if they are associated to the corresponding coordinates:

- u for the 1st configured axis (typically X)
- v for the 2nd configured axis (typically Y)
- w to the 3rd configured axis (typically Z).

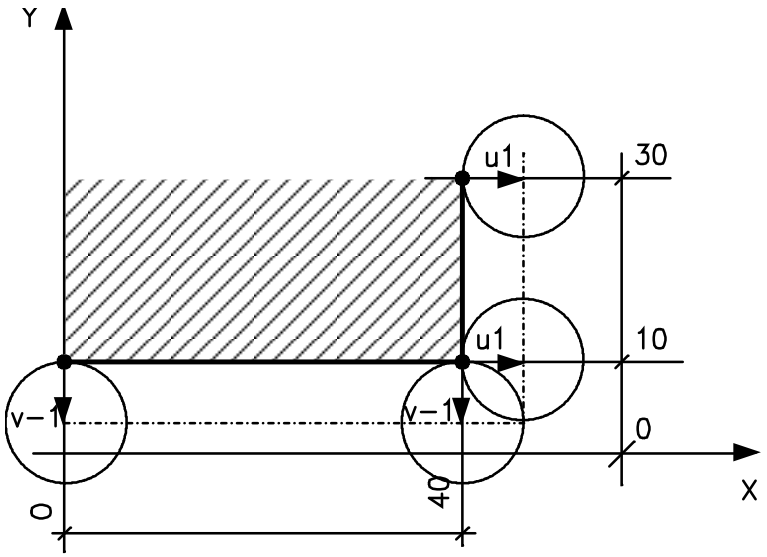
5.9.1 *Examples of compensation factor applications u, v, w*

Example1:



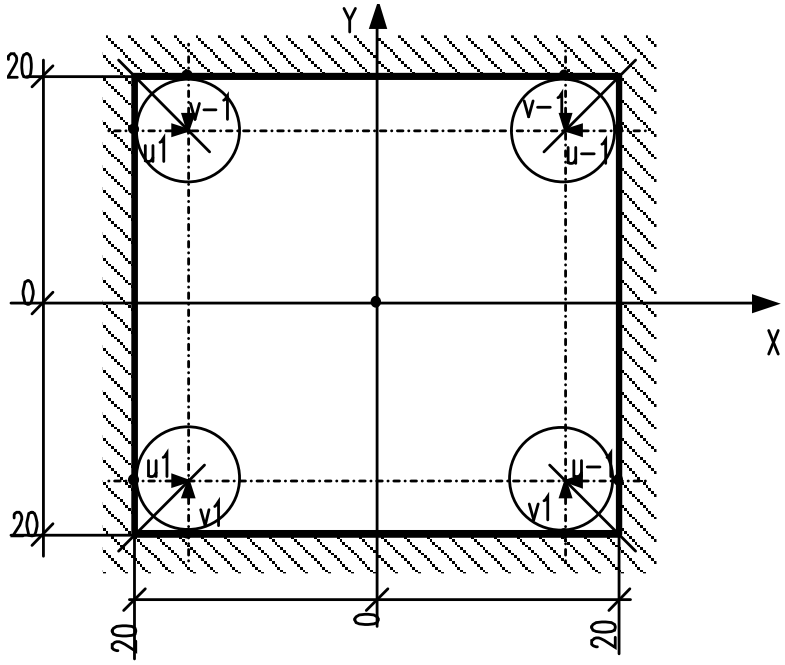
Program:
N5 T1.01 M6 S. . F. .
N6 G X Y30
N7 G1 Y10 v1
N8 X40 u-1
N9 Y30
N10 G X. . Y. .

Example 2:



Program:
N13 G X Y
N14 Y10 v-1
N15 X40 u1
N16 Y30
N17 G X. . Y. .

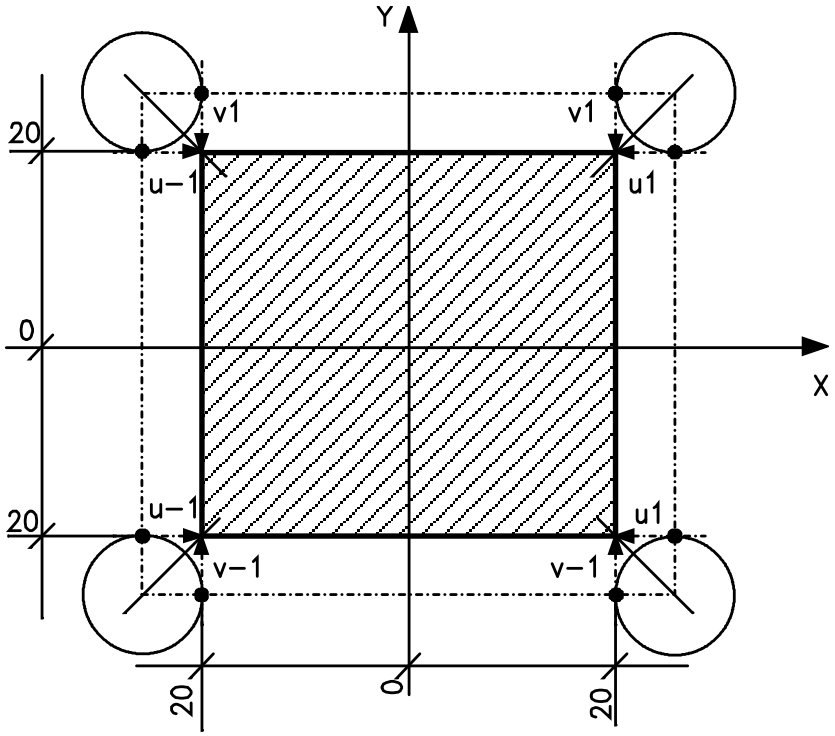
Example 3:



Program:
N13 G X Y
N14 G1 Z-10
N15 X-20 Y-20 u1 v1
N16 X20 u-1
N17 Y20 v-1
N18 X-20 u1
N19 Y-20 v1
N20 G X Y

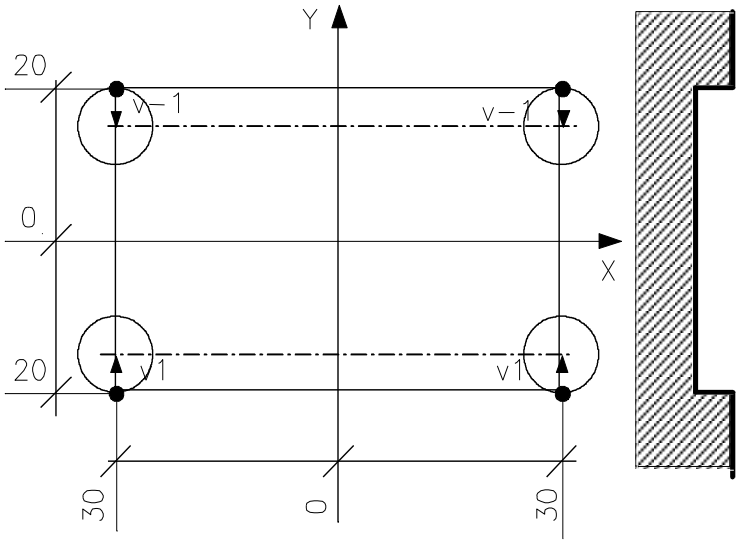
FIG57.PIC

Example 4:



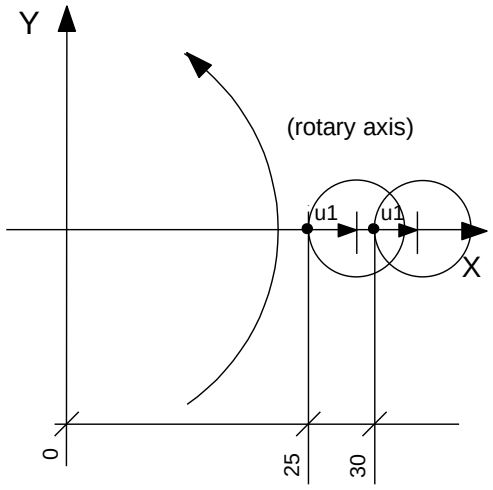
Program:
N13 G X-35 Y-35
N14 Z-10
N15 G1 X-20 Y-20 u-1 v-1
N16 X20 u1
N17 Y20 v1
N18 X-20 u-1
N19 Y-20 v-1
N20 GZ

Example 5:



Program:
N12 . . .
N13 G X-30 Y
N14 G1 Y20 v-1
N15 X30
N16 Y-20 v1
N17 X-30
N18 . . .

Example 6:



Program:
N12 G X40 Y B0
N13 G1 X30 u1
N14 X25 B360 u1
N15 G X40

5.10 U V W – Definition of axes for paraxial compensation

It is possible to change the axes on which paraxial compensation is performed.
As a rule, paraxial compensation is applied, within each process, on the first three axes configured (the first three to appear on the screen). With triliteral (**UVW**) the axes on which paraxial compensation is performed may be changed.

Syntax

(UVW, *axis1 axis2* [*axis3*]**)**

---

where:

axis1…axis3 are the names of the axes to be associated with the **uvw** paraxial compensation: axis 1 is associated with parameter **u**, axis 2 with parameter **v**, and axis 3 with parameter **w**.

Characteristics:
If only two axes are specified in the command, the paraxial compensation will be activated only for parameters **uv**, and the third parameter (**w**) will be ignored.

Once defined, the association between the axes and the uvw parameter remains active until the next RESET; then the default association (the first three axes configured) is restored. A GTA command will also restore the default association, unless all the axes (identified by axis ID) on which the uvw parameters are active remain present even after the execution of the command.

Example:

```
N10 (UVW,XYZ)              Activates uvw on X Y and Z respectively
N20 h1
N20 X10u1Y20Z40
N30 (DAN, XP, YQ, ZD)
N40 (UVW,XYZ)              Activates the uvw parameters on PQD (DAN renamed them in XYZ)
N50 X10u1Y20v0.3Z40w0.3

N60 X0u1Y5v0.50Z4w-0.5
```

5.11 MSA – Defining a Machining Stock Allowance

The MSA command defines the value of the machining stock allowance along the profile. It is used in roughing and pre-finishing cycles. MSA can be programmed in a block, assigned in MDI or entered via softkey.

Syntax

MSA = *value*

---

where:

value It can be a decimal number or an E parameter. It is programmed in the measuring unit (G70-G71) that is currently active in the program.

Characteristic:

The system uses the programmed MSA to calculate the offset value to be applied to the profile when cutter compensation (G41 or G42) is active.
The offset value is the sum of the tool radius and the MSA.

Example:

MSA	= 0.5	Assigns a 0.5 machining allowance
E30	= 1.5	
MSA	= E30	Assigns a 1.5 machining allowance

When measuring unit changes from G70 to G71 or vice versa, the programmed value for MSA is set to zero.

6. VIRTUALIZATION and 3D TRANSFORMATIONS

The virtualization and 3D transformation functions can be identified as a range of features allowing the management of the real axes of a machine tool in any three-dimensional space.
Virtualization is used if the real axes of the machine tool do not correspond to the orthogonal Cartesian axes. The 3D transformation functions allow definition of a roto-translated Cartesian reference system compared to that of the real machine or the direct programming of movement of the tool tip without having to take account of the machine kinematics

6.1 Virtualization available

COMMAND	FUNCTION
UVP	Polar coordinates programming
UVC	Cylindrical coordinates programming
UVA	Non-orthogonal axes programming

6.1.1 *UVP – Polar coordinates programming*

The UVP command allows programming of a machine having a polar reference system (one linear axis and one rotary axis) as if it was equipped with two orthogonal axes.

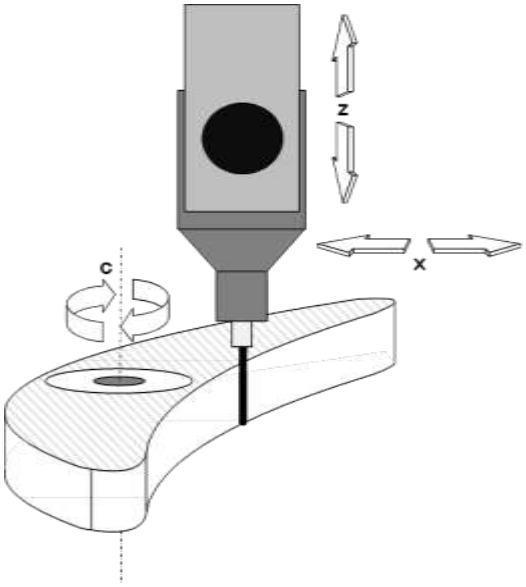
Syntax

(UVP, *af1af2*, *av1*,*av2*, *r***)**
(UVP)

where:

af1	Name of the physical linear axis
af2	Name of the physical rotary axis
av1	Name of the abscissa virtual axis
av2	Name of the ordinate virtual axis
r	Minimum radius that the tool path must not enter. It can be specified directly using a real number or a variable
no parameter	Disables the UVP mode

Characteristics:
This type of virtualization is used to move one linear axis (X) and one rotary (C) by programming the positions on a Cartesian plane (UV); the position of a generic pair of coordinates (U,V), in the virtual plane is represented by the physical axes coordinates (X,C).



The r parameter (minimum radius) must be programmed considering the programmed feed rate, in order to avoid the rotary axis being required to move at a feed higher than the rapid.

The minimum radius is calculated using the formula below:

$$r = \frac{F}{V_{max}} * \frac{360}{2\pi}$$

where:

r minimum radius

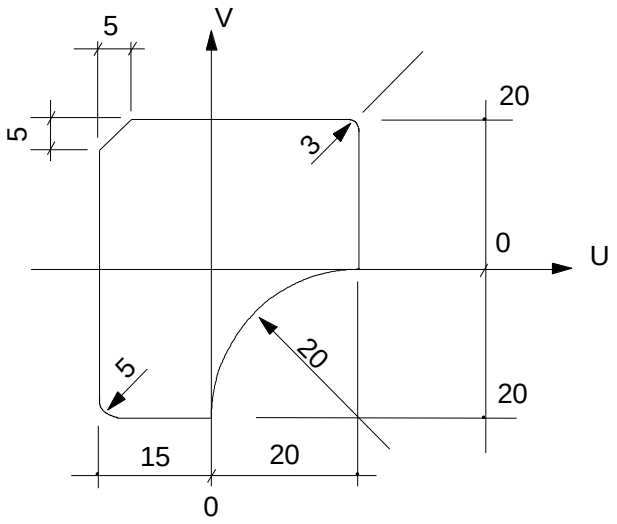
f programmed feed rate (mm/min or inch/min)

$\mathbf{V_{max}}$ rotary axis rapid feed

The limitation on the feed rate is performed dynamically to allow traverse at high speed when working with points distant from the machining centre.

Example of programming in polar coordinates

E0=110*180/(3.14159*800)
T1.1M6
S1000M3
GC0X50Y0
(UVP,XC,UV,E0)
G16 UV
G1G42U20VF110
V20
r3
U-15
b5
V-20
r5
U0
G40G2U20V0I20J-20
(UVP)
GX50



6.1.2 *UVC – Cylindrical Coordinates programming*

The UVC command allows to program a machine having a cylindrical reference system as it was equipped with two orthogonal axes.

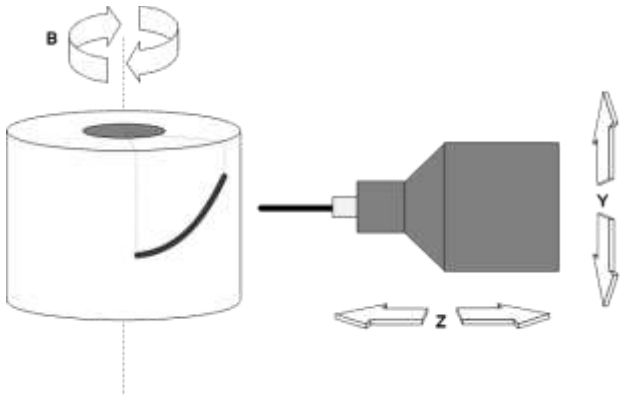
Syntax

(UVC,*af1*,*av1*,r**)**
(UVC)

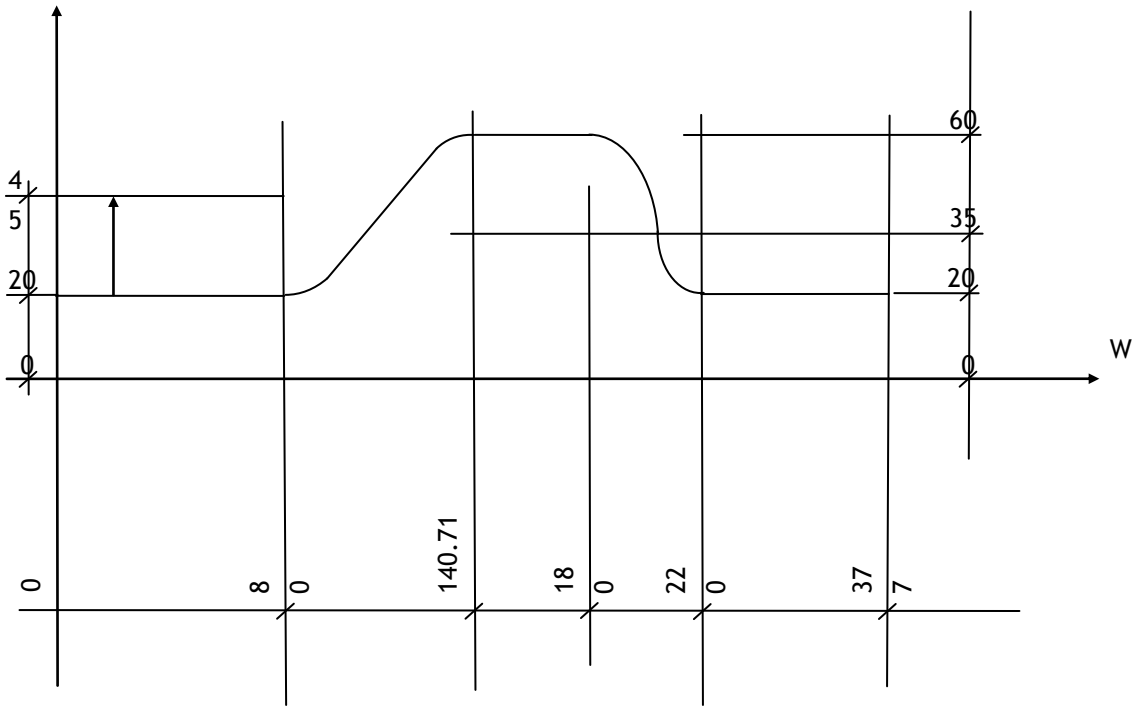
where:

af1	Rotary axis name
av1	Virtual axis name
r	Value of the cylindrical radius where the profile is made. It can be specified directly using a real number or a variable
no parameter	Disables the UVC mode

Characteristics:
This type of virtualization is used to move a linear axis (Y) and a rotary axis (B) by programming their positions in a Cartesian plane (WY). All the moves of the virtual Y coincide with those of the physical Y, but each move of W corresponds to a circle arc, a function of the cylinder radius, that has to be converted in an angular move of the physical axis B.



Example




```
(DIS,"UVC EXAMPLE");- Cylindrical coordinates programming -
T1.1M6
S2000F300M3
B0
XY20Z10
E30=60;CyLINDER RADIUS
(UVC,B,W,E30)
G16WY
E31=2*3.1415*E30
G41G1W0Y20
Z-12
G1W80Y20
G2W98.77813Y28.49601I80J45
G1W122.4554Y52.08126
G2W140.171Y60I140.71J35
G1W180Y60
G2W205Y35I280J35
G3W220Y20I220J35
G40G1W277
GZ20
(UVC)
M30
```

6.1.3 *UVA – Non-orthogonal axes programming*

The UVA command allows to program non-orthogonal axes as they were orthogonal.

Syntax

(UVA,*af1af2*,*av1av2*,*alpha*)
(UVA)

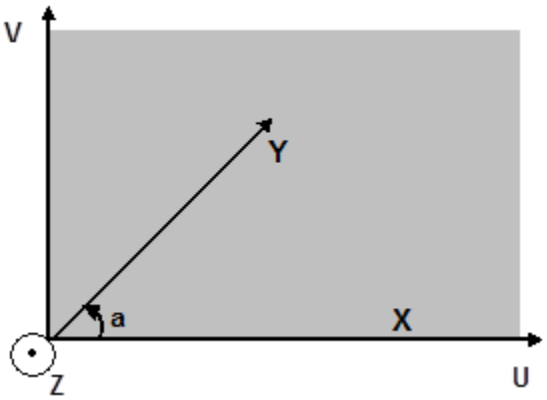
where:

af1	Name of the first physical axis
af2	Name of the second physical axis
av1	Abscissa virtual axis name
av2	Ordinate virtual axis name
alpha	angle, in degrees, subtended by two semi-positive axes of the physical linear axes. It can be specified directly using a real number or a variable
no parameter	Disables UVA mode

Characteristics:

This type of virtualization is used when two linear axes (X and Y) which are not orthogonal by programming the positions on an orthogonal virtual plane (UV). The U axis, abscissa of the new pair of axes, will be coaxial with the X axis while the V axis will be perpendicular to U, forming an angle of 90° - alpha with the axis Y.

Origin of UV will be coincident with that of XY. The position of the generic point P of coordinates (U, V) in the virtual plane is translated to the coordinates (X, Y) of the physical axes. While movements of U coincide with the axis X every physical movement of V is translated into a path motion of X and Y.

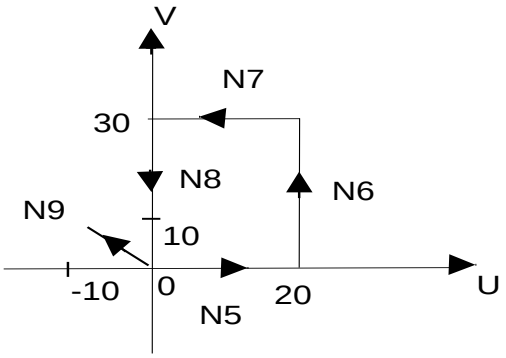


The sign of the parameter alpha is positive if the X axis, when positioning on the Y axis rotates counter clockwise, negative if the direction is clockwise (in the figure alpha is positive).

Alpha values of 0° or 180° cannot be accepted, as it would mean having two parallel physical linear axes.

Example of polar coordinates programming

N1 T1.1M6
N2 S1000M3
N3 GX0Y0
N4 (UVA,XY,UV,45)
N5 G1U20VF110
N6 V30
N7 U0
N8 V0
N9 U-10V10
N10 (UVA)
N11 GX50



6.2 3D transformations available

COMMAND	FUNCTION
UPR	
G53	Cartesian tern (XYZ) roto-translation in space
G54	
TCP	Programming in workpiece positions Movement on tool direction

6.2.1 *UPR – Cartesian tern rotation*

UPR (USE PLANE ROTATED) instruction allows definition of a roto-translation of a Cartesian axes tern and a rotation of the rotary axes.

Syntax

(UPR, *type, ax1 ax2 ax3* [, [a*n1*] *r1*, [a*n2*] *r2*, [a*n3*] *r3*] [, o*1*, o*2*, *o3*] [, *rr1*, *rr2*]**)**
(upr, *type, ax1 ax2 ax3* [, [*n1*] *r1*, [*n2*] *r2*, [*n3*] *r3*] [, o*1*, o*2*, *o3*] [, *rr1*, *rr2*]**)**
(UPR, 99)
(UPR)

where:

type Represents the UPR type with the following values allowed:

- **0** Absolute: roto-translations refer to physical axes
- **1** Incremental: roto-translations refer to the tern used during the programming
- **2** **Absolute with transformation on 5 axes:** features are similar to the 0 type, but the linear axes rotation affects the position of the two rotary axes. When the UPR function is enabled, rotary axes get a new value that identifies the tool axis position relative to the new virtual system.
- **3** **Incremental with transformation on 5 axes**: features are similar to the 1 type, but the previous considerations have to be added.
- **4** **Rotation plane automatically determined according to the tool axis direction**
- **6** as 4 but with 5 axes transformation (see 2)
- **10** Absolute global: same features as 0 type, but the programmed rotation remains active also with following non-global UPRs
- **12** **Type 2 global**: features are similar to type 2, but the programmed rotation remains enabled also with following non-global type UPR.
- **14** **Type 4 global**: features are similar to type 4, but the programmed rotation remains enabled also with following non-global type UPR.
- **16** **Type 6 global**: features are similar to type 6, but the programmed rotation remains enabled also with following non-global type UPR.
- **99** with no additional parameters, it disables all the UPR modes active as well as the global roto-translation as well

ax1 linear axes names on which apply the roto-translation
ax2
ax3

an1r1, an2r2, an3r3 pair of parameters indicating the axis name and the rotation value (in degrees) relative to that axis. If the axis name is not specified, then the rotation is applied to the axis corresponding to the position in which appears the rotation value.

an1r1
an2r2
an3r3

parameters indicating axis name and its rotational value

If axis name is not specified, the rotation applies on the axis corresponding to the position where the rotation value appears.

o1
o2
o3

origins translation values towards the current reference system. Translations refer to axes specified by the parameters *ax1*, *ax2* and *ax3*.
In case of absolute UPR, the coordinate refers to the origin in use when UPR was programmed.
In case of incremental UPR, the coordinate represents the increment towards the "UPR" origin already in use.

rr1
rr2

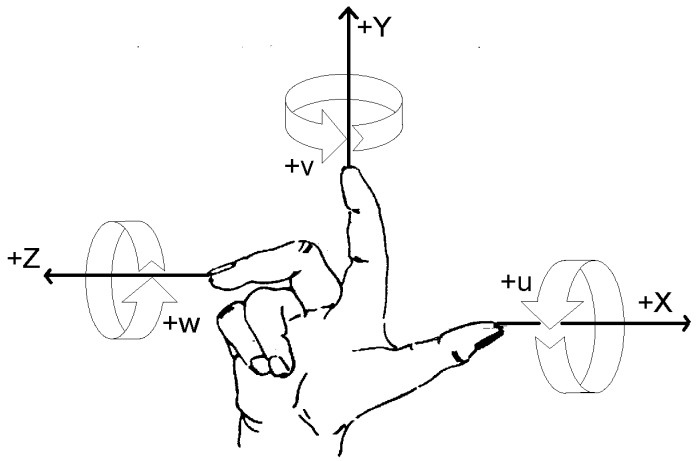
Angles values (degrees) internal to the perigon (between -360° and +360°) identifying the origin of the two rotary axes. They have to be considered always as physical axes translation.

no parameter

(UPR) Without parameters disables the UPR mode leaving only the global (if present).
If no global rotation has been programmed, UPR and UPR,99 instructions are equivalent.

upr

When programming the three-letter code as minuscule, angles and origins can change without disabling the UPR algorithm and without exit the "CONTINUOUS MOVEMENT" mode.
upr must be used only to change UPR parameters if they are non-global and previously programmed. It is not possible to change axes sequence.
It is no possible to program a global rotation.
In order to define the rotary direction of the virtual tern, reference has to be made to the right hand rule.



Right hand rule.

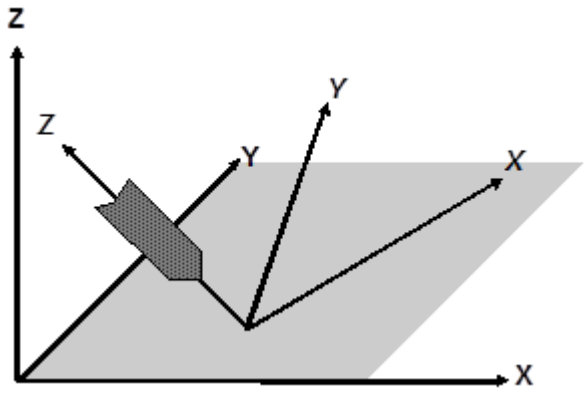
Characteristics:
UPR allows any machine function to be programmed within a rotated space at any angle relative to the Cartesian tern of the tool machine.
This allows the operator to program the definition of the profile to be machined within the regular Cartesian system (XYZ). The CNC recalculates axes motion according to the planes derived by the rotation.

Global UPR programming permits the application of a starting roto-translation to which all the following UPRs refer. Even if the part is clamped to the machine with different orientation, this programming allows the same part-program containing other geometrical instructions to be executed.
Global UPR activation doesn't affect the incremental or absolute programming.
Global UPR activation resets any previous virtualization resetting all the possible displacements.

In the case of use of **UPR** type **2, 3, 4** and **6** or of the corresponding global type, the system requires the characteristics of the machine which must be incorporated in the **TCP** table according to the rules defined by the same TCP and then activated. When UPR and (**TCP, 5**) are active at the same time, the movements of the tool axis will influence the movement of the virtual axes.

The **modes 2 and 3** make that the tool axis, identified by the programmed rotary axes, assumes in the virtual reference system the same position as in the reference system identified by the three Cartesian axes.

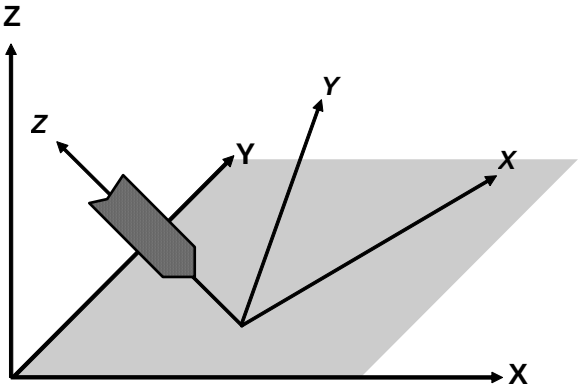
The **modes 4** and **6** and the corresponding global type, determine the new rotation plane according to the tool direction (thus based on the position of the rotary axes). The new plane, assuming the programming (UPR, 4, XYZ, ...) will have the W axis coinciding with the tool direction, the U axis lying in the first quadrant of the XY plane original, the V axis accordingly based on the rule of description of a Cartesian tern.



UAO or **UTO** origin **can't be** programmed if a virtualisation is active**.**

On rotary axes, origins activated with a **UIO** instruction are not managed.

The **RESET** instruction disables the UPR mode.



In case of **global UPR**, *r1, r2* and *r3* values must be between –90° and +90°. When swapping from one virtualization to another, the algorithm generating the rotary axes position could create wrong rotations for angles exceeding this range. The rotations created would still be congruent to the tool position.

UPR is not affected by DAN: names of physical axes are specified within the upr, no matter which DAN instructions have been previously executed.

Rotation mode:

Cartesian tern rotation is sequential towards programmed rotation angles, therefore:

A. *ax1 ax2 ax3* coordinates system rotates by an *r1 angle* around *ax1*.

B. The new system of coordinates (*ax1' ax2' ax3')* deriving from the aforementioned rotation (see A) rotates around *ax2'* with angle *r2*.

C. The new system (*ax1" ax2" ax3")* deriving from the aforementioned rotations (see A and B) rotates around *ax3"*, with a angle *r3*.

These operations lead to the reference system identified by *ax1 ax2 ax3* axes.

A) B) and C) operations are sequentially executed towards angles and axes programmed within the three-letter code, therefore the angles and physical axes order is very important.

6.2.2 *UPR examples*

Examples below assume the reference system as the Cartesian tern XYZ.
In examples 1, 2, 3 ,4 ,5 the reference system origin corresponds to the origin in use during the UPR programming:

Example 1

```
GYXZ
(UPR, 0, XYZ, 30,45,60)
G1F5400
X100.4 Y9.12 Z-70
X70.345 Z-20
```

Rotated system is obtained by:
30° rotation of the XYZ system around X axis (XYZ → X'Y'Z')
45° rotation of the XYZ system around Y axis (X'Y'Z' → X"Y"Z")
60° rotation of the X"Y"Z" system around Z"

Example 2

```
GYXZ
(UPR, 0, XYZ, X10,Z80,0)
G1F4000
X50 Y70
X90
```

Rotated system is obtained by:
10° rotation of the XYZ system around X axis (XYZ → X'Y'Z')
80° rotation of the system X'Y'Z' around Z' axis

Example 3

```
GXYZ20
(UPR, 0, ZYX, 80, 0, 10)
G1F3000
Z50 Y70
X90
```

Rotated system is obtained by:
80° rotation of the ZYX system around Z axis (XYZ → X'Y'Z')
10° rotation of the Z'Y'X' system around X' axis

Example 4

```
GXYZ
(UPR, 0, ZYX, Y50, Z60,0)
G1F3000
Z50 Y70
X90
```

Rotated system is obtained by:
50° rotation of the ZYX system around Y axis (ZYX → Z'Y'X')
60° rotation of the Z'Y'X' system around Z' axis

Example 5

```
GXYZ
(UPR, 0, XYZ, 30, 0,0)
G1F3000 X50Y0
Y30

.
(UPR,1,XYZ,60, 0, 0)
```

Rotated system is obtained by:
30° rotation of the XYZ system around Z axis (XYZ → X'Y'Z')

New system is obtained by:
60° rotation of the X'Y'Z' system around X' axis

Example 6

```
GXYZ
(UPR, 0, XYZ, 30, 45, 60, 10.8, 20, -30.2)

G1F5400
X100.4 Y9.12 Z-70
X90
```

Rotated system is obtained by:
Translation of the reference system origin in X10.8, Y20, Z-30.2 within the active reference system.
30° rotation of XYZ around X axis (XYZ → X'Y'Z')
45° rotation of X'Y'Z' around Y' axis
(X'Y'Z' → X"Y"Z")
60° rotation of X"Y"Z" around Z" axis

Example 7

```
GXYZ
(UPR, 0, XYZ, 30, 45, 60, 10.8, 20, -30.2)
G1F5400
X100.4 Y9.12 Z-70
X90
```

Rotated system is obtained by:
Origin translation towards the active reference system.
30° rotation of XYZ around X axis (XYZ → X'Y'Z')
45° rotation of X'Y'Z' around Y' axis
(X'Y'Z' → X"Y"Z")
60° rotation of X"Y"Z" around Z" axis

```
(UPR,1,XYZ, 10, 0, 0, 3, 8, 5)
```

Rotated system is obtained by:
Origin translation towards the active reference system (X"Y"Z").

10° rotation around X" axis

Example 8

In this case, a starting 90° rotation around Z axis is applied to the part-program in the example 2

```
GYXZ
(UPR,10,XYZ, 0,0,90)
(UPR, 0, XYZ, X10,Z80,0)
G1F4000
X50 Y70
X90
```

Rotated system is obtained by:
90° rotation of XYZ around Z axis (XYZ → X'Y'Z')
10° rotation of XYZ around X axis
(X'Y'Z' → X"Y"Z")
80° rotation of X"Y"Z" around Z' axis

6.2.3 *G53 G54 Cartesian tern rotation*

G53 and G54 allows definition of a roto-translation of three axes belonging to the process. This is a special case of UPR.

Syntax

G53 *a1 o1 a2 o2 a3 o3* [**a** *r1* **b** *r2* **c** *r3*]
G54 *ax1 o1 ax2 o2 ax3 o3* [**a** *r1* **b** *r2* **c** *r3*]
G53
G54

---

where:

ax1	axis name referring to the first parameter of the roto-translation
ax2	axis name referring to the second parameter of the roto-translation
ax3	axis name referring to the third parameter of the roto-translation
o1	value of the translation to be applied on the corresponding axis
o2 *o3*	the value can be expressed directly using a real number or indirectly using an E parameter or a variable
r1	value of the rotation to be applied on the corresponding axis
r2 *r3*	the value can be expressed directly using a real number or indirectly using an E parameter or a variable
no parameter	If no parameter is et, the two codes disable the roto-translation

Characteristics:
G53 corresponds to a 10 type UPR (absolute global) and **G54** corresponds to a 0 type UPR (global).

G53 execution without parameters corresponds to (UPR) i.e. removal of all active UPRs. **G54** execution without parameters corresponds to UPR removal keeping active only global UPR.

These three rotations happen in the following order:

1. Rotation around the third axis specified within the three-letter code (*ax3)*
2. Rotation within the second axis of the tern generated by the first rotation
3. Rotation around the third axis of the tern generated by the second rotation

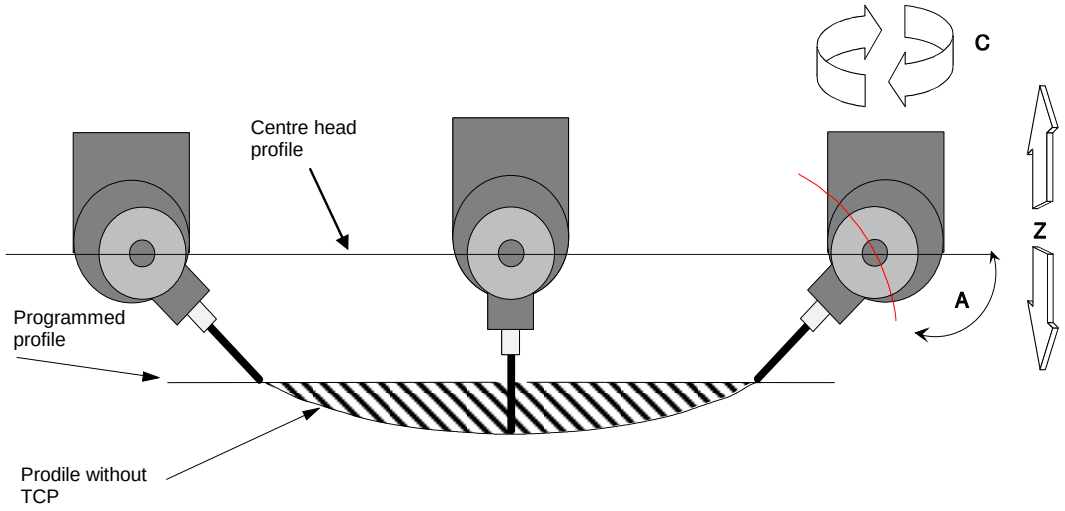
6.2.4 *Tool Centre Point (TCP)*

TCP allows a 5-axis machine programming (3 linear and 2 rotary axes)referring more to the tool head than the axis rotation centre (head centre). Up to 4 settings can be defined each of them having 4 different toolholder. When activating TCP, it is possible to set which kinematics and toolholder should be enabled.

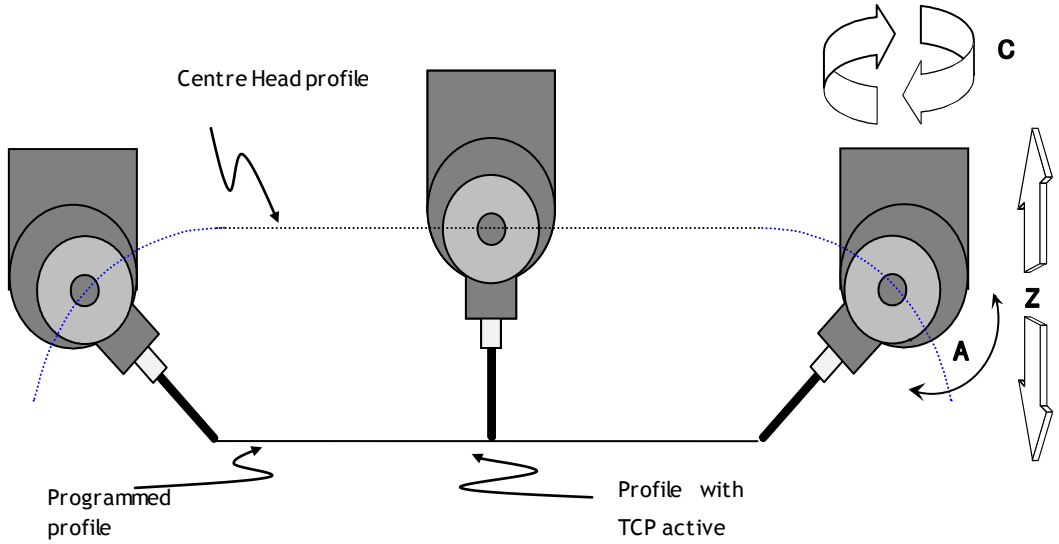
This algorithm is activated directly from the program using TCP. There are two Tool Centre Point modes:

- **Programming in workpiece positions (TCP, 1):**
  Positions programmed refer to the workpiece.
  This mode is used when the profiles to trace refer to both linear axes motions (1÷3) or rotary axes motions (1÷2), having the tool always in touch with the surface to be machined. In addition, linear axes positions refer to the workpiece.
  System, in fact, offsets the motions regulating automatically the three axes position according to the rotary axes. This procedure allows to keep the tool head on the profile defined by linear axes.

  The following figures show the trajectory of a tool tip according to the A axis motion or to the TCP activation/deactivation. The first figure shows the motion when TCP is disabled: this is the case when A (rotary) changes the tool tip position along the Z axis direction, creating a profile that is different from the one programmed.



When TCP is enabled (second figure), control introduces a motion along Z allowing to reset the tool tip position change on the programmed profile.

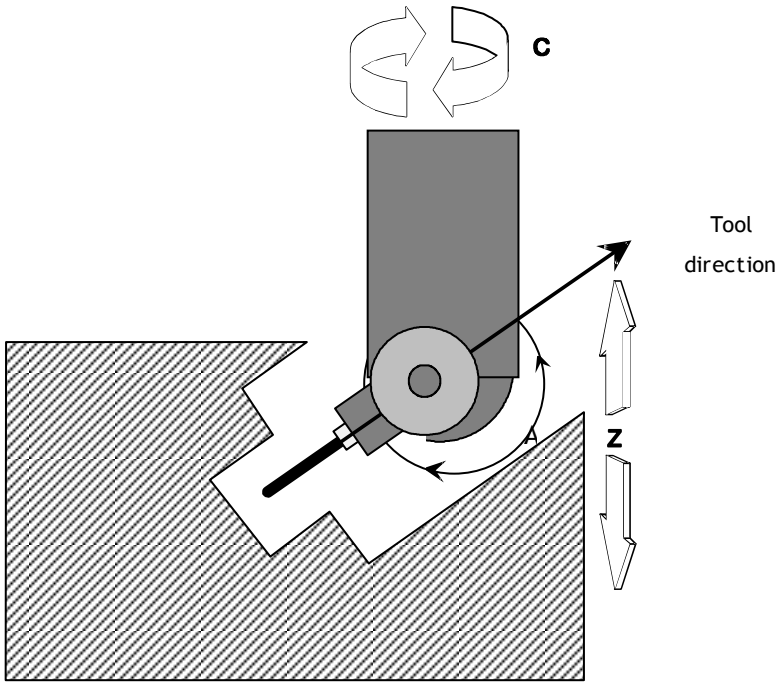


- **Motion along the tool direction (TCP, 5):**

  This mode allows execution of motions along the tool direction. The activation of this mode leads the system to create a virtual axis having a name previously defined in AMP or within the $TDNAME variable.

  Virtual axis motion leads to linear axes movement with the tool moving along the position determined by the rotary axes.

  $TDNAME.



This virtual axis can be programmed alone or with other axes; in this case, (TCP,5) features are added to the (TCP,1). The axis can be manually moved as well.

Mode 5 can be useful either to drill inclined holes or to extract the tool from the workpiece in case of an anomaly that stopped the machining.

In the latter case, we can have 3 different situations:

1. **When activating (TCP,5) rotary axes are not referred and the previous interruption occurred during the machining using any TCP mode.**
   In this case, system automatically identifies the tool direction according to the positions of the rotary axes before the anomaly.
   As these positions are stored in a non-volatile memory, they can also be recovered if the system is powered off. Values identifying rotary axes positions can be changed using the MANUAL SETUP data entry or using the PLC with the ToolCenterPointWrite function (useful to change the tool extraction direction).

2. **When activating (TCP,5), rotary axes are not referred and the previous interruption happened during machining without using the TCP mode.**
   In this case, tool direction must be communicated to the system as the positions previously stored by the system may not be significant. Positions could be communicated using either PLC ToolCenterPointWrite function or MANUAL SETUP data entry.

3. **When activating (TCP,5) axes are referred.**
   In this case, even if the interruption didn't happen during the TCP mode machining, the current rotary axes positions determine the tool direction.

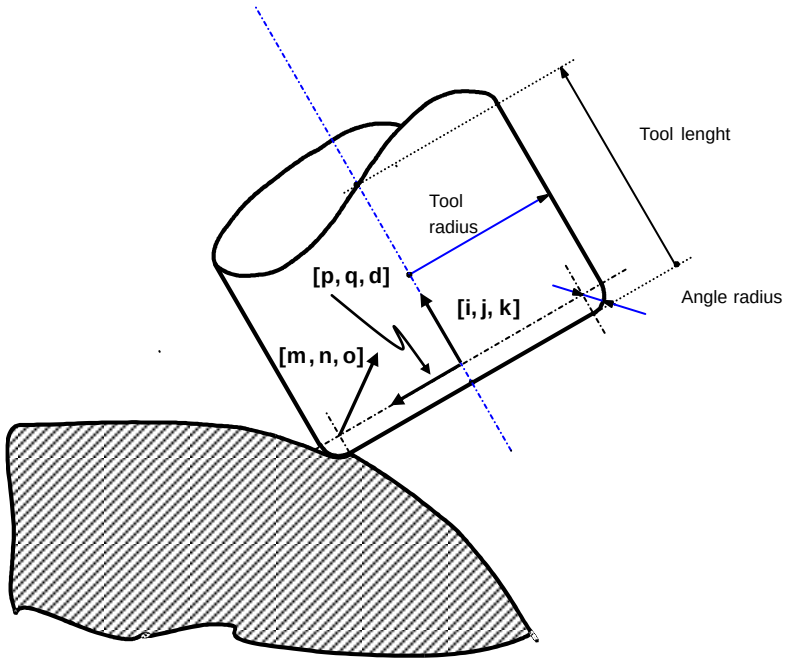
NOTE:

- To use ToolCenterPointWrite or MANUAL SETUP data entry, refer to 4Control Library and User Manual.

- In all TCP modes, machine logic can communicate to the interpolator (in real time) a possible variation of the tool radius/length of the angle radius and of the angles touching the workpiece. This allows offsetting of wear and tear on the tool during the matching.

6.2.5 *Tool direction/compensation vectors programming*

In all the TCP modes aforementioned it is possible to program not only the axes positions, but also different vectors/versors.



Available terns are:

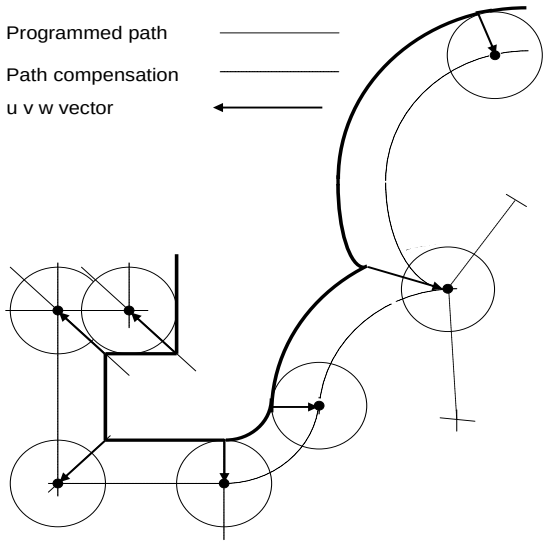
- **i, j, k:** tool direction versors
  i, j, k parameters specify tool direction axis oriented from the tool head towards its mount

- **m, n, o:** versor of the direction perpendicular to the machining profile
  m, n, o parameters define the direction perpendicular to the plane machining.

- **p, q, d:** versor of the tool radius application direction
  p,q,d parameters define the vector where the radius compensation is applied (starting from the tool centre).

- **u, v, w:** paraxial offset versor for the tool radius offset
  u, v, w parameters represent an alternative to mno and pdq versors for calculation of the tool radius compensation. The compensation doesn't work on the direction perpendicular to the programmed profile (mno), but is obtained by moving the programmed point of a distance equal to the radius along the uvw direction.

  uvw vector could be a non-unitary module as well



6.2.6 *TCPDEF – Default kinematics identifier*

This instruction specifies the default kinematics identifier to which TCP commands refer.

Syntax

TCPDEF = *idCin*

---

where:

idCin It's a numerical value specifying the ID of the TCP default kinematics. It can be specified directly using an integer from 1 to 4 or indirectly using an E parameter.

Characteristics:

If no kinematics identifier is specified in the three-letter code, the value of this variable is used for the TCP activation.

6.2.7 *TCP – Tool Centre Point for machines with Prismatic head*

This feature allowsprogramming of a 5 axes machine (3 linear and 2 rotary) referred to the tool head rather than to the axes rotation centre (head centre).
Algorithm activation is made directly in the program using the three-letter code TCP.

Double-Twist head is a special case of the machine.

TCP mode is disabled using (TCP) or RESET.

Some parameters can be changed without disabling TCP or closing the "CONTINUOUS MOVEMENT" mode, by programming the lower case three-letter code (tcp, n).

(tcp, 1) must be used only to change the (TCP, 1) parameters previously programmed. In mode 5, parameters can't be changed during a continuous movement.

Syntax

(TCP,1[**,** *idCinem*]**)**
(TCP,5 [**,** *idCinem*, *mode*]**)**
(tcp,1)

---

where:

idCinem Kinematics ID to activate. If this parameter is missing, the default kinematics (defined by TCPDEF) will be enabled.

mode Mode active for TCP tool direction, the values below are valid:

1. Generates the tool direction virtual axis

2. Generates the tool direction axis and the perpendicular virtual plane

3. Generates the tool direction axis and the projection on the XY plane of the plane perpendicular to the tool

Characteristics:

In order to work properly, the TCP algorithm needs information about the machine geometry while tool information is set by the system according to the tool used. The system manages up to 4-tool heads and each of them could be angular too.

In the following TCP features description, the Cartesian tern of the three TCP linear axes is identified by the XYZ tern where X corresponds to the first linear axis, Y to the second and Z to the third. The two rotary axes will be identified as horizontal and nutating (nodding) head. Configuration parameters evaluation has to be made after positioning the head as the tool to be set is vertical along Z direction. In addition, nutating rotary axis has to be parallel to the XZ plane as shown in figures 2.9, 2.10, 2.11.

When configuring AMP using ODM, the PLC uploads these parameters in their variables, using the functions required or assigning the values directly from part program.

If the parameter of the second rotary axis inclination (nutating axis) is 90°, prismatic head is like a Double Twist head type.

Table below lists all the parameters needed to configure a prismatic head.

DESCRIPTION	VARIABLE NAME	VALID VALUES
Machine with prismatic head	$CINTYPE(*idCinem*)	1
First linear axis ID	$ABSID(*idCinem*)	1 ÷ 64
Second linear axis ID	$ORDID(*idCinem*)	1 ÷ 64
Third linear axis ID	$VRTID(*idCinem*)	1 ÷ 64
First rotary axis ID (horizontal axis)	$ROTPID(*idCinem*)	1 ÷ 64
Second rotary axis ID (nutating axis)	$ROTTID(*idCinem*)	1 ÷ 64
Tool hold enabled	$TOOLHOLD(*idCinem*) *	1 ÷ 4
Horizontal head direction. Defines rotation direction of the horizontal head for positive programming and is: +1 CW - 1 CCW	$DIRROTP(*idCinem*) *	-1 ; 1
Nutating axis direction. Defines rotation direction of the nutating head for positive programming and is: +1 CW - 1 CCW	$DIRROTT(*idCinem*) *	-1 ; 1
Inclination of the axis around which rotates the nutating axis towards the machine vertical (90°	$TILTBASC(*idCinem*)	0 ÷ 90°

for Double-Twist head type).		
Horizontal head offset (primary rotary axis) It is the offset value associated with the horizontal head in order to match the tool plane with the XZ machine plane. To determine the offset, rotate the horizontal axis as having the plane with the tool complying with XZ plane. After this rotation, a positive movement of the nutating head directs the tool towards Z-. Multiply the position displayed by the rotation direction of the horizontal head, then change the sign. The resulting motion corresponds to the horizontal axis rotation needed to place the head as displayed in figure 2.9	$OFFSROTP(*idCinem*) *	-360 ÷ 360°
Tilting head (secondary rotary axis). It is the offset value associated to the nutating head in order to place the tool along Z- direction. The offset is determined by rotating the nutating axis so as to place the head in the same position as shown in figure 2.9 Multiply the position displayed by the rotation direction of the nutating head, then change the sign.	$OFFSROTT(*idCinem*, *ToolHold*) *	-360 ÷ 360°
A parameter (on X, this is the distance between Pc and Pr) Distance between **Pc** (control point) and **Pr** (rotation point) on X axis. This distance is measured in mm and has a positive value when moving along the axis negative direction.	$PCPRX(*idCinem*)	
B parameter (on Y this is the distance between Pc and Pr). Distance between **Pc** (control point) and **Pr** (rotation point) on Y axis. This distance is measured in mm and has a positive value when moving along the axis negative direction.	$PCPRY(*idCinem*)	
C parameter (on Z this is the distance between Pc and Pr). Distance between **Pc** (control point) and **Pr** (rotation point) on Z axis. This distance is measured in	$PCPRZ(*idCinem*)	

mm and has a positive value when moving along the axis negative direction.		
D parameter (on X, this is the distance between Pr and Pu) Distance between **Pr** (rotation point) and **Pu** (tool mount point) on X axis. This distance is measured in mm and has a positive value when moving along the axis negative direction.	$PRPUX(*idCinem*, *ToolHold*)	
E parameter (on Y, this is the distance between Pr and Pu) Distance between **Pr** (rotation point) and **Pu** (tool mount point) on Y axis. This distance is measured in mm and has a positive value when moving along the axis negative direction.	$PRPUY(*idCinem*, *ToolHold*)	
F parameter (distance along Z and between Pr/Pu). Distance between **Pr** (rotation point) and **Pu** (tool mount point) on Z axis. This distance is measured in mm and has a positive value when moving along the axis negative direction.	$PRPUZ(*idCinem*, *ToolHold*)	
Length 1 (Right angle head). Distance between right angle head centre and spindle centre (length L1 figure 2.12).	$L1(*idCinem*, *ToolHold*)	
Length 2 (right angle head). Distance between centre head and right angle head centre (length L2 figure 2.12).	$L2(*idCinem*, *ToolHold*)	
α angle (Right angle head). Angle of deviation relative to the direction Y- (angle α in figure 2.12)	$ALFA (*idCinem*, *ToolHold*)	-180 ÷ 180°
ß angle (Right angle head). Angle of deviation relative to the vertical (ß angle in figure 2.12).	$BETA(*idCinem*, *ToolHold*)	-90 ÷ 90°
Horizontal axis inclination (alternative to Id horizontal head) Consider the head horizontal to the specified position. It's a signed value, a function of the configured "horizontal head direction".	$NUTATINGROTP(*idCinem*)	
Vertical axis inclination (alternative to Id vertical head) It will be considered the vertical	$NUTATINGROTT(*idCinem*)	

head to the position specified. It's a signed value, a function of the configured "vertical head direction".		
Tool radius compensation type: **0** no compensation **1** compensation with $TOOLANG, $TORANG angles **2** compensation with versors (mno) **3** compensation with versors (mno and pqd)	$COMPRAD(*idCinem)*	0 ÷ 3
Enabling real-time tool compensation: **0** real-time compensation disabled **1** real-time compensation enabled	$COMPRT(*idCinem*)	0 ÷ 1
Tool-workpiece contact angle. It's the subtended angle between the tool centre and the tool-workpiece contact point (see figure 2.13).	$TOOLANG(*idCinem*)	0 ÷ 360°
Contact angle (α) for spherical or toroidal tools	$TORANG(*idCinem*)	0 ÷ 90°
Operating and rotary tool radius TODO TODO TODO	$OPRADP(*idCinem*)	
Tool rotary operative radius TODO TODO TODO	$OPRADT(*idCinem*)	
Negative operational limit	$NEGLIMOP(*idCinem*)	
Positive operational limit	$POSLIMOP(*idCinem*)	
Operational limits mode: The bits 1,2 and 3 (starting at bit 0) are used to activate the control of the operating limit referred to the tool tip, respectively for the first, second and third axis TCP. For example assigning the variable value 7 requests the activation of the real-time control of operating limits both with respect to the tool tip and to the physical locations for which the axes X and Y, Z maintaining active for only one referring to the physical position	$OPLIMMODE(*idCinem*)	0 ÷ 15
Tool direction axis name. It is used only for tool direction mode (TCP,5). The axis name has to be valid and when activating TCP no axis should have the same name.	$TDNAME(*idCinem*)	
Axis name "abscissa for tool direction". It is used only for tool direction mode	$TANAME(*idCinem*)	

(TCP,5). The axis name has to be valid and when activating TCP no axis should have the same name.		
Axis name "ordinate for tool direction". It is used only for tool direction mode (TCP,5). The axis name has to be valid and when activating TCP no axis should have the same name.	$TONAME(*idCinem*)	

Figures below illustrate the characterization of the compensation parameters applied to a machine designed in the frontal, side and up view.

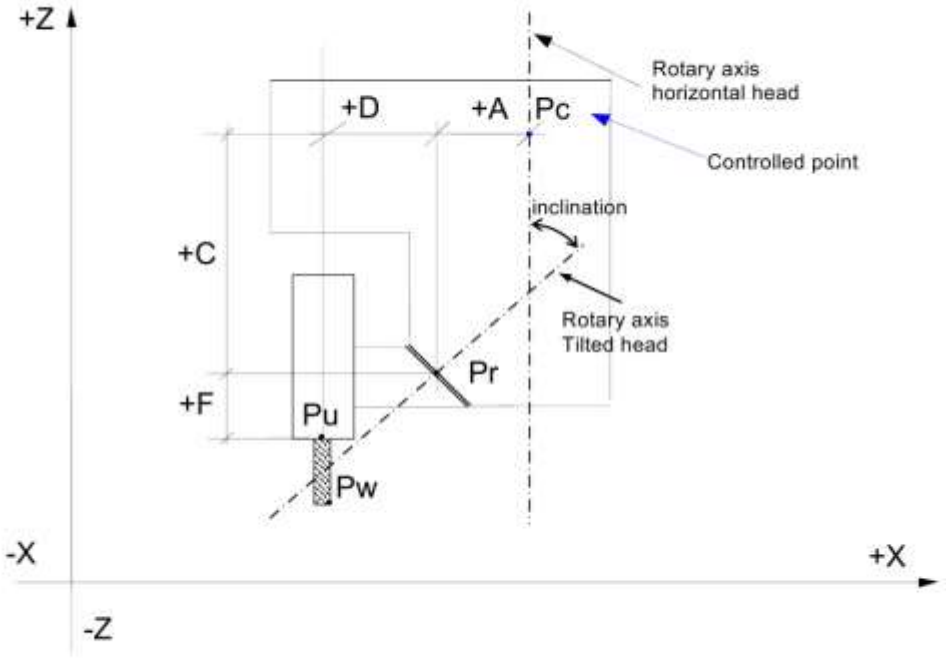


Fig. 2.9 Frontal view

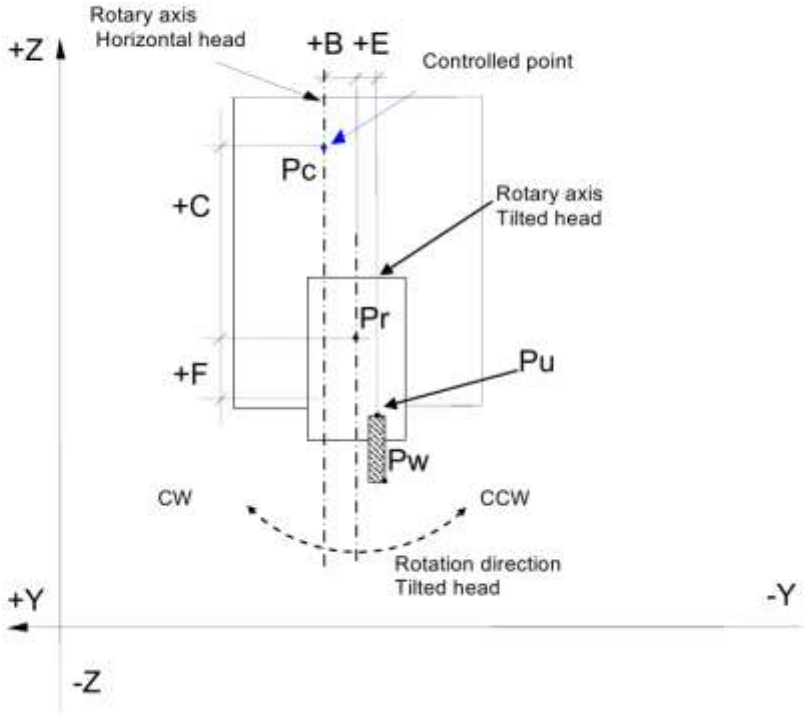


Fig. 2.10 Side view

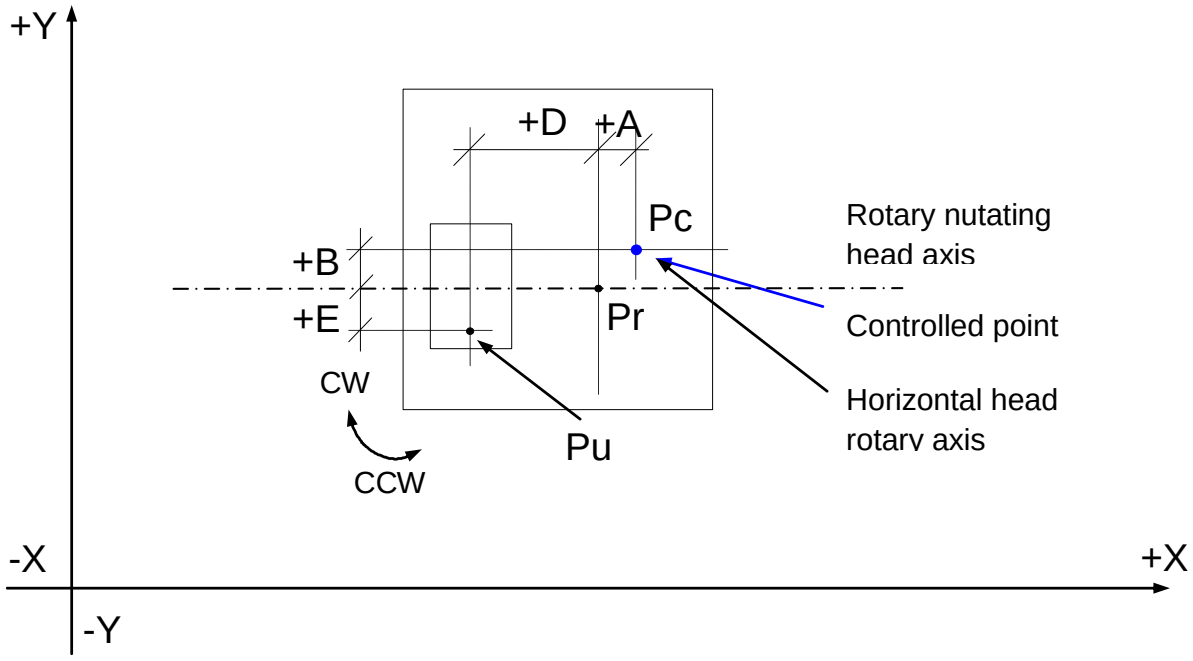


Fig. 2.11 Upper side view

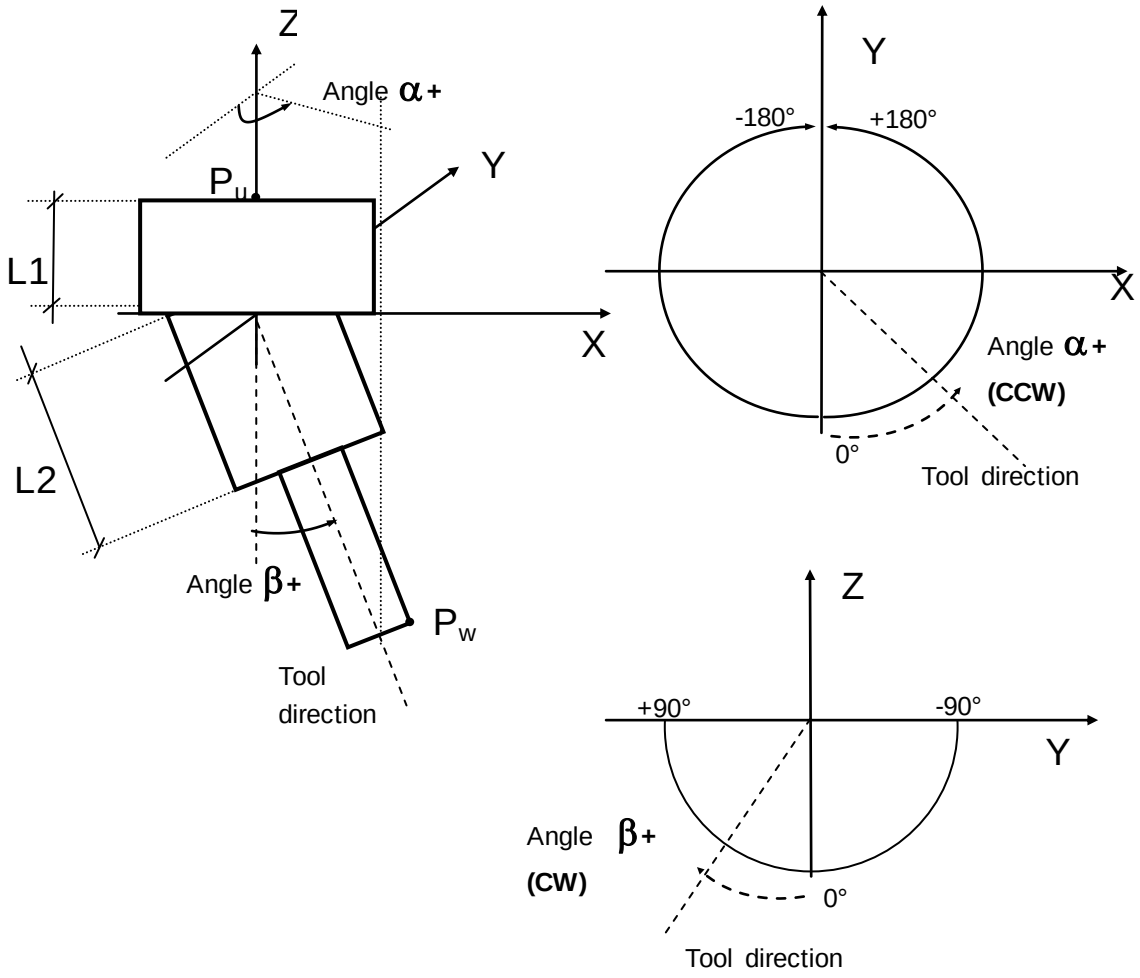


Fig. 2.12 Right angle head

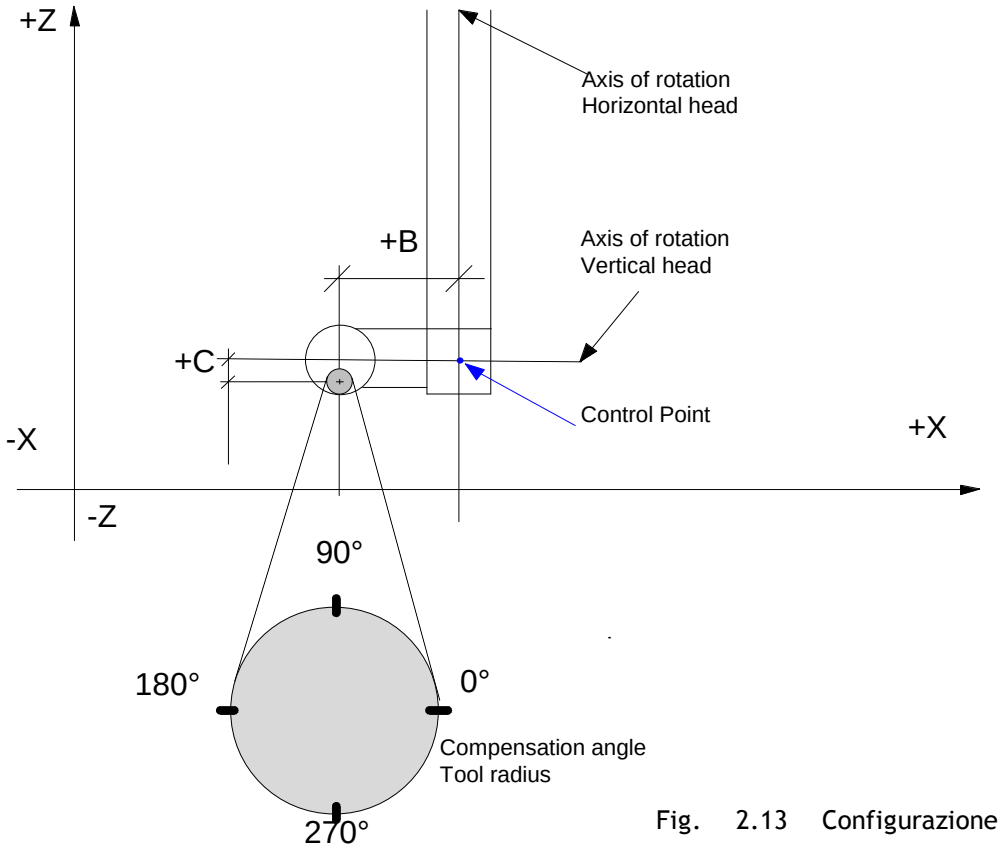


Fig. 2.13 Configurazione angoli di

6.2.8 *TCP – Tool Centre Point of the tool-length only*

This feature programs a 5-axis machine (3 linear and 2 rotary) regardless of whether the tool or the table moves (spherical or toroid). The position controlled by the system coincides with the point connecting the tool tip and the spindle.
TCP activates the algorithm.
TCP mode is disabled with **(TCP)** or RESET.

Syntax

(TCP,1,[*idCinem*]**)**
(TCP,5,[*idCinem*, *mode*]**)**
(tcp,1)

---

where:

idCinem Identifier of the kinematics to be enabled. If this parameter is missing, the default kinematics specified by the TCPDEF variable will be uploaded.

Mode Mode enabled for tool direction TCP, following values are accepted:

- **1** Generates the tool direction virtual axis
- **2** Generates the tool direction axis and the perpendicular virtual plane
- **3** Generates the tool direction axis and the projection on the XY plane of the plane perpendicular to the tool

Characteristics:
In order to work properly, TCP algorithm needs information about the tool in use.
When configuring AMP using ODM, PLC uploads these parameters in their variables, using the functions required or assigning the values directly from part program.

For each programmed motion, TCP require the definition of both the tool direction (orientation) and the direction perpendicular to the surface to machine. Table below lists parameters needed to install the machine kinematics.

DESCRIPTION	VARIABLE NAME	VALID VALUES
Kinematic described	$CINTYPE(*idCinem*)	2
First linear axis identifier	$ABSID(*idCinem*)	1 ÷ 64
Second linear axis identifier	$ORDID(*idCinem*)	1 ÷ 64
Third linear axis identifier (horizontal or rotary axis closer to the workpiece)	$VRTID(*idCinem*)	1 ÷ 64
First rotary axis identifier (horizontal or rotary axis closer to the workpiece)	$ROTPID(*idCinem*)	1 ÷ 64
Second rotary axis identifier (nutating axis or rotary axis closer to the tool)	$ROTTID(*idCinem*)	1 ÷ 64
Active tool holder	$TOOLHOLD(*idCinem*)	÷ 4
Tool radius compensation type: **0** no compensation **1** compensation with angles $TOOLANG, $TORANG	$COMPRAD(*idCinem*)	÷ 3

2 compensation with versor (mno) **3** compensation with versors (mno and pqd)		
Enabling real-time tool compensation **0** real-time compensation disabled **1** real-time compensation enabled	$COMPRT(*idCinem*)	0 ; 1
Contact angle between the tool and the workpiece. This is the angle between the tool centre and the tool point contact with the workpiece (see figure 2.13).	$TOOLANG(*idCinem*)	0 ÷ 360°
Contact angle (α) for toroid or spherical tools.	$TORANG(*idCinem*)	0 ÷ 90°
Operative radius rotary workpiece TODO TODO TODO	$OPRADP(*idCinem*)	
Operative radius rotary tool	$OPRADT(*idCinem*)	
Operational limits mode	$OPLIMMODE(*idCinem*)	
Name of the tool direction axis. It is used only for tool direction mode (TCP,5). It has to be a valid axis name and, when TCP is activated, it has to be the only axis with that name.	$TDNAME(*idCinem*)	
Axis name "abscissa for tool direction" It is used only for tool direction mode (TCP,5). It has to be a valid axis name and, when TCP is activated, it has to be the only axis with that name.	$TANAME(*idCinem*)	
Axis name "ordinate for tool direction" It is used only for tool direction mode (TCP,5). It has to be a valid axis name and, when TCP is activated, it has to be the only axis with that name.	$TONAME(*idCinem*)	

Definition of tool direction and direction orthogonal to the surface

This type of TCP requires the specification of both the tool direction and of the direction orthogonal to the work surface. Each programmed point is associated to a versor indicating tool direction (identified by **i, j, k**) and to a versors perpendicular to the work surface (identified by **m, n, o**).
Before activating TCP the starting position of the two vectors must be defined programming their values.
If TCP is active, **m, n, o, i, j,** or **k** can't be programmed without having a corresponding TCP axes motion. They define a vector that can't be null and, if parameters are not programmed on a block, the two directions are considered unchanged.

6.2.9 *TCP – Tool Centre Point for generic machines*

This performance defines a range of geometric transformations describing kinematics for 5-axis machines (3 linear and 2 rotary) or for 6 axes machines (3 linear and 3 rotary).
The algorithm is directly activated within the program by the TCP.
(TCP) or RESET can both disable TCP mode.

Syntax

(TCP,1,[*idCinem*]**)**
(TCP,5,[*idCinem*, *mode*]**)**
(tcp,1)

where:

idCinem Identifier of the kinematics to be enabled. If this parameter is missing, the default kinematics specified by the TCPDEF variable will be uploaded.

Mode Mode enabled for tool direction TCP, following values are accepted:
- **1** Generates the tool direction virtual axis
- **2** Generates the tool direction axis and the perpendicular virtual plane
- **3** Generates the tool direction axis and the projection on the XY plane of the plane perpendicular to the tool

Characteristics:
The most important operation to define machine kinematics is the identification of the "**Kinematic Chain**" defining the axes motions sequences and how the axis motion extends to the following one. This chain starts from the tool finishing on the part.

PLC manages kinematics for generic machines require 2 or 3 rotary axes and 3 linear axes with the option to manage rotary axes according to the following instructions:

a) Two or three rotary axes mounted on the head (Double twist/prismatic) moving around the workpiece. In this the case kinematic chain is (**TOL** type kinematics):

Tool → 1° Rotary head → 2° Rotary head → [3° Rotary head →] Linear axes → Part

b) A rotary axis associated to a rotary table on which is placed the workpiece and one or two rotary axes placed on the head. In this the case kinematic chain is (**MIX** type kinematics):

Tool → 1° Rot head → [2° Rot head →] → Linear axes → Rotary table → Part

c) Two or three rotary axes referred to the rotary tables where the workpiece is placed and one rotary axis mounted on the head. In this the case kinematic chain is (**WRK** type kinematics):

Tool → Linear axes → 1° table → 2° table →[3° table →] Part

d) One or two rotary axes associated with the respective rotary table on which the workpiece is placed and a rotary axis mounted on the head. In this case the kinematic chain is defined as follows (MIXM kinematics type):

Tool → Rot head → Linear Axes → 1° Rot table → [2° Table →] Part

This structure can be resumed as the table below:

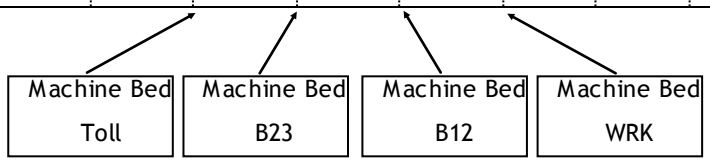
Kinematics	Tool	Rot	Rot	Rot	Lin 3	Lin 2	Lin 1	Rot	Rot	Rot	Part
TOL	T	TOL	MID	WRK	LI3	LI2	LI1	---	---	---	P
MIX	T	**---**	TOL	MID	LI3	LI2	LI1	WRK	---	---	P
WRK	T	**---**	---	---	LI3	LI2	LI1	TOL	MID	WRK	P
MIXM	T	**---**	---	TOL	LI3	LI2	LI1	MID	WRK	---	P

Where abbreviations mean:

LI1 First linear axis of the kinematic chain.
LI2 Second linear axis of the kinematic chain.
LI3 Third linear axis of the kinematic chain.
WRK Rotary axis closer to the tool part.
TOL Rotary axis closer to the tool.
MID Intermediate rotary axis

Another important element in the kinematic chain is "**Machine Bed**" that is the **"Fixed point".**
This is a "theoretical" point referred to the ground, with respect to which the axes movement sequence (chain) can be divided in two groups. The movements of the first axes group do not affect the movements of the second one.
Below a scheme of the 4 different "**Machine Bed**". displacements the kinematic chain can have:

Kinematics	Tool	Rot	Rot	Rot	Lin 3	Lin 2	Lin 1	Rot	Rot	Rot	Part
TOL	T	TOL	MID	WRK	LI3	LI2	LI1	---	---	---	P
MIX	T	**---**	TOL	MID	LI3	LI2	LI1	WRK	---	---	P
WRK	T	**---**	---	---	LI3	LI2	LI1	TOL	MID	WRK	P
MIXM	T	**---**	---	TOL	LI3	LI2	LI1	MID	WRK	---	P



There is an extension of the machine kinematics that allows you to position the rotary axes (only two) over the linear axes. This is the description for machines using the "sleeve" kinematic solutions. Additional kinematic chains are:

e) One or two rotary axes referred to the rotary tables where the workpiece is mounted and a "sleeve" associated to the second linear axis. In this case the kinematic chain is defined as follow (**WC1** kinematics type):

Tool -> Lin axes (3° e 2°) -> Rot -> Lin axis (1°) -> 1° Rot table -> [2° Table->] Workpiece

f) One or two rotary axes referred to the rotary tables where the workpiece is mounted and a "sleeve" associated to the third linear axis. In this case the kinematic chain is defined as follow (**WC2** kinematics type):

Tool -> Lin axes (3°) -> Rot -> Lin axes (2° and 1°) -> 1° Table ->[2° Table ->] Workpiece

g) A rotary axis mounted on the head moving around the workpiece and a "sleeve" associated to the second linear axis. In this case the kinematic chain is defined as follow (**TC1** kinematics type):

Tool -> 1° Rot head ->[2° Rot ->] Linear Axes (3°)-> Rotary -> Linear Axis (1°) -> Workpiece

h) A rotary axis mounted on the head moving around the workpiece and a "sleeve" associated to the third linear axis. In this case the kinematic chain is defined as follow (Type **TC2** kinematics):

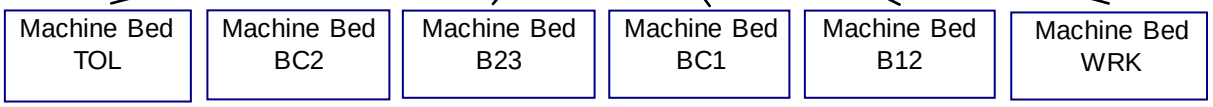
Tool -> Rotary head -> Linear Axis (3°) -> Rotary -> Linear axes (2° e 1°) -> Workpiece

i) Last configuration having 2 sleeves; this is the case where the kinematic chain is defined as below (**C12** kinematics type):

Tool -> [Rot ->] Lin Axis (3°) -> Rot -> Lin Axis (2°) -> Rot -> Lin Axis (1°) -> Workpiece

This structure could be resumed using the following table (with reference to the symbolism previously used):

Kinematics	Tool	Rot	Rot	Lin 3	Rot	Lin 2	Rot	Lin 1	Rot	Rot	Part
WC1	T	---	---	LI3	---	LI2	TOL	LI1	MID	WRK	P
WC2	T	---	---	LI3	TOL	LI2	---	LI1	MID	WRK	P
TC1	T	TOL	MID	LI3	---	LI2	WRK	LI1	---	---	P
TC2	T	TOL	MID	LI3	WRK	LI2	---	LI1	---	---	P
C12	T	---	TOL	LI3	MID	LI2	WRK	LI1	---	---	P



Description of the machine Kinematic chain

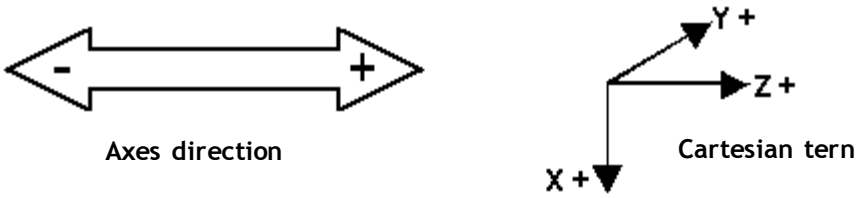
The description of machine kinematic chain defines the way an axis motion affects the following axes movements. When passing from one reference system to another, translations and rotations must be defined for all "real" machine axes (linear and rotary) and for the three "imaginary" axes (tool axis, workpiece axis and right angle head axis).

To move from one Cartesian tern to another, the rotations must always be executed by following the order X axis, Y axis and Z axis.
Rules to follow in order to define these transformations are:

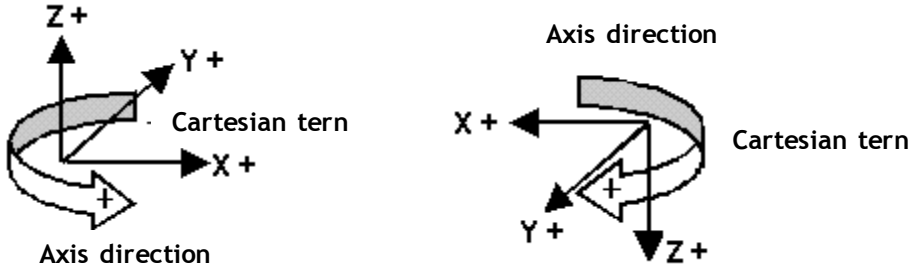
- First, it must be defined an arbitrary Cartesian tern referred to the machine bed and then a Cartesian tern for each component of the kinematic chain.
- To define terns while machine is in Home position

Cartesian terns of the kinematic components must respect the following rules:

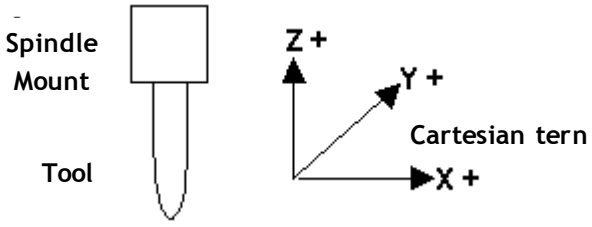
- **Linear axes**: the Z^+ direction of the Cartesian tern must correspond to the axis positive direction, X and Y directions do not have any special meaning.



- **Rotary axes**: the Z^+ direction of the Cartesian tern is orthogonal to the positive rotation direction of the rotary axis (CCW). X and Y directions do not have any special meaning.

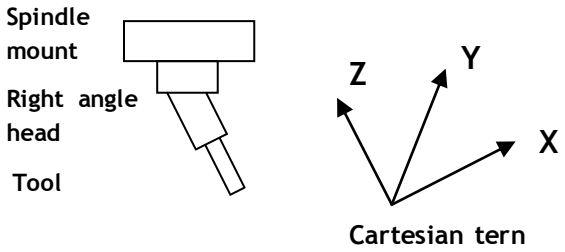


- **Tool axis**: the Z^+ direction of the Cartesian tern is from the tool tip to the spindle mount. Let's suppose to remove the right angle head, X and Y directions do not have any special meaning. The origin of this tern must be placed on the tool connection point with the spindle (as the tool length is 0).



- **Right angle head axis**: direction Z^+ of the Cartesian tern starts from the tool head towards the spindle mount while axes Y and X directions do not have any special meaning. The tern origin has

to be placed on the tool connection point with the spindle (as the tool length=0).



Cartesian tern

- **Workpiece axis**: Cartesian tern must correspond to the reference system that programmed the workpiece and how it is fastened on the machine.

Table below lists parameters to configure machine kinematics.

DESCRIPTION	VARIABLE NAME	VALID VALUES
Machine with generic configuration	$CINTYPE(*idCinem*)	10
First linear axis ID	$ABSID(*idCinem*)	1 ÷ 64
Second linear axis ID	$ORDID(*idCinem*)	1 ÷ 64
Third linear axis ID	$VRTID(*idCinem*)	1 ÷ 64
First rotary axis ID (rotary axis closer to the workpiece)	$ROTPID(*idCinem*)	1 ÷ 64
Second rotary axis ID (rotary axis closer to the tool)	$ROTTID(*idCinem*)	1 ÷ 64
Active toolholder	$TOOLHOLD(*idCinem*)	1 ÷ 4
Kinematics type: - **0** machine has an a) type kinematics (**TOL**) - **1** machine has a b) type kinematics (**MIX**) - **2** machine has a c) type kinematics (**TOL**) - **3** machine has a d) type kinematics (**WC1**) - **4** machine has an e) type kinematics (**WC2**) - **5** machine has an f) type kinematics (**TC1**) - **6** machine has a g) type kinematics (**TC2**) - **7** machine has a h) type kinematics (**C12**) - **8** machine has an i) type kinematics (**MIXM**)	$MAC(*idCinem*)	0 ÷ 8
Machine bed position: - **0** point is between the workpiece and the linear axes (after the first axis) (**WRK**) - **1** point is between the first sleeve and the first linear axis (**B12**) - **2** point is on the first sleeve (after the second linear axis) (**BC1**) - **3** point is between the second sleeve and the second linear axis (**B23**) - **4** point is on the second sleeve (after the	$BED(*idCinem)*	0 ÷ 5

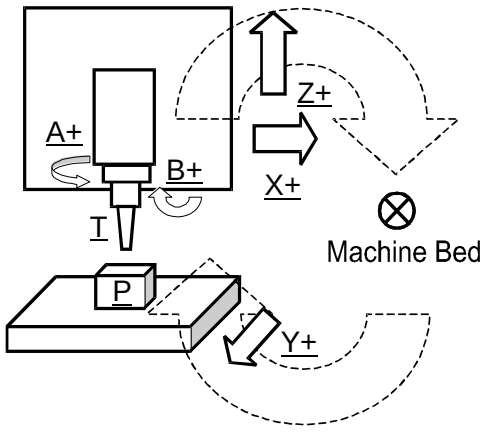
third linear axis) (**BC2**) › **5** point is between the tool and the third linear axis (**TOL**)		
First linear axis translation along the abscissa	$CINABS(*idCinem*, 0)	
First linear axis translation along the ordinate	$CINABS(*idCinem*, 1)	
First linear axis translation along the vertical axis	$CINABS(*idCinem*, 2)	
First linear axis rotation along the abscissa	$CINABS(*idCinem*, 3)	
First linear axis rotation along the ordinate	$CINABS(*idCinem*, 4)	
First linear axis rotation along the vertical axis	$CINABS(*idCinem*, 5)	
Second linear axis translation along the abscissa	$CINORD(*idCinem*, 0)	
Second linear axis translation along the ordinate	$CINORD(*idCinem*, 1)	
Second linear axis translation along the vertical axis	$CINORD(*idCinem*, 2)	
Second linear axis rotation along the abscissa	$CINORD(*idCinem*, 3)	
Second linear axis rotation along the ordinate	$CINORD(*idCinem*, 4)	
Second linear axis rotation along the vertical axis	$CINORD(*idCinem*, 5)	
Third linear axis translation along the abscissa	$CINVRT(*idCinem*, 0)	
Third linear axis translation along the ordinate	$CINVRT(*idCinem*, 1)	
Third linear axis translation along the vertical axis	$CINVRT(*idCinem*, 2)	
Third linear axis rotation along the abscissa	$CINVRT(*idCinem*, 3)	
Third linear axis rotation along the ordinate	$CINVRT(*idCinem*, 4)	
Third linear axis rotation along the vertical axis	$CINVRT(*idCinem*, 5)	
First rotary axis translation along the abscissa	$CINROTP(*idCinem*, 0)	
First rotary axis translation along the ordinate	$CINROTP(*idCinem*, 1)	
First rotary axis translation along the vertical axis	$CINROTP(*idCinem*, 2)	
First rotary axis rotation along the abscissa	$CINROTP(*idCinem*, 3)	
First rotary axis rotation along the ordinate	$CINROTP(*idCinem*, 4)	
First rotary axis rotation along the vertical axis	$CINROTP(*idCinem*, 5)	
Second rotary axis translation along the abscissa	$CINROTT(*idCinem*, 0)	
Second rotary axis translation along the ordinate	$CINROTT(*idCinem*, 1)	
Second rotary axis translation along the vertical axis	$CINROTT(*idCinem*, 2)	
Second rotary axis rotation along the abscissa	$CINROTT(*idCinem*, 3)	
Second rotary axis rotation along the ordinate	$CINROTT(*idCinem*, 4)	
Second rotary axis rotation along the vertical axis	$CINROTT(*idCinem*, 5)	
Translation of the third rotary axis along the abscissa axis	$CINROTM(*idCinem*, 0)	
Translation of the third rotary axis along the ordiante axis	$CINROTM(*idCinem*, 1)	
Translation of the third rotary axis along the vertical axis	$CINROTM(*idCinem*, 2)	
Rotation of the third rotary axis along the axis of abscissa	$CINROTM(*idCinem*, 3)	
Rotation of the third rotary axis along the axis of ordinate	$CINROTM(*idCinem*, 4)	
Rotation of the third rotary axis along the vertical axis	$CINROTM(*idCinem*, 5)	
Work axis translation along the abscissa	$CINW(*idCinem*, 0)	

Work axis translation along the ordinate	$CINW(*idCinem*, 1)	
Work axis translation along the vertical axis	$CINW(*idCinem*, 2)	
Work axis rotation along the abscissa	$CINW(*idCinem*, 3)	
Work axis rotation along the ordinate	$CINW(*idCinem*, 4)	
Work axis rotation along the vertical axis	$CINW(*idCinem*, 5)	
Tool axis translation along the abscissa	$CINT(*idCinem*, *toolHold*, 0)	
Tool axis translation along the ordinate	$CINT(*idCinem*, *toolHold*, 1)	
Tool axis translation along the vertical axis	$CINT(*idCinem*, *toolHold*, 2)	
Tool axis rotation along the abscissa	$CINT(*idCinem*, *toolHold*, 3)	
Tool axis rotation along the ordinate	$CINT(*idCinem*, *toolHold*, 4)	
Tool axis rotation along the vertical axis	$CINT(*idCinem*, *toolHold*, 5)	
Translation of the right angle head axis along the abscissa	$CINR(*idCinem*, *toolHold*, 0)	
Translation of the right angle head axis along the ordinate	$CINR(*idCinem*, *toolHold*, 1)	
Translation of the right angle head axis along the vertical axis	$CINR(*idCinem*, *toolHold*, 2)	
Rotation of the right angle head axis along the abscissa	$CINR(*idCinem*, *toolHold*, 3)	
Rotation of the right angle head axis along the ordinate	$CINR(*idCinem*, *toolHold*, 4)	
Rotation of the right angle head axis along the vertical axis	$CINR(*idCinem*, *toolHold*, 5)	
Tool radius compensation type: **0** no compensation **1** compensation with $TOOLANG, $TORANG angles **2** compensation with versors (mno) **3** compensation with versors (mno and pqd)	$COMPRAD(*idCinem*	0 ÷ 2
Real-time tool offset activation: **0** real-time offset disabled **1** real-time offset enabled	$COMPRT(*idCinem*)	0 ÷ 1
Tool-workpiece contact angle It's the subtended angle between the tool centre and the tool-workpiece contact point (see figure 2.13).	$TOOLANG(*idCinem*)	0 ÷ 360°
Contact angle (α) for spherical or toroidal tools (see figure 2.13).	$TORANG(*idCinem*)	0 ÷ 90°
Operative radius rotary workpiece TODO TODO TODO	$OPRADP(*idCinem*)	
Operative radius rotary tool TODO TODO TODO	$OPRADT(*idCinem*)	
Negative operating limit	$NEGLIMOP(*idCinem*)	
Positive operating limit	$POSLIMOP(*idCinem*)	
Operational limits mode TODO TODO TODO TODO TODO	$OPLIMMODE(*idCinem*)	

Tool direction axis name. It is used only for tool direction mode (TCP,5). It must be a valid axis name and when TCP is activated no other axes should have the same name.	$TDNAME(*idCinem*)	
Axis name “abscissa for tool direction” It is used only for tool direction mode (TCP,5). It must be a valid axis name and when TCP is activated no other axes should have the same name.	$TANAME(*idCinem*)	
Axis name “ordinate for tool direction” It is used only for tool direction mode (TCP,5). It must be a valid axis name and when TCP is activated no other axes should have the same name.	$TONAME(*idCinem*)	

Example

Assume that a machine has a workpiece fastened to the table (Y axis) with a bi-rotary head (axes A and B) referred to X and Z axes. These axes will move independently of the Y axis.



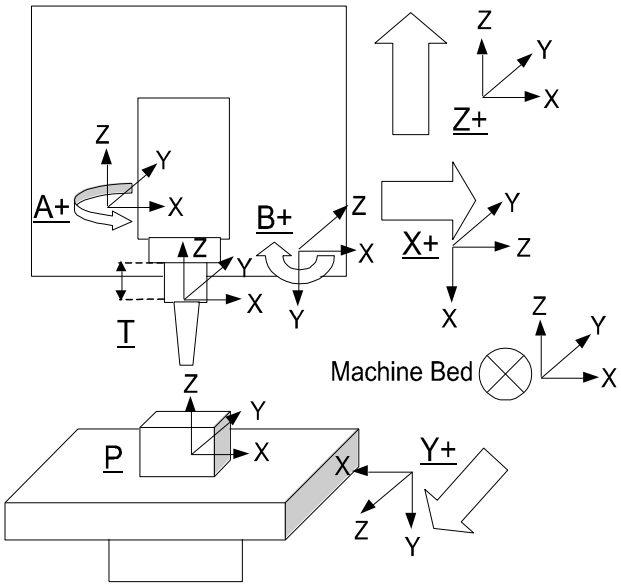
From the tool towards the workpiece, the kinematic chain is as follow:

Tool	Rot	Rot	Rot	Lin 3	Lin 2	Lin 1	Rot	Rot	Rot	Workpiece
T	---	B	A	Z	X	Y	---	---	---	P
		TOL	WRK	LI3	LI2	LI1				

Machine has a “**TOL**” type kinematics (two rotary axes next to the tool). “Fixed point” is **B12** type, placed between X and Y axes, and the two sections of the kinematic chain moves independently.

Example

With reference to the previous machine, let's define all the Cartesian terns needed to configure the machine kinematics.



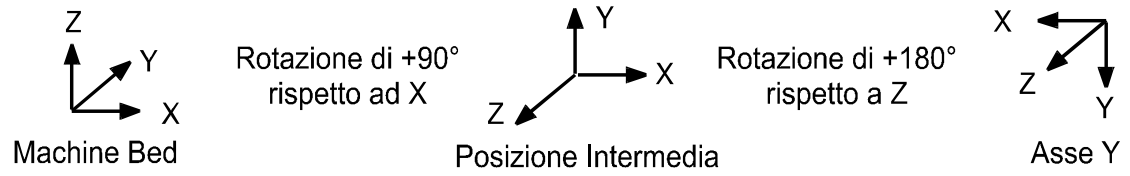
First of all, let's place the Machine Bed on the rotary A axis centre: this position will correspond to the "zero" of the X, Y, and Z linear axes as well. This positioning allows the Machine Bed and the origins of the terns associated with A, X, Y, Z to be on the same point. In addition, assume the workpiece programming origin is at the same position.

The origin changes can be programmed using the normal programming in ISO code. At this point the kinematic chain could be defined and therefore the roto-translation sequences to apply to the aforementioned terns can also be defined.

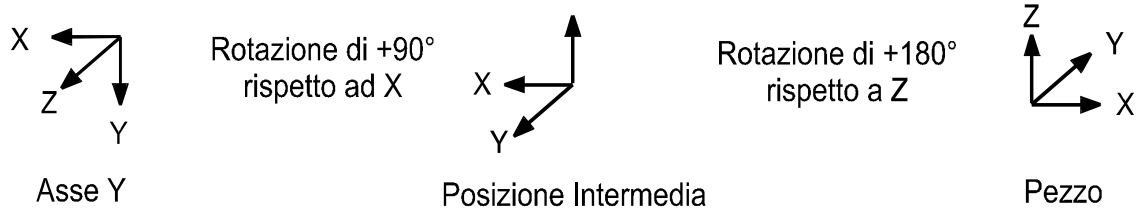
The first segment of the kinematic chain starts from the Machine Bed towards the workpiece, therefore the distance to cover is defined by the following sequence:

Machine bed -> Y Axis -> Workpiece.

Machine bed → Y Axis



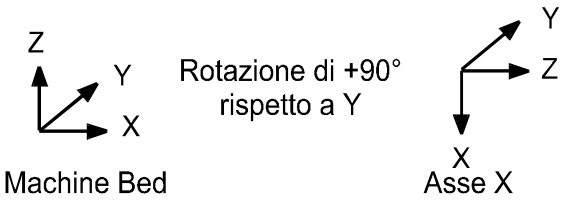
Y Axis → Workpiece



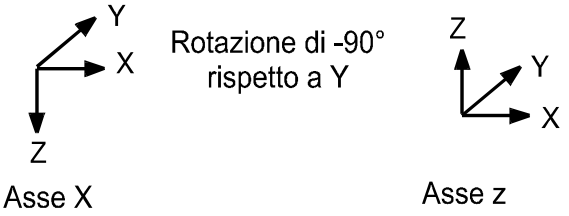
The second segment of the kinematic chain always starts from the Machine Bed and reaches the tool, describing the path as follow:

Machine bed -> X Axis -> Z Axis -> A Axis -> B Axis -> Tool.

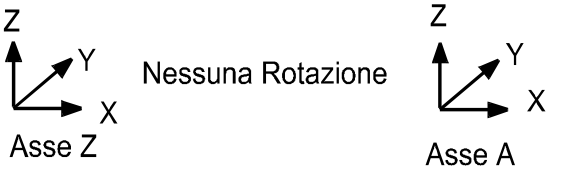
Machine bed → X Axis



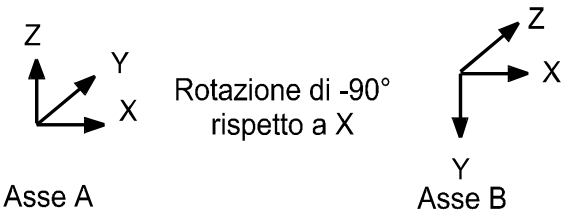
X Axis → Z Axis



Z Axis→ A Axis

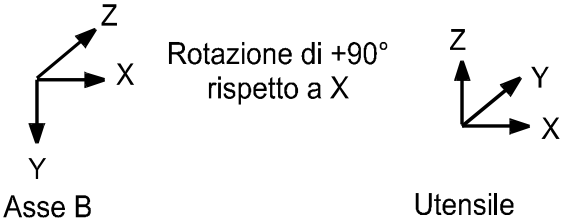


A Axis → B Axis



If the rotation centre of the two axes is on the same point, no translation needs to be defined; on the other hand, having two different centres, the translation from "Axis A" to "Axis B" must be defined. The translations must be applied before the rotation and is calculated according to the starting tern (in that case is "Axis A").

B Axis → Tool



The offset between the B rotation centre and tool-spindle mount must be defined. As this value is a translation, it has to be applied according to the "B axis" tern (in our case equals K value according to Y axis).

As there is no right angle head, translations/rotations do not have to be defined and the Cartesian terns ("tool axis" and "right angle head axis") will coincide.

So far, the kinematic chain could be resumed as follow:

Axis to reach	Offset X	Offset Y	Offset Z	Rotation on X	Rotation on Y	Rotation on Z
Y	0	0	0	+90	0	+180
Workpiece	0	0	0	+90	0	+180
X	0	0	0	0	+90	0
Z	0	0	0	0	-90	0
A	0	0	0	0	0	0
B	0	0	0	-90	0	0
Tool	0	K	0	+90	0	0
Right angle head	0	0	0	0	0	0

6.2.10 *TCP – UPR and tool offsets*

If machining is executed when there is an active kinematic, a Cartesian plane translation or there are some tools, this is the programming order to follow:

…	
hn or Tx,n	Tool offset enabling
…	
(TCP,1) or (TCP,5)	Machine kinematics enabling
…	Roto-translated plans enabling
(UPR,0,…	
…	
	Roto-translated planes disabling
(UPR)	Machine kinematics disabling
(TCP)	Tool offset disabling
h0	

7. PROGRAMMING THE SPINDLE

The functions described in this chapter must be managed by PLC.

SPINDLE FUNCTIONS

With G96/G97 the S address programs the spindle speed in rpm or as constant surface feed. In addition, several M functions affect the spindle on/off state, range, etc.

7.1 G96 G97 – CSS and RPM Programming

The following G codes control spindle speed programming.

G96 Speed programming in constant surface speed
G97 Speed programming in rpm

Syntax

G97 [*G-codes*] [*operands*]
G96 [*G-codes*] [*operands*]

where:

G-codes Other G codes that are compatible with G96 and G97 (see Table "Compatible G codes" in Chapter 2).

operands Any operand or code that can be used in a G function block.

Characteristics:

G97 activates address S programming in rpm. G97 is the default mode of the control and is modal (like G96).

G96 activates constant surface speed (CSS) programming for address S in feet per minute (G70), or metres per minute (G71). G96 forces the spindle speed to be controlled by the position of the diameter axis, so that it remains constant on the machining surface.

When G96 is activated the position of the diameter axis is assumed to be the radius for which constant surface speed (S) is programmed.

7.1.1 *Changing the S address during G96. Cancelling CSS mode*

If you want to change the S word value while G96 is active, the control must be in G00 or G29 mode. To cancel G96 CSS, the control must be in G00 mode and a block with G97 and an S word that defining the spindle rpm must be programmed.

Example:
The following example illustrates CSS programming.

- Assumed power-up G codes: G00, G27, G70, G95, G97
- Assumed gear range: 1 (M41) = 800 rpm max.

```
N1  G90 G00
N3  U5                  ;Presets diameter (U) axis
N4  SSL=700             ;700 rpm limit
N5  G96 S400 M3         ;CSS at 400 ft /min
N6  G1 U0 Z -2 F10      ;Sets contouring mode and U in feed 10 ipm
N7  U5 Z0
N8  G00                 ;Prepares to change S
N9  G01 S300 U0 Z-2     ;New surface feed = 300ft/min
N10 G00                 ;Prepares to cancel G96
N11 G97 S100            ;Set G97, S=100 rpm
N12 G01 Z0              ;Contouring to G0
N13 U5
N14 G00                 ;Rapid mode, contouring off
N15 M05                 ;Stop spindle
N16 M02                 ;Program end
```

S operators cannot be programmed when the process is on hold.

7.2 SSL – Spindle Speed Limit

The SSL command is used with G96 to set the maximum rpm that the spindle is allowed to run during CSS.

Syntax

SSL=*value*

where:

value value that can be programmed directly with a decimal number or indirectly with an E parameter.

Characteristics:

As the diameter axis approaches the spindle centreline, the spindle approaches maximum speed to maintain the programmed S word value. The SSL command limits the spindle to some value below the maximum rpm. If you program the value of SSL above the maximum rpm for the current gear range, the current gear range limit will be the maximum allowable rpm.

The SSL command must be programmed before the number of rounds **S**.

Make sure you enter this value in the part program prior to entering G96 blocks.

Example:

```
SSL = 2000          Assign a spindle speed limit of 2000 rpm
E32 = 1500
SSL = E32           Assign a spindle speed limit of 1500 rpm
```

7.3 M19 – Oriented spindle rotation stop

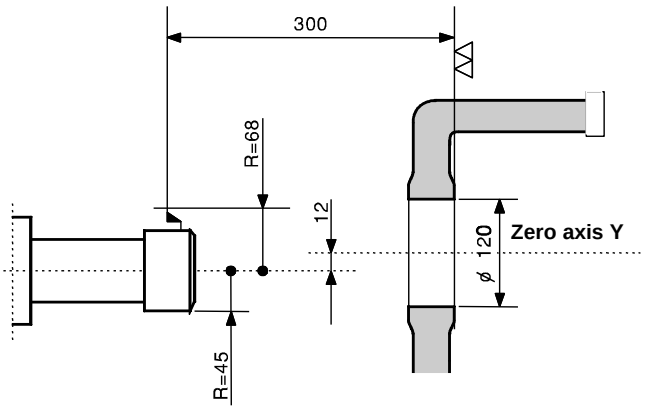
Typically, the M19 function programs a spindle stop with a predetermined angular orientation. This feature is convenient in back spot-facing operations, because it allows the spindle to position, move the X or Y axis (depending on blade orientation), enter the hole, reposition the spindle and start machining.

M19 can also be used for increased accuracy in boring operations, to avoid scoring the bored surface during the return move of the Z axis. In this case, when the hole is finished, the spindle is oriented, move the X or Y axis is moved according to the blade orientation, and the Z axis returns.

M19 is deleted by M03, M04, M13, M14. When the control reads M19 in a block that also contains movement information, M19 precedes the movement.

The M functions that are implemented in an application may be different than those described here. Consult the machine documentation for more information on application-specific M functions.

Example:



Program:

```
N32 (DIS,"BACK SPOT- FACING BAR D=136")
N33 S115 F20 T7.7 M6
N34 G X250 Y-12 M19
N35 Z-306
N36 Y M3
N37 DWT=2
N38 G1 G29 G4 Z-300
N39 G Z-302 M5
N40 Y-12 M19
N41 Z
```

In block N34 the X and Y axes are positioned and spindle orientation is controlled so that the tool can go through the hole. In the next block, the Z axis is positioned to start spot-facing.

In block N36 the tool axis is moved to coincide with the spot-facing axis.

Spot-facing is performed in block N38. In subsequent blocks, the tool is oriented, withdrawn from the workpiece and then positioned along the Y axis, so that it can go through the hole.

7.4 GTS – Spindle sharing

This three-letter code enables each process to release its spindle, to change its status, or to acquire a new one in a new mode. For the various spindle conditions, see characterisation manual.

Syntax

(GTS)
(GTS, *spindle_id***,** *mode***)**

where:

spindle_id Logic identifier of the spindle axis concerned.

mode Mode requested, **X** = exclusive, **S** = shared.

(GTS)

This is used to release a spindle. With this syntax the process tells the system it does not want to use the spindle any longer and the spindle is made available for use by other processes. Axis status is changed to RELEASED only if the release is performed by the last owner process; if an axis is still being used by other processes, its status continues to be SHARED.

(GTS, ***spindle_id***, S)

This is used to share the spindle with id *spindle_id* with other processes. The axis cannot be used by another process on an exclusive basis. Possible cases include:

- A process does not own any spindle and wants to acquire a new spindle in shared mode.
- A process owns the spindle on an exclusive basis and wants to make it available for sharing.
- A process wants to release the current spindle and to acquire a new one.

(GTS, **spindle_id**, X)

This is used to use a spindle with id *spindle_id* on an exclusive basis. The axis will not be shared with other processes. Possible cases include:

- A process does not own any spindle and wants to acquire one for exclusive use.
- A process is the only owner of a shared spindle and wants exclusive control of it.
- A process wants to release the current spindle and acquire a new one.

Non envisaged cases generate a process error.

Example:
(GTS, 4, S)

Characteristics
The second field (spindle id) can be a variable; the third field (mode) can only assume the values described. Moreover, a call to the entire part program forces the block-calculation synchronism.

8. AUXILIARY M FUNCTIONS

8.1 Standard M functions

In this chapter we briefly describe the auxiliary functions (hereinafter M functions) that are considered standard for most applications. These functions are listed in the table below.
Since the machine integrator has programmed the interface between the control and your application, the M functions that are implemented in a particular machine may be different to those described here. Consult your machine documentation for more information on application-specific M functions.

CODE	ENABLED AT MOVE START/END		CANCELLED BY	DESCRIPTION
M0		x	Cycle Start	Program Stop
M1		x	Cycle Start	Optional Program Stop
M2		x		End of Program/Subroutine exit
M3	x		M4-M5-M14-M19	Spindle CW
M4	x		M3-M5-M13-M19	Spindle CCW
M5		x	M3-M4-M13-M14	Spindle Stop
M6		x		Tool Change
M7	x		M9	Auxiliary Coolant On
M8	x		M9	Main Coolant On
M9		x	M7-M8	Coolant Off
M10	x		M11	Axes Lock
M11	x			Axes Unlock
M12	x		M11	Rotary Axes Lock
M13	x		M4-M5-M14-M19	Spindle CW and Coolant On
M14	x		M3-M5-M13-M19	Spindle CCW and Coolant On
M19	x		M3-M4-M5-M13-M14	Spindle Stop and Angular Orientation
M30		x		End of Program and Reset to 1st Block
M40	x			Deactivate Spindle Range
M41	x		M42-M43-M44-M40	
M42	x		M41-M43-M44-M40	Spindle Range

M43	x		M41-M42-M44-M40	1-2-3-4
M44	x		M41-M42-M43-M40	
M45	x		M41-M42-M43-M44	Automatic Range Change
M60		x		Part Change

AMP configuration determines whether an M function can be disabled by a control reset.

The block that includes an expedite M code must also program an axis move. Depending on how the M has been configured, the move may be point-to-point (G29) or continuous (G27-G28).

M CODE	FUNCTION DESCRIPTION
M0	**Program Stop**: M0 stops program execution, spindle rotation and coolant flow after the control has performed all the operations of the block in which it appears. The control retains all current status information after executing an M0.
M1	**Optional Program Stop**: If the command is enabled using user interface, M1 operates like the M0 code.
M2	**End of Program:** M2 defines the end of a program or forces a subroutine exit before the subroutine is finished.
M3	**Spindle Rotation Clockwise**: M3 defines spindle clockwise rotation. It is enabled by the control either when it is entered from the keyboard or when it is read from a program. In a block it is enabled before any other axis move.
M4	**Spindle Rotation Counter clockwise**: M4 defines spindle clockwise rotation. It is enabled by the control either when it is entered from the keyboard or when it is read from a program. In a block it is enabled before any other axis move.
M5	**Spindle Stop**: M5 stops spindle rotation. It is enabled by the control either when it is entered from the keyboard or when it is read from a program. In a block it is enabled after any axes moves.
M6	**Tool Change**: M6 temporarily stops the system from reading the program. It activates the offsets selected by the T function. In a block it is enabled before any other axis move. If it also stops spindle rotation and coolant flow, then M6 does not disable M3, M4, M7, M8, M13, M14. These functions become active again after M6 is completed.
M7	**Auxiliary Coolant On**: M7 turns the auxiliary coolant on. It is enabled by the control either when it is entered from the keyboard or when it is read from a program. In a block it is enabled before any other axis move
M8	**Main Coolant On**: M8 turns the main coolant on. It is enabled by the control either when it is entered from the keyboard or when it is read from a program. In a block it is enabled before any other axis move.
M9	**Coolant Off:** M9 turns all coolant systems off. It is enabled by the control either when it is entered from the keyboard or when it is read from a program. In a block it is enabled before any other axis move.

M10	**Linear and Rotary Axes Lock:** M10 locks the axes that are not involved in the current machining process.
M11	**Deactivates M10 and M12**
M12	**Rotary Axes Lock**: M12 locks only the rotary axes that are not involved in the current machining process.
M13	**Spindle Rotation Clockwise and Coolant On**
M14	**Spindle Rotation Counter clockwise and Coolant on**
M19	**Oriented Spindle Stop**: In a block, M19 spindle is oriented before any motion in the block. M19 deactivates M3, M4, M13, M14.
M30	**Automatic Reset at End of Program:** M30 deletes all information from the control dynamic buffer. Absolute origin 0 is automatically enabled and the selected program is set for restart. M30 does not deactivate the offset of the tool in the spindle.
M40	**Spindle Range Reset**
M41	**Spindle Range 1**
M42	**Spindle Range 2**
M43	**Spindle Range 3**
M44	**Spindle Range 4**
M45	**Automatic Spindle Range Change**
M60	**Workpiece Change**

8.2 M functions with parameters

Up to 8 numerical parameters can be associated with a M function call. PLC will acquire these values.

Syntax

M*n***[***param1***[, *param2* ….,*param8*]]**

where:

n a valid code of an auxiliary M function. The code can be specified directly using an integer number or using an E parameter. The valid range is 0 up to 9999

param1 … param8 a numerical value or a variable of type double

9. PARAMETRIC PROGRAMMING

Parametric programming lets the programmer use system and local variables to define geometrical and technical values for working cycles. System and local variables differ in the way they are stored and, subsequently, their scope is very different. System variables are stored in the dual port memory of the system. All the active processes can access them and they are retained after the system is switched off. Local variables are stored in a memory area local to the system and are seen only by the process they refer to.

Their value is lost when the system is switched off. At power up they are re-initialised with the value defined in AMP configuration.

The following table summarises the variables available with the system.

VARIABLE	TYPE	FUNCTION
E	local	E parameters
!*nam*	local	User variables
SN	system	System number
SC	system	System character
LS	system	System strings
TIM	system	System timer (read only)
@*nam*	system	PLC variables
L	System	System number

Except for System Characters, all system and local variables can be used in mathematical or trigonometric operations in place of the geometrical and technological data of the machining cycle.

A mathematical operation is 40 characters maximum long and is formed by arithmetic operators, functions and operands (variables or numerical constants). The following are arithmetic operators:

- addition (+)
- subtraction (-)
- multiplication (*)
- division (/)

9.1 Trigonometric functions

Trigonometric functions used by the system are listed in the table below.

FUNCTION	DESCRIPTION
SIN (A)	Calculates sine of A
COS (A)	Calculates cosine of A
TAN (A)	Calculates tangent of A
ARS (A)	Calculates arc sine of A
ARC (A)	Calculates arc cosine of A
ART (A)	Calculates arc tangent of A
ATAN2 (A, B)	Calculates the abscissas axis and the arc tangent of the angle made by A,B coordinates and the vector centred in the origin.

The arguments of a function (A,B) can be variables or numerical constants. When the control solves a mathematical equation, it considers the priority of brackets and signs. The result is converted into the format of the variable written to the left of the equal sign.

Arguments of trigonometric functions (SIN, COS, TAN) must be expressed in degrees. The result of inverse trigonometric functions (ARS, ARC, ART) is also expressed in degrees.

9.2 Mathematical functions

Mathematical functions available are listed below:

FUNCTION	DESCRIPTION
SQR (A)	Calculates the square root of A
ABS (A)	Calculates the absolute value of A
INT (A)	Calculates the integer part of A
NEG (A)	Inverts the sign of A
MOD (A,B)	Calculates the rest of the division between A and B
POW (A, B)	Calculates A value raised to B power

9.3 Logarithmic and exponential functions

Trigonometric functions used by the system are listed in the table below.

FUNCTION	DESCRIPTION
EXP10 (A)	Compute A-raised to the power of 10 (10^A)
EXP2 (A)	Compute A- raised to the power of 2 (2^A)
EXPE (A)	Compute A- raised to the power of *e* (*e*^A)
LG2 (A)	Compute the logarithm to the base 2 of A
LN (A)	Compute the natural logarithm of A
LOG (A)	Compute the logarithm to the base 10 of A

9.4 Boolean functions

Boolean functions available with the system are listed in the table below.

FUNCTION	DESCRIPTION
AND(A,B)	Executes END bit-a-bit between two numbers having value from -32768 to +32767*
OR(A,B)	Executes OR bit-a-bit between two numbers having value from -32768 to +32767*
EXOR(A,B)	Executes EXOR (exclusive OR) bit-a-bit between two numbers having value form -32768 and +32767*
NOT(A)	1s Complement of a number in the -32768 +32767 range*
BSET(A, B)	Returns the value obtained by forcing to 1 B bit of number A*
BCLR(A, B)	Returns the value obtained by forcing to 0 bit B of number A*
BTST(A, B)	Returns the value of number A bit B
A && B	Executes logical AND between two operands
A || B	Executes logical OR between two operands
A ^^ B	Executes logical XOR (exclusive OR) between two operands
NEG(A)	Inverts operand sign

* The parameters of Boolean functions must be integers in the -32768 +32767 range (short format with sign). They may be long real E (double) parameters as long as they remain in this range.

9.5 LOCAL VARIABLES

9.5.1 *E parameters*

The maximum number of E parameters must be defined during system configuration. In theory, there can be up to 8000 E parameters.

E parameters are of the Double type, which allows 17 digits in total, 17 maximum before the decimal point and 16 decimals. The system accepts several statements per block, the only restriction being block length. When in block-by-block mode, multiple statements will be executed as if they were in a single block. Two levels of parametric indexes are allowed. For example: E(E(E..)).

E parameters receive values in special assignment blocks. The format of an assignment block is:

En = *expression*

---

where:

n Is the identification number of the E parameter.

expression Can be a numerical value, a character or a mathematical equation whose result is stored in the E parameter identified by *n*.

Examples:
The following are assignment blocks for parameter calculation.

E37=(E31*SIN(E30)+123.4567)/SQR(16)	Solves the mathematical equation and assigns the result to parameter E37.
E39=-0.00000124+5	Calculates the expression and assigns the result to parameter E39.
E40=TAN(35)	Finds the tangent of 35° and assigns the result to parameter E40.
E7=81	Assigns the value 81 to parameter E7.
E25=E25+30	Adds 30 to the current value of E25 and assigns the result to E25.
E29=1,2,3,4,5	Assigns the value 1 to parameter E29, the value 2 to parameter E30, the value 3 to parameter E31, the value 4 to parameter E32 and the value 5 to parameter E33.
E(E42)=1729	Assigns the value 1729 to parameter corresponding to the value memorized in E42 parameter.

E parameters can be used inside programs and subroutines. To display the current value of a parameter, use the DIS command.

Example:
(DIS,E39) displays the current value of E39.

Example:
This example shows how to assign an ASCII character:
SC0="P" Assigns the P character to the SC0 string variable
E1=SC0
(DIS,E1) Displays 80 (ASCII code of P)

Examples of motion blocks or commands with parameters.
XE1
X-E1
X(E1)
X(-E1)
X(E8-14*SQR(E14))
X(-(E8-14*SQR(E14)))
X(E(E(E3)))

FE1
SE2
TE1.E2

9.6 ! – User variables

User defined variables can be of two types:
- Double
- Character

User variables must be defined in the ODM configuration.

A user variable name can be up to 8 characters long. The first character must be !. The extension of a user variable may be .LR or .CH.

With user variables of the character type apply this rule:

!name_var [**(***index***)**] **[.***number of characters***]CH** = *parameter*

---

where:

index — Number indicating the starting character in the variable character array. If index is not specified, it is taken to be zero. If specified, it must be programmed between round brackets.

number of characters — Specifies how many characters after the index must be read/written. The default value is 1. The sum of *index*+*number of characters* must not be greater than the maximum number of characters configured for the specified variable.

parameter — Can be:
- a string constant enclosed between apexes or quotes;
- a string variable not longer than *length*
- a numerical constant in the 0 - 255 range
- a numerical variable in the 0 - 255 range.

Example:
!ABC(1) = 125
G0 X(!ABC(1))

125 is assigned to the !ABC(1) user variable, then this variable is used as argument of the X address in a G0 block.

!CHAR(2).8CH="ABC"
This instruction writes "ABC" in the first three characters of the !CHAR user variable, starting from the second character. The remaining 5 characters (8-3) will be automatically set to zero; to prevent this, program !CHAR(2).3CH="ABC"
!CHAR(1).CH="A" or!CHAR(1).CH=65

Double type user variables have the following format:

!name_var[(*index*)]= *expression*

---

where:

index Number identifying the variable.
If *index* is not specified, it is assumed to be zero.
If specified, it must be programmed between round brackets.
It can be a number or an E parameter.

expression Can be a numerical value or a mathematical expression, whose result is stored in the user variable identified by the index.

Example:
!ABC(1) = 125
G0 X(!ABC(1))
125 is the value assigned to the !ABC(1) user variable; this variable is then used as an argument of address X in a G0 code.

NOTES:
In motion blocks, the user variable must always be written in brackets.

Examples of motion or command blocks with Double user variables:

X(!USER1(2))
X(!USER1(2)*10)
F(!USER1(1))
S(!USER1(1))
T(!USER1(1).(!USER1(2))

9.7 System variables

There are four types of system variables that can be used in part programs:

- System Number
- System Character
- System String
- System Timers
- Plus Variables

These variables can be used to read or write values or strings for assignment operations within part programs.

9.7.1 *SN – System Number*

System Number variables are of the Double type, which allows 17 signed digits with 17 integer digits maximum or 16 decimal. Up to 25 System Numbers can be defined in the 200 byte area that is available in the dual port memory of the system.

The format of a System Number variable is as follows:

SN*n* = *expression*

---

where:

n Is the identification number of the System Number variable. The n parameter can be a number or an E parameter.

expression Can be a numerical value, an equation, or a character whose result is stored in the System Number identified by *n*.

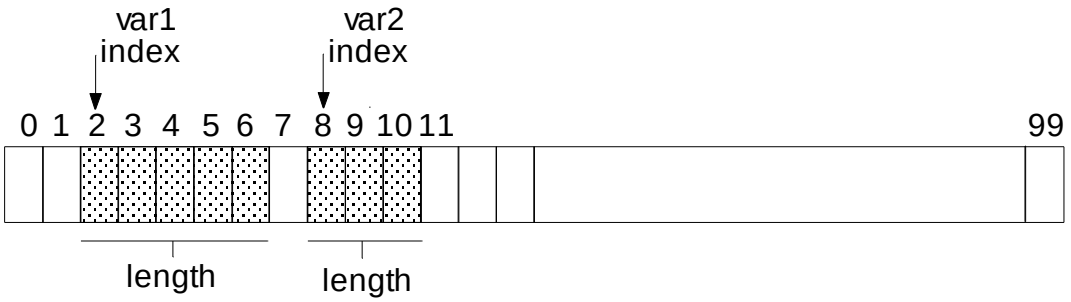
NOTE:
You can assign one System Number variable to each defined System Number.

Examples:
SN20 = 326.957
The decimal value 326.957 is assigned to the SN20 variable.
SN20 = (SN9*SIN(30) + 12.5)/SQR(81)
The result of the mathematical expression is assigned to the SN20 variable.

Examples of motion or command blocks with SN variables:
X(SN0)
X(SN0*SN1)
F(SN1)
S(SN2)
T(SN1).(SN3)

9.7.2 *SC – System Character*

System Character variables are of the character type and are stored in the dual port memory of the system, where a 100 byte array is reserved for them. This means that all the defined System Characters cannot occupy more than 100 bytes.
Each System Character is identified by an index that specifies the start address inside the array and by a length that specifies how many bytes the variables occupies starting from that address.



The format of a System Character variable is as follows:

SC*index***.***length* = *parameter*

---

where:

index Is the index that specifies the start position of the variable in the array. It may be in the 0 to 99 range. The index of the variable can be a number or an E parameter.

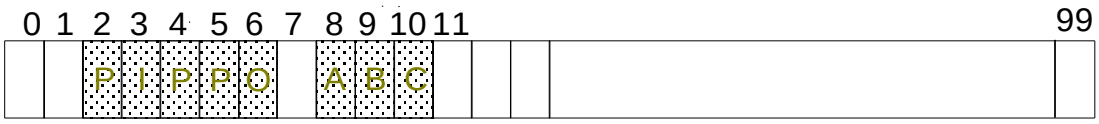
length Is the length of the variable expressed in number of characters (bytes). The length can be a number or an E parameter.

parameter A parameter may be:
- a string constant enclosed between apexes or quotes;
- a string variable not longer than *length*
- a numerical constant in the 0 - 255 range
- a numerical variable in the 0 - 255 range.

NOTES:

- For each variable, i*ndex* + *length* sum must not exceed 100.
- One System Character variable can be assigned to another System Character variable.
- Numeric variables (decimal) corresponding to ASCII characters can be assigned to SC variables. In this way it is possible to assign those characters that are not displayed. For example, 10 (LF) and 13 (CR).
- Numerical values are programmed without double quotes (" ").

Example:
SC2.5="PIPPO"
SC8.3="ABC" or SC8.3=65,66,67



The string "PIPPO" is written starting from byte 2 of the array and occupies 5 bytes.

The string "ABC" is written starting from byte 8 of the array and occupies 3 bytes.

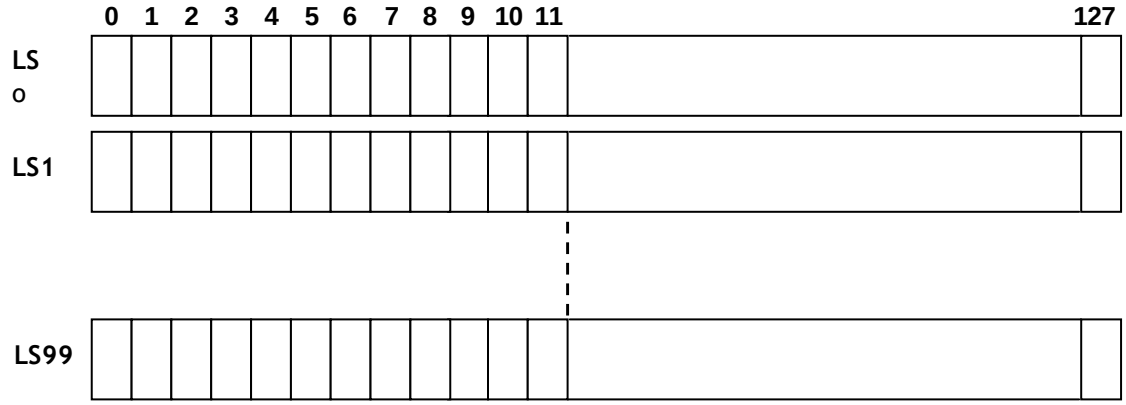
When defining the index and the length, care must be taken not to overlap two variables.

Example:
SC0.1=80 assigns 80 to the string variable
(DIS,SCO.1) displays P (ASCII code 80)

E1=80 assigns 80 to E1
SC0.1E1
(DIS,SC0.1) displays P (ASCII code 80)

9.7.3 *LS – System Strings*

A System String variable is an array of 100 strings with up to 128 characters. This means that system string variables take up 128000 bytes in system memory.
Each System String is identified by an index specifying its starting address; its length need not be specified, since the system reads or writes a string from its first character to the line terminator or up to the 128 character limit.



Syntax:

LS*index* = *parameter*

where:

index Identifies the string in the memory. It may have values in the 0 to 99 range and it may be a number or an E parameter. If no index is specified, the string is assumed to be a 0 position string

parameter It may be:

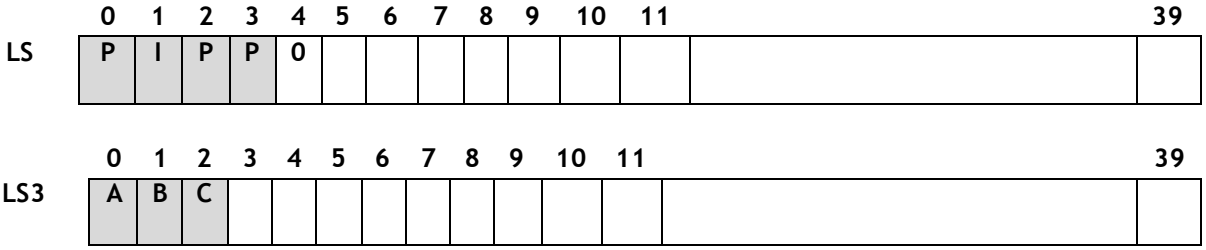
- a string constant delimited by single or double superscripts
- a string variable of up to 128 characters
- a numerical constant whose value ranges from 0 to 255
- a numerical variable whose value ranges from 0 to 255

NOTES:

- A System String variable may be assigned to another System String variable.
- Numerical variables corresponding to ASCII characters can be assigned to LS variables. In this way, it is possible to assign characters that are not visualised, e.g., 10 (LF) and 13 (CR).
- Numerical values are programmed without double subscripts (" ").

Example:

LS ="PIPPO"
LS3="ABC" o LS3=65,66,67 Assigns an "ABC" string to variable LS with index 3



Example:

Assigning a character variable value to an LS variable.
SC3.5=”PLUTO” Assigns a PLUTO string to variable SC
LS10 = SC3.5 Assigns an SC string with five characters starting from index 3 to the LS variable with index 10
(DIS,LS10) Displays PLUTO
E1=80 Assigns 80 to the variable E1
LS1=E1 Assigns the content of E1 to the string variable
(DIS,LS1) Displays P (ASCII 80 character).

9.7.4 *TIM – System Timer*

The TIM instruction defines a variable used by the programmer to read the time shown by the control timer. Its value is expressed in seconds.
A TIM instruction can be read, displayed or stored in a support variable.
Computation of this value starts as the control is switched on.

Example:
(DIS, TIM)
E10=TIM

NOTE:
TIM contents cannot be modified from part program.

9.8 @ - WinPLUS variables

From part-program it is possible to read or write the PLC variables managed by the machine logic.

Syntax

@*name*

---

where:

name Is the name of the variable declared in the configuration (AMP).
There are three types of PLUS variables:

- Short
- Boolean
- Double

The system has 2000 Short variables. Short variables names and their correspondence with WinPLUS variables are configured in AMP. They are made up of 16 bits and can contain values between -32768 and +32767.
You can also address a single bit of a Short variable with a Boolean variable. There are 256 Boolean variables, but a part program can only access 128 of them. The names of the Boolean that can be accessed by part programs must be configured in AMP.

System has 1000 Double variables, all accessible by the part program. The names of the Double variables that can be accessed by a part program must be configured in AMP.

The total number of Boolean, Short and Double variables must be configured in AMP.
The values of all the PLC variables configured in AMP are loaded when the system is started up. If the *value* field of a variable is empty, the variable is not initialised and retains the value it had when the system was switched off.

WinPLUS variables can be used in a part program as follows:

- Assignment blocks and three-letter codes

```
E10=@LOG1
(GTO,END,@LOG3=1)
```

- Motion or command blocks

```
G0 X (@LOG2)
X(@LOG2*2)
F(@LOG2)
S(@LOG3)
T(@LOG2)·(@LOG3)
```

Because these variables are directly linked to the PLC, the machine tool manufacturer must provide you with the list of the variables used in your application as well as with all the information you need in order to handle them correctly.

9.9 L variables

L variables have Double format, with 17 signed figures and a maximum of 17 integer figures. There are 3000 L variables indexed from 0 to 2999.

These variables conform with the user tables available in the table editor and the PLC environment. They can be utilised both in the programming and the logic environment, either separately or for communications between both environments.

Examples:
L10 = 26.9570
L15 = (L10*SIN (30)+9)/SQR(81)
(GTO,END,L2=1)
G0XL15
X(L15)
X(L15∗L1)
FL1
SL1
TL1.L3

Because these variables are directly linked to the PLC, the machine tool manufacturer must provide you with the list of the variables used in your application as well as with all the information you need in order to handle them correctly.

9.10 Multiple assignments

With the multiple assignment operators **{ }** it is possible to assign the contents of other variables to a certain number of variables.
Multiple assignment is accepted only for numeric variables.

Syntax

---

destination_variable **= {** *source_variable*, *number_variables* **}**

---

where:

destination_variable is the first destination variable.

source_variable is the first of the source variables.

number_variables is the number of variables to transfer. It can be an integer or a local or system variable.

Examples:
E0 = { SNO,4 }
is equivalent to the following four assignments:

E0 = SN0
E1 = SN1
E2 = SN2
E3 = SN3

E100 = 5
E50 = **{** LO,E100 **}**
is equivalent to the following five assignments:

E50 = L0
E51 = L1
E52 = L2
E53 = L3
E54 = L4

10. CANNED CYCLES

10.1 Canned cycles G8N

Codes G81 through G89 define canned cycles that let the user program multiple operations (drilling, tapping, boring, etc.) without repeating parameters and commands that are common to all the operations.

In the block where the G81-G89 canned cycle is programmed it is not possible to program additional axes moves. The cycle is stored, but not yet executed.
Canned cycle execution starts from the block that follows the definition of the G81-G89 block. To repeat a cycle a second time the user just has to program coordinates of the points at which the cycle must be executed.

The spindle axes for the canned cycle can be assigned in the G81-G89 block. For example, in the G81 R Y-20 block the Y axis is the spindle of the canned cycle. G8n functions are modal. Before programming a new canned cycle the user must cancel the current one with G80. The G80 function must be programmed in the block that follows the last canned cycle. A user cannot program a G8n block if cutter diameter compensation (G41/G42) is active.

When a canned cycle such as G82, G83, G89 requires a dwell time is possible to:

- use the default dwell time defined in AMP
- program a block containing the variable DWT=time (expressed in seconds)

10.2 Canned cycles features

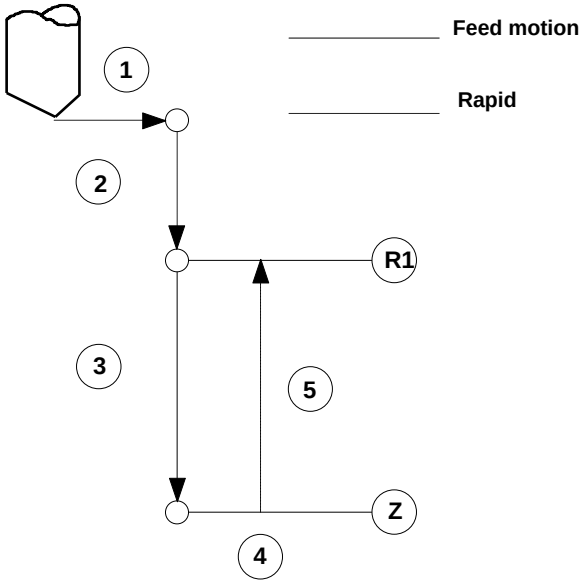
Table below lists canned cycles available and their features

CANNED CYCLE		INFEED	DWELL	ROTATION	RETURN
G81	drilling	Working	no	normal	rapid
G82	spot-facing	Working	yes	normal	rapid
G83	deep drilling with chip removal	intermittent working (down at machining rate interrupted with rapid retracts)	no	normal	rapid
G84	tapping	working spindle rotation start	no	rotation reversal	working to R1 rapid to R2 if present
G85	reaming or tapping by Tapmatic	working	no	normal	working to R1 rapid to R2 if present
G86	boring	working spindle rotation starts	no	stop	rapid
G89	boring with spot facing	working	yes	normal	working to R1 rapid to R2 if present
G80	deletes the canned cycle				

10.3 Canned cycle moves

When the system reads a canned cycle programmed in a block the axes execute the following sequence of moves:

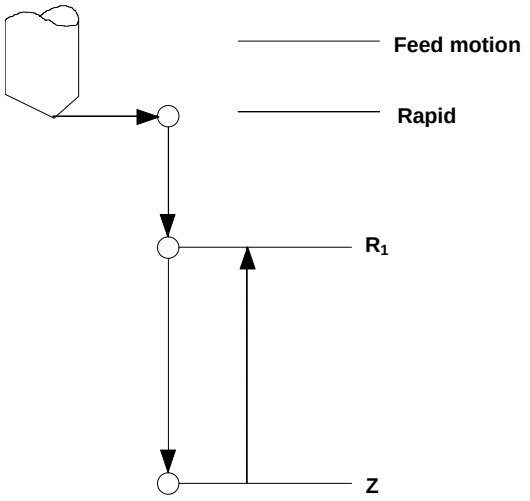
1. Rapid positioning to the centre line of the hole
2. Rapid approach to the work plane (R1 dimension)
3. Machining feedrate down to the programmed depth dimension (Z)
4. Cycle functions at the bottom of the hole
5. Rapid or machining feedrate return to the R dimension (R_2 if the return dimension is different from the approach R_1)



Examples of Canned Cycles

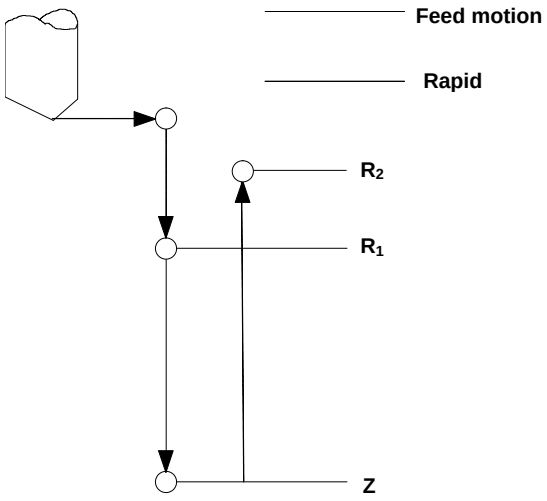
Two typical canned cycles are shown in this section.

This is a canned cycle having approaching position = return position.



G81 R_1 . . Z. .
R = approaching position
Z = hole depth

In this canned cycle the return position is different from the approach position.



G81 R. . R. . Z. .

R = approaching position (R_1)
R = return position (R_2)
Z = hole depth

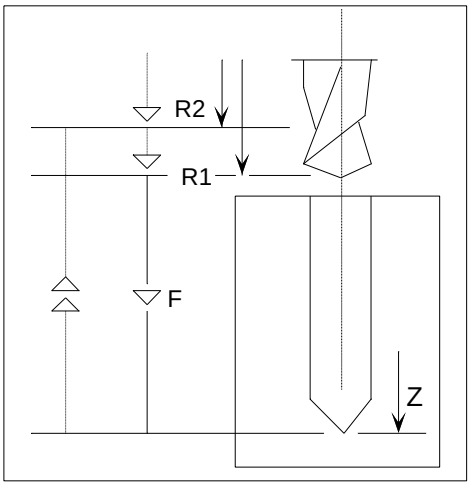
10.4 **G81 – Drilling cycle**

Syntax

G81 [*G-codes*] **R**.. [**R**..] *axis*.. [**F**..] [*auxiliary*]

where:

G-codes Other G codes compatible with G81 (see "Compatible G codes" Table in Chapter 2).



R Approach dimension (mandatory). Defines the coordinates for rapid positioning on the machining plane when the cycle starts. The **R** address is followed by the rapid approach value. It can be programmed directly with a decimal number or indirectly with an E parameter.

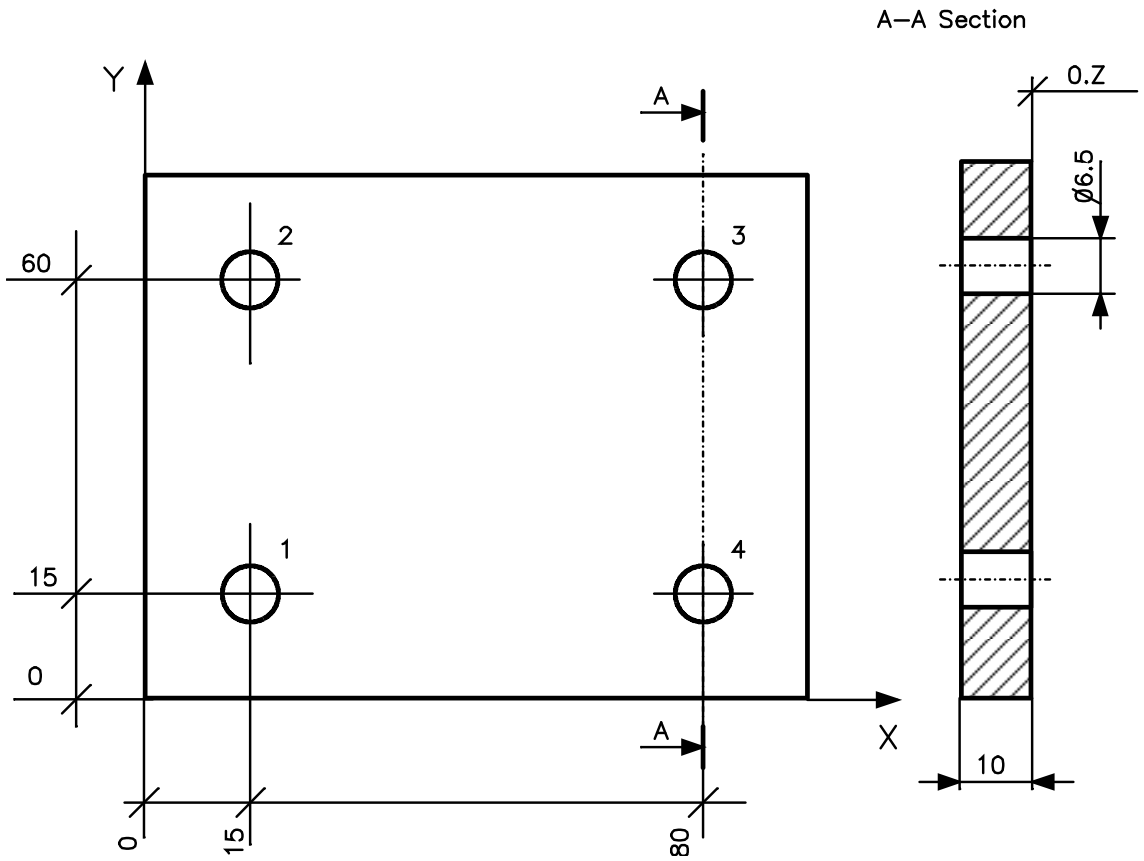
R Defines the coordinates for return after machining (R planes). The **R** address is followed by the return value. If it is omitted, the control assumes by default the approach dimension (R). R can be programmed directly with a decimal number or indirectly with an E parameter.

axis Defines the hole depth. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

F Defines the feedrate used in the canned cycle operation. It is programmed with the F address followed by the feedrate value.

auxiliary M, S or T programmable auxiliary functions. In a canned cycle block you can program up to four M functions, one S (spindle speed) and one T (tool selection).

Example:



NOTE:
0.Z means Z=0

Program:

```
      N31 (DIS,"TWIST DRILL D=6.5")
      N32 S1100 T3.03 M6
      N33 X0Y0Z10
      N34 G81 R3 Z-10 F95 M3
1     N35 X15 Y15
2     N36 Y60
3     N37 X80
4     N38 Y15
      N39 G80 Z50 M5
      N40 M30
```

10.5 **G82 - Spot facing cycle**

Syntax

G82[*G-codes*] R.. [R..] *axis*.. [F..] [*auxiliary*]

where:

G-codes Other G codes compatible with G82 (see "Compatible G codes" Table in Chapter 2).

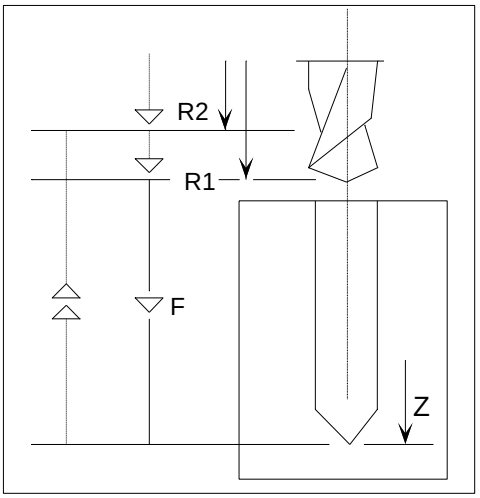
R Defines the value of hole start (see G81).

R Defines the coordinates for return (see G81).

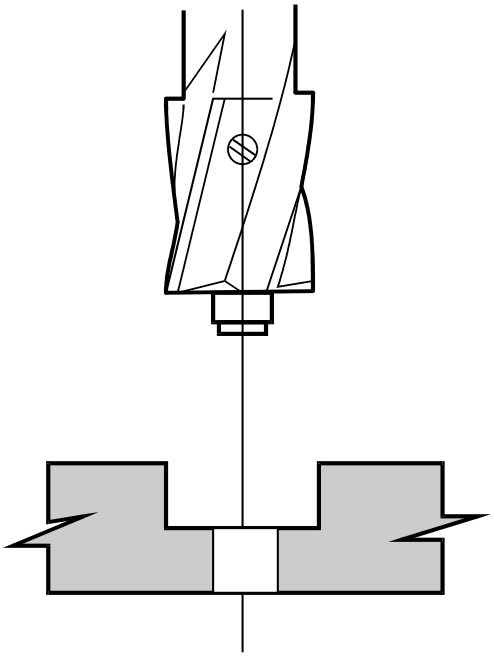
axis Defines the value of hole end (see G81).

F Defines the value of feedrate(see G81).

auxiliary M, S or T programmable auxiliary functions.



Example:
The following figure shows a spot facing cycle.



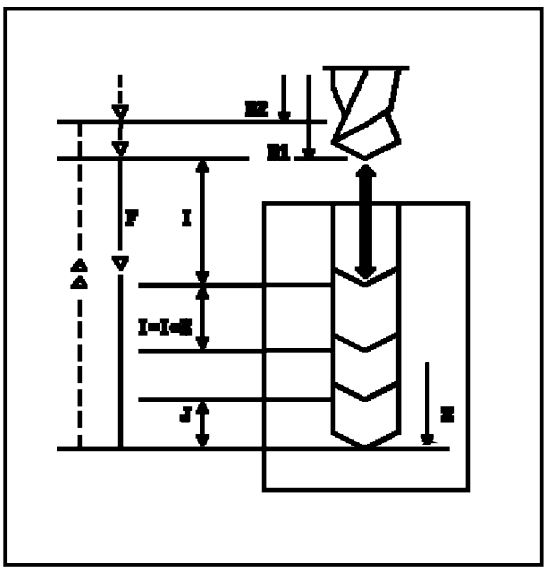
G82 R5 Z-15 F.. M..
X.. Y..

10.6 G83 – Deep drilling cycle

Syntax

G83 [*G-codes*] R.. [R..] *axis*.. I.. [J..] [K..] [F..] [*auxiliary*]

where:



G-codes Other G codes compatible with G83 (see "Compatible G codes" Table in Chapter 2).

R Defines the value of hole start (see G81).

R Defines the coordinates for return (see G81).

axis Defines the value of hole end (see G81).

I Defines the depth increment after each pull-out for chip discharge. The I address is followed by the depth increment value.

J Defines the minimum depth increment after which the cycle applies a constant increment. The J address is followed by the minimum depth increment value.

K Defines the reduction factor applied to *I* until the *J* value is reached. The K address is followed by a numerical value.

F Defines the value of feedrate (see G81).

auxiliary M, S or T programmable auxiliary functions.

Characteristics:

Depending on the value and presence/absence of **I**,**J**, and **K** parameter G83 generates different moves.

If **I**,**J** and **K** are programmed axes behave as follows:

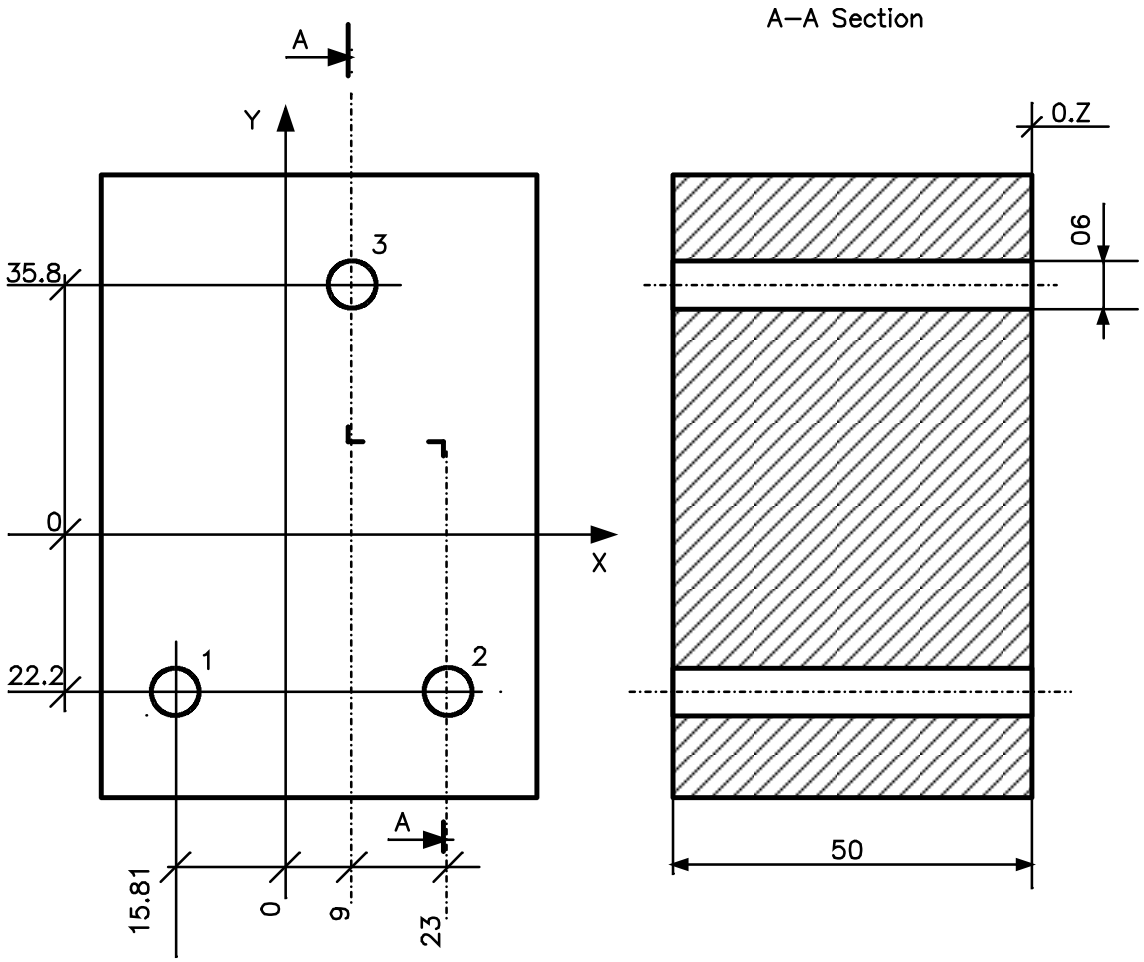
1. Rapid approach to the hole centre line
2. Rapid approach to the $\mathbf{R_1}$ coordinates
3. Machining feed rate to the $\mathbf{R_1}$ + **I** coordinates
4. Rapid retract to the initial $\mathbf{R_1}$ coordinates each time a chip discharge occurs.
5. Calculation of the new **R** value with the formula:
   R = R_{old} + I - X (where X is 1 by default, or, if the DRP variable is programmed, it assumes the value assigned to this variable)
6. Calculation of the new I value with the formulas:
   I = I_{old} * K if I * K > J
   I = J if I * K < J
7. Repetition of points 2 through 6 until the Z dimension is reached.

10.6.1 *Chip breaking cycle*

If **J** and **K** parameters are not programmed, the following movements are generated:

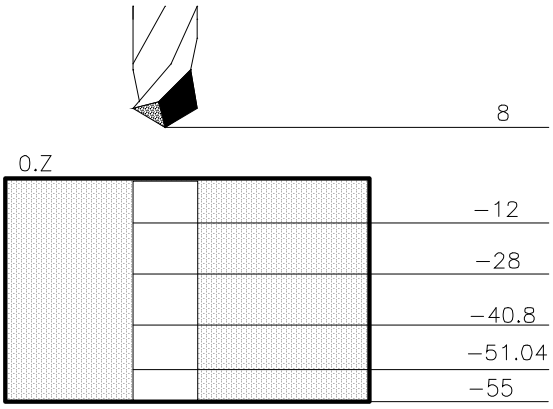
1. Rapid approach to the hole centre line
2. Rapid approach to the R coordinates
3. Machining rate to R + I
4. Spindle dwell for the time programmed in the DWT function, or for the characterized time if DWT is not programmed.
5. Repetition of points 3 and 4 until the Z dimension is reached.

Example:



NOTE:

0.Z means Z=0



Program:

```
        N65 (DIS,"TWIST DRILL D=6")
        N66 S930 F65 T6.6 M6
        N67 G83 R8 Z-55 I20 K.8 J6 M13
1       N68 X-15.81 Y-22.2
2       N69 X23
3       N70 X9 Y35.8
        N71 G80 Z50 M5
        N72 M30
```

10.7 DRP – G83 hole reworking distance

The DRP command defines in mm the hole reworking distance in G83 cycles (with IJK versors programmed).
The system initializes this value at 1. It can be assigned from the Keyboard or the Part Program.

Syntax:

DRP = *value*

---

where:
value can be programmed directly or indirectly with an E parameter.

Example:
DRP=0.01 hole reworked at 0.01 mm
DRP=5 hole reworked at 5 mm

RESET restores the value of 1.

10.8 G84 – Tapping cycle without transducer

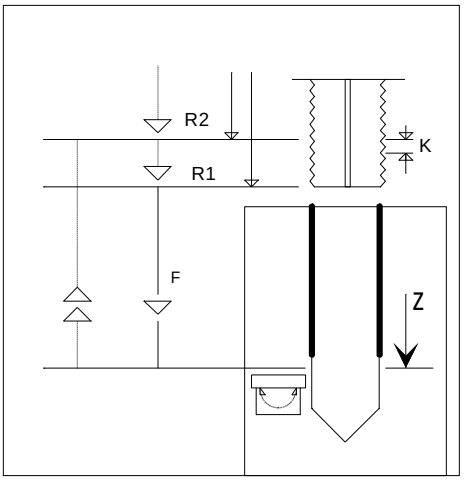
This G84 code operates when a transducer is not mounted on the spindle. The spindle must belong to a process in exclusive mode.

Syntax

G84 [*G-codes*] **R**.. [**R**..] *axis*.. [**F**..]

where:

G-codes Other G codes compatible with G84 (see "Compatible G codes" Table in Chapter 2).



R Approach dimension (mandatory). Defines the coordinates for rapid positioning on the machining plane when the cycle starts. The **R** address is followed by the rapid approach value. It can be programmed directly with a decimal number or indirectly with an E parameter.

R Defines the coordinates for return after machining (R planes). The **R** address is followed by the return value. If it is omitted, the control assumes by default the approach dimension (R). R can be programmed directly with a decimal number or indirectly with an E parameter.

axis Defines the end of tapping points. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

F Defines the feed rate used in the canned cycle operation. It is programmed with the F address followed by the feed rate value.

Characteristics:

The tool that is approaching the workpiece at rapid must stop at five times the tap pitch (for depths less than 3.O) or seven times the tap pitch (for depths greater than 3.O) from the workpiece.

Tapping feedrate must be calculated with the following formula:

$$F = S * p * 0.9$$

where:

S spindle rotation speed

p tap pitch

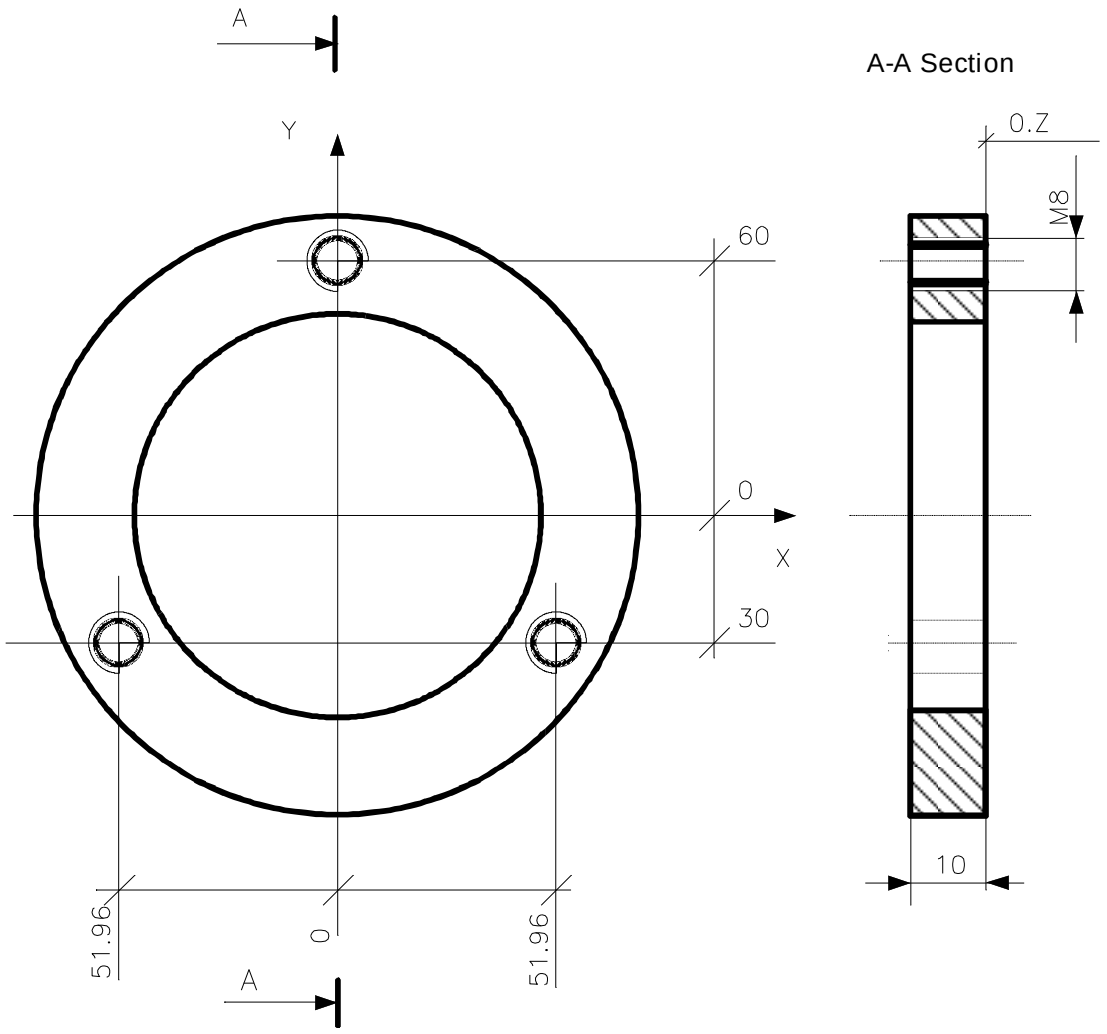
0.9 feedrate decrease factor to keep the tool holder spring compensator stretched

The final position must be decreased by 10% of the actual tap working travel.

The final tapping position must be long enough for the axis to reach the programmed feedrate and stop with controlled deceleration. It must be calculated according to the time it takes spindle rotation to stop. If the final tapping position is not long enough, the control displays an error.

M functions are not allowed in the G84 block.

Example:



NOTE:
0.Z means Z=0

Program:
N90 (DIS,"TAP M8-TRACTION TYPE COMPENSATOR")
N91 S280 F315 T8.8 M6 M13
N92 G84 R7 Z-15
N93 X-51.96 Y-30
N94 X51.96
N95 X Y60
N96 G80 Z50 M5

This program is valid for R.H. tapping operations, because M13 is programmed in block N91. For L.H. tapping operations, simply program M14 (or M04) instead of M13 (or M03).

10.9 G84 – Tapping cycle with transducer on the spindle

This G84 code operates when a transducer is mounted on the spindle.

Syntax

G84 [*G-codes*] R.. [R..] *axis*.. K.. [F..] [*auxiliary*]

where:

G-codes Other G codes compatible with G84 (see "Compatible G codes" Table in Chapter 1).

R Approach dimension (mandatory). Defines the coordinates for rapid positioning in the machining plane when the cycle starts. The R address is followed by the rapid approach value. It can be programmed directly with a decimal number or indirectly with an E parameter.

R Defines the coordinates for return after machining (R planes). The R address is followed by the return value. If it is omitted, the control assumes by default the approach dimension (R). R can be programmed directly with a decimal number or indirectly with an E parameter.

axis Defines the end of tapping points. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

K Defines the thread tap pitch. The K address is followed by a value.

F Defines the feedrate used in the canned cycle operation. It is programmed with the F address followed by the feedrate value.

auxiliary M, S or T programmable auxiliary functions (see G81).

Characteristics:

With a transducer mounted on the spindle, G84 code can be programmed as follows:

- by calculating the F feedrate as if there were no spindle on the transducer.
- by using the K thread pitch. In this case the control automatically calculates the feedrate by multiplying K by the spindle speed in rpm.

During the tapping cycle the control ignores the CYCLE STOP pushbutton (except in the rapid traverse approach section) and the FEEDRATE OVERRIDE selector (or softkey).
The SPINDLE SPEED OVERRIDE selector must be disabled by the machine logic. To abort the tapping cycle, logical functions may be used (see variable description in PLC Variables Manual).

Example:

```
N90 (DIS,"TAP M8")
N91 S280 T8.8 M6 M3
N92 G84 R7 Z-15 K1
N93 X-51.96 Y-30
N94 X51.96
N95 X Y60
N96 G80 Z50 M5
```

10.10 G84 – Solid or Rigid tapping cycle with a transducer mounted on the spindle

Syntax

G84 [*G-codes*] **R**.. [**R**..] *axis*.. **K**..[*auxiliary*]

where:

G-codes Other G codes compatible with canned cycle G84 (see "Compatible G codes" Table in Chapter 2).

R Approach dimension (mandatory). Defines the coordinates for rapid positioning in the machining plane when the cycle starts. The R address is followed by the rapid approach value. It can be programmed directly with a decimal number or indirectly with an E parameter.

R Defines the coordinates for return after machining (R plane). The R address is followed by the return value. If it is omitted, the control assumes by default the approach dimension (R). R can be programmed directly with a decimal number or indirectly with an E parameter.

axis Defines the end of tapping point. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

K Defines the male threading pitch. It is given by the K address followed by a value.

auxiliary Other M, S, T functions (see G81).

Characteristics:

To perform the tapping cycle correctly it is necessary to load two machine parameters with the appropriate values, which, as a rule, vary from machine to another.
These parameters are called TKG and TAG.
Accordingly, rigid tapping cycles must be preceded by the definition of these two parameters with the values obtained at the machine installation stage.

It is recommend to create a paramacro for the loading of these values (for instance, G840), to be called only once within the program that performs the rigid tapping operation.
Hence, the paramacro will be structured as follows:

TKG=……
TAG=……

If a RESET command (which stops the program) is executed, the TKG and TAG parameters are deleted and will have to be assigned again for the correct execution of rigid tapping.

N.B. Methods for the determination of the correct values of the parameters are described in the “Characterization Manual”.

10.11 TKG – Constant speed gain in solid or rigid tapping

TAG – Acceleration gain in solid tapping

The variables describe the delay offset coefficients in the servo system axes control and spindle application of the command in case of solid tapping.

Syntax

TKG = *value*
TAG = *value*

where:

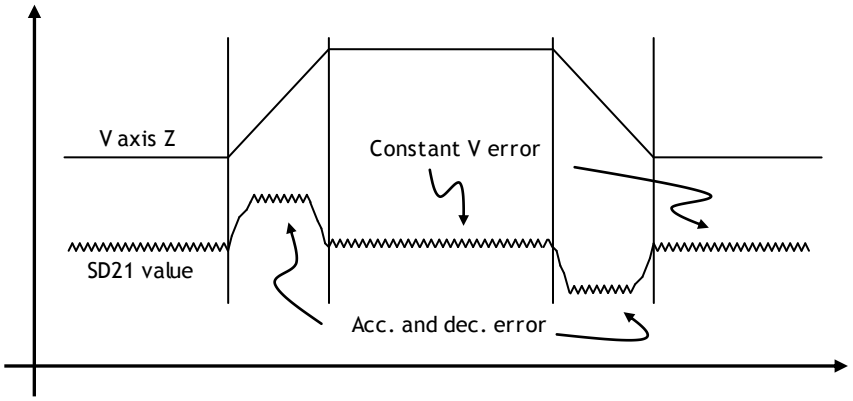
value is a real value describing the commands application delay at constant speed (**TKG**) and during acceleration/deceleration phases (**TAG**).

Characteristics:
In order to have a good performance of the solid tapping cycle, it is important to determine and store the parameters of the constant speed gain and of the acceleration/deceleration. This procedure and storage uses the same VVf configuration used during the machining.

The correct values to be assigned to these two parameters are identified by tapping cycles without a workpiece.
During the tapping cycle, the error between the theoretical position and the position read on the tapping axis is stored in the SD21 variable (when working with process 1, otherwise the value will be stored in the SD related to the process in use). The current position of the tapping axis is saved in SD22 variable.
The best TKG and TAG values are the ones reducing the error, therefore reducing the SD21 value.

From an operational perspective, it is better to work first on the TKG parameter trying to set the error as close as possible to 0 at constant feed, ignoring the movement of the error when accelerating. Once the TKG value is calculated, the user can start the search for the best value of the TAG parameter.

Values set for TAG and TKG, will be valid for all kinds of tapping, even with pitch or spindle feed different from those used during the tests



10.12 TRP – Tapping Return Percentage

The TRP command defines the percentage change to the feedrate applied in the retract phase of the tapping cycle. This command is normally defined in a program, but can also be used in blocks entered with a keyboard command or by means of a softkey.

Syntax

TRP = *value*

---

where:

value can be programmed directly with an integer number or indirectly with an E parameter of the byte type.

Example:

TRP = 110 represents +10% of the programmed F

TRP = 10 represents - 90% of the programmed F

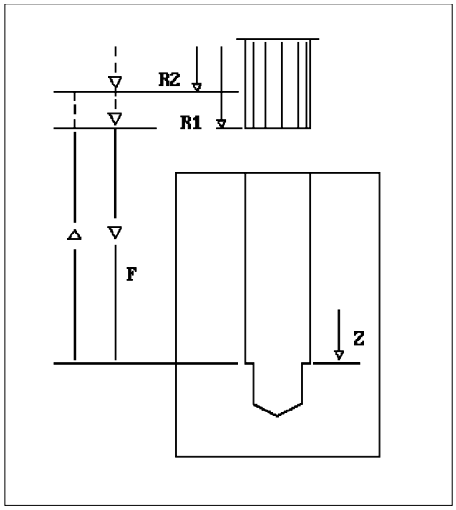
10.13 **G85 – Reaming cycle (or tapping by Tapmatic)**

Syntax

G85 [*G-codes*] R.. [R..] *axis*.. [F..] [*auxiliary*]

where:

G-codes Other G codes compatible with G85 (see "Compatible G codes" Table in Chapter 2).



R Approach dimension (mandatory). Defines the coordinates for rapid positioning in the machining plane when the cycle starts. The **R** address is followed by the rapid approach value. It can be programmed directly with a decimal number or indirectly with an E parameter.

R Defines the coordinates for return after machining (R planes). The **R** address is followed by the return value. If it is omitted, the control assumes by default the approach dimension (R). R can be programmed directly with a decimal number or indirectly with an E parameter.

axis Defines the end of reaming/tapping point. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

F Defines the feedrate used in the canned cycle operation. It is programmed with the F address followed by the feedrate value.

auxiliary M, S or T programmable auxiliary functions (see G81).

10.14 **G86 – Boring cycle**

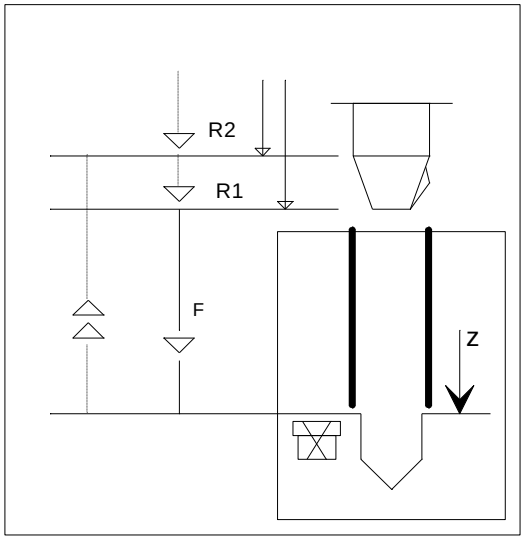
The spindle must belong to a process in exclusive mode.

Syntax

G86 [*G-codes*] **R**.. [**R**..] *axis*.. [**F**..] [*auxiliary*]

where:

G-codes Other G codes compatible with G86 (see "Compatible G codes" Table in Chapter 2).



R Defines the rapid approach coordinates and the fixed return to work. This parameter has the **R** address followed by an integer. It can be programmed directly (using decimals) or indirectly (using E parameters). The approaching position is mandatory.

R Defines the rapid return coordinates after machining. This parameter is set with **R** address followed by a return value. If this position is missing, the approach position is automatically set by the control as return position. It can be programmed directly (using decimals) or indirectly (using E parameters).

axis Defines the end of boring point. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

F Defines the feedrate used in the canned cycle operation. It is programmed with the F address followed by the feedrate value.

auxiliary M, S or T programmable auxiliary functions (see G81).

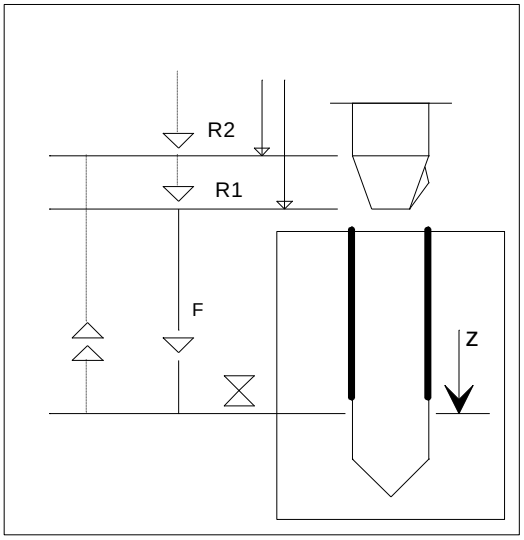
10.15 **G89 – Boring cycle with spot facing**

Syntax

G89 [*G-codes*] **R..** [**R**..] *axis*.. [**F**..] [*auxiliary*]

where:

G-codes Other G codes compatible with G89 (see "Compatible G codes" Table in Chapter 2).



R Approach dimension (mandatory). Defines the coordinates for rapid positioning in the machining plane when the cycle starts. The **R** address is followed by the rapid approach value. It can be programmed directly with a decimal number or indirectly with an E parameter.

R Defines the coordinates for return after machining (R planes). The **R** address is followed by the return value. If it is omitted, the control assumes by default the approach dimension (R). R can be programmed directly with a decimal number or indirectly with an E parameter.

axis Defines the end of boring point. It is defined by an axis name (typically Z) followed by the depth value, which can be programmed directly with a decimal number or indirectly with an E parameter.

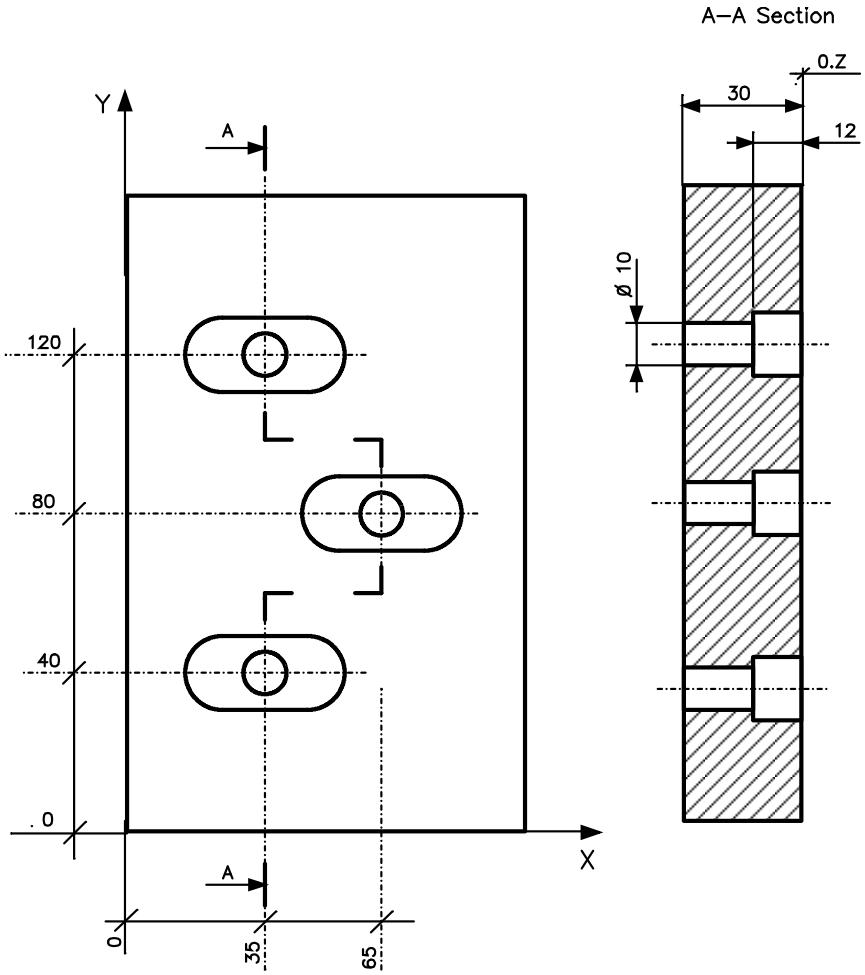
F Defines the feedrate used in the canned cycle operation. It is programmed with the **F** address followed by the feedrate value.

auxiliary M, S or T programmable auxiliary functions. (see G81).

10.15.1 *Using two R positions in a canned cycle*

You can program two Rs in a canned cycle when holes must be drilled, tapped, reamed, etc., on the same plane, but are separated by obstacles (clamps, holes on repeated sub-planes, etc.)

Example:



NOTE:
0.Z means Z=0

Program:
N42 (DIS,"TWIST DRILL D=10")
N43 S850 F100 T4.4 M6
N44 G81 R-10 R2 Z-36 M3
N45 X35 Y40
N46 X65 Y80
N47 X35 Y120
N48 G80 Z50 M5

10.15.2 *Updating Canned Cycle Dimensions*

When a canned cycle is active in a program, you can program blocks with rapid approach, return and depth coordinates in order to update the cycle moves without re-programming the cycle.

Program blocks whose format is X, Y, R (approach), R2 (return), Z (where Z is the canned cycle axis) are performed in this order:

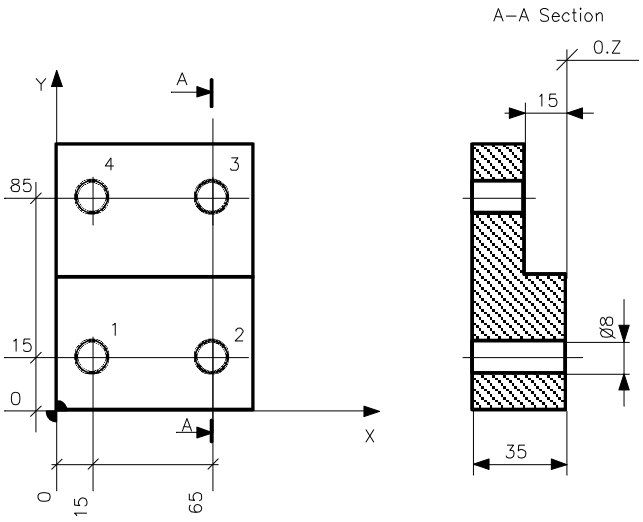
1. X and Y
2. Updated R1 - new rapid approach coordinates
3. Updated Z - new depth
4. Updated R2 - new return coordinates

The table below summarizes the program block formats used for updating canned cycles:

BLOCK	ACTION
X.. Y.. Z..	Performs a canned cycle at XY with new Z depth
X.. Y.. R..	Performs a canned cycle at XY with new R (approach) rapid plane
X.. Y.. R.. R..	Performs a canned cycle at XY with new (R) rapid plane and (R) return dimension
X.. Y.. R.. Z..	Performs a canned cycle at XY with new R rapid plane, and Z depth
X.. Y.. R.. R.. Z..	Performs a canned cycle at XY with new R rapid plane, R return dimension, and new Z depth
R..	Updates the R rapid plane and does not perform canned cycle at the current location
R.. R..	Updates R rapid plane and R return dimension; does not perform a canned cycle at the current location
R.. Z..	Updates R rapid plane and Z depth; does not perform a canned cycle at the current location
R.. R.. Z..	Updates R rapid plane, R return, and Z depth dimensions; does not perform a canned cycle at the current location

10.15.3 *Updating R dimensions (upper limit and lower limit) during execution*

Example 1:

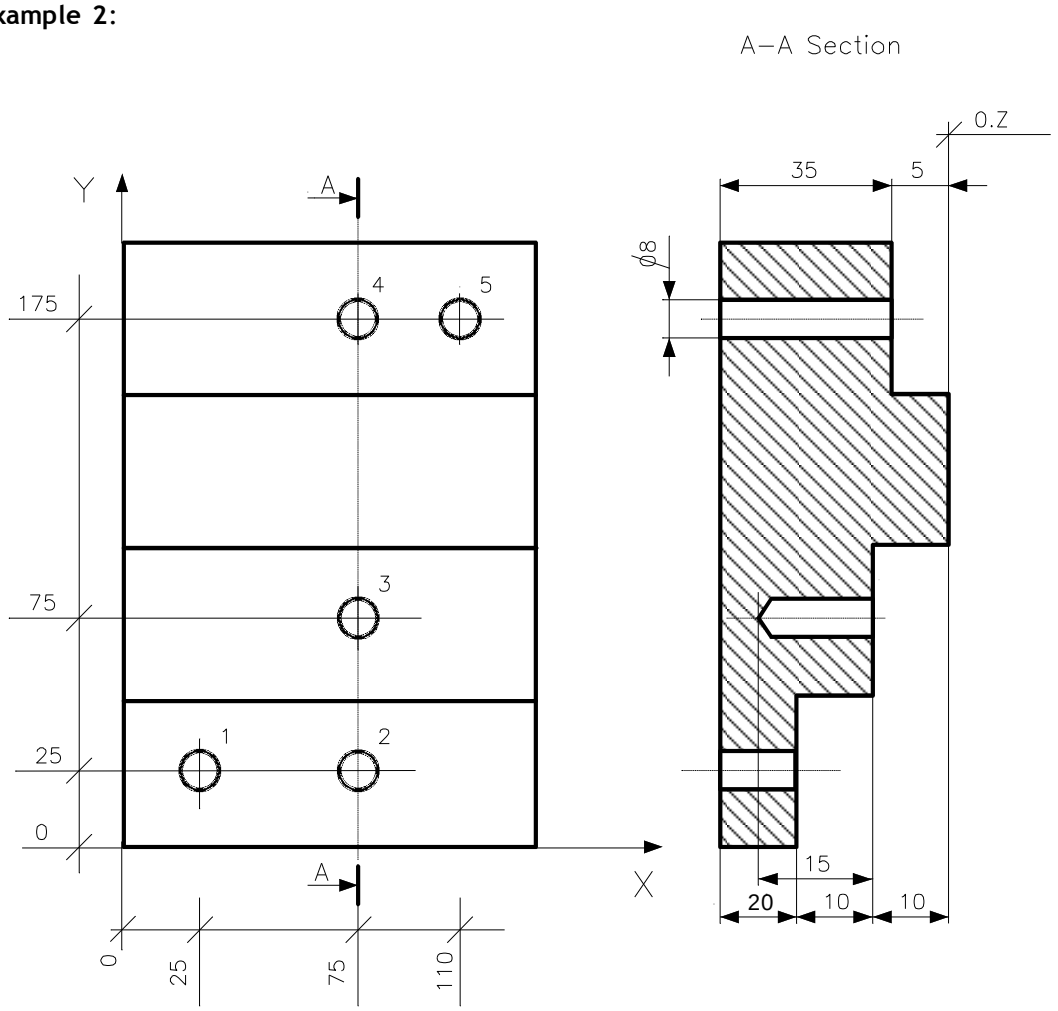


NOTE:
0.Z means Z=0

Program:

```
      N35 (DIS,"TWIST DRILL D=8")
      N36 S1000 F100 T4.4 M6
      N37 G81 R3 Z-42 M3
1     N38 X15 Y15
2     N39 X65
3     N40 Y85 R-13
4     N41 X15
      N42 G80 Z50 M5
```

Example 2:

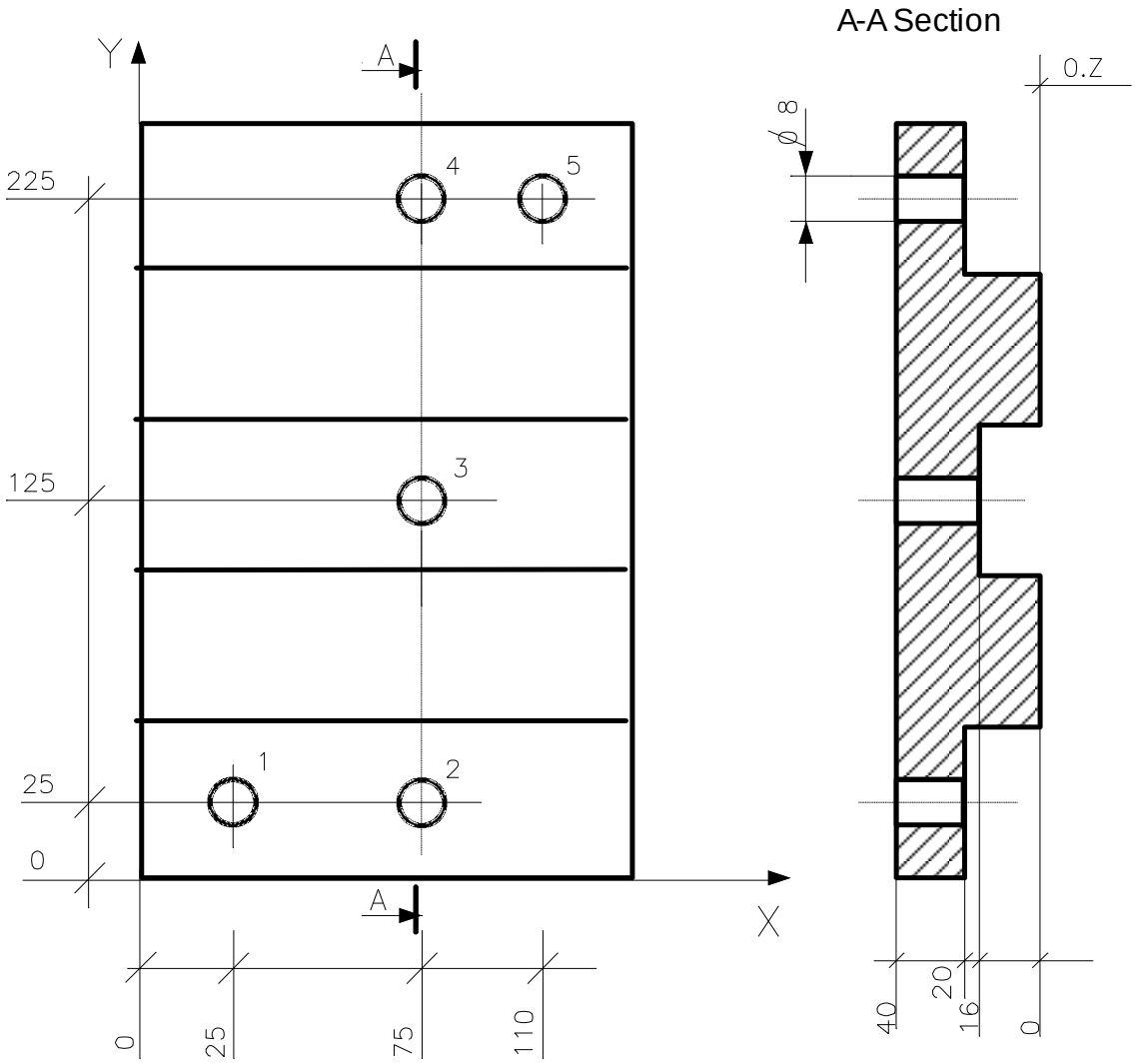


NOTE:
0.Z means Z=0

Program:

```
      N42 (DIS,"TWIST DRILL D=8")
      N43 S1000 F100 T5.5 M6
      N44 G81 R-18 Z-46 M13
1     N45 X25 Y25
2     N46 X75 R-18 R-8
3     N47 Y75 R-8 R2 Z-25
4     N48 Y175 R-3 Z-46
5     N49 X110
      N50 G80 Z50 M5
```

Example 3:



NOTE:
0.Z means Z=0

Program:

N42 (DIS,"TWIST DRILL D=8")
N43 S1000 F100 T4.4 M6
N44 G81 R-18 Z-42 M3 M8
1 N45 X25 Y25
2 N46 X75 R-18 R4
3 N47 Y125 R-14 R4
4 N48 Y225 R-18
5 N49 X110
6 N50 G80 Z2 M5

11. PARAMACRO

11.1 Paramacro Definition

Paramacro subroutines can be used in user-defined cycles. They are called using a 3-digit code.
Modal paramacros are active only in motion blocks that do not include M functions. No other function or code can be programmed when a modal paramacro is active.

Syntax

G*n* [*ar-name-1* [*valore-1*] . . . [*par-name-n value-n*] . . . ["*string*"]

where:

n	Is a number from 300 to 998
par-name-1. . . *par-name-n*	Are letters (refer to the correspondence table)
value-1 . . . *value-n*	Can be a number or an E parameter or a parametric expression.
string	Characters string (max. 99)

Characteristics:
There are two groups of paramacros:

- from G300 to G699 non modal paramacros
- from G700 to G998 modal paramacros

Modal paramacros are reset by G999.

The H, HF and HC parameters are used in the paramacros, The system offers 100 H parameters. Some of them can be used in paramacros. H parameters are classed as follows:

- Parameters from H0 to H51 are associated with letters and cannot be used in paramacros (see equivalence table).
- Parameters from H52 to H59 are associated to parameters used by M functions used as paramacro (see equivalence table).
- Parameters from H60 to H99 can be used for math operations in paramacros.

H parameters can be used in paramacros to perform all the operations available to E parameters.
Paramacros allow 4 nesting levels.

According to the ISO compatibility type defined n AMP, H variables will exist to global and local level.
When a series of paramacros are nested, the H, HF and HC parameters in a nested paramacro will reset the equivalent H, HF and HC parameters in higher paramacro levels.

In case of local H variables, values of the H, HF and HC parameters having a lower level do not change values of corresponding H, HF and HC variables at the upper level. The H variables values are maintained towards the lower level, but not from the HF variables values.
When a letter is programmed, the corresponding HF parameter is set to 1 (Boolean type). It makes it possible to check that all parameters of a paramacro have been programmed.

Example:
G300 A1 B2 C3 D4

H0 is assigned to 1
H1 is assigned to 2
H2 is assigned to 3
H3 is assigned to 4

HF0 is set
HF1 is set
HF2 is set
HF3 is set

All other HF are reset.

If G300 calls:
G400 A10 C30,
If H variables are configured as local or global, the result will be different. In case of global H variables then the value of H0 will be 10 and the value of H2 will set to 30. HF0 and HF2 will be set and all other HF's will be reset.

In case of local H variables, H0=10, H2=30, HF0/HF2 will be forced at 1 and all the other HF will be set at zero, H1=2 and H3=4. Values given to H0 and H2 are not lost as at the end of G400 execution all H variables will have again the value they had before the G400 call.

E parameters (and all other parameters available in CNC OPEN*control*) can be used in paramacros if needed. However, to avoid interaction with parameters used elsewhere, it is preferable to use H parameters within G paramacros.

11.2 HC Parameters

If a block calling a paramacro also includes a character string between inverted commas, this string is made available to the paramacro in the HC character array (100 characters).
If no such string is specified, the entire HC array is reset.

Array is always unique regardless the ISO compatibility type configured by the ODM.

If the string was programmed, it is put in the HC variables with a terminator '0' added; for this reason, maximum length of the string is 99, even if the HC array is 100 characters long.

Examples:

```
!PROFILE=.....
G600 "PROFILE" A50 ;call to paramacro G600
. .
. .

;------------------ in the paramacro.....
(DIS,HC0.10)
(CLS,?HC0.10)
```

State of the HC parameters after the call to the paramacro:

HC0="P" HC1="R" HC2="O" HC3="F" HC4="I" HC5="L" HC6="O" HC7=0
H0=50 HF0=1

The following table shows the correspondence between letters and H parameters.

LETTER	PARAMETERS		LETTER	PARAMETERS	
	H	HF		H	HF
A	H0	HF0	a	H26	HF26
B	H1	HF1	b	H27	HF27
C	H2	HF2	c	H28	HF28
D	H3	HF3	d	H29	HF29
E	H4	HF4	e	H30	HF30
F	H5	HF5	f	H31	HF31
G	H6	HF6	g	H32	HF32
H	H7	HF7	h	H33	HF33
I	H8	HF8	i	H34	HF34
J	H9	HF9	j	H35	HF35
K	H10	HF10	k	H36	HF36
L	H11	HF11	l	H37	HF37
M	H12	HF12	m	H38	HF38
N	H13	HF13	n	H39	HF39
O	H14	HF14	o	H40	HF40
P	H15	HF15	p	H41	HF41
Q	H16	HF16	q	H42	HF42
R	H17	HF17	r	H43	HF43

S	H18	HF18	s	H44	HF44
T	H19	HF19	t	H45	HF45
U	H20	HF20	u	H46	HF46
V	H21	HF21	v	H47	HF47
W	H22	HF22	w	H48	HF48
X	H23	HF23	x	H49	HF49
Y	H24	HF24	y	H50	HF50
Z	H25	HF25	z	H51	HF51

The following table shows the correspondence between parameters used by M paramacro and H parameters.

Mcode	**PARAMETERS**		**Mcode**	**PARAMETERS**	
parameter	**H**	**HF**	**parameter**	**H**	**HF**
Param1	H52	HF52	Param5	H56	HF56
Param2	H53	HF53	Param6	H57	HF57
Param3	H54	HF54	Param7	H58	HF58
Param4	H55	HF55	Param8	H59	HF59

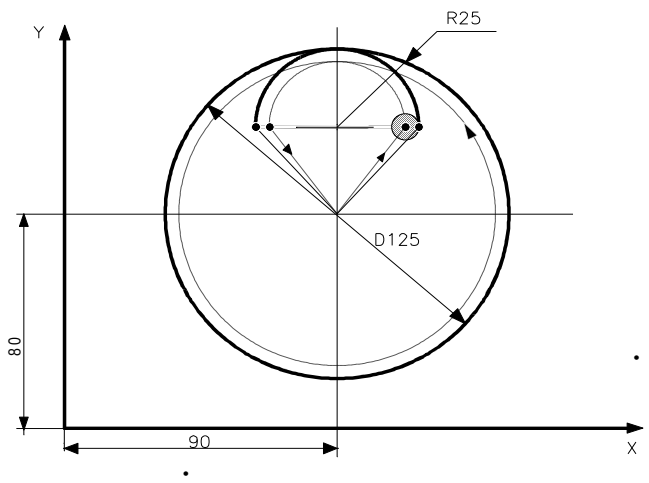
Example 1:
N45 G777 A(E8) R22.5 F(E2) S(E3+5-E1)

E8 is passed to H0
22.5 is passed to H17
E2 is passed to H5

The result of (E3+5-E1) is passed to H18

In this example Boolean parameters HF0, HF17 and HF5 are set to 1.

Example 2:



D=H3=Major diameter
R=H17=Minor radius
F=H5=Feedrate

Program:
This is an example of milling/boring cycle using paramacros.

```
;MAIN PART PROGRAM
.
.
.
N20 G601 D125 R25 F160
N21 . . .
```

```
; PARAMACRO G601
G0 X90 Y80
G92 XY
(GTO,END,HF3=0)
(GTO,END,HF17=0)
(GTO,END,HF5=0)
H57=H3/2
H58=H57-H17
G1 G41 XH17 YH58 F2000
G3 X0 YH57 I0 JH58 FH5
I0 J0
H59=NEG(H17)
G40 XH59 YH58 I0 JH58
G1 X0 Y0 F2000
(GTO,F)
"END"
G99
(DIS,"OMITTED PARAMETERS")
M ..
"F"
```

11.3 DAN – Define Axis Name

This command associates the name of a characterised axis with the name used in a paramacro.

Syntax

(DAN,*par-ax1 char-ax1*[**,***par-ax12 char-ax12*]**)**
(DAN)

where:

char-ax1 **...** *char-ax12*	Are the names of up to twelve characterised axes.
par-ax1 ... par-ax12	Are the names of twelve axes used in a paramacro.
no parameters	(DAN) without parameters disables (DAN) mode.

Example 1:

(DAN,PX,QY,DZ) X,Y,Z are replaced by P,Q,D so X,Y,Z are not usable.

Example 2:

(DAN,PX,QY,DZ) X,Y,Z are replaced by P,Q,D so X,Y,Z are not usable.

(DAN,WA) A is replaced by W, furthermore X,Y,Z are re-enabled and P,Q,D,A are now not usable.
By reprogramming (DAN,...) all previous associations are cancelled and the current ones become active.

After the three-letter mnemonic DAN has been used, if the axes concerned are those of the interpolation plane, this plane must be redefined using the new names.

11.4 SCRLPAR – Scrolling paramacro status

This variables defines the default behaviour for paramacro scrolling and execution.
It applies to all paramacros, regardless of their type (M,S,T and G).

Syntax

SCRLPAR = *value*

where:
value It is a Boolean value. 0 disables scrolling and 1 enables scrolling for all paramacros.

Characteristics:
If SCRLPAR is 0 and the selected mode is BLOCK-by-BLOCK, then the entire paramacro will execute with a single push of the CYCLE START button.
SCRLPAR value applies for the execution level where it is programmed and all lower levels (e.g. paramacros and subroutines called from that level), but does not apply to higher levels (e.g. calling program / paramacro).
If SCRLPAR is 0 for an higher execution level, then any modification to its value (e.g. setting it to 1) is ignored.
Default value for SCRLPAR is defined in AMP using ODM tool.

Example:

```
Main program:          G302   G303
N1 SCRLPAR=1           N10 GX10Y10   N21 G1X-10Y-10
N2 G302                N11 SCRLPAR=0 N22 SCRLPAR=1
N3 G303                N12 G303
```

The execution of the program will be:

```
N1  SCRLPAR=1      ; SCRLPAR set to 1, no effect
N2 G302            ; entering the paramacro. Since SCRLPAR is 1, then paramacro blocks will
                   ; appear on screen
        N10 GX10Y10   ;
        N11 SCRLPAR=0; SCRLPAR set to 0
        N12 G303      ; Entering the paramacro. Since SCRLPAR is 0 then paramacro blocks
                      ; will not appear on screen.
              N21 G1X-10Y-10       ; The screen shows block G303
              N22 SCRLPAR=1; Since SCRLPAR has been set to 1 then this assignment is unseen
              ; the screen still shows block G303

N3  G303      ; Entering the paramacro. At this level of execution SCRLPAR has not been modified;
              paramacro blocks will appear on screen
              N21 G1X-10Y-10       ;
              N22 SCRLPAR=1        ; SCRLPAR remains at 1
```

Triliteral DGM has higher precedence than SCRLPAR, so a paramacro containing this triliteral will always execute with scrolling disabled.

11.5 DGM – Disable paramacro scrolling

This triliteral disables scrolling for a paramacro. It disables block by block execution, too.

Syntax

(DGM)

Characteristics:
DGM triliteral disables visualization and scrolling of the blocks of a paramacro.
It can be programmed in any position in a paramacro, it affects the entire paramacro.
If current execution mode is block-to-block, then the paramacro, and nested paramacros and subroutines, will be executed entirely by a single push of the *cycle* key.

The three-literal is taken into account only by paramacros. It is ignored if it appears in the main part-program or any subroutine.

As DGM has the priority over the SCRLPAR variable, the paramacro containing this three-letter code will be always executed with scrolling disabled.

11.6 PMS – S as a paramacro, PMT – T as paramacro, PMM – M as a paramacro

These commands enable and disable the management of the related auxiliary functions S, T, M as paramacros.
For further information on how to program these paramacros see the paragraph on the "S, T, M auxiliary functions executed as paramacros" in this chapter.

Syntax

PMS=value
PMT=value
PMM=value

where:

value This value can be:
0. Disables the management of the S, T, M functions as paramacros – these functions are executed in standard mode

1. Enables the management of the S, T, M functions as paramacros – these functions associate the execution of the corresponding paramacro

NOTE:
The values ascribed to PMS, PMT and PMM variables can be changed as follows:

- Through ODM at the configuration stage
- Through a part program with the syntax described

RESET command restores the characterisation values for these variables.

11.6.1 *S, T, M auxiliary functions executed as paramacros*

The execution of the S T M auxiliary functions can be associated with paramacro type subprograms, i.e. in the presence of a block containing at least one S, T and/or M function, previously configured as a paramacro, the corresponding subprogram is executed, according to the criteria currently defined for paramacros.

Syntax:
same as the syntax used for auxiliary functions executed in standard mode.

Auxiliary code	**Paramacro**
S	In the presence of an "Sx" function, e.g. on process "*n*", paramacro "P*n*S" is executed with parameter H18 = x
T	In the presence of a "Tx.y" function, e.g., on process "*n*", paramacro "P*n*T" is executed with parameter H19 = x.y. The following additional variables are defined for the parameters, the tool code and the offset associated with the programming of T: › **HI** (H Integer), double type, and corresponding HFI flag: contains the whole number to the left of the decimal point (tool code). In the example HI = x

- **HD** (H Decimal), double type, and corresponding HFD flag: contains the whole number to the right of the decimal point (offset tool code).
  In the example HD = y
- **HS** (H String), string type, with the corresponding HFS flag: contains the ASCII ToolName, if programmed. HS is a 40-character string, but is reduced to 32 when used for T

M In the presence of an "Mx" function, e.g. on process "*n*", paramacro "P*n*Mx" is executed with parameter H12 = x. It is allowed to set up to 8 parameters using the following syntax:

- **M**x**[***param1***[,** param2 ... **,** *param8***]**

These eight parameters are stored in H variables starting from index 52.

Characteristics:

- The functions generating paramacro calls cannot not be programmed at the start of the block.
- Leading zeroes in process number and M code are not taken into account for the paramacro name. This means that M code 45 on process 1 will correspond to paramacro P1M45.
- If M, S and/or T are programmed simultaneously, the name of the procedure to be called is determined by the first function which is configured as a paramacro, and any other function programmed in the block is called with the respective parameters H, HF and HC, as described in this chapter. For example, if both M and T are configured as paramacros, programming TxMy, will generate a call to macro PnT whilst programming MyTx will call the "PnMy" macro, with parameters H19=x and H12=y in either case.
- The value for the M code can be specified using an E parameter, for instance ME3 will call the M function whose code is specified by the content of the parameter E3.
- For the execution of the paramacro in MDI mode, see paragraph "Executing blocks entered from the keyboard" in the user manual.
- At the end of the paramacro, the execution mode (MDI, AUTO, STEP) in which the system was working before the call is restored.
- The type of paramacro being executed can be read in the **SW21** variable and it is:
  - 1000 for a paramacro called by the S function
  - 2000 for a paramacro called by the T function
  - 3xxx for a paramacro called by the Mxxx function
- The *pathname* to be used to call the paramacros can also be specified with the PTH mnemonic described below in this chapter. If the pathname is omitted, the search for the directory containing the subroutine is performed as follows:

  1. case: no pathname has been specified through a PTH instruction.
  System searches for the paramacro in the directory containing the calling program and, if it does not find it, searches for it in the default directories specified in AMP.
  2. case: a pathname has been specified through a PTH instruction.
  System searches for the paramacro in the directory specified with a PTH and, if it does not find it, searches for it in the configured default directory.

Configuration: three process variables (PMM, PMS, PMT, previously defined), if set to 1 or 0, enable or disable the execution of the corresponding functions as paramacros.
Moreover, the execution of an auxiliary function M as a paramacro is identified by a bit, defined in AMP.

Hence, if PMM = 1 and the "Mx" function is:

- configured in the "n" process but its bit = 0, the paramacro is not executed (standard Mx is executed instead)
- configured in the "n" process and its bit = 1, the "PnMx" paramacro is executed
- not configured in the "n" process, an "M not defined" error message is displayed

Example:
Execution of the block described below in process 1, in which the auxiliary function M2 is configured with the paramacro bit enabled

N20 X10 M2 T1.0 S2000

PMM	PMT	PMS	Paramacro name	Parameters
1	1 or 0	1 or 0	P1M 2	H23 = 10; H12 = 2; H19 = 1.2; HI = 1; HD = 20 H18 = 2000 e HF23 = 1; HF12 = 1; HF19 = 1; HFI = 1; HFD = 1; HF18 = 1
0	1	1 or 0	P1T	H23 = 10; H12 = 2; H19 = 1.2; HI = 1; HD = 20 H18 = 3000 e HF23 = 1; HF12 = 1; HF19 = 1; HFI = 1; HFD = 1; HF18 = 1
0	0	1	P1S	H23 = 10; H12 = 2; H19 = 1.2; HI = 1; HD = 20 H18 = 3000 e HF23 = 1; HF12 = 1; HF19 = 1; HFI = 1; HFD = 1; HF18 = 1

Example:
Execution of the following block in process 1 where variable PMT was set to 1.

T"ToolName".20 calls paramacro P1T with parameters: H19=0.2, HF19=1; HFI = 0; HD = 20 and HFD = 1; HS =" ToolName" and HFS = 1

This is an example of how paramacros are developed:

;PARAMACRO P1M2
E0 = H12
PMM = 0
ME0
"TEST_T"
(GTO,TEST_S,HF19=0)
E1 = H19
PMT = 0
TE1
"TEST_S"
(GTO,END,HF18=0)
E2 = H18
PMS = 0
SE2
"END"

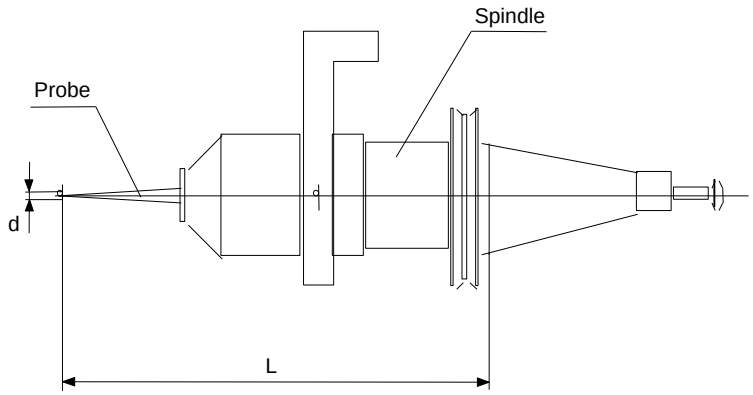
;PARAMACRO P1T
E1 = H19
PMT = 0
TE1
"TEST_M"
(GTO,TEST_S,HF12=0)
E0 = H12
PMM = 0
ME0
"TEST_S"
(GTO,END,HF18=0)
E2 = H18
PMS = 0
SE2
"END"

;PARAMACRO P1S
E2 = H18
PMS = 0
SE2
"TEST_M"
(GTO,TEST_T,HF12=0)
E0 = H12
PMM = 0
ME0
"TEST_T"
(GTO,END,HF19=0)
E1 = H19
PMT = 0
TE1
"END"

12. PROBING CYCLES

12.1 MANAGING AN ELECTRONIC PROBE

An electronic probe is a measuring device that can be mounted on the spindle and controlled like a tool. The probe can also have length and diameter offsets associated with it. It can also be mounted in a fixed position and used as an electronic gauge to requalify the tool length.
The figure that follows shows an electronic probe with its relevant dimensions.



Electronic Probe
Some G codes perform specific measuring or probing cycles when an electronic probe is mounted on the spindle.

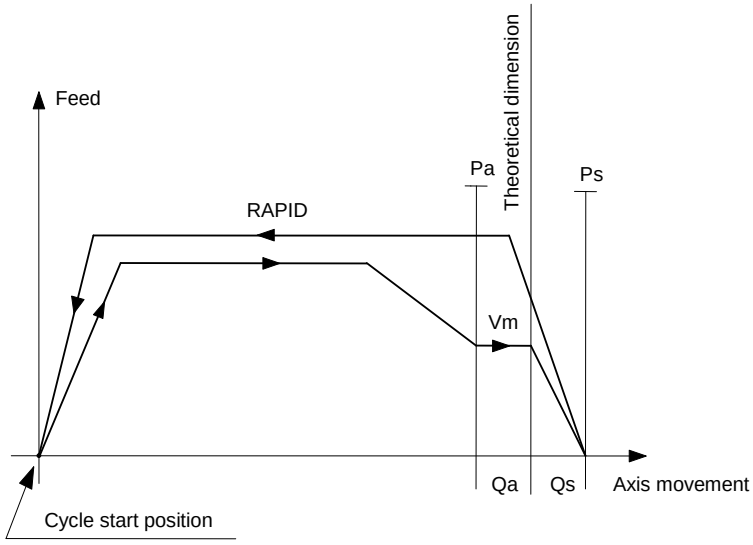
G CODE	FUNCTION
G72	Measures the coordinates of a point
G73	Measures the coordinates of the circle centre and radius
G74	Measures variances from theoretical points (for probes mounted in a fixed position rather than on the spindle).

The control stores the measured values in E parameters defined in the probing cycles (G72-G74).

When the control executes a probing cycle, it measures a point through the following sequence of moves:

1. Rapid to the approach point (Pa).
2. Move at measuring speed (Vm) to the point where the probe triggers, then stop and store the dimensions. If the probe does not trigger, move only as far as the end safety point (Ps).
3. Rapid return to the start position of the probing cycle (hole centre in G73). Execution of that phase depends on machine parameters characterised in AMP or the corresponding DPP three-letter block mode parameter (DPP).

The diagram below shows the moves and feed of a probing cycle.



The following errors may occur during a probing cycle:

- If the probe does not trigger before reaching the safety point, it returns to the start point of the cycle. The control panel displays a message and the system goes into an error condition.
- If the probe does not reset properly on the way back to the start point after a successful probing cycle, a message is displayed and the system goes on error condition.
- If the probe is carried out during the rapid approach phase, it returns to the start point of the cycle. The control panel displays a message and the system goes into an error condition.

NOTE:
If the probing cycle is executed with G27 or G28 active, both the approaching movement and probing are carried out in continuous mode, while the stopping movement after the probe input has been found is executed in point to point mode.

Values obtained from probing cycles are always expressed in the current measurement unit.

12.2 PRESETTING A PROBING CYCLE

When a probe is used for the first time, or whenever probing cycle conditions change, it is compulsory to:

1. define the probing parameters
2. dynamically measure the diameter of the probe ball
3. requalify the probe with respect to the spindle axis
4. dynamically measure the length of the probe.

12.2.1 *DPP – Defining Probe Parameters*

Use the DPP command to define the probing parameters from the keyboard or in a program. These parameters can also be defined on the control with a specific data entry via user interface. When these are not defined the system uses the values defined in AMP.

Syntax

(DPP,*approach*,*safety*,*speed* [, *mode*] **)**

where:

approach Is the approach tolerance expressed in mm or inches.

safety Is the safety tolerance expressed in mm or inches.

speed Is the measuring speed expressed in mm/min or inches/min.

mode Is the value which indicates whether to perform the rapid return phase at the end of the probe. It can be one of the following values :

0 = perform the return phase

1 = do not perform the return phase

If no mode parameter is missing then the default 0 is assumed.

Example:
Instruction used from the program
(DPP,10,12,1000)

"Approach", "speed" and "safety" values are defined in the unit of measurement that is configured in AMP.

12.2.2 *Dynamic Measurement of the Ball Diameter*

To dynamically measure the apparent diameter of the probe ball, use a requalification ring, cylinder, or similar device with a centre assumed to be the absolute origin 99 of axes X and Y.

12.2.3 *Probe Requalification*

Typically, before using the probe, the centre of the probe ball is not centred on the spindle axis. To centre the probe ball on the spindle axis, it's compulsory to requalify the probe.
Use the hole of the requalifying ring, cylinder, or similar device for this operation. The centre of the hole is assumed to be origin 99 for axes X and Y.

12.2.4 *Dynamic Measurement of the Probe Length*

To measure the length of the probe dynamically, use the requalification ring, cylinder, or similar device whose reference surface (top) is assumed to be origin 99 for the length axis Z.

12.2.5 *Probe Presetting*

The following steps must be executed to pre-set the probe:

1. Use a requalification ring for reference.
2. Position absolute origin 99 for axes X and Y at the centre of the ring hole.
3. Position absolute origin 99 for the Z axis on the top surface of the ring.

The initial values stored in the probe offset are:
length1 = nominal length of the probe with respect to the axis of the ball.
Diameter = 0

Example:

```
N1  (DIS, "DPP, UPA, UPO, ")
N2 T30.30 M6                        - probe on spindle
N3  (UAO, 99)
N4  (DPP, 10, 12, 600)              - defines probing parameters
N5  UPA=0
N6  UPO=0
N7  E30=...                         - assigns diameter to the hole of the sample ring
N8  E31=E30/2
N9  E32=...                         - assigns distance from Z=0 to probing surface on the  Z axis
                                      (usually = 0)
N10  E33=E31+10
N11  GB0                            - only if the ring is mounted on the indexing table
N12  XY
N13  Z-4
N14  G73 rE31 E40                   - measures hole co-ordinates (centre and radius)
N15  Z100
N16  (DIS, "UPA=", E40, "UPO=", E41)
N17  M0
N18  UPA=E40                        - requalifies probe abscissa
```

```
N19  UPO=E41                - requalifies probe ordinate
N20  E34=(E30-E42*2)        - diameter of apparent ball
N21  (DIS, "DIAMETER=", E34)
N22  M0
N23  (RQP, 30, 30, dE34)    - stored ball diameter on d offset
N24  T30.30 M6              - enables new offset
N25  GXYE33
N26  G72 ZE32 E43           - measures Z dimension on ring surface
N27  E35=E43-E32            - variance between nominal and real value
N28  Z100
N29  (DIS "VARIANCE.Z=", E35)
N30  M0
N31  (RQP, 30, 30, LE35)    - requalifies Z length offset
N32  M30
```

12.3 PROBING CYCLES

These are G codes that define probing cycles:

G72 Point measurement (with probe ball diameter compensation)

G73 Measurement of hole parameters

G74 Point measurement (without probe ball diameter compensation)

In the cycles G72, G73 and G74, E parameters are associated with the axes in the order in which the latter are characterised (AMP) and displayed, and not in the order in which they are programmed.

12.3.1 *G72 - Point Measurement with Compensation*

G72 is used to measure the co-ordinates of a point using linear moves and a probe.

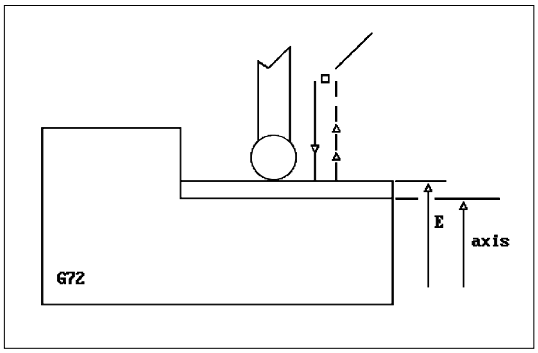
Syntax

G72 *axis1* [*axis2*] [*axis3*] *E-par*

---

where:

axis1 . . . axis3 Are the axes that will move during the probing cycle. The dimensions programmed in the G72 block are the theoretical positions where the probe is expected to impact the target point. You can program up to three axes and dimensions in the G2 block.



E-par Defines the E(x) parameter for storing the measured dimension of the first axis programmed in G72; the dimensions of other axes, are stored sequentially in order of configuration in E(x+1) , E(x+2) and E (x+3) in virtual axes probing.

Characteristics:
The co-ordinates measured by the probe are stored in E parameters that are defined by the G72 cycle. The control stores the co-ordinates of the axes beginning with the E parameter specified in the cycle. Measurements are taken after applying cutter diameter compensation to the probe (tool length plus sphere probe radius, as per 10 Series). To ensure best accuracy the surface must be perpendicular to the measuring move.

Example:
The axes are configured in AMP in the following order: XYZ
G72 X100 Y50 E32
G72 Y50 X100 E32
In both cases the values that the control calculates for X and Y are stored sequentially in E32 and E33.

12.3.2 *G73 – Hole probing cycle*

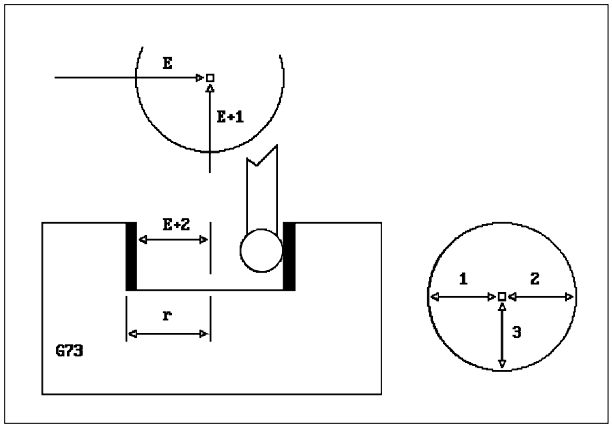
G73 uses a probe to measure the dimensions of a hole on the active interpolation plane storing these records in E parameters.

Syntax

G73 r*value*.. *E-par1*

where:

r*value* Defines the theoretical hole radius. r is followed by the radius length. It can be programmed directly or with an E parameter.



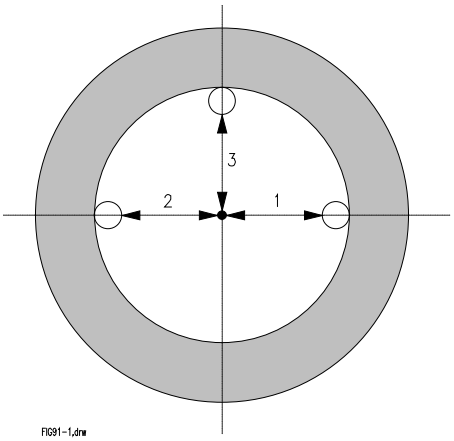
E-par Is the first E parameter in which the system starts storing values:

- first E parameter: centre abscissa (typically X)
- second E parameter: centre ordinate (typically Y)
- third E parameter: radius

Characteristics:

If only one E parameter is programmed, the three values detected during the probing operation (circle centre coordinates and radius) are stored in E parameters sequentially starting from the specified E parameter.

Before activating the G73 cycle, the axes of the machine tool must be positioned on the hole centre. Measurements are taken after applying cutter diameter compensation to the probe.



Probing moves

Example:

G73 r100 E55 probing moves

The coordinates of the circle centre (abscissa and ordinate) and the actual radius are stored in E55, E56 and E57 respectively.

12.3.3 *G74 – Tool requalification cycle*

G74 is a tool requalification cycle. G74 uses a fixed probe (such as an electronic gauge) to measure variances from theoretical size of the tool mounted in the spindle.

Syntax

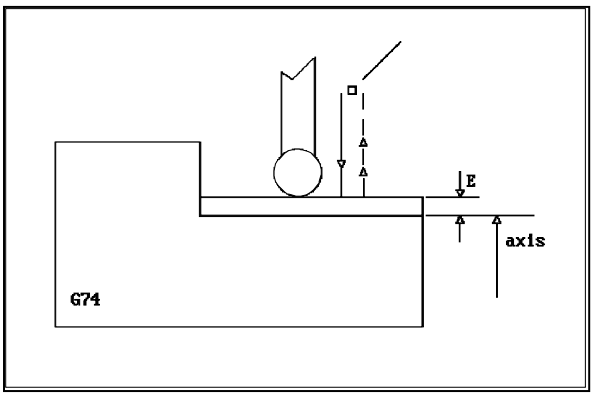
G74 *axis1* [*axis2*] [*axis3*] *E-par*

---

where:

axis1. . . axis3 Are up to three simultaneous axes.

E-par Defines the E(x) parameter for storing the deviation from the point measured and the point programmed for the first axis programmed in G74. The deviation values of the other axes, if programmed, are stored sequentially in E(x+1) and E(x+2).



Characteristics:

G74 can be used for requalifying a tool or checking tool wear. The calculation of measured dimensions does not take into account the tool offset, since the cycle is checking the actual "tool" dimension.

The steps in the G74 cycle are similar to those in the G72 cycle. The difference between both cycles is how the control executes calculations based on measured dimensions. The control does not consider the diameter of the probe ball and stores the variance from the theoretical dimensions in the parameters specified in the G74 block.

Example:

G74 X60 E41

where: E41 is given by the formula: E41 = Pm - Pt

in which: Pm is the measured point

Pt is the theoretical point

12.4 UPA – Update probe abscissa

Defines the probe requalification value for the abscissa (typically X).

Syntax

UPA = *value*

where:

value is the abscissa requalification value expressed in millimetres or inches. UPA value is not volatile, so it is not reset after shutdown.

12.5 UPO – Update probe ordinate

Defines the probe requalification value for the ordinate (typically Y).

Syntax

UPO = *value*

dove:

where:

value is the ordinate requalification value expressed in millimetres or inches. UPO value is not volatile, so it is not reset after shutdown.

12.6 ERR – Managing probing errors

G72 – G73 – G74 cycles enable the handling of probing errors either automatically or from part program. The choice between these two modes is made setting ERR parameter.
For further information on probing error management see Appendix C.

12.7 OPERATIONS WITH A NON-FIXED PROBE

With probing cycles G72 and G73 is possible to:

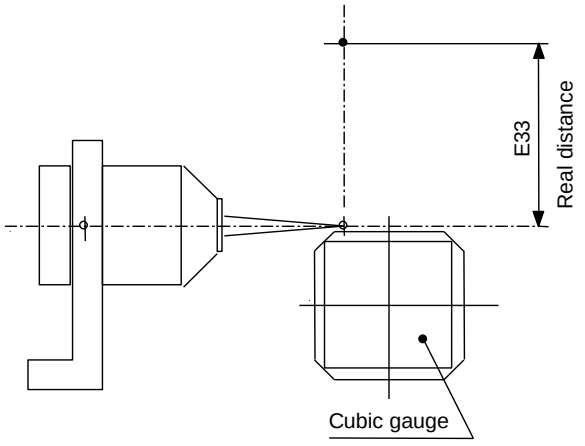
Requalify origins by:
- probing reference surfaces
- centring on a hole

Check the dimensions of:
- diameters
- planes and depth of holes.

12.7.1 *Requalifying Origins by Probing Reference Surfaces*

There are two conditions that may require the requalification of origins by probing a reference surface. Origins may change due to:
- Changes in temperature (thermal drift)
- New pallet used for machining

Thermal drift
This procedure uses a requalifying cube that is placed at a precise location on the machine.



Main Program:

.
.
.

N99 E33=-300

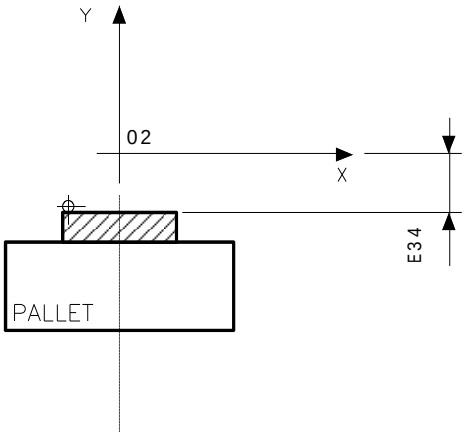
.
.
.

/N100 (CLS,TEST3)

Subprogram TEST3:

```
N500 (DIS,"RQO-DT")
N501 G72 Y(E33) E32          ;Measured distance stored in E32
N502 E32=E32-E33
N503 (RQO,1,Y(E32))          ;Requalify origin 1 for Y axis
N504 (RQO,2,Y(E32))          ;Requalify origin 2 for Y axis
N505 (RQO,3,Y(E32))          ;Requalify origin 3 for Y axis
```

New pallet for machining



Main Program:

```
M . . . ;                    ;Pallet Change
N199 T30.30M6;               ;Probe on spindle
N200 (UAO,2)
N201 GXY
N202 E10=2
N203 E34=-250
/N204 (CLS,TEST4)
```

TEST4 subroutine:

```
N500 G72 Y(E34) E30
N501 E31=E30-E34
N502 (RQO,E10,Y(E31))        ;Requalify origin 2 for Y axis
```

12.7.2 *Requalifying Origins by Centering on a Hole*

Before performing this requalification cycle, X and Y axes have to be positioned on the hole axis and the probe has to be positioned into the hole.

Example:

```
N200 (DIS,"REQUALIFYING ORIGIN FOR X AND Y AXES")
N201 T11.11 M6
N202 GX180 Y60
N203 Z-130
N204 G73 r50 E35                ;Measuring cycle diameter 100. X and Y coordinates are
                                ;stored in parameters E35 and E36
N205 E35=E35-180                N206 E36=E36-60
N206  E36=E36-60
N207 (RQO,1,X(E35),Y(E36))      ;Requalifies origin 1 for X (E35) and Y (E36)
```

12.7.3 *Checking hole diameters*

The diameter of holes can be checked with a probe. With proper programming it is possible to compare detected values with allowed values, and to decide whether to continue machining or to branch to a block containing a label.

In the following program, the control measures the deviation between the actual and the theoretical diameter of a hole and compares this deviation with the tolerance. The comparison may have three different results and courses of actions:

- Hole diameter is within the allowed tolerance - the program continues.
- Hole diameter is greater than the allowed tolerances - tool is automatically timed out of use (label A3). The program stops (M00) and the part is rejected.
- Hole diameter is less than the allowed tolerance - tool is automatically timed out of use (label A4). The program branches to label A1, where the hole reaming cycle is repeated using the alternative tool that is specified in the tool life file.

Example:
The following is an example of diameter checking.
Hole diameter and tolerance: D=100 +0.02/-0.015
Program:

```
"A1" N111
N112 GZ-150
N113 (DIS,"REAMING HOLE D=100")
N114 F..S..T13.13 M6
N115 GX-120 Y80 M13
       .
       .
       .
N129 (DIS,"HOLE TOLERANCE CHECK D=100")
N130 T14.14 M6                          ;probe in spindle
N131 GX-120 Y80                         ;XY positioning to hole centre
N132 Z-85                               ;Z positioning
N133 G73 r50 E30                        ;hole radius stored in E32
N134 E32=E32*2
N135 (DIS,E32)
N136 (GTO,A3,E32>100.02)
N137 (GTO,A4,E32<99.985)
N138 GZ150
N139 (DIS,"WORKPIECE IN TOLERANCE")
       .
       .
       .
N2100 M30
"A3" N2101 (DIS, "Hole too big")
N2102 M00
"A4" N2103 (DIS, "Hole too small")
/N2104 (GTO,A1)
```

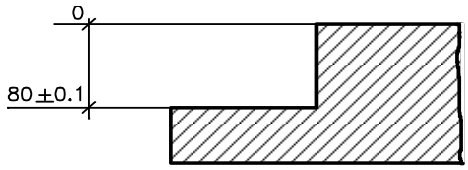
12.7.4 *Checking surface positions and hole depths*

A probe can be used to check the positions of surfaces and the depths of holes. With proper programming it is possible to compare detected values with allowed values, and to decide whether to continue machining or branch to a block containing a label.

In the following programming example, the control detects the deviation of actual dimensions from programmed dimensions. It compares this deviation with allowed tolerances. The comparison has three possible results:

- Continue machining
- Stop machining and reject the part because the dimension is too long
- Requalify the tool and repeat the program from label C1 because the dimension is too short

Example:
The following example check of the dimensions of a plane
Theoretical dimension, with respect to Z axis zero: - 80 ± 0.1



Program:

```
      .
      .
      .
"C1" N252
N253 (DIS," . . .")
N254 F. . S. . T23.23 M6
      .
      .
      .
N267 (DIS,"CHECK PLANE DIMENSION 80")
N268  T30.30 M6                    ;Probe in spindle
N269 GX150 Y35                     ;XY positioning on the point to be checked
N270 G72 Z-80 E30                  ;Distance measured and stored in E30
N271 (GTO,C2,E30 <-80.1)           ;Branch to C2 if E30 < min coordinate
N272 (GTO,C3,E30>-79.9)            ;Branch to C3 if E30 > max coordinate
N273 (DIS,"WORKPIECE IN TOLERANCE")
      .
      .
      .

N2100 M30
"C2" N2101 (DIS,E30)               ;Dimension too long
N2102 M00
"C3" N2103 (DIS,E30)               ;Dimension too short
N2104 E31=-80-E30                  ;Variance between virtual and actual dimension
N2105 (RQT,23,23,LE31)             ;Requalification of length offset 23 (Z)
N2106 (GTO,C1)
```

12.8 Operations using a fixed probe

Using the G74 probing cycle with a probe that is fixed in position (like an electronic gauge) and a tool mounted on the spindle, is possible to:

- requalify tools (automatic tool offset modifications)
- check tool wear.

Example 1:
This is an example of tool length offset requalification:

```
N170 G X100 Y100          ;Positioning on the measuring point (probe position)
N180 G74 Z-50 E30         ;Deviation measured and value stored in E30
N190 (RQT,10,1,LE30)      ;Requalification of offset 1 for tool 10 in length (Z) by E30
```

Example 2:
This is an example of tool length offset and tool diameter offset modification.

```
N200 G X100 Y100          ;Positioning on the measuring point (probe position)
N210 G74 Z-50 E30         ;Measured deviation (Z axis and value stored in E30)
N220 G X150
N230 Z-60
N240 G74 X130 E31         ;Measured deviation (X axis) and value stored in E31
N250 E31=E31*2
N260 (RQT,10,1,LE30,dE31) ;Requalify offset 1 of tool 10 in length (Z) by E30 and in
                                  diameter (d) by E31
```

Example 3:
This is an example of tool wear inspection with probe error management (Error management methods are described in Appendix C):

```
N480 (DIS,"TOOL D=10")
N490 T10.10M6
. . . . .
. . . . .
N600 (DPP,10,5,500)
N610 (UAO,9)
N620 GXY
N630 ERR=1                ;Error managed by program
N640 G74 Z0 E35           ;Tool length measurement
N650 (GTO,A2,STE=1)       ;Branch to A2 if tool is broken (no point probed within the
                          ;5 mm safety distance)
N660 ERR=0                ;Error managed by system
. . . . .
N1500 M30
"A2" N2001 (TOU,10)       ;Declares tool 10 out of use
N2002 (DIS, "TOOL K.O.")
N2003 M0
```

13. MANAGING INFORMATION EXCHANGE

13.1 General

This chapter describes the commands used to send/receive information using a **part program**.
These commands allow control of:

- Displays
- ASCII and binary files
- Serial ports

13.1.1 *Commands to manage the displays*

Command in this class is

COMMAND	FUNCTION
DIS	Displays numbers, strings or the content of variables

13.1.2 *File access commands*

Commands in this class are

COMMAND	FUNCTION
OPN	Opens (read or write) a communication channel associated with a file
WRT	Writes to a file
REA	Reads from a file
CLO	Closes one or more open files
DEL	Deletes a file
CPY	Copies a file
REN	Renames a file

13.1.3 *Command to access serial ports*

COMMAND	FUNCTION
SOP	Opens the connection to a serial line
SCL	Closes one or all the connections opened to serial lines
SRS	Resets the specified serial line
GET	Enables reception of data from the serial port
PUT	Enables transmission of data to the serial port

13.2 Commands to manage the display

13.2.1 *DIS – Displaying values on screen*

The DIS command can display variable values to the operator. The control will show the value in the screen area that is reserved for communications with the operator.

Syntax

(DIS, *operand* [**,***operand*] [**,***operand*] [**,***operand*] [**,***operand*]**)**

---

where:

operand It can be a number, a variable or an ASCII string. Up to five operands can be displayed. All 5 operands in total cannot be more than 100 characters long.

If *operand* is a number it has to belong within the normal range for variables (5.5 format).

If *operand* is a variable it can be any variable used in assignment blocks.

If *operand* is an ASCII string, it can be a message for the operator. The message can be up to 100 ASCII characters long. In this case the operand should be surrounded by quotes (" ").

Examples:

(DIS,100)	displays the value 100
(DIS,E27)	displays E27 current value on the screen
(DIS,MSA)	displays current MSA value (machining allowance)
(DIS,"THIS IS AN EXAMPLE")	displays the following string: THIS IS AN EXAMPLE

13.3 File access commands

13.3.1 *OPN – Opening a communication channel*

This command opens a file, associating it to a channel number and specifying the type of operation that is to be executed on the file.
The channel is an integer between 1 and 8 associated with the file specified in the command line, that simplifies file management for other instructions.

Syntax

(OPN, *channel, filename,* [*fileType*,] *mode***)**

---

where:

channel integer between 1 and 8 which is associated with the file that is opened. Can be expressed directly as an integer, or indirectly using a local variable or system

filename name of the file that will be associated with the specified channel. This parameter can be an alphanumeric string (possibly enclosed between two quotes) containing name and path of the file to be opened. The name of the file can also be indirectly specified using a string variable type. In this case double quotes cannot be used and the name of the variable must be preceded by **?**. When opening a file for writing, it is necessary to specify the filename and also a logical path to access the file.

fileType type of the file to be opened. Is a parameter of type character and is:

- **A** for text files (ASCII)
- **B** for binary files

The file type choice affects of the format of read/write operations for the file. If this parameter is not specified, the file is assumed to be an ASCII type.

mode indicates the file opening mode. Three modes are available, each indicated by a character::

- **R** Opening the file for reading, the file must exist
- **W** Opening the file for writing, if the file does not exist it is created. if it does exist. The new data is appended
- **A** Opening the file for writing, adding new data in queue

Characteristics:

The maximum number of files (and therefore channels) that a process can open at the same time is 8. If the file type is undefined, the control considers it as an ASCII type. If the command is for reading the file reading, it must exist. If the file is opened for writing (or append) it does not have to exist: in this case i t will be created by the control.
Some errors generated by the OPN command can be automatically managed or managed within the part program. The choice is made by setting the ERR variable.
For more information about part program management of errors, see Appendix C.

Example1:
(OPN, 1, PROGRAM\TEST, A, W)
Opens file located in the directory PROGRAM the file TEST for writing in ASCII and associates it with the channel 1

Example 2:
E1 = 3
SC0.8 = “DATA”
(OPN, E1, ?SC0.8, B, R)
Opens the file DATA located in a directory visible to the part program for reading in binary mode and associates it with channel 3

Example3:
(OPN, 8, “TEST”, R)
Opens the ASCII file TEST located in a directory visible to the part program for reading and associates it with channel 8

13.3.2 *WRT – Writing a file*

This command a record to be written to the file corresponding to the channel specified. A record can be made by one or more data or by variables specified in the command. If several values are specified, they have to be written in the record (and in the file) maintaining the same order that they had when specified in the command.

Syntax

(WRT, *channel*,[**R** *value*,] *value*[**,** ..., *value*]**)**

---

where:

channel integer between 1 and 8 that identifies the file that is to be read. Can be expressed directly or indirectly using a local variable or system.

R *value* record number where the data is to be written. This parameter is valid and accepted only for a binary file.

value list of variables or constants that are to be written to the file. Up to 32 values can be written. If the file is ASCII, the command accepts only one variable (character-string type).There are no limitations for constant strings.
For the binary files, all records must maintain the same structure as the first record that is written . You can specify a range of variables that need to be written (e.g. E0 | E9 shows the first ten E variables).
This syntax can be used with E, H or L variable types and is:
[**E**|**H**|**L**]*indi*|[**E**|**H**|**L**]*indl*
Indi and *indl* are the first and the last interval index, both increasing and decreasing intervals are accepted. The first and the last variables must be of the same time (E, H or L).

Characteristics:

If the file is binary and the record number was not specified in the WRT command, the writing will be made to the end of the file, that is after the last record in the file; or if the file is empty, the first record of the file.

The first step in writing a new binary file identifies the type of records in the file. All subsequent operations must be carried out with the same number, type and sequence of variables as the first record.

If, for instance, the first record was writtenwith the command below:

(WRT, 1, E1, SC3.4, @PLUS_SHORT)

that is with a double type variable, four characters and one short type variable, subsequent commands to access that file must comply with these types,dimensions and order.

Therefore the following command are valid:

(WRT, 1, L1, “ABCD”, @SHORT_PLC)
(REA, 1, !USE, !PROVA.4CH, @SHORT_PLC)

Commands that contain other types or number of variables or different sequences of the same, will cause a runtime error.

If the file is ASCII, the new data will always be written after the last record of the file. To change the record of an ASCII file, it is necessary to open the file for reading and, reading sequentially, write the records read, amended if necessary, to a new file different from the original.

To read a file that has been opened for writing it is necessary to close the corresponding channel using the CLO command and then reopen it for reading.

Some errors generated by the WRT command can be managed automatically or by the part program. The choice between these two modes is performed by setting the variable ERR.
For more information on the management of part program errors, see Appendix C.

Example 1:
(WRT, 1, *SC5*.80)
Writes 80 characters in the file associated to the channel, from the SC5 variable.

Example 2:
(WRT, 3, R12, E7, E8, E9)

Writes in record 12 of the binary file associated with channel 3, taking data from E7, E8 and E8 variables. The file pointer is positioned at the beginning of record 13.

13.3.3 *REA – File reading*

This command is used to read a record from a file corresponding to the specified channel. A record can be created from a single or multiple data items or from variables.

Syntax

(REA, *channel*, [**FRB** | **R** | **FRC** *value*,] *variable* [**, ...,** *variable*]**)**

---

where:

channel integer between 1 and 8 that identifies the file that is to be read. Can be expressed directly or indirectly using a local variable or system.

FRB *value*
FRC *value*
R *value* Number of records to be read. This parameter only accepted for binary files. If FRB or R is used *value* indicates the position of the record from the beginning of file, if FRC is used *value*, it indicates the offset from the current position

variable list of variables or string constants that must be read from the file. The maximum number of values that can be read is 32. For binary files, the format of the variable parameters must be consistent both in number and type to match the type of record used by the file. A range of variables to be read may be specified (e.g. E0 | E9 shows the first ten E variables).
This syntax can be used with E, H or L variable types and is:

[E|H|L]*indf***|[E|H|L]***indl*

Indf and *indl* are the first and the last indices, both increasing and decreasing indices are accepted. The first and the last variables must be of the same type (E, H or L).

Characteristics:
If the file is ASCII or binary type, and the command does not contain the number of the record to be read, then it will read the record following the last that was read. or for the first read operation it will read the first record in the file.
Some errors generated by the REA command can be handled automatically or by the part program. Set the variable ERR to select the desired mode. For more information on the management of part program errors, see Appendix C.

ASCII type files
The conversion of a numeric parameter to the format of the input variable is done automatically. The command considers a space, a sign or an alphabetic character as valid terminators for numeric fields; if, during the reading of a numeric variable, at least one numeric character is not read, the system reports a format error. When reading a string field that does not contain a real terminator, the command will try to read a number of characters (bytes) equal to what was specified by the variable present in the command.

If reading is specified for a number of variables different to number of the fields in a file record, no error is signalled, but variables are loaded in sequence until the record is complete.
Example 1:
(REA, 1, SC5.20)

From the file associated with channel 1, it reads 20 characters placing them in the first twenty characters of the variable SC5.

Example 2:

(REA, 3, R42, E1, SN1, H1)
Reads record 42 of the binary file associated with channel 3, placing its data in the variables E1, SN1, H1. The file pointer is positioned at the beginning of record 13.

Example 3:
Reading record of the ASCII file associated to channel 1 with the following format:

-5.00 +3400 PLUTO

Reading using different command types will lead to different readings:

(REA, 1, E1, E2, E3) → Format error as a numerical field cannot be assigned to the E3 variable
(REA, 1, SC0.8, E1, SC10.80) → SC0 = "-5.00 +3" E1 = 400 SC10 = "PLUTO"
(REA,1,E1,E2,SC10) → E1=-5.00, E2=+3400,SC10 = "PLUTO"
(REA, 1, E1) → E1 = -5.00

After each reading, the pointer is moved to the record following the one just read, so for the last example the fields following -5.00 in the record are lost.

13.3.4 *CLO – Close Channel*

This command closes the file associated with the channel specified.
If no channel is specified, all open channels belonging to the process will be closed.

Syntax

(CLO[, *channel*])

---

where:

channel integer between 1 and 8 that is associated to the opening file. It can be expressed directly or indirectly using a variable. If this parameter is not specified, all channels associated with the current process will be closed.

Characteristics:
If the channel number is omitted, this command closes all files opened by the process.
System RESET closes all open files, so it is the equivalent of executing CLO without parameters.

Some errors generated by the *CLO* command can be handled automatically or by the part program. Set the variable ERR to select the desired mode. For more information on the management of part program errors, see Appendix C.

Examples:

(CLO, 2) → Closes the file associated with channel 2

(CLO) → Closes all files opened by the process

13.3.5 *DEL – Deleting a file*

This command deletes the specified file.

Syntax

(DEL, *filename***)**

where:

filename This parameter can be an alphanumeric string (enclosed in double quotes if necessary) that contains the name and path of the file to be deleted. You can also specify the name of the file indirectly with a string variable. In this case double quotes cannot be used and the variable name must be preceded by the symbol **?**.

Characteristics:
You can delete a file only if the system is not using it and in the case of part program files, not selected by any process. If the file to be deleted does not exist, the command generates a runtime error.

Some errors generated by the *DEL* command can be handled automatically or by the part program. Set the variable ERR to select the desired mode. For more information on the management of part program errors, see Appendix C.

13.3.6 *CPY – Copying a file*

This command copies a file.

Syntax

(CPY, *filename1***,** *filename2**)***

where:

filename1 This parameter can be an alphanumeric string (enclosed in double quotes if necessary) that contain the name and location of the original file. The name of the file may be specified indirectly with a string variable. In this case it is double quotes are not allowed and the variable name must be preceded by the symbol **?** .

filename2 This parameter can be an alphanumeric string (enclosed in double quotes if necessary) that contain the name and location of the destination file. The name of the file may be specified indirectly with a string variable. In this case double quotes are not allowed and the variable name must be preceded by the symbol **?** .

Characteristics:
Some errors generated by the *CPY* command can be handled automatically or by the part program. Set the variable ERR to select the desired mode. For more information on the management of part program errors, see Appendix C.

13.3.7 *REN – File rename*

This command renames a file.

Syntax

(**REN**, *filename1*, *filename2*)

where:

filename1 This parameter can be an alphanumeric string (enclosed in double quotes if necessary) that contains the name and path of the original file. The name of the file may also be specified indirectly with a string variable. In this case double quotes are not allowed and the variable name must be preceded by the symbol **?** .

filename2 This parameter can be an alphanumeric string (enclosed in double quotes if necessary) that contains the name and path of the original file. The name of the file may also be specified indirectly with a string variable. In this case double quotes are not allowed and the variable name must be preceded by the symbol **?** .

Characteristics:
This command is equivalent to copying the first file to the second file and then deleting the first file.
Some errors generated by the *REN* command can be handled automatically or by the part program. Set the variable ERR to select the desired mode. For more information on the management of part program errors, see Appendix C.

13.4 Commands for serial port management

The serial line management allows the following commands:

- Enable and configure serial lines defining the communication parameters and all terminators necessary to manage data and messages.
- Transmit any kind of data such as numerical, data in numerical variables, alphanumeric data (strings), alphanumeric data contained in string variables.
- The transmitted data constitutes the **message**, which can have a maximum length of 128 characters including data and message terminators.
- Receive data of any kind such as numerical data, data in numeric variables, alphanumeric data (strings), alphanumeric data contained in string variables.
- The length of the received message should not be greater than 180 characters including data and message terminators.
- Close the serial lines when communication finishes.
- Reset a serial line.

13.4.1 *Enabling and configuring the serial port*

This command reserve the specified serial port and configures the physical characteristics of the communication.

Syntax

(SOP, *Portnum*, *Baud*, *size*, *bitStop*, *parity*, *protocol*, *flowtype*, *TXtimeout*, *TXretry*, *RXtimeout*, *RXretry* [*/ numSepDataTx*, *terminator*, *numSepDataRx*, *terminator*, *numTermMessTx*, *terminator*, *numTermMessRx*, *terminator*]**)**

where:

Portnum serial port number that you want to open. The OPEN*control* can manage up to 4 serial ports

Baud Transmission speed in bauds. You can use the following values:

50	75	110	134	150	300
600	1200	1800	2000	2400	3600
4800	7200	9600	14400	19200	38400
56000	57600	115200	128000	256000	

Size words length in bits, values accepted are:
5, 6, 7, 8

bitStop number of stop bits, values accepted are 1 (1 stop bit), 15 (1 bit and a half) and 2 (2 bits)

protocol Protocol used for the transmission, two values are available:

- **0 no protocol**
- **1 logic protocol (X_{on}, X_{off})**

Flow type Bit mask to identify the communication flow type

FlowCtrl	Description
0x0001	DTR enabled
0x0002	DTR HandShake
0x0004	RTS enabled
0x0008	RTS HandShake
0x0010	CTS management enabled
0x0020	DSR management enabled
0x0040	Ignores receipt when DSR off
0x0080	X_{on} X_{off} continuous
0x0100	X_{off} management
0x0200	X_{on} management
0x0400	Management terminator receiving
0x8000	Clear buffers after receiving RTS off (valid only for CE5.0)

Parity Parity control type, the types available are:

0 No parity
1 Odd parity
2 Parity
3 Mark parity
4 Space parity

Txtimeout transmission timeout in milliseconds. If 0, the timeout is not active

Txretry number of attempts to repeat transmission in case of error

Rxtimeout receive timeout in milliseconds. If 0, the timeout is not active

Rxretry number of attempts to repeat reception in case of error

numSepDataTx
numSepDataRx number of characters that make up the data separator in transmission or in reception. Valid values belong to the interval 3-0. If the data separator is 0, the separator is not used.

numTermMessTx
numTermMessRx number of characters that make up the terminator message in transmission or in reception. Valid values belong to the range 3-0 If the message terminator is 0 its terminator is not used.

terminator is the set of characters used as data separator or as a message terminator. It can be:

- an integer
- an alphanumeric character placed in quotation marks
- a set of alphanumeric characters placed within quotation marks

In case the number of separators / terminators is greater than 1 (for example numSepDataRx=2) and alphanumeric characters are used as separators / terminators, they can all be put together between the same superscripts or placed individually in single quotes and separated each other by commas.

Characteristics:

All parameters of the triliteral SOP can be expressed directly using a number or indirectly via variables. After executing the SOP command, in order to reconfigure one or more parameters of the serial line, it is necessary execute the command to close the SCL line and then a new SOP command with the updated parameters.

It is possible that the configuration parameters valid for a given serial line are less than those actually configurable via the triliteral SOP. In this case, consult your hardware documentation on the serial port you want to configure. A process can enable simultaneously all the serial lines available to the OPEN*control* system (up to 4).

For the SOP command to run without errors, the enabled serial line can only be used by the process that executed the command and cannot be used by any other process.

Examples:

(SOP, 1, 9600, 8, 1, 2, 0, 1024, 500, 3, 750, 0/0, 1, 10, 2, " ", 10, 1, 13)
Serial line 1, communication at 9600 baud, 8-bit word, one stop bit, even parity, no protocol, management terminator received, 500 milliseconds timeout 3 retry transmission and transmission, 750 milliseconds timeout in receiving and no retry in reception, no separator transmission data, 1 separator received data (the number 10), two terminators message transmission (the space character and the number 10) and a terminator message on reception (the number 13)

(SOP, 2, 2400, 7, 2, 2, 1, 1152, 4000, 0, 0, 3 / 2, " "," ", 3, " ", 9, 13, 2, 13, 10, 1, 13)
Serial line 2, communication at 2400 baud, word 7 bits, 2 stop bits, even parity, logic protocol (Xon, Xoff), management and message terminator Xon, Xoff continuous, 4 second timeout retry transmission without, no retry timeout and 3 in reception, two separators given in transmission (two space characters), three separators given in reception (the space character and the numbers 9 and 13), two terminators message transmission (the numbers 13 and 10), a terminator message (number 13)

(SOP, 3, 38400, 5, 1, 1, 0, 0, 5000, 0, 1000, 5)
Serial line 3, baud 38400 baud rate, word of 5 bits, 1 stop bit, odd parity, no protocol logic, no configuration for the type of flow, transmission timeout of 5 seconds and no retry, receive timeout of 1 second and 5 retry, no data separator and no terminator message either in transmission or in reception.

13.4.2 *GET – Get data in serial line*

This command sets the process in which it is programmed to wait for data from the specified serial line. The received data are stored in the variables in the order of arrival and programming (the first data iem received is stored in the first programmed variables).

Syntax

(GET, *Portnum*, *variable*[,….., *variable*]**)**

where:

Portnum serial port number from which you want to read. The valid values range from 1 to 4.

Variable List of variables in which the incoming data from the serial line will be placed. Up to a maximum of 32 variables may be specified.

Characteristics:

The port by which the data is to be received must have been enabled and configured by the active process through the SOP triliteral. Data contained in the receiving message will be saved in the variables specified in the triliteral; the message consists of single or multiple data items separated from each other with separator characters data reception, and ends with one of the message reception terminator characters. If none of these terminators is specified when configuring the serial port, then the control will assume that the message is made up of a single data type character (1 byte).

The triliteral GET suspends the execution of the part program until the GET is considered finished; the command can be considered terminated if one of the following conditions occur:

- a character has been received and reception message terminators have not been configured
- a RESET or a CYCLE STOP has been requested by the process that is executing the command
- the reception timeout has expired or all configured retries have been executed. In this case, all characters received are lost and the serial line is reinitialized.

If the expected data is numeric (for example, it is to be placed in the variable E1), then the incoming message must contain a numeric value expressed in ASCII characters, that will automatically be converted into binary form. This conversion is not made if:

- the specified variable is a character type (string)
- one character at a time is received, i.e. no reception message terminator has been specified.

If several variables have been programmed in the GET triliteral, when configuring via the triliteral SOP at least one character terminator data reception and at least one character terminator message reception must have been specified.

Some errors generated by the *GET* command can be handled automatically or by the part program. Set the variable ERR to select the desired mode. For more information on the management of part program errors, see Appendix C.

Examples:

(GET, 1, SC10)

Waiting for a 10 character data item (excluding the terminator message reception) from the serial port 1.

(GET, 3, E1)

If the reception message terminator has been configured: waiting for numerical data encoded in ASCII from serial port 3.
If the termination character has not been configured: waiting for a single character, saved in E1 without any conversion.

(GET, 2, L30, L31, !MYVAR, SC0.10)
Waiting for three double data types and a string of 10 characters from the serial line 2.

13.4.3 *PUT – Data transmission on the serial line*

This command allows data exchange on the serial line specified.

Syntax

(PUT, *Portnum*, *data* **[**,…..,*data***])**

where:

Portnum serial port number used to send the data. The valid values range from 1 to 4.

Data data to be transmitted: it can be a constant or a local or system variable.

Characteristics:

The port used to send the data must already have been enabled and configured by the active process through triliteral SOP. Numerical data programmed into the PUT command, both immediate and those managed with variables are converted to ASCII before being transmitted.

The strings and the contents of string variables do not undergo any conversion and are sent directly. The set of characters derived from the constants or variables specified in the triliteral PUT is the message to be transmitted: it can be up to 180 characters, including separators and transmission terminators (if configured).

The transmission finishes:

- with expiration of the transmission timeout and the number of transmission attempts (number of retry transmission) is exceeded: in this case the line is automatically reinitialized
- at the conclusion, without errors, of the serial transmission
- with the arrival of a RESET or of a cycle stop in the process running the command

Some errors generated by the PUT command can be managed by the part program for a full list of these errors can be found in Appendix C.

This command cannot be programmed in MDI mod with the CYCLE_STOP state active.

Examples

(PUT,2 SC10)

Transmission on line 1 of a message made only of a 10 character data and of any message terminators.

(PUT, 1, E1)

Transmission on line 1 of the E1 variable contents and of any message terminators.

(PUT, 4, “Message”, E1, SC0.4, $CINTYPE(1))

Transmission on the serial line 4 of a constant string of 9 characters, a double numeric variable, a variable string of 4 characters and a short variable. If separators were configured they will be placed between the variables, the message ends with terminators message if configured.

13.4.4 *SRS – Serial line reset*

This command re-initializes a serial line that is configured and already open.

Syntax

(SRS [**,** *Portnum*]**)**

where:

Portnum serial port number that you want to disable. The valid values range from 1 to 4. If not specified all serial lines belonging to the current process will be disabled.

Characteristics:
This command resets one or all open serial ports configured by the current process. This triliteral cannot be programmed in MDI if the CYCLE_STOP state is active.

Examples:
(SRS, 2)
Serial line 2 reinitialization

(SRS)
Reinitialize all serial lines opened by the current process

13.4.5 *SCL – Serial port disabled*

This command disables the specified serial line making it available to other potential users. If not the number of the serial line is not specified, it disables all serial lines in the active process.

Syntax

(SCL [**,** *Portnum*]**)**

where:

Portnum serial port number that you want to disable. The valid values range from 1 to 4. If not specified all serial lines belonging to the current process will be disabled.

Characteristics:
If the serial line for which disable is request is not active, the command does not do anything and does not generate any error. This triliteral cannot be programmed in MDI with the CYCLE_STOP state active.

Examples:
(SCL, 4)
Serial line 4 disable

(SRS)
Disable all serial lines opened by the current process

14. MODIFYING THE PROGRAM EXECUTION SEQUENCE

14.1 GENERAL

This chapter discusses commands that can modify, disable or suspend the program execution sequence by:

- repeating part programs
- executing subprograms
- modifying the flow of the program
- delaying and disabling execution
- releasing the program or suspending its execution

14.1.1 *Block repetition commands*

These commands allow a specified set of program blocks to be executed several times. They can be used for repetitive machining operations such as drilling multiple holes.

COMMAND	FUNCTION
RTP	Opens the set of program blocks to be repeated
ERP	Closes the set of program blocks to be repeated

14.1.2 *Commands for executing subprograms*

Commands in this class are:

COMMAND	FUNCTION
CLS,CLD,CLT	Calls a subroutine and executes it
$(name)	Calls a subroutine and executes it
PTH	Defines and activates the default pathname for the subroutine

A subroutine is a series of blocks defining a machining cycle. The subroutine is stored as a separate file with its own name. Control passes to blocks of the subroutine each time that they are called by the CLS, CLD or CLT command. The subroutine may be called at any time and from any part of the main program.

14.1.3 *Commands for modifying program flow*

The commands in this class are:

COMMAND	FUNCTION
EPP	Executes a section of a part-program delimited by one or two labels
EPB	Executes a block of a part-program
GTO	Performs a jump or skip during the execution of the program
IF ELSE ENDIF	Executes sessions of a part-program depending on certain conditions

The GTO command makes the execution of a program jump to a block which contains a specific label. A jump may be unconditional or conditional based on E parameters, machine logic signals or numeric values.
A conditional jump is performed only if the result is true. No jump is performed if the condition is false.
The commands IF, ELSE, ENDIF allow the flow of the program to be modified conditionally without the need to define labels and program skips.

14.1.4 *Commands for delaying program execution and disabling blocks*

Commands in this class are :

COMMAND	FUNCTION
DLY	Causes a delay in the execution of the program
DSB	Disables slashed blocks

The delay command can be used to delay the execution of the program for reasons due to synchronisation.
The disable command lets the user choose to execute or not execute slashed blocks.

14.1.5 *Commands for the releasing or suspending the execution of a program*

The commands in this class are:

COMMAND	FUNCTION
REL	Releases the part-program
WOS	Puts the program in wait mode until a signal is received

14.2 RPT and ERP – Commands for program blocks repetition

The RPT and ERP commands define a set of part program blocks that must be executed a specified number of times. The set of blocks begins with the RPT command and ends with the ERP command.

Syntax

(RPT,*n***)**
.
.
.
blocks
.
.
.
(ERP)

---

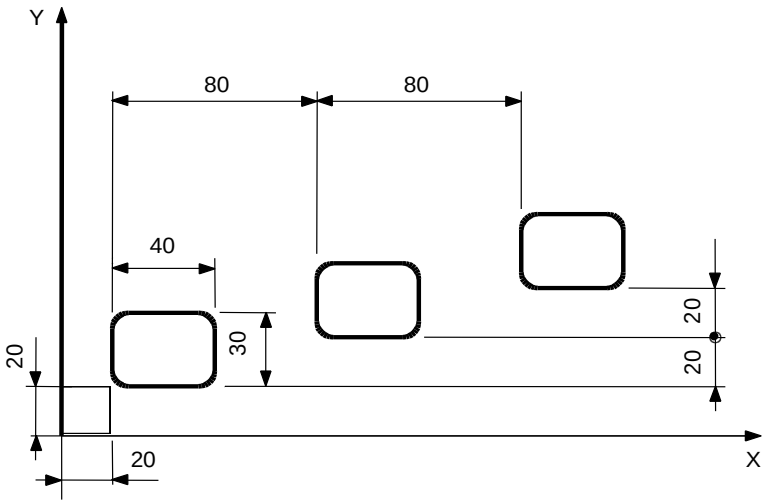
where:

n Is the number of times the specified block must be executed. It is an integer from 1 to 65535 that can be programmed directly with a number or indirectly with an E parameter. The control allows five nesting levels, i.e. in a repeat block you can program up to four repeat commands.

blocks It is the set of blocks that must be executed *n* times.

The GTO command can be programmed inside an RPT-EPP section. However, if at program end at least one of the cycles programmed with RPT is not executed, the system returns the following error: '**23/166 RPT/EPP CYCLE OPEN AT END OF PROGRAM**'.

Example 1:
The following is an example of the command.

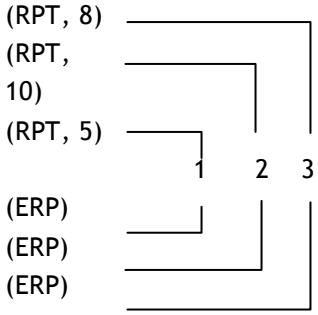


Program:

```
(DIS,"N.3 POCKETS")
(DIS,"MILL D12")
N1  S600 T6.6 M6
N2  (RPT,3)
N3  X40 Y35 M3
N4  Z 2
N5  (RPT,2)
N6  G91 Z-8
N7  G90 G1 G41 X40 Y20 F300
N8  X60
N9  Y50
N10 X20
N11 Y20
N12 G40 X40
N13 Y35 F1000
N14 (ERP)
N15 G Z2
N16 (UIO,X80,Y20)
N17 (ERP)
N18 (UAO,0)
N19 Z20
N20 X Y M30
```

Example 2:

This example shows how three repeat levels are nested.

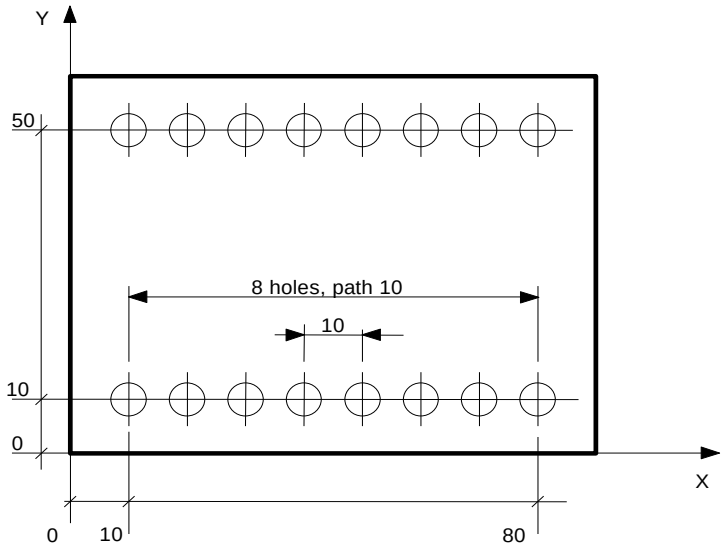


14.2.1 *Machining Equidistant Holes*

The following example uses a repeat command for machining equidistant holes.

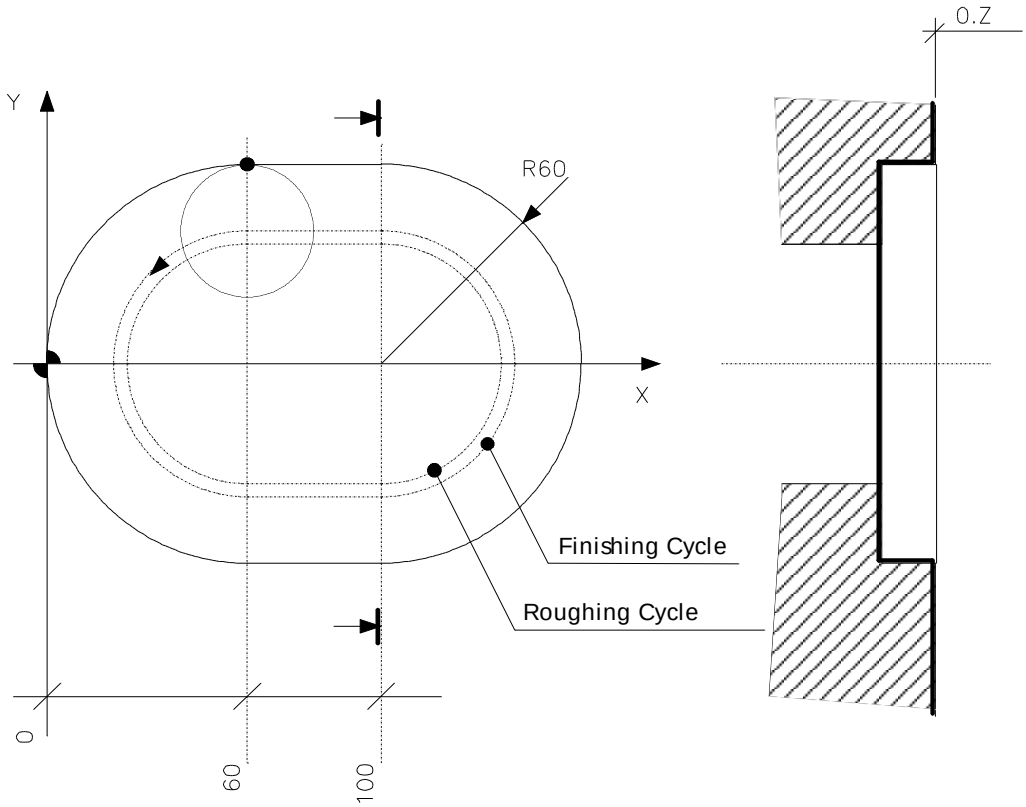
Program:

```
(DIS,"EQUIDISTANT HOLES")
N1  F200 S900 T1.1 M6
N2  G81 R5 Z-10 M3
N3  X10 Y10
N4  (RPT,7)
N5  G91 X10
N6  (ERP)
N7  Y40
N8  (RPT,7)
N9  X-10
N10 (ERP)
N11 G80 G90 XY M5
```



14.2.2 *Machining with Roughing and Finishing Cuts*

The following example shows repeat commands for machining with one roughing cut and one finishing cut.



Program:

```
(DIS,"DEFINITION OF MACHINING ALLOWANCE")
N1  S350 T6.6 M6
N2  X60 Y M3
N3  Z-50
N4  MSA=0.5
N5  (RPT,2)
N6  G1 G41 X60 Y60 F500
N7  G3 Y-60 I60 J
N8  G1 X100
N9  G3 Y60 I100J
N10 G1 G40 X60
N11 MSA=0
N12 (ERP)
N13 GZ20 M5
N14 X Y M30
```

14.3 COMMANDS FOR SUBROUTINES EXECUTION

14.3.1 *CLS – CLD – CLT – S() Call subroutines*

Commands CLS, CLD, CLT and S() recall the execution of a subroutine, i.e. a separate program stored in a file. These commands can call a subroutine from a main program or from another subroutine. Up to four nesting levels are possible.

CLS	Call to a static subroutine:	analysed upon the activation of the main part program
CLD	Call to a dynamic subroutine:	analysed during the execution of the part program and only if the name has not been found earlier
CLT	Call to a temporary subroutine:	analysed during the execution of the part program whenever its call is encountered
S()	Call to a static subroutine:	analysed upon the activation of the main part program

Syntax

(CLS,*name* **[,** *parameter1* **[,** *parameter2* **…[,** *parameterN* **]…])**
(CLD, *name* **[,** *parameter1* **[,** *parameter2* **…[,** *parameterN* **]…])**
(CLT, *name* **[,** *parameter1* **[,** *parameter2* **…[,** *parameterN* **]…])**
$(*name***) [,** *parameter1* **[,** *parameter2* **…[,** *parameterN* **]…])**

where:

name Subroutine file name.

name must be specified through either a string of capital characters or a string variable preceded by key "?" (question mark).

name may be simply a file name, or it may be specified with a logic address, through a DRIVE configured in AMP. If a path is specified, the system works out automatically the physical address associated with the DRIVE. 127 alphanumerical characters are available to specify the file name, including the path and the extension, as applicable. For further details on how to specify the file name, see paragraph “PROGRAM FILE” in chapter 1 of the following manual

parameter1
parameterN list of the optional parameters sent to the subroutine. Parameters can be programmed directly by with a decimal value or indirectly by an E parameter. All values will be available within H variables.

Example:
(CLS,\DRIVEUSER\PROGRAM.PRG) ;call to a subroutine with logic path

Example:
(CLS,PROGRAM.MIO) ;call to a direct subroutine

NOTE:

- If *name* is a string variable preceded by the "?" code, the subroutine is not analysed upon the activation of the main part program; it is analysed instead during the execution of the latter. This has the following implications, depending on the triliteral used to call the subroutine:
  - CLS or $(): the subroutine cannot contain skip instructions
  - CLD or CLT: the subroutine can contain skip instructions as it is analysed at runtime.
- The *pathname* to use for calls to subroutines may also be declared through the three-letter code PTH described later in this chapter. If the pathname is omitted in the call to a subroutine, the search for the subroutine is carried out as follows :

  1. No pathname specified via PTH

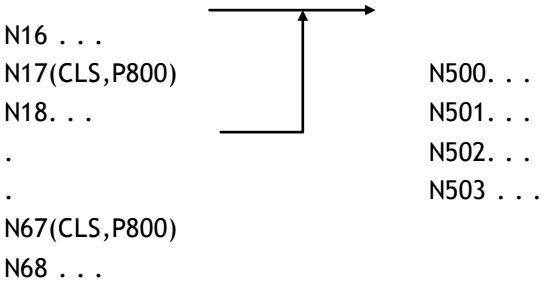
  The system searches for the subroutine in the directory in which the calling program is found and if it is not there, it looks for it in the default DOS directories defined at the machine characterisation stage.

  2. Pathname specified via the PTH instruction

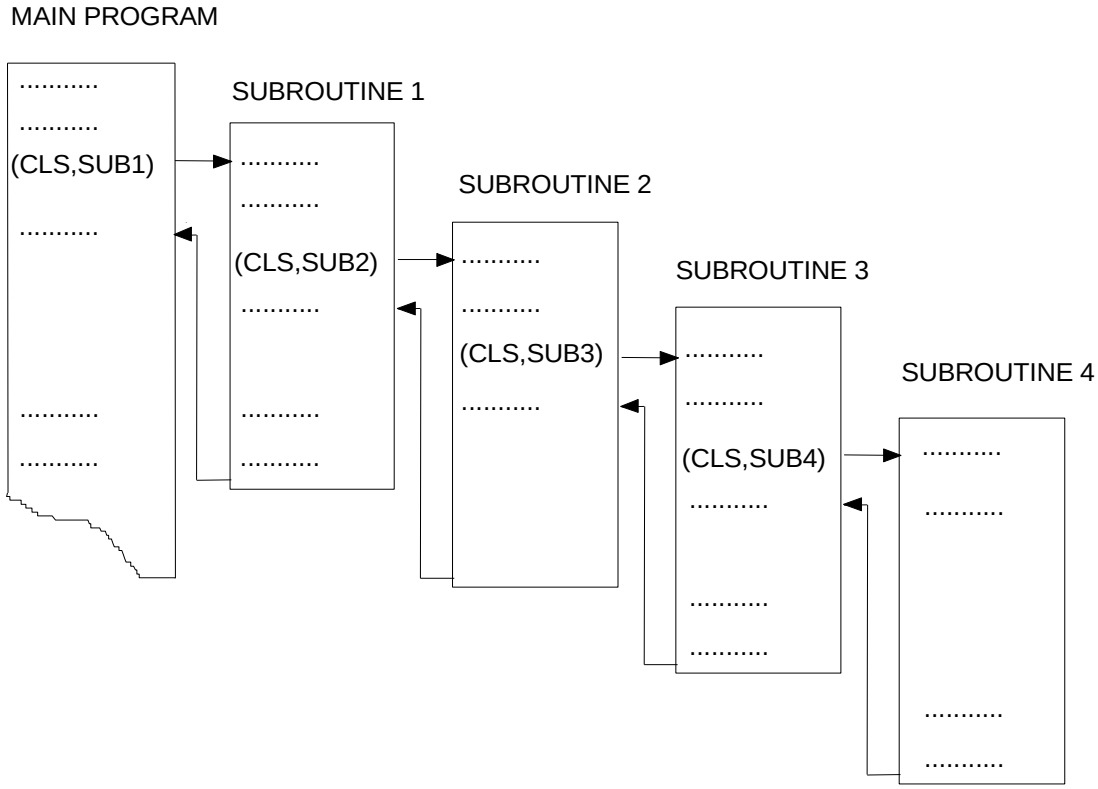
  The system looks for the subroutine in the directory specified with PTH, if it is not there, it looks in the DOS directories defined at the machine characterisation stage.
- The **number of subprograms and "static" labels** managed by the control depends on the corresponding values configured in AMP. For full tables, a memory management procedure is activated, which, as far as possible, removes the dynamic and paramacro subroutines from the tables and compacts the static ones ($() and CLS). When it is not possible to free any further space in the memory, a table full message is given out and the system shifts to the same error conditions as would have occurred during part program selection. At this point, the operator can proceed either by changing the part program or increasing the maximum limits specified for the tables.
- When the tables are full and the **execution of a program is not blocked** thanks to the memory compacting feature, program execution is slowed down, during the subsequent stage of analysis, which inevitably takes place in the event of a blocking error.
- The successful completion of a program activation stage is no guarantee that the program will be executed without errors, if the memory is full. This is the case, for instance, of a CLD calling a number of CLS or $() instructions large enough to fill up the tables.
- **Re-executing a main program** may generate a tables full message even if the program has been activated in its entirety the first time (after its activation). This is because after the activation of a program, the tables do not contain the file names and the static subroutine tables called either by static subroutines or by dynamic or temporary subroutines.
- A command for a **call to the same program name** is entered through triliterals of different types (e.g. first CLT and then CLS) a label unknown message may be generated; the message can be obviated by replacing the CLS call with a CLD call.
- A **run time change to a subroutine** not followed by the reactivation of the main program will be interpreted only if the call is made through a CLT; otherwise, the one previously analysed will be re-executed, with unpredictable effects on program execution.
- Optional parameters calling a subroutine are assigned to H variables. Parameter in N position is mapped on H variable of N-1 index.

Example 1:

An example of call to a subroutine is shown below, a path with PTH is not specified.
Main program Subroutine P800 ;is searched for in the directory where the main program resides and in the default directories configured in AMP



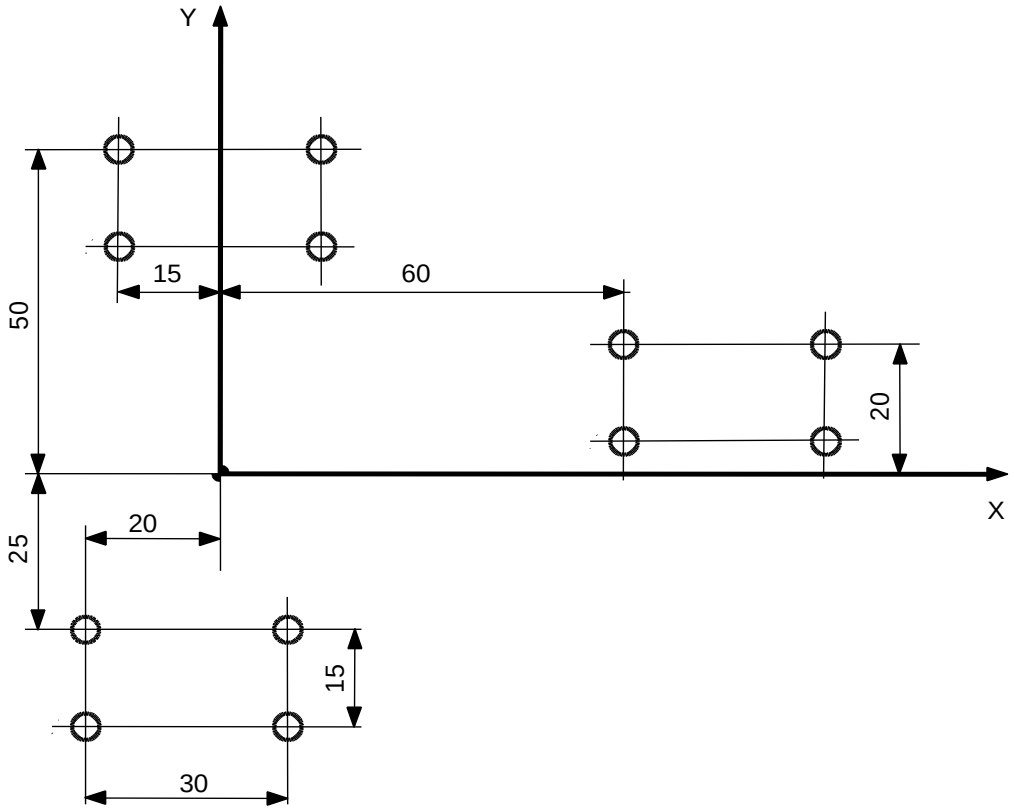
Example 2:
Sequence for execution of four nested subroutines.



Example 3:
Call to subroutine specified in implicit mode
SC0.5="PIPPO"
(CLS,?SC0.5)

Example 4:

This example uses subroutines for repeated drilling operations.



Main Program:

```
N19 (DIS,"...")
N20 S2000 F180 T2.02M6
N21 (UTO,1,X-20,Y-25)
N22 (CLS,S600)
N23 (UTO,1,X-15,Y50)
N24 (CLS,S600)
N25 (UTO,1,X60,Y20)
N26 (CLS,S600)
N27 Z0
```

Subroutine S600

```
N501 G81 R-108 Z-130 M3
N502 XY
N503 Y-15
N504 X30
N505 Y0
N506 G80
```

Example 5:

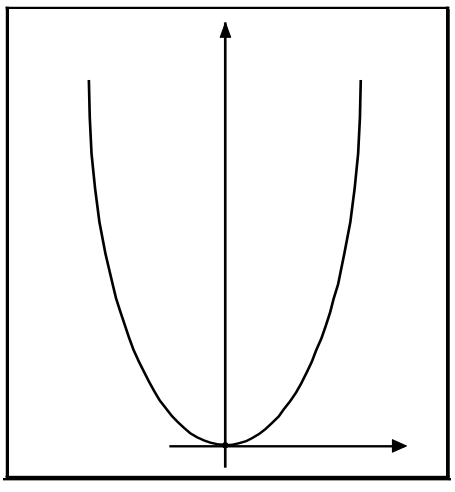
Parabolic profile programming with a parametric subroutine.

Main Program:

```
;N10 MAIN PAR
N20 T1.1M6 S1000 M3 F700
N30 E30=72.795                    ;start X
N40 E31=24.28                     ;focal length (twice the focus)
N50 E32=2                         ;Y increment
N60 E33=108.24                    ;start Y
N70 E34=0                         ;final Y
N80 GX0 Y120
N90 (CLS,PAR)
```

Subprogram PAR:

```
;N500 PAR
;N501 Parametric subroutine: Complete parabola execution, internal profile.
N502 G1 G42 XE30 YE33
N503 E36 = E33
"START" N504
N505 E36=E36-E32
N506 (GTO,END,E36<E34)
N507 E35=SQR( 2*E31*ABS(E36))
N508 XE35 YE36
N509 (GTO,START)
"END" N510
N511 E35=SQR (2*E31*ABS(E34))
N512 XE35 YE34
;N513 Second part of the parabola.
N514 E42=E32
N515 E43=E34
N516 E44=E33
N517 XE35 YE43
"START2" N518
N519 E43=E43+E42
N520 (GTO,END2,E43>E44)
N521 E35= -(SQR (2*E31*ABS(E43)))
N522 XE35 YE43
N523 (GTO,START2)
"END2" N524
N525 E35= - (SQR(2*E31*ABS(E44)))
N526 G40 XE35 YE44
N527 GX0
```



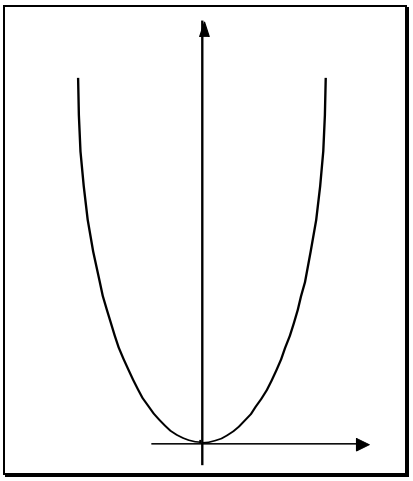
Example 6:
Call to the dynamic subroutine, specified in an implicit manner with the PAR subprogram of the previous example.

Main program:

```
;N10 MAIN PAR
N20 T1.1M6 S1000 M3 F700
N30 E30=72.795                ;Initial value of X
N40 E31=24.28                 ;Focal length
N50 E32=2                     ;Increment of Y
N60 E33=108.24                ;Initial value of Y
N70 E34=0                     ;Final value of Y
N80 GX0 Y120
SC0.5="PAR"
(CLD,?SC0.5)
```

Example 7:
Execution of the parabolic profile (Ex. 5) using direct passing of parameters to the subroutine.

Main program:
;N10 MAIN PAR
N20 T1.1M6 S1000 M3 F700
N30 GX0 Y120
N40 (CLS,PAR, 72.795, 24.28, 2, 108.24,0)
PAR subroutine:
;N500 PAR
;N501 Subroutine parameter: complete parabolic profile execution.
N502 G1 G42 XH0 YH3
N503 E36 = H3
"START" N504
N505 E36=E36-H2
N506 (GTO,END,E36<H4)
N507 E35=SQR(2*H1*ABS(E36))
N508 XE35 YE36
N509 (GTO,START)
"END" N510
N511 E35=SQR (2*H1*ABS(H4))
N512 XE35 YH4
;N513 Second part of the parabolic profile
N514 E42=H2
N515 E43=H4
N516 E44=H3
N517 XE35 YE43
"START2" N518
N519 E43=E43+E42
N520 (GTO,END2,E43>E44)
N521 E35= -(SQR (2*H1*ABS(E43)))
N522 XE35 YE43
N523 (GTO,START2)
"END2" N524
N525 E35= - (SQR(2*H1*ABS(E44)))
N526 G40 XE35 YE44
N527 GX0



14.3.2 *PTH – Declaration of the default pathname*

The PTH command declares the *pathname* to be used as the default in calls to subroutines and paramacros.

Syntax

(PTH, *mode* [,*pathname*]**)**

---

where:

mode *mode* may have the following values:

mode=0
Takes as the default *pathname* for CLS calls the *path* of the main program.
mode=1
Declares the *path* to use when there is a (PTH,2) instruction active.
This mode may be of use when a *pathname* defined initially with (PTH,1,*pathname*) has to be called numerous times in the part program with (PTH,2).
mode=2
Activates the *path* specified in the instruction or, if none is specified, the one declared previously with a (PTH,1,*pathname*) instruction.

pathname Path to be declared as the *path*; this is an optional parameter. It is not a physical address and hence it must contain a logic DRIVE that was associated with a physical path in AMP at the machine characterisation stage.

NOTE:
The *pathname* declared with three-letter code PTH is preserved even after a RESET of the process in which it was programmed.

14.3.3 *EPP – Executing a Portion of Program*

The EPP command executes a subprogram, i.e. a portion of a program delimited by one or two blocks with label fields.

Syntax

(EPP, *label1***,** [*label2*]**)**

---

where:

label1 Label field of the first block to be executed. A label is a sequence of up to six alphanumeric characters.

label2 Label field of the last subprogram block.

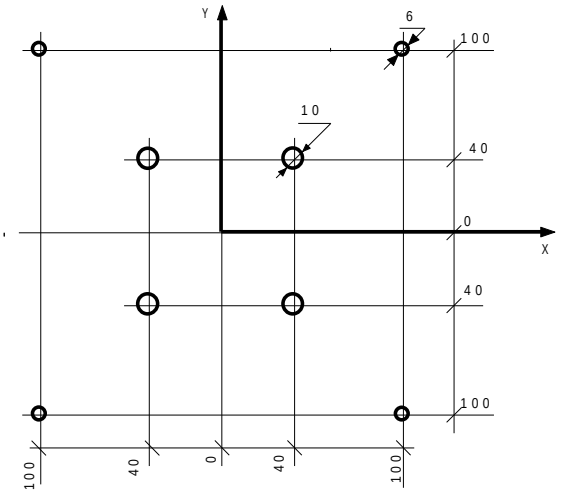
Characteristics:
Labels have to be programmed between quotes ("LABEL1") in program blocks, while in the EPP command they are declared without quotes. The system accepts up to 5 nesting levels.
If the command is programmed with one label field, then the return to the calling point happens when executing an M code of type “exit from subroutine” (an M02, typically).
If the end of part program is reached when executing an EPP command with one label field, then execution ends and there is no return to the calling point.
In contouring operations, the EPP command can be used for finish milling with the same program blocks used for roughing. During the roughing phase, the MSA command could be used to program the machining allowance for finishing.
In positioning operations, after programming points for a centering operation, the EPP command calls for different tools in order to execute different operations on each hole. The EPP command can be used to execute a complete machining operation at different orientations on the active interpolation plane.

Example 1:

```
      .
      .
"START"N25                    First block with label
      .
      .                       An EPP cannot occur here
      .
"END"N100                     Last block with label
      .
      .
      .
N150 (EPP,START,END)          EPP command that specifies the labels. The control executes
                              blocks N25 to N100; after this point execution continues on the
                              block written after EPP (N150).
```

The '**23/165 NESTING OF EPP GREATED THAN 5**' error occurs if more than five instructions are nested in an EPP command.
GTO command can be programmed inside an EPP section. However, if at program end at least one of the cycles programmed with EPP are not fully executed, the system returns the following error: '**23/166 RPT/EPP CYCLE OPEN AT END OF PROGRAM**'.

Example 2:
This example shows how to use the EPP command in a positioning operation:



Program:

```
N1 (DIS,"DRILLING CENTRE HOLES")
N2 F300 S2000 T1.1 M3 M6
N3 G81 R0 Z-3
"D6"N4
N5 X100 Y100
N6 X-100
N7 Y-100
N8 X100
"D10"N9
N10 X40 Y40
N11 X-40
N12 Y-40
N13 X40
"END"N14
N15 G80
N16 (DIS,"BIT D6")
N17 F200 S1800 T2.02 M3 M6
N18 G81 R Z-22
N19 (EPP,D6,D10)
N20 G80
N21 (DIS,"BIT D10")
N22 F220 S1600 T3.3 M3 M6
N23 G81 R Z-24
N24 (EPP,D10,END)
N25 G80
```

14.3.4 *EPB – Execute Part-Program Block*

The three-letter code EPB allows execution of a part-program block.
Execution can be conditioned by the result of a comparison specified within the command; If the condition is not met an alternate part-program block may be defined so as it can be executed.

Syntax

(**EPB,** *block1* [**,** *condition* [, *block2*] **)**

where:

block1 Is the part-program block to be executed.
This can be a string in inverted commas or a character, local or system variable.

par1 Is a local or system variable or a constant whose value is compared to the value contained in the parameter *par2*.

condition Is a condition formed by operations and logic operands.

block2 Is the part-program executed when if the specified condition for *block1* is not met. It is an optional parameter. This can be a string in inverted commas or a character, local or system variable.

Characteristics:
If *par1 ,operator* and *par2* conditions are not specified, the program always executes the *part_program_block1 block*.
Only one level of nesting is accepted by the system with the three-letter code EPB.

Example:
(EPB, "(EPB,'E1=1')") : accepted by the system
(EPB, "(EPB,'(EPB,SC0.100)')") : NOT accepted by the system (error: FORMAT ERROR)

The part-program blocks specified in the three-letter code EPB are not analysed in the activation phase of the part-program, therefore it is up to the programmer to see that these blocks do not cause any malfunctioning of the part-program.

Example:

(EPB,"(CLS,SUBROUT)")	the subprogram SUBROUT is not pre-analysed in the activation phase, therefore it cannot contain any jump (GTO) instructions.
(EPB,' "LABEL" ')	The label LABEL will not be inserted in the label table of the part-program.
(EPB, " (DIS, E1)")	equivalent to the block (DIS, E1)
SC0.30= " E1 =10"	
(EPB, SC0.30)	equivalent to the block E1=10
SC40.30 = "# X10"	
(EPB, SC40.30)	equivalent to the block #X10
(EPB,"(CLS,SUBROUT)",E1=34)	calls the subprogram SUBROUT only if E1=34
(EPB," E1=100", SN1=25," E1=0")	assigns 100 to E1 only if SN1=25, otherwise assigns 0 to E1
(EPB,"(EPP,LAB1,LAB2)",SC0.2="OK","(EPP,LAB3,LAB4)")	executes from the label LAB1 to the label LAB2 if SC0.2 is equal to OK, otherwise executes from the label LAB3 to the label LAB4.
(EPB, "(EPB,' E1 = 2', E0 < 100)", E0 > 70)	assigns 2 to E1 only if E0 is between 70 and 100.
(EPB," (EPB,' E0 = 5', E0 < 5) ",E0 < 10,E0=10)	assigns the value 5 to E0, if E0<5; assigns 10 to E0 if E0>10. If 5<E0<0, E0 does not change.

14.4 BRANCHING AND DELAY COMMANDS

14.4.1 *GTO - Branch command*

The GTO command performs a jump to a block specified using a label.
The command can contain a condition, in this case if the condition is met then the jump is allowed. If the condition is false then no jump is performed.

Syntax

(GTO,*label* [**,***condition*]**)**

---

where:

label Label of the program block to branch to. A label is a string of up to 6 alphanumeric characters. In the destination block the label must be between quotes while in the GTO command the label must be programmed without quotes, it may also be a character variable preceded by "?"

condition Is a condition formed by operations and logic operands.

Characteristics:
If *par1, operator* and *par2* are not specified, the program always jumps to the block marked by the label.
The label the part program must branch to can also be expressed as a local or system variable (preceded by the "?" character).
This facilitates execution of programs having a great number of blocks. In particular, by programming the label as a variable it is possible to program a default label and call it "DEFLAB": The part program jumps to "DEFLAB" if the label in the variable does not exist.

Example 1:
;SC0.3 contains the label

(GTO,?SC0.3)
"ONE"
.
.
.
.
"TWO"
.
.
.
.
"THREE"
.
.
.
"DEFLAB"
.
.
.
If SC0.3 does not contain "ONE", "TWO" OR "THREE" the part program jumps to the "DEFLAB" block. If the "DEFLAB" label does not exist, error **'23/156 Undefined label'** is displayed and part program execution is interrupted.

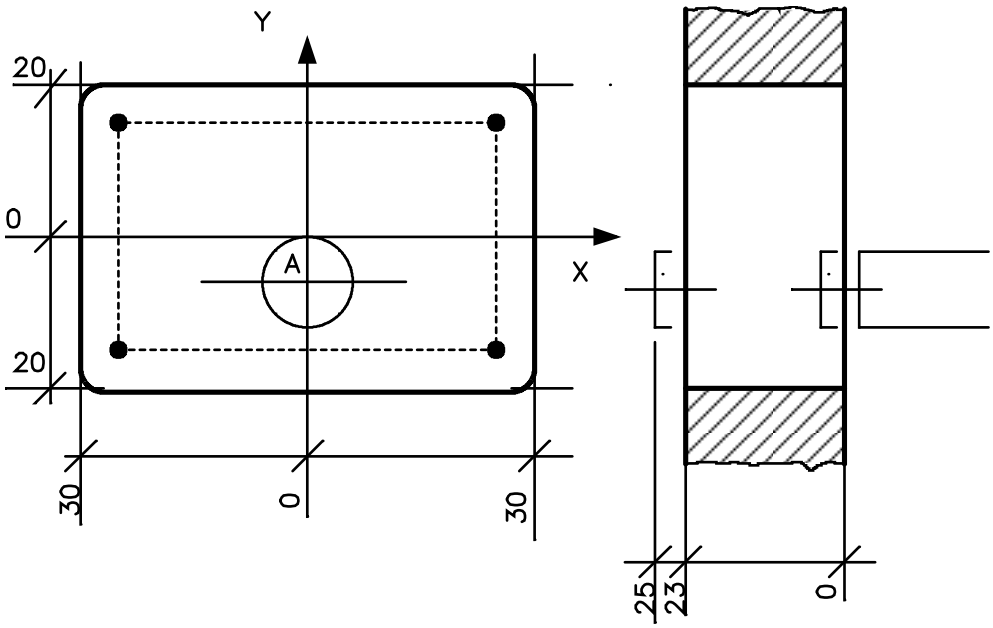
Example 2:

N01 (GTO,START)	the part program always branches to "START"
N10 (GTO,END,E1 > 123)	branch to "END" if the value of E1 is greater than 123
N20 (GTO,LAB1,@COOLANT = 1)	branch to "LAB1" if the PLUS variable @COOLANT = 1
N30 (GTO,START,E1 <> E5)	branch to "START" if the value of E1 is different to that of E5
N40 (GTO,LAB1,SC1.2H = "OK")	branch to "LAB1" if the two characters beginning with SC1 are equal to OK
N50 SC1.3="ABC"	prepares the variable for the following block
N60 (GTO,?SC1.3)	branch to ABC label

Example 3:
The instruction:
(GTO,END,SC2.3 = "ABC")
branches to the "END" label in the program if the three characters (.3) beginning from SC2 are ABC.

NOTE:
The character string to be compared in the branch command must always be programmed between quotes.

Example 4:
Here is an example of conditional branching in slot milling.

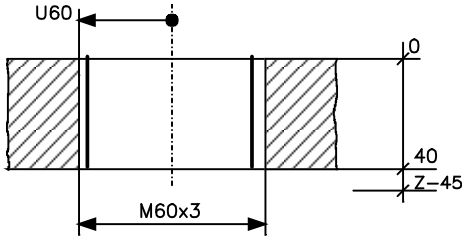


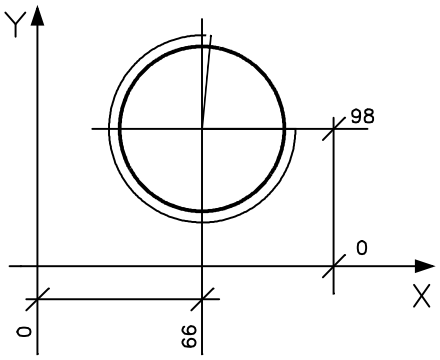
Program:

```
N1 (DIS,"MILL A SLOT")
N2 F500 S2000 T1.1 M3 M6
N3 E31=-3.5
N4 E32=-24
"START"
N6 G X Y-10
N7 Z(E31)
N8 G1 G42 X Y-20
N9 X-30
N10 Y20
N11 X30
N12 Y-20
N13 G40 X
N14 Y-10
"END"
N16 E31=E31-3.5
N17 (GTO,START,E31>E32)
N18 E31=-25
N19 (EPP,START,END)
N20 G Z10
```

Example 5:
This is an example conditional branching in cylindrical thread machining.

E30 = Diameter of Next Cut
E31 = Cut Depth Increment (Diameter)
E32 = Return Diameter
E33 = Final Diameter





Program:

```
N1 (DIS,"THREAD DIA 60")
N2 S150 T5.5 M6
N3 G0 X66 Y98 Z5 M3
N4 E30=56.8
N5 E31=0.5
N6 E32=50
N7 E33=60
"I" N8
N9 G0 Z5
N10 U(E30)
N11 G33 Z-45 K3
N12 GU(E32)
N13 E30=E30+E31
N14 (GTO,F,E30>E33)
N15 (GTO,I)
"F" N16
N17 GU(E32)
N18 Z5
N19 U(E33)
N20 G33 Z-45 K3
N21 GU(E32)
N22 Z20
```

14.4.2 *IF ELSE ENDIF*

The IF command allows the execution of a part-program section which will run only if a condition specified within the command is met. This part-program section must terminate with the ENDIF command : the ELSE command can be optionally included in the IF command, and it defines, up until the ENDIF command, a part-program section which will be executed if the condition specified in the IF command is not met.

Syntax

(IF, *condition***)**
.
blocks

.
(ELSE)
.
blocks
.
(ENDIF)

---

where:
condition Is a condition formed by operations and logic operands.

blocks blocks array to be executed

Characteristics.
If the condition specified in the IF command is met, the blocks specified up until the ELSE command (if present) or until the ENDIF command (if the ELSE command is not present) will be executed, otherwise they will be ignored.

The blocks between the ELSE command and the ENDIF command will be executed only if the condition specified in the IF command is not met.

Up 32 levels of nesting are allowed with the IF ELSE ENDIF command.
Each IF command must have a corresponding ENDIF command.

Examples:

```
( IF , E0 = 3)
( DIS , " E0 is equal to 3 ")
(ELSE)
( IF E0 > 3)
( DIS , " E0 is greater than  3 ")
(ELSE)
( DIS , " E0 is less than  3 ")
( ENDIF)
( ENDIF)

 ( IF , E0 >10 )
( IF , E0 >20 )
( IF , E0 >30 )
( IF , E0 >40 )
( IF , E0 >50 )
( DIS , " E0 is greater than 50  ")
( ENDIF)
( ENDIF)
( ENDIF)
( ENDIF)
( ENDIF)
( ENDIF)
```

14.4.3 *DLY – Defining delay time*

The DLY command specifies a delay in program execution.

Syntax

(DLY,*time***)**

where:

time Delay time in seconds (minimum value 0.1).Delay is defined using a numerical constant or an E parameter.

14.4.4 *DSB – Disable Slashed Blocks*

This command enables/disables slashed blocks.
Slashed blocks have "/" symbol as the first character and their execution is conditioned by DSB value.
Syntax

DSB = *value*

where:
value The value may be:
0 slashed blocks are executed
1 slashed blocks are not executed

14.4.5 *REL – Releasing the part program*

The REL command releases the part program and all subroutines.

Syntax

(REL)

Characteristics:
The REL function disables the part program. It can be entered through MDI.
After the program has been released, the following message will be displayed: ***'23/160 END OF FILE'***.

14.4.6 *WOS – Wait on signal*

Wait on signal command puts the part program into a wait status until the specified condition is met.

Syntax

(WOS, *par1 operator par2***)**

where:

par1 Is a local or system variable or a constant to be compared to the value of *par2*.

operator Logic operators that can be used in expressions:

=	equal to
<	lower than
>	higher than
< >	different than
<=	lower or equal to
>=	higher or equal to

par2 Is a local or system variable or a constant to be compared to the value of *par1*.

Characteristics:
The process where the WOS function is programmed goes into WAIT until the condition is satisfied.
By default the WOS command is synchronised.

If the system is in WAIT during program debugging (because a WOS function has been programmed), it is possible to assign to the *par1* variable the value that satisfies the condition specified in WOS.

To make this assignment, follow these steps:

1. Press CYCLE STOP
2. Key in the correct value of the *par1* variable in MDI.
3. Press CYCLE STOP

Example:

```
(WOS,E1>4)
(WOS,SC10.5 = "GOOFY")
(WOS, TOOL_STATUS>=E3)
(WOS,!USER_VAR3.5CH 0 SC4.5)
```

15. MULTIPROCESS MANAGEMENT COMMANDS

15.1 General

This chapter discusses inter-process commands, i.e. instructions that allow the programmer to synchronise parallel part program execution on various processes or to shift the "axes" resource between processes in order to meet specific requirements.

The following table summarises these commands:

Three-letter codes	Functions
PRO	Specifies the default process to which synchronisation codes and commands must be sent.
SND	Sends a synchronization message to the specified process.
WAI	Puts the current process into WAIT until a message arrives from another process.
EXE	Activates automatic part program execution by the specified process.
ECM	Executes an MDI block in the specified process
RTP	Reads the number of the active / programmed tool for the specified process
ROP	Reads the number of the active / programmed offset for the specified process
GPS	Reads status and mode for the specified process
RAP	Read axis position for the specified axis and process
CON	Execute a CYCLE ON in the specified process
COF	Execute a CYCLE OFF in the specified process
HON	Execute an HOLD ON in the specified process
HOF	Execute an HOLD OFF in the specified process
RES	Execute a RESET in the specified process
SMD	Select the operative mode (SETMODE) for the specified process
SAX	Select axis for manual movements for the specified process
DIR	Select direction for manual movements for the specified process
JOG	Defines increments for jog movements
FHO	Send stop movement signal for the specified process
GTA	Shifts the "axes" resource between processes
PLS	Reads one or more SW variables belonging to WinPLUS
PLR	Reads a status variable of type WORD
PLD	Reads a status variable of type LREAL

By default SND and WAI are synchronised.

15.2 Synchronization between Processes

15.2.1 *Notes on "WAIT" function:*

The synchronization messages received by a process are recorded in a memory area called MESSAGE QUEUE. This local area is process-dedicated: there is one message queue for each process. If the (WAI, P*n*) instruction is executed by a part program, the process has to wait for a message from process <n>. The system looks for the message in the MESSAGE QUEUE:

- ***If the message is not found in the MESSAGE QUEUE***, the process puts itself in WAIT and suspends part program execution.
- ***If the message is in the MESSAGE QUEUE***, part program execution continues and the message is deleted from the message queue.
- ***If the queue contains a message coming from a process other than <n>,*** the message is not deleted but it is kept aside in order to be used when a WAI for the appropriate process is programmed.

The process can exit from WAIT mode only when the queue contains a message coming from the process specified in WAI. If the process specified in the WAI command is process 0, then any process can “awaken” the destination process from the WAIT state.

15.2.2 *Notes on "SEND" function:*

Three-letter code SND allows synchronisation messages to be sent to the specified process; the messages may be synchronous or asynchronous:

- ***If the message is synchronous***, the process that sent SND goes into WAIT and waits for an ACK (acknowledge to use the message) from the destination process;
- ***If the message is asynchronous***, the process that sent SND continues executing the part program without waiting for the message to resume the SND destination process.

A process (e.g. process 1) cannot send another process (e.g. process 2) more than one message at a time, at least not until process 2 exits from WAIT and the message is cleared from the queue. However, the sender can send messages to other processes (e.g. process 3). Resetting the process or enabling another part program clears the message queue.

15.2.3 *Exchanging data*

Commands SND and WAI permit the exchange between processes of data, such as local or system variables, constants, strings. The data is sent to (SND) or received (WAI) from the requested processes and may be specified in programming of the respective functions; for further details, see the syntax of the two three-letter codes on the following pages.

15.2.4 *Resetting synchronised processes*

If a reset is made, it must be made both on the processes sending the synchronisation messages (SND) and on the processes receiving the synchronisation messages (WAI).
This is to avoid unwanted messages remaining in the "MESSAGE QUEUE" of the processes, which could lead to malfunctions.

15.3 PRO – Definition of the default process

The PRO command allows the programmer to specify which process the synchronisation commands refer to. The process number is an optional parameter in all commands of this type.

Syntax

(PRO, *process number***)**

---

where:

process number It can be a number or a system or local variable in the 1-24 range It defines the default process for multiprocess commands.

A reset of the system or activation of a part program resets the previous definitions of the default process.

15.4 SND – Send a synchronization message

SND allows a synchronisation message and data to be sent to the specified process.

Syntax

(SND [**,P** *process number*,] *synchro* [,*data*] [,*data*] ..... [,*data*]**)**

where:

process number Destination process number (from 1 to 24). It can be a number, a local or system variable.
If it is the number of the process the SND command is executing, a **"23/500 Current process number programmed"** message will be generated.
If it is omitted, the information will be sent to the default process declared with PRO.

synchro It specified the execution mode for the command. Two modes are available:

S (synchronous) the sender or current process goes into WAIT until the destination process has not received the message, cancelled it from the queue and resumed part program execution.

A (asynchronous) part program execution continues immediately after the message has been sent.

MS (multi-synchronous) SND can be synchronously activated on a process having some multi-asynchronous messaged on wait.

MA (multi-asynchronous) several SND commands van be grouped towards the same process

data The data (optional and max. 20 items) may be Long Real numbers, strings between quotes, and local or system variables. The system reads the programmed data and transmits them to the specified process together with the synchronisation message. It may send up to 20 parameters, which may occupy up to 570 bytes.

The length of the various data types is as follows:

Data type	Length (in byte)
Boolean / Byte	2
Short	3
Real	5
Double	9
String	number of characters + 2

Characteristics:
Synchronous and asynchronous modes allow to send only one message to a process and if a second message is sent, the CN will display error **23/511 – Message still in queue** for both processes.

Multi-synchronous ad multi-asynchronous modes allow to send several messages to a destination process without generating any error. Only one multi-synchronous message can be sent as its execution sets the sending process in WAIT status.

Example:
(SND,P1,S,E1,SN12,"TOOL",SC0.30,@BOOL_PLUS(0),33.6)

The number of bytes transmitted by this block is:

Variable or constant:	Type:	Length (in byte):
E1	Double	9
SN12	Double	9
"TOOL"	String	6
SC0.30	String	32
@BOOL_PLUS(0)	Boolean	2
33.6	Double	9
	Total:	67 byte transmitted

The system generates the "**23/509 Data sending too long**" error if there is an attempt to send more than 570 bytes.

- **Error situations:**
  The "**23/510 Data loading failed**" error message occurs in two cases:

  Example 1 (SND synchronous mode)
  Process 1 sends a synchronous send (SND,P2,S,E1) to process 2 while it is in WAIT (WAI,P1,SC0.30). Because the types of sent and expected variables are different, then:
  Both processes will display the "**23/510 Data loading failed**" error message and their programs will remain in WAIT.

  Example 2 (SND asynchronous mode)
  Process 1 sends the asynchronous send (SND,P2,S,E1) to process 2, in which a wait (WAI,P1,SC0.30) is programmed.

  In this case, as the transmitted data is not consistent with that expected by process 2, and the SND is asynchronous, the result is that:
  - process 1 resumes part program execution
  - process 2 displays the **23/510 Data loading failed** error message and its part program is not allowed to start.

15.5 WAI – Wait for a synchronization message

WAI puts the process into WAIT - waiting for a synchronization message to be sent by a specified process. If the message is available in the message queue, part program execution continues.
If send message was of “synchronized” type than the receiver responds to the sender.

Syntax

(WAI [,**P** *process number*] [**,***variable*] [**,***variable*] ..... [**,***variable*]**)**

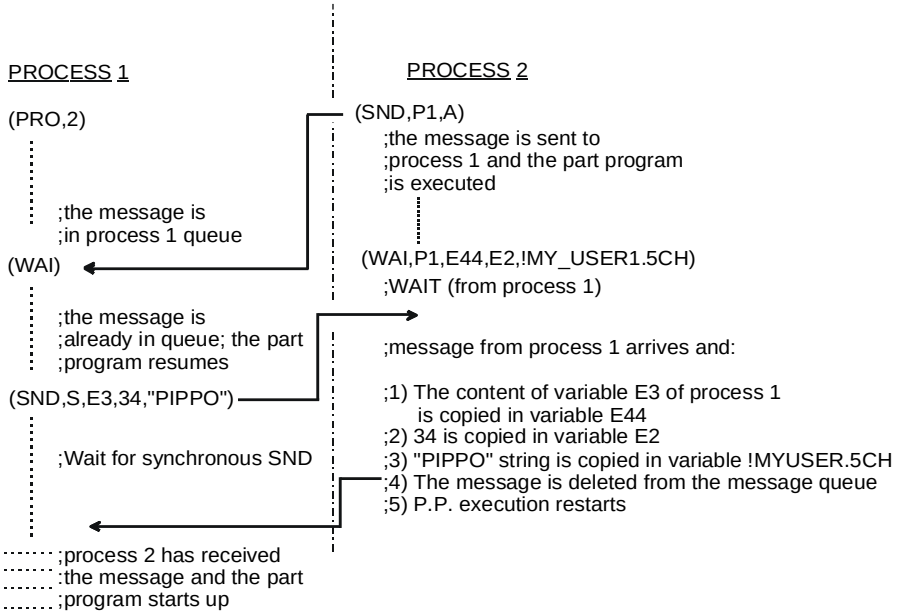
---

where:

process number Sender process number (from 0 to 24). It can be a number, a local variable or a system variable.
If it is the number of the process in which the WAI command is executing, a **23/500 Current process number programmed** message will be generated.
If it is omitted, the information will be sent to the default process declared with PRO.

variable List of the variables, maximum 20, in which to receive the alphanumeric values arriving with the synchronisation message. If number or type of the variables programmed is not consistent with those sent, the sender process is sent error message **23/510 Data sent too long** (see also the SND command for further details).

NOTE:
When a process is in WAIT and a synchronisation message with data arrives, the ACK requested by the sender is sent after the data has been written to the programmed variables. If data fails to be written because of format or process number errors, a negative ACK is sent to the sender.
When *process number* is equal to 0 then any process will be considered the requested process for synchronisation. If synchronisation demands associated data then the sending process has to respect the number and type of the requested data.



Example of using SND and WAI commands

15.6 EXE – Automatic part program execution

EXE starts execution of a part program for the specified process.

Syntax

(EXE, *part program name* [**,P** *process number*]**)**

where:

part program name Name of the part program to be executed in AUTO mode.
It may be a string of characters (not between quotes) or a character variable preceded by ?.

part program name could be a filename or be specified using a logic address with a DRIVE configured in AMP. If a path is specified, then the system automatically generates the physical address associated with DRIVE. For further details on how specify a part program name, please refer to paragraph "The program files" in Chapter 2 of this manual.

process number Number of the process (from 1 to 24) that runs the program. It can be a number, a local variable or a system variable.
If it is the number of the process that is executing the EXE command, a **23/500 Current process number programmed** message will be generated.
If it is omitted, the message will be sent to the default process declared with PRO.

Characteristics:
The EXE command forces execution of the specified process in AUTO mode, activates and executes the part program. Commands executed on target process SETMODE, SPG and CYCLE ON could be filtered by the PLC.

If SETMODE, SPG or CYCLE ON are received by a process that cannot execute them because it is in RUN or HOLD, **23/512 EXE or ECM failed** error messages will be displayed.

Destination process, after program execution, will remain in AUTO.

Errors returned by the EXE command can be managed by a part program through the STE and ERR variables. For more details, see appendix C.

Examples:

```
(EXE, MAIN_PROG ,P3)        activates execution of the MAIN_PROG program by process 3

(PRO,2)
SC1.10="MAIN_2"
(EXE,?SC1.10)               activates execution of the MAIN_2 program by process 2
```

15.7 ECM – MDI block execution in a process

ECM executes a block in the specified process.

Syntax

(ECM, *command* **[,P** *process number*] **[,S])**

where:

command It is the block of code that will be executed. It may be a string variable or a string between apexes or quotes.
The apexes are used when in the command there are quotes.

process number Number of the process (from 1 to 24) that will execute the block of code. It can be a number, a local variable or a system variable.
If it is the number of the process executing the EMC command, an **23/500 Current process number programmed** message will be generated.
If it is omitted, the information will be sent to the default process declared with PRO.

S optional parameter forcing synchronous mode operation

Characteristics:
The EMC command forces the destination process in MDI mode, sends it to the specified command and executes it on that process.
The commands to be executed on the target process (SETMODE and CYCLE ON) can be filtered by the PLC.

The ECM command by default (without the S parameter) is of the asynchronous type: this means that after executing ECM, the part program continues with the following blocks irrespective of whether the command sent via ECM to another process has been concluded or not.
It is therefore important not to execute another ECM command on the same process until the execution of the previous command has been completed.

If this occurs, the **"23/512 EXE or ECM failed"** error is signalled.
The ECM command with the S parameter is of synchronous type; this means that after the execution of ECM, before proceeding with the following blocks the part program waits until the command has been completed on the recipient process.

If MDI or CYCLE ON are received by a process that cannot execute them because it is in RUN or HOLD, the **23/512 EXE or ECM failed** error message will be displayed.
Errors returned by the ECM command can be managed by a part program through the STE and ERR variables. For further details, see appendix C.

Example 1:

(ECM, '(DIS, "TEST")', P2, S) Process 2 will visualize the string TEST. Current process waits for the end of execution of DIS command on process 2

(ECM,"E1=12",P2) Sets to 12 the value of local variable E1 in process 2

Example 2:

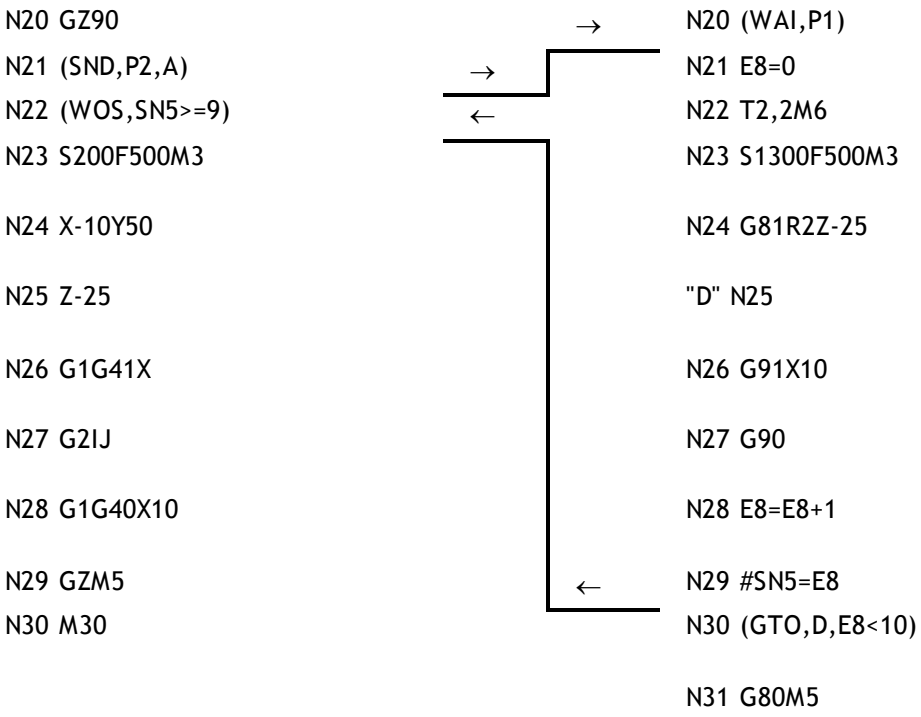
(PRO,2) Other example:

SC0.5="E1=12"Prepares the command in variable SC0.5

(ECM,SC0.5) Executes ECM for a command written contained by the variable

Example of synchronisation of two process using EXE:

;N1 SNA_N		;N1 SNB_N
;N2 MULTIPROCESS: SYNCHRONISATION		;N2 MULTIPROCESS; SYNCHRONISATION
N3 (DIS,"PROGRAM SYSTEM 1")		N3 (DIS,"PROGRAM SYSTEM 2")
N4 (EXE,SNB_N,P2)		N4 SN5=0
N5 T2.2M6		N5 T4.4M6
N6 S1500F500M3	→	N6 (WAI,P1)
N7 XYZ		N7 S400F400M3
N8 E25=0		N8 XY
N9 G81R-14Z-35		N9 E25=0
N10 (RPT,6)		N10 G84R-10Z-22
N11 (ROT,E25)		N11 (RPT,6)
N12 X-80Y		N12 (ROT,E25)
N13 E25=E25+60		N13 X-80Y
N14 (ERP)		N14 E25=E25+60
N15 G80Z50		N15 (ERP)
N16 (ROT,0)		N16 (ROT,0)
N17 (SND,P2,A)	→	N17 G80ZM5
N18 (CLS,MA)	→	N18 (WAI,P1)
N19 (SND,P2,S)	→	N19 GZ40


N20 GZ90
N21 (SND,P2,A)
N22 (WOS,SN5>=9)
N23 S200F500M3
N24 X-10Y50
N25 Z-25
N26 G1G41X
N27 G2IJ
N28 G1G40X10
N29 GZM5
N30 M30
N20 (WAI,P1)
N21 E8=0
N22 T2,2M6
N23 S1300F500M3
N24 G81R2Z-25
"D" N25
N26 G91X10
N27 G90
N28 E8=E8+1
N29 #SN5=E8
N30 (GTO,D,E8<10)
N31 G80M5

15.8 RTP – Reading the number of the programmed or active tool

This instruction reads the number of the active or programmed tool and, if requested, the record (page) where this tool is mapped, from the specified process (or from the default process defined with PRO) and then writes the number into the specified variable.

Syntax

(RTP [**,P** *num pro*]**,** *type***,** *variable1* [**,***variable2*]**)**

where:

num pro *(optional)* Is the number of the process that includes the tool number. If it is omitted, the tool number will be read from the default process specified with PRO.
It is a number from 1 to 24. It may be written as a numerical value or read from a local or system variable.

type It specifies whether the tool number refers to the active (**A**) or the programmed (**N**) tool.

variable1 Is the numerical variable where the requested tool number will be written.

variable2 Is the numerical variable where the related record will be written.

Characteristics
If the default process has not been defined and the **P***num pro* parameter is omitted, the system will read the tool number from the current active process.

Examples:
(RTP,N,E1)
This command reads the number of the programmed tool from the current process and writes it into the E1 variable.

(RTP,P3,A,E3)
This command reads the active tool number from process 3 and writes it into the E3 variable.

(RTP,A,E1,E2)
This commands writes the current tool number into the current process E1 variable, and it writes the page or record where the tool is mapped into the E2 variable.

15.9 ROP – Reading the number of the programmed or active tool offset

This instruction reads the number of the active or programmed tool offset from the specified process (or from the default process defined with PRO) and then writes the number into the specified variable.

Syntax:

(ROP [**,P** *num pro*]**,** *type***,** *variable***)**

---

where:

num pro Is the number of the process from which the tool offset number will be read. If it is omitted, the tool number will be read from the default process specified with PRO. It is a number from 1 to 24. It may be written as a numerical value or read from a local or system variable.

type It specifies whether the tool offset number refers to the active (**A**) or the programmed (**N**) tool offset.

variable Is the number of the variable in which the tool offset number must be written.

Characteristics:

If the default process has not been defined and the **P***num pro* parameter is omitted, the system will read the tool number from the current active process.

Examples:

(ROP,N,SN3)

This command reads the number of the programmed tool offset from the current process and writes it into the SN3 variable.

(ROP,P2,!ABC(1))

This command reads the active tool offset number from process 2 and writes it into the !ABC(1) user variable.

15.10 GPS – Reading the process state, sub-state or mode

This instruction reads the state, sub-state or mode of the specified or default process and then writes it into the specified variables.

Syntax:

(GPS [**,P** *num pro*]**,** *variable1***,** *variable2***,** *variable3***)**

where:

num pro Is the number of the process. If it is omitted, the state, sub-state or mode will be that of the default process specified with PRO.
It is a number from 1 to 24. It may be written as a numerical value or read from a local or system variable.

variable1
variable2
variable3 Are three numerical variables in which the process state, sub-state and mode are written. The meaning of these numerical values is as follows:

Value	State
1	IDLE
2	RUN
3	HOLD
4	RUNH
5	HRUN
6	ERROR
7	WAIT
8	RESET
9	EMERGENCY
10	reserved

Value	Sub-state
1	DUMMY
2	reserved
3	reserved
4	BLOCK RETRACE
5	reserved
6	MAS
7	reserved
8	
9	
10	

Value	Mode
1	MDI
2	AUTO
3	STEP
4	MANUAL JOG
5	INCR. JOG
6	RETURN PROFILE
7	HOME
8	HANDWHEEL
9	
10	

For a detailed description of the states, sub-states and modes listed in the table refer to "OPEN*control* User Manual".

Characteristics
If the default process has not been defined and the **P***num pro* parameter is omitted, the system will read information from the current active process.

Examples:
(GPS,E1,E2,E3)
This instruction reads the state, sub-state and mode of the current process and writes them into variables E1, E2 and E3.

(GPS,P4,!UT(1),!UT(2),!UT(3))
This instruction reads the state, sub-state and mode of the current process and writes them into user variable !UT(1 ÷ 3).

15.11 RAP – Reading the axis coordinates

This instruction reads the position of the specified axis.

Syntax

(RAP [**,P** *num pro*]**,** *axis name***,** *type of coordinate***,** *variable***)**

where:

num pro Is the number of the process. If it is omitted, the system will read the position of the axis in the default process specified with PRO. It is a number from 1 to 24. It may be written as a numerical value or read from a local or system variable.

axis name Is the ASCII name of the axis whose position will be read by the system.

type of coordinate It is a number from 0 to 10. It may be written as a numerical value or read from a local or system variable. The meaning of this value is as follows:

- **0** absolute interpolated coordinate (distance to the axis zero)
- **1** PROGRAMMED coordinate being executed
- **2** DISTANCE to go (DTG)
- **3** absolute real coordinate (distance to the axis zero)
- **4** ERROR coordinate
- **5** interpolated coordinate (includes the origins and the tool offset)
- **6** real coordinate (includes the origins and the tool offset)
- **7** PROGRAMMED coordinate at the time RAP instruction is analysed
- **8** INTERPOLATED position (comprehends origins and tools offset)
- **9** PROBE CARTESIAN position
- **10** PROBE JOINT position (ABSOLUTE REAL)

variable Is the local or system variable where the axis position will be written.

Characteristics

The coordinates read by the RAP triliteral are expressed in the unit of measure configured for the process. Rotary axes coordinates are always expressed in degrees.

If the axis name is P, it is mandatory to specify the process number (even if it is the default process). In type 7 readings, the command takes into account the virtualisation and DANs currently active on the programmed axis, if any.

Examples:

To read the absolute real coordinate of the X axis in the default process, write it into the E4 variable and display it on the screen, program the following instructions:

```
(RAP,X,3,E4)
(DIS,E4)
```

To read the DISTANCE TO GO for the Z axis in process 3 and write it into the SN5 variable, program the following:

```
(RAP,P3,2,SN5)
```

15.12 CON – Cycle ON command

This instruction sends the CYCLE ON command to the specified process, which is equivalent to pressing the **CYCLE START** button.

Syntax

(CON [, **P** *numproc*]**)**

where:

numproc Is a number between 1 and 24 identifying the process on which the CYCLE ON command will be executed. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

Characteristics

The result of issuing this command depends on the current mode of operation.
If the **P** *numproc* parameter was not specified, the CYCLE ON command will be given to the default process, i.e. the one set earlier with the PRO instruction.

Example:

(CON, P3)
The CYCLE ON command will be issued to Process 3 .

15.13 COF – Cycle OFF command

This instruction sends the CYCLE OFF command to the specified process, which is equivalent to releasing the **CYCLE START** button.

Syntax

(COF [, **P** *numproc*]**)**

where:

numproc Is a number between 1 and 24 identifying the process on which the CYCLE OFF command will be executed. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

Characteristics

The result of issuing this command depends on the current mode of operation.
If the **P** *numproc* parameter was not specified, the CYCLE OFF command will be given to the default process, i.e. the one set earlier with the PRO instruction.

Example:

(COF)
The default process will be issued with the CYCLE OFF command.

15.14 HON – HOLD ON command

This instruction sends the HOLD ON command to the specified process, which is equivalent to pressing the **CYCLE STOP** (**HOLD**) button.

Syntax

(HON [**, P** *numproc*]**)**

where:

numproc Is a number between 1 and 24 identifying the process on which the HOLD ON command will be executed. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

Characteristics:
If the **P** *numproc* parameter was not specified, the HOLD ON command will be given to the default process, i.e. the one set earlier with the PRO instruction.
If no default process was set earlier and the **P** numproc parameter was not included in this command, the command will be executed on the current process (i.e. the one executing the current instruction).

Example:
(HON, P2)
Process 2 will be issued with the HOLD ON command.

15.15 HOF – HOLD OFF command

This instruction sends the HOLD OFF command to the specified process, which is equivalent to pressing the **CYCLE STOP** (**HOLD**) button in **CYCLE STOP** or **HOLD** mode.

Syntax

(HOF [**, P** *numproc*]**)**

where:

numproc Is a number between 1 and 24 identifying the process on which the HOLD OFF command will be executed. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

Characteristics
If the **P** *numproc* parameter was not specified, the HOLD OFF command will be given to the default process, i.e. the one set earlier with the PRO instruction.

Example:
(HOF)
The default process will be issued with the HOLD OFF command.

15.16 RES – RESET command

This instruction sends a RESET command to the specified process, which is equivalent to pressing the **RESET** on the operator panel.

Syntax

(RES [**,** *P numproc*]**)**

where:

numproc Is a number between 1 and 24 identifying the process. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

Characteristics:

If the **P** *numproc* parameter was not specified, the RESET command will be given to the default process, i.e. the one set earlier with the PRO instruction.

If no default process was set earlier and the **P** *numproc* parameter was not included in this command, the command will be executed on the current process (i.e. the one executing the current instruction).

Example:

(RES,P4)

Process 4 will be issued with the RESET command.

15.17 SMD – Set operating mode

This instruction defines mode of operation for the specified process.

Syntax

(SMD [**,P** *numproc*] **,***mode***)**

where:

numproc Is a number between 1 and 24 identifying the process. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

mode The mode of operation desired. Possible values for this parameter are:

- **1** MDI
- **2** Auto
- **3** Block/block
- **4** Continuous manual
- **5** Incremental manual
- **6** Return on profile
- **7** Homing cycle
- **8** Handwheel

Characteristics:

If the *P numproc* parameter was not specified, the SMD command will be given to the default process, i.e. the one set earlier with the PRO instruction.

If no default process was set earlier and the *P numproc* parameter was not included in this command, the command will be executed on the current process (i.e. the one executing the current instruction).

Example:

(SMD,P4,3)

The block/block operating mode is set on process 4.

15.18 SAX – Select axis for manual movement

This instruction selects the axis on which a manual movement is to be executed.

Syntax

(SAX [**, P** *numproc*] **,** *axis name***)**

where:

numproc Is a number between 1 and 24 identifying the process. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

axis name Is the ASCII name of the axis to execute the movement on.

Characteristics

If the **P** *numproc* parameter was not specified, the SAX command will be given to the default process, i.e. the one set earlier with the PRO instruction.

If no default process was set and the **P** *numproc* parameter was not included in this command, the command will be executed on the current process (i.e. the one executing the current instruction).

Example:

(SAX,Z)

Axis Z of the current process is selected for manual movement.

If the axis name is P, it is mandatory to specify the process number (even if it is the default process).

15.19 DIR – Direction of manual movements

This instruction defines the direction of the axis selected during the manual movement.

Syntax

(DIR [**,** *P numproc]* **,** *sign***)**

where:

numproc Is a number between 1 and 24 identifying the process. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

sign Sign of the direction of manual movement. Possible values are **+** or **-**.

Characteristics

If the **P** *numproc* parameter was not specified, the DIR command will be given to the default process, i.e. the one set earlier with the PRO instruction.

If no default process was set earlier and the **P** *numproc* parameter was not included in this command, the command will be executed on the current process (i.e. the one executing the current instruction).

Example:

(DIR,P3,+)

A positive manual movement is selected for the axis selected earlier with the SAX command in process 3.

15.20 JOG – Jog increment value

This instruction defines the value of the increment during jog manual movement in the specified process.

Syntax

(JOG [**,** *P numproc]* **,** *jog value***)**

where:

numproc Is a number between 1 and 24 identifying the process. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

jog value Is the numeric value for the step in jog manual movement.
It may be a number or a local or system variable.
Will be in mm or inches depending on the units configured for the process.

Characteristics

If the **P** *numproc* parameter was not specified, the JOG command will be given to the default process, i.e. the one set earlier with the PRO instruction.
If no default process was set earlier and the **P** *numproc* parameter was not included in this command, the command will be executed on the current process (i.e. the one executing the current instruction).

Example:

(JOG,P2,0.1)
A value of 0.1 mm (or 0.1 inches) will be taken as the increment for jog manual movement in process 2.

15.21 FHO – Enable/disable FEEDHOLD

This instruction stops the machining of a specific process, forcing the working feed rate to zero.

Syntax:

(FHO [**, P** *numproc*] **,** *enable***)**

---

where:

numproc Is a number between 1 and 24 identifying the process. This parameter may be a number directly identifying the process or a local or system variable containing the number of the process desired.

enable Is a number either 0, 1 or 2 representing the FEEDHOLD enable or disable command. With a value of 1 the FEEDHOLD command is enabled, i.e., the movement is stopped at once; with 2 the programmed movement currently being executed is stopped; with 0 the movement is re-enabled.

Characteristics

If the **P** *numproc* parameter was not specified, the FEEDHOLD command will be given to the default process, i.e. the one set earlier with the PRO instruction.

Example:

(FHO,P4,1)

Machining movement is stopped in process 4.

15.22 PLS – Reading WinPLUS SW variables

This instruction reads bits or words (16 bits) from WinPLUS SW variables and writes them into the specified local or system variables.

Syntax

(PLS, *SW number*, *bit number*, [**&** *mask*], *variable*
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]
[,*SW number*, *bit number*, [**&** *mask*], *variable*]**)**

where:

SW number Is the number of the SW variable. It can be specified directly as an integer or indirectly as an E parameter using a local or system variable.

bit number Is the number of a bit in the specified SW variable. It may be written as a numerical value or read from a local or system variable.
It is a number from 0 to 16 that may be written as a numerical value or read from a local or system variable specified in the command. Values from 0 to 15 indicate single bits, whereas 16 indicates the whole word. It can be specified directly as an integer or indirectly as an E parameter using a local or system variable.

mask This parameter can be programmed only if *bit number* is 16 (whole word). *mask* provides a decimal description of the bits of the word setting those be read to 1. It is applied in an AND to the contents of the specified SW variable before saving it in the specified local or system variable.
It may be written as a numerical value or read from the local or system variable specified in the instruction. It can be specified directly as an integer or indirectly as a local or system variable.

variable Is the local or system numerical variable in which the contents of the specified bit or word will be written.

Characteristics

This instruction may specify up to 10 readings, which may be bits of the same SW variable, or the bits/word from different variables.
The meaning of the bits and words of each SW variable is described in Chapter 3 (sections "System state flags" and "Process state flags") of the WinPLUS Application Manual.

Examples:
The following command
(PLS,1,16, E1)
will save the active Security level in E1.

The following command
(PLS,23,2,SN2)
writes 1 into the SN2 variable if G18 is active and writes 0 if G18 is not active.

With the following command
(PLS,26,16,&1792,E0)
1792 masks bits 8,9 &10 equivalent to G72, 73, 74
0 will be written in E0 if there is no probing cycle in process 1.

With this command
(PLS,27,0,E1,47,0,E2,67,0,E3,87,0,E4)

The results of the specified command will be as follows:
E1 will be 0 if no canned cycle is active in process 1.
E2 will be 0 if no canned cycle is active in process 2.
E3 will be 0 if no canned cycle is active in process 3.
E4 will be 0 if no canned cycle is active in process 4.

15.23 PLR – Reading WinPLUS WORD variables

This instruction reads the value of the WinPLUS variables of type WORD (MW, GW, PW, SW) and writes them into the specified local or system variable.

Syntax

(PLR, *class*, *word_index*, *variable***)**

where

class It is an integer that defines what class of WORD variable will be read. Allowed values are:
0 : GW class
1 : MW class
2 : PW class
3 : SW class

word_index Index for the chosen variable class.
It may be written as a numerical value (integer) or read from a local or system variable.

variable Is the local or system numerical variable in which the contents of the specified bit or word will be written.

15.24 PLD – Reading WinPLUS DOUBLE variables

This instruction reads the value of the PLC variables of type double (MD, GD, PD, SD) and writes them into the specified local or system variable.

Syntax

(PLD, *class*, *word_index*, *variable***)**

where

type It is an integer that defines what class of DOUBLE variable will be read. Allowed values are:
0 : GD class
1 : MD class
2 : PD class
3 : SD class

word_index Index for the chosen variable class.
It may be written as a numerical value (integer) or read from a local or system variable.

variable Is the local or system numerical variable in which the contents of the specified bit or word will be written.

15.25 NOTES ABOUT “SHARED AXES” FUNCTION

15.25.1 *General*

The "shared axes" feature allows one or more processes to acquire or release axes resources. An axis is acquired or released by associating its name (X,Y,Z) to one of the identifiers configured in AMP (from 1 to 64).

- A name cannot be assigned to different axes in a process.
- An identifier cannot be assigned to different machine axes.

15.25.2 *Conditions for axis acquisition*

Before acquiring an axis check whether the following conditions apply:

- the axis is configured in AMP (the identifier exists in the configured processes or in the PLC)
-
- the axis has been released from all processes or it is shared by the PLC.
- the axis name is a name accepted in programming;
- the identifier (ID) must not be associated with an auxiliary axis
- the identifier (ID) must not be associated with a spindle axis
- the identifier (ID) is a number between 1 and 64 and is associated neither to a spindle nor to an auxiliary axis;
- the name to be assigned to the axis is not used for a pseudo axis in the process.

15.26 GTA – Get Axes

GTA allows one or more axes to shift between processes. In addition, it redefines the axis/offset association and sets the sequence in which axes are displayed.

Syntax

(GTA, [T*value*,] *axis1 ID1* [, *axis2 ID2*] … [,*axis12 ID12*] [/ [*axis1C*] [,*axis2C*]…[, *axis5C*]]**)**
(GTA)

where:

value It is an integer value. Its allowed values are
0: deactivate origins (this is the same as omitting the T0 parameter)
1: maintain active origins on the programmed axes with a different (or the same) name but with the same ID and order.
2: maintain active origins on all set axes, regardless of the programming origin of the ID's.

axis1…axis12 It is the name of the axis that will be shown at position *n*.

ID1 … ID12 It is the ID associated with the axis. It can be programmed directly using an integer from 1 to 64 or indirectly using an E parameter.

axis1C…axis5C Are the axes to which length offsets will be applied the next time an offset is enabled.

Characteristics:
In a GTA block all the controlled axes have to be declared, including those already belonging to the process. If an axis is not specified in a GTA command it will be released and made available to other processes.
When a GTA command is executed the axes are displayed in the order in which they are specified. GTA can be used for changing this order.

The RESET command ***does not remove*** the changes made by GTA. To restore axis configuration the system has to be rebooted.
When the control executes a tool change or enables an offset, the axes to which length offsets are associated must already be controlled by the process.

15.26.1 *Inhibiting axis acquisition and release*

Axes acquisition/release with GTA may be inhibited in the following cases:

- When a cutter compensation offset is active (G41 / G42)
- When a canned cycle (G81-89) is active
- When the system is in HOLD or IDLE-MAS
- When the tool correction mode (h or T) is active and you are releasing an axis where the tool correction has been applied or are acquiring a new axis to which the tool correction must be applied.
- Release of axes shared with the logic.

15.26.2 *System parameters after GTA*

A GTA command initialises the following information:

Interpolation plane

- The interpolation plane is defined by two of the first three axes programmed by the current G (G17, G18, G19).
- If GTA defines less than three axes, the first and second axes will be forced as abscissa and ordinate.
- To ensure correct operation, the desired interpolation plane must be programmed with G16.

Origins

- If the "Tvalue" parameter is omitted or set to "value=0", the active origins will remain current.
- If the "Tvalue" parameter is set to value=1, the origins will be maintained on the reprogrammed axes with the same ID and order.
- If the Tvalue" parameter is set to value=2, the origins will be maintained on all the reprogrammed axes even with different ID.

Mirror

- Mirroring is reset for all axes

Scale factor:

- Scale factors are reset for all axes

ROT rotation:

- Rotation programmed with a ROT command will be cancelled.

Travel limits:

- Travel limits for all the axes are restored to configured values.

Axis selection for manual movements:

- After a GTA command, the first displayed axis will be selected for manual move execution.

Length offsets

- The axes identifiers to which length offsets are associated can be programmed as an option with GTA. Default associations defined in AMP will be lost. They will be restored after the control has been switched on again.

15.27 Information retained after a GTA command

- Locked axes
- Incremental and temporary origins

Example 1:

Initial conditions:

- Controlled axes are X (with ID 1) and Y (with ID 2).

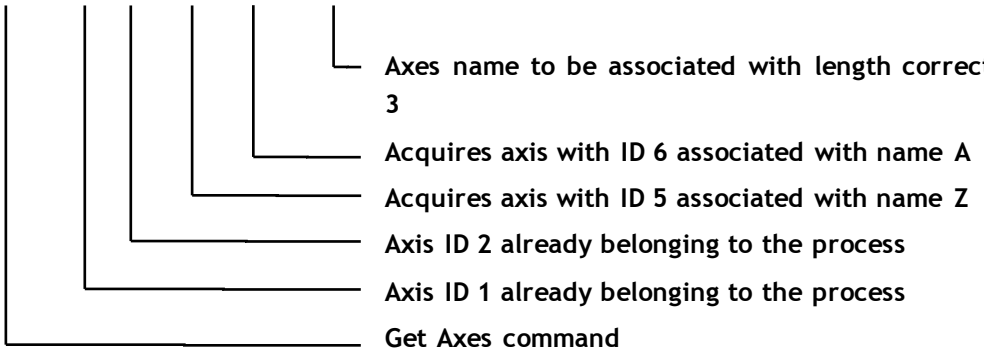
Goals:

- Maintain control over axes X and Y (ID 1 and 2)
- Acquire axes associated with ID 5 and 6 and name them Z and A
- Associate length offset 1 with axis Y (ID 2)
- Associate length offset 2 with axis Z (ID 5)
- Associate length offset 3 with axis X (ID 1)
- Obtain the following display order: X, Y, Z, A.

Programming example:

Programming all the axes (new and old) is mandatory. It reassigns axes names and including those already belonging to the process is mandatory and allows redefines both the axis names and the display order.

(GTA,X1,Y2,Z5,A6/Y,Z,X)



Example 2:

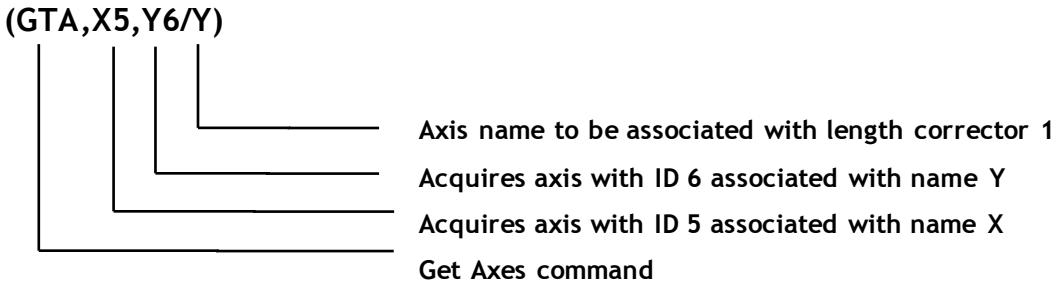
Initial conditions:

- Controlled axes are X (with ID 1) and Y (with ID 2).

Goals:

- Release axes X and Y (ID 1 and 2)
- Acquire axes associated to ID 5 and 6 and name them X and Y
- Associate length offset 1 with axis Y (ID 6)
- Obtain the following display order: X, Y.

Programming example:



Example 3:

Initial conditions:

- An indefinite number of controlled axes

Goals:

- Release all its axes

Programming example:

Example 4:

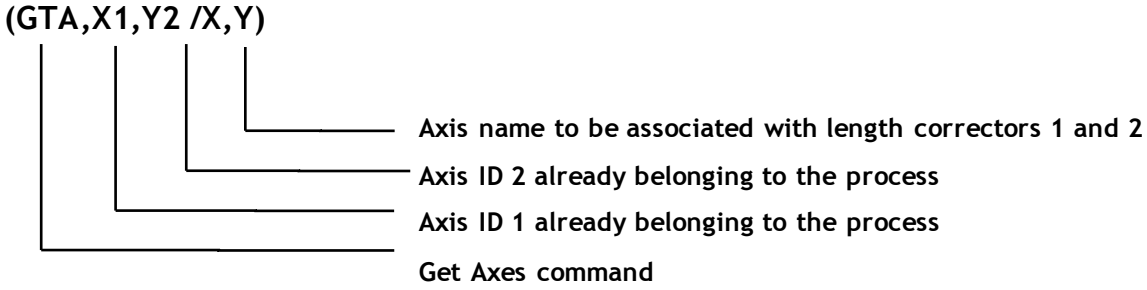
Initial conditions:

- Controlled axes are X,Y,Z,A,B with ID 1,2,3,4,5.

Goals:

- Retain control of X and Y axes (with ID's 1 and 2)
- Release axes Z, A and B (with ID's 3, 4 and 5)
- Associate length offset 1 with axis X (ID 1)
- Associate length offset 1 with axis X (ID 2)
- Obtain the following display order: X, Y.

Programming example:



Example 5:

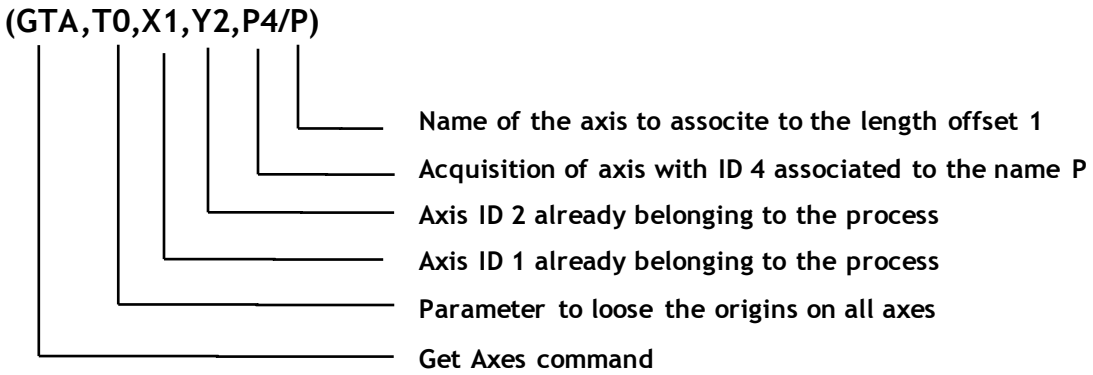
Initial Conditions:

- Controlled axes are X,Y,Z axes with ID 1,2,3.
- Origin 1 is associated to the X,Y,Z axes.

Goals:

- Retain control of the axes with ID 1 and 2 (X,Y).
- Release the axis Z (ID 3).
- Acquire the P axis with ID 4.
- Associate length offset 1 with the P axis (ID 4).
- Obtain the following display order: X,Y,P.
- Deactivate the origin associated to all axes.

Programming example:



Example 6:

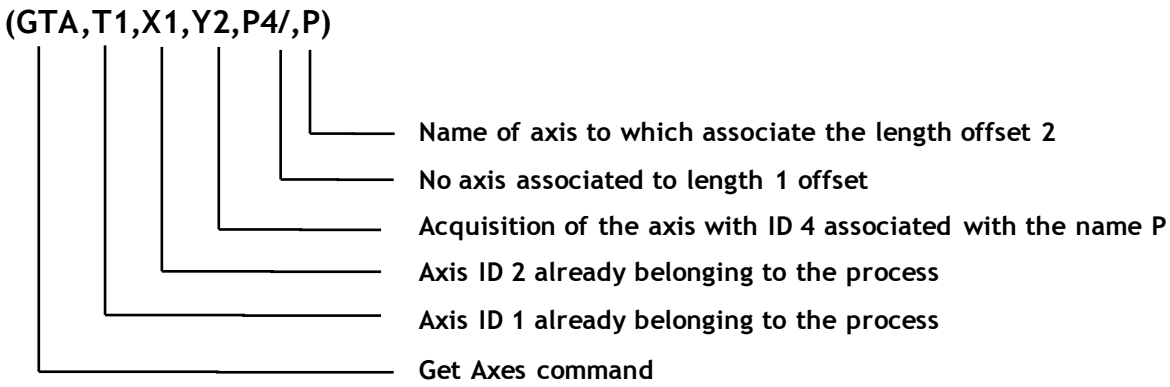
Initial Conditions:

- Controlled axes are X,Y,Z axes with ID 1,2,3.
- Origin 1 is associated with X,Y,Z axes.

Goals:

- Retain control of the axes with ID 1 and 2 (X,Y).
- Release the axis Z (ID 3).
- Acquire the P axes with ID 4.
- Associate the length offset 2 with the P axis (ID 4).
- Obtain the following display order: X,Y,P.
- Retain the associated origins with the axes with ID 1 and ID 2.

Programming example:



Example 7:

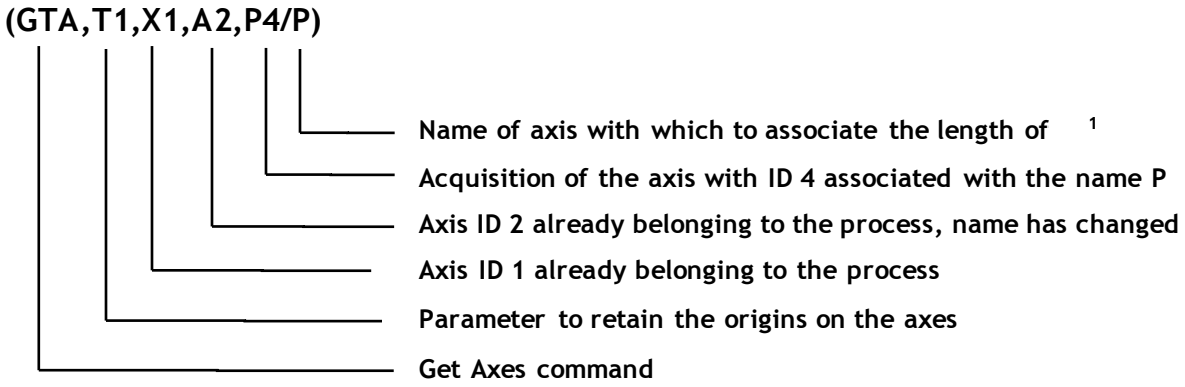
Initial Conditions :

- Controlled axes are the X,Y,Z axes with ID 1,2,3.
- Origin 1 is associated to X,Y,Z axis.

Goals:

- Retain control over the axis with ID 1 (X).
- Retain control over the axis with ID 2 ID 2 (Y) changing the name to (A).
- Release the Z axis (ID 3).
- Acquire the P axis with ID 4.
- Associate the length offset 1 with the P axis (ID 4).
- Obtain the following display order: X,A,P.
- Retain the origin associated with ID 1 and ID 2.

Programming example:



Example 8:

Initial conditions:

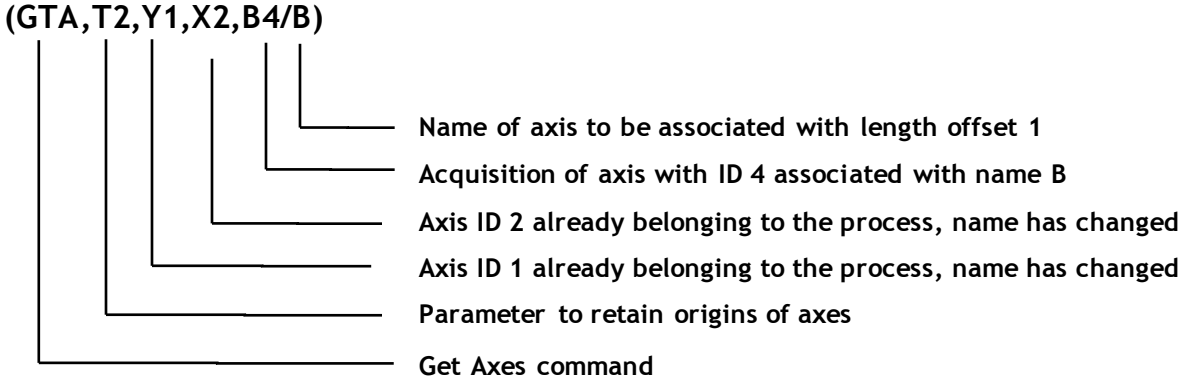
- Controlled axes are X,Y,Z axes with ID 1,2,3.
- Origin 1 is associated with axes X,Y,Z.
- A 2nd process retains axes A, B with ID 4, 5, with origin 2 associated.

Goals:

- Retain control over axes with ID 1, 2 changing the names
- Release axis Z (ID 3).
- Acquire axis B, with ID 4, previously released by process 2.
- Associate length offset 1 with axis B (ID 4).
- Obtain the following display order (on system video): Y,X,B.
- Maintain the origins associated with all the axes, i.e. origin 1 for axes with ID's 1 (Y), 2 (X) and origin 2 for axis with ID 4 (B).

Programming example:

(GTA) on process 2 for the release of A and B



Example 9:

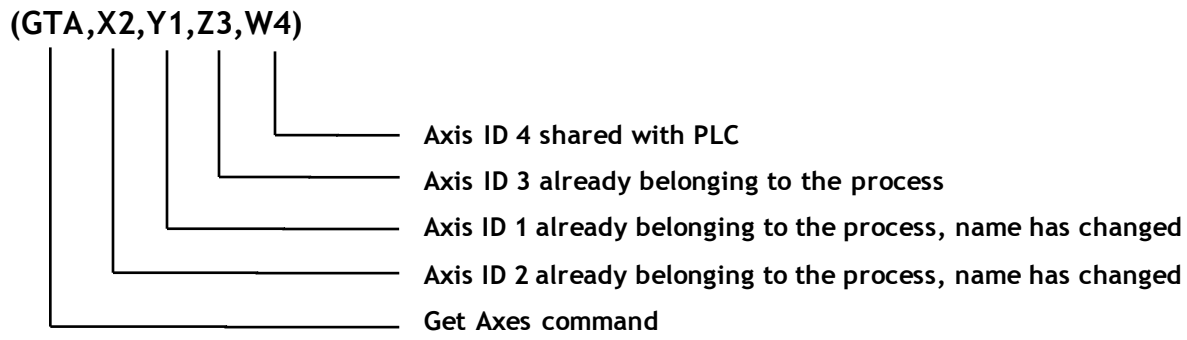
Initial conditions:

- Controlled axes are X,Y,Z axes with ID 1,2,3.
- Axis with ID 4 is “shared with the PLC”.

Goal:

- Retain control over axes with ID 1, 2, 3
- Exchange the abscissa with the ordinate
- Acquire axis with ID 4
- Obtain the following display order (on system video): X, Y, Z, W.

Programming example:



NOTE:
An axis shared with the logic cannot be released through a GTA command (the system gives error message 22/117) and since it belongs to the PLC, even though it is visible from the process environment, it cannot be programmed.
Hence to use the GTA command on a process involving one or more axes shared with PLC, such shared axes must first be reconfirmed as described in the example.

15.27.1 *Error Management*

If an error such as axis resource not available, ID axis not found, etc. occurs during part program execution, the programmer may stop the program and display the error or manage it from the program through the ERR and STE variables.

For further information about these variables, refer to Appendix C.

Example of axes acquisition from a process:

```
;N1 SINUS_AM
;N2 EXECUTION IN PROC.1
;N3 ACQUISITION OF ID9-ID10 (NEW AXES A-B) FROM PROC. 2
N6 (EXE,GTA1_AM,P2)
N7 (DLY,0.1)
N8 (GTA,X1,Y2,Z3,A9,B10,-Z)
N9 T1.1M6
N10 S1000M3
N11 G0G90XYZAB
N12 G1G91G28
N13 E25=0.03
N14 X0.2713Y0.0947Z-0.1852A-0.1815B0.1007tE25
N15  X0.2711Y0.0886Z-0.1887A-0.1776B0.1066tE25
N16 X0.2708Y0.0825Z-0.1922A-0.1736B0.1125tE25
N17 X0.2703Y0.0764Z-0.1955A-0.1696B0.1184tE25
N18 X0.2697Y0.0702Z-0.1987A-0.1654B0.1241tE25
N19 X0.2689Y0.0640Z-0.2017A-0.1610B0.1299tE25
.....
.....
.....
N2010 X0.2697Y0.1241Z-0.1654A-0.1987B0.0702tE25
N2011 X0.2703Y0.1184Z-0.1696A-0.1955B0.0764tE25
N2012 X0.2708Y0.1125Z-0.1736A-0.1922B0.0825tE25
N2013 X0.2711Y0.1066Z-0.1776A-0.1887B0.0886tE25
N2014 X0.2713Y0.1007Z-0.1815A-0.1852B0.0947tE25
N2015 G0
N2016 (SND,P2,S)
N2017 (GTA,X1,Y2,Z3)
N2018 M30
```

16. FILTERS

16.1 General

Nowadays filters are used in the widest variety of applications, e.g., to regularise sets of values, to process data after eliminating a specific portion thereof, to analyse signals by cutting the high frequencies, or in the finite band pre-compensation of control loops.

This chapter describes the various types of filters that can be configured and applied in OPEN*control* units, designed to improve machine tool performance from both the geometric and dynamic point of view and hence the finished quality of the parts produced.

16.2 FLT – Filters programming

The FLT triliteral makes it possible to configure and enable filters that act on the geometry and the dynamic movement of the axes.

The FLT instruction, depending on the mode defined in the command, is used as a preparatory command (preparation of the filter parameters) and as an executive command (activating and deactivating the filter).

Syntax

(FLT, *mode1*, *filter_id*, *name_axis* [,*parameters*])
or
(FLT, *mode2* [,*filter_id* [,*axes_names*]])

---

where:

mode1 is the preparatory mode for the filter:

R : parameter reading; it is possible to read the values of the parameters of a filter previously written with the same triliteral or configured in AMP. Parameters must be numerical variables, otherwise a format error is notified.

Syntax:

(**FLT, R**, *filter_id* [, *axis_name*, *parameters*)

For this mode, the number of parameters programmed must be the same as the number of parameters managed by the filter specified in the three-letters code.

W : parameter writing; configures the filter , by writing the relevant parameters, but does not enable it: in this mode, the parameters can be both variables and numerals.

N.B.: When a filter is configured for the first time, a "filter resource" is allocated, A total of 32 filter resources can be allocated for the entire CNC.

Syntax:

(**FLT, W**, *filter_id*, *parameters*)

(**FLT, W**, *filter_id*, *axis_name*, *parameters*)

For this mode, the number of parameters programmed must be the same as the number of parameters managed by the filter specified in the three-letter code.

S : filter status request; -1 = filter not configured
0 = filter inactive
1 = filter active

The status is stored in the variable specified by the programmed in this command

Syntax:

(**FLT, S**, *filter_id* [, *axis_name*, *parameter*)

mode2 Is the executive mode that can be performed with the filter :

E = enable ; allows the simultaneous activation of one or more of the filters on one or more axes of the process listed in *axes_names*

D = disable; simultaneously deactivates one or all the filters on one or more axes of the process listed in *axes_names*

C = delete ; releases a "filter resource" which is no longer required. This command can be used in the event of a resources exhausted error (-5) during the configuration of a new filter.

Syntax:

(FLT, *mode2*)

(FLT, *mode2*, *filter_id*)

(FLT, *mode2*, *filter_id*, *axes_ names*)

filter_id Identification of the filter. This is required if the filter is programmed in *mode1*. For further information on the value to be inserted in this field, see the paragraph "Types of filters that can be configured"

axis Programmed with *mode1,* defines the axes where the filtering algorithm is applied:

- Id axis is programmed by name, one ASCII character
- If programmed by ID, '/' character followed by an integer

If no axis is specified, the system applies the write command (the only operation permitted) to all process axes.

axes_names Programmed with *mode 2*: it is a list of the names of the process axes to which the filtering algorithm is to be applied.

Axes can be specified by:

- Id axis is programmed by name, one ASCII character
- If programmed by ID, '/' character followed by an integer

parameters To be used only in *mode1* programming, it is a list of the parameters associated with the filter. The number of parameters must be consistent with the number managed by the filter.

Characteristics:

Writing these parameters is permanent, i.e., the original value is only restored at next power-up, not with RESET.

Types of filters that can be configured:

Filter_id	*Filter type* Description	Parameters
1	*Floating average on the command* It is possible, with this type of filter, to average all the points and speeds sent to the servo amplifiers during a configurable number of samples. This function makes it possible to improve the dynamic behaviour of the electromechanical system, in case of programming by points performed with low enough tolerances. By calculating a floating average of the points it is possible to round off the corners of the polygonal trajectory generated by CAD systems	Lower and upper sampling limits **(L).** The total number of points considered for calculation of the average reference value applied to the system will be 2*L+1: current point + L previous points and L subsequent points. The value of **L cannot exceed 32**
2	*Centripetal acceleration compensation* This type of filter can introduce a compensation factor on the programmed position of a physical point. Compensation factor equals acceleration on a point divided by filter gain. An extra contribution is added if VFF is on.	Filter gain (Kv). It has to be a positive value and it is measured in thousands of rpm
3	*Compensation on direction reversal* This type of filter can modify axis behaviour when it changes movement direction	Reversal ramp-up gain (Ks) Reversal ramp-down gain (Kd) Both gains belong to the 0.0-1.0 range. Application amplitude: defines lower and higher position around reversal where filter is applied.

Example 1:
Configure, on the linear axes of the process, a filter that calculates the floating average of the points, based on three samples.

(FLT, W, 1, X , E1) E1 = L = 3

(FLT, W, 1, Y , E1)
(FLT, W, 1, Z , E1) The average will be calculated on 7 reference values: current point + 3 previous and 3 subsequent points

Example 2:
To configure the filter of the previous example on all process axes:

(FLT, W, 1, E1)

Example 3:

enable the filter of example 1 only on the linear axes	=	(FLT, E, 1, XYZ)
	>	
determines the enable status of the filter on the Y, A and B axes	=	(FLT, S, 1,
	>	Y, E2)
after the command execution, E2 = 1, E3 = 0, E4 = -1		(FLT, S, 1, A, E3)
		(FLT, S, 1, B, E4)
enables the filter of example 1 on all the axes in the process	=	(FLT, E, 1)
	>	
disables all filters on all the axes in the process	=	(FLT, D)
	>	

17. TECHNOLOGICAL VARIABLES

17.1 Introduction

The ISO language implemented in the OPEN*control* can access some technological variables belonging both to PLC tables and defined in other functions.
This chapter describes technological variables and how to access them.

17.2 Variables and tables

There are four tables containing technological variables; and values of inputs and outputs and some information stored in PD and PW (axes related) variables can be accessed using technological variables.

The four tables are mainly managed by the PLC (for further details check the WinPLUS Application Manual).

Each table is made up of a given number of pages which contain the parameters that describe the objects of the table. The number of pages for each table is:

Table	N° of pages	Index
Origins Table	100	Origin Identifier
Tool Table	250	Number of the record in table
Tool Offset Table	300	Tool Offset Identifier
User Table	100	User variable number

The table page number corresponds to the index that follows the variable name.
The type, number and symbol of a parameter varies from table to table. The sections that follow illustrate a typical page of each table. For further information about these tables, refer to the "WinPLUS Application Manual".

The LCK instruction enables write protection of WinPLUS tables. More information about the LCK triliteral is provided in the final section of this chapter.

Values read or written in PLC tables are not affected by the current unit of measure (G70/G71) but are considered as absolute values. It should be remembered therefore that any numerical values representing lengths are with reference to the machine's unit of measure configured in AMP; it is up to the operator to perform any conversion required.

17.2.1 *Origins Table*

Origins table can contain up to 100 records, Each record contains data related to an origin. These data are organised into fields.

For each record in the table there is a field with an origin associated with each configured axis.
The number (index) of the field related to an axis corresponds to its physical address as defined in the machine configuration.

Since the system supports up to 64 axes, each record contains 64 Origin fields.

Field	Variable Name	Format	Meaning	Structure
OriginValue	$ORIG	LREAL ARRAY	Value for origin	ARRAY[1…64]
ExType	RESERVED			
ExInde	RESERVED			

17.2.2 *Tools table*

Tools table can contain up to 250 records, Each record contains data about a tool. These data are organised into fields.
Each record manages fields about tool life, current tool status, associated tool offset, 10 SHORT (integer) fields and 10 LREAL (long real) fields. These two fields are reserved for the user.

Field	Variable Name	Format	Meaning	Structure
ToolCode	$TNAME	STRING	Tool name	32 characters
Status	$TSTATUS	UNSIGNED SHORT	Tool status	
LifeType	$TLIFETYPE	SHORT	Type of tool life	
MaxLife	$MAXLIFE	LREAL	Initial life for the tool	
RemLife	$REMLIFE	LREAL	Remaining life for the tool	
OffsNum	$OFFSNUM	SHORT	Number of the associated tool offset	
ExType	RESERVED			
ExInd	RESERVED			
dUser	$TDUSER	LREAL ARRAY	User variables	ARRAY[1...10]
sUser	$TSUSER	SHORT ARRAY	User variables	ARRAY[1...10]

17.2.3 *Tool offset table*

Tool offsets table can contain up to 300 records, i.e. up to 300 tool offsets can be managed, so it is possible to define more than one offset for each tool. Each record contains data about a tool offset. These data are organised into fields.

Each record manages fields about lengths, diameters and orientation of the tool, 10 SHORT (integer) fields and 10 LREAL (long real) fields. These two fields are reserved for the user

Field	Variable Name	Format	Meaning	Structure
Initial Len	$OFFORIGVAL	LREAL	Initial tool length	ARRAY[1…5]
MaxChangeLen	$OFFMAXRQ	LREAL	Max wear for tool length	ARRAY[1…5]
ActChangeLen	$OFFCURRQ	LREAL	Current wear for tool length	ARRAY[1…5]
InitialDim	$DIAORIGVAL	LREAL	Initial tool diameter	ARRAY[1…2]
MaxChangeDiam	$DIAMAXRQ	LREAL	Max wear tool diameter	ARRAY[1…2]
ActChangeLen	$DIACURRQ	LREAL	Current wear for tool diameter	ARRAY[1…2]
Orient	$ORIENT	SHORT	Tool orientation type	
ExType	RESERVED			
ExInd	RESERVED			
dUser	$ODUSER	LREAL ARRAY	User variables	ARRAY[1…10]
sUser	$OSUSER	SHORT ARRAY	User variables	ARRAY[1…10]

17.2.4 *User table*

User table can contain up to 100 records. Each record contains 4 fields of LREAL (long real) type, they are reserved for the user and the system does not access them.

Field	Variable Name	Format	Meaning	Structure
UserVal	$USERVAR	LREAL ARRAY	User variables	ARRAY[1…4]

17.2.5 *Input/output WinPLUS variables*

WinPLUS input/output are organized as 512 WORD variables for a total amount of 8192 I/O signals.

17.2.6 *Variables not in the table*

The following variables do not belong to the tables and contain values related to the axes features. These variables correspond to the data within the Axes table of the S10-OSAI controls.
These variables are read only.

Each variable can be considered as an array of 64 elements, one for each axis on the machine. The index of each element corresponds to the physical identifier of the axis.

Variable Name	Format	Meaning
$AXOWNER	SHORT	Process owning the axis (0 if PLC owns it)
$AXNAME	SHORT	Axis name in numerical ASCII (ASCII code of axis name)
$AXORIG	LREAL	Value of current origin
$AXOFFG92	LREAL	Current G92 offset value
$AXTOFF	LREAL	Current tool offset value
$TOT_OFFS	LREAL	Total offset of the axis

17.2.7 *Table variables access*

There are two ways to access the technological variables according to the variable type (ARRAY OR SINGLE VARIABLE) .

Single variable syntax

varname **(***record***)**

Array variable syntax

varname **(***record***,** *index***)**

where:

varname Technological variable name

record Record number to which the required variable belongs

index Required array variable index

Example:

```
(DIS, $TNAME(6))      ; displays the name of the tool stored in the record 6 of the tools table
$ORIG(45, 3) = -10    ; Set origin value = 45 for the axis ID 3
```

17.2.8 *Input variables access*

There are three syntaxes to access input variables. The first two allow to read/write one bit of an input WORD; the third allows to read/write one WORD of whole input.

Syntax

$I(*toralindex***)**
$I(*word,bit***)**
$IW(*word***)**

where:

totalindex Bit index to read/write is calculated as follows:
Word * 16 + bit index

word WORD to access

index Bit index to read/write within the selected WORD

Examples:

```
E0 = $I(129)   ; E0 =  value of the word 8 bit input
E0 = $I(8 , 1) ; E0 = value of the word 8 bit input
E0 = $IW(12)   ; E0 = value of the word 12 bit input
```

17.2.9 *Output variables access*

There are three syntaxes to access input variables. The first two allow to read/write one bit of an input WORD; the third allows to read/write one WORD of whole input.

Syntax

$O(*totalindex***)**
$O(*word,bit***)**
$OW(*word***)**

where:

totalindex Bit index to read/write is calculated as follows:
Word * 16 + bit index

word WORD to access

index Bit index to read/write within the selected WORD

Examples:

```
$O(129) = E0  ; E0 = bit 1 of 8 output word
$O(8,1) = E0  ; As previous
$OW(12) = 45  ; 12 word output = 45
```

17.2.10 *Synchronized output variables*

At the end of a motion element, a change of status can be set on output variables.
"Standard" outputs or fast outputs type can be changed according to the three-letters code.

Syntax

($OW, *index*, *value*, *mask* [**,** *delay*]
($OF, *value*, *mask* [**,** *delay*]

---

where:

index Output word index containing bits to change at the element movement execution end. It can be expressed whether using an integer from 0 to 511 or an E parameter or a variable.

value is the value word output gets when the previous movement element stops. It can be expressed whether using an integer from 0 to 65535 or an E parameter or a variable.

mask 16 bit mask to identify which bits of the word have to be forced at the end of the movement. It can be expressed whether using an integer from 0 to 65535 or an E parameter or a variable.

delay Optional parameter allowing the introduction of a delay (in milliseconds) between the motion end and the output status change. It can be expressed whether using an unreal number or an E parameter or a variable.

Characteristics:
Value and *mask* can be expressed by an hexadecimal annotation using the prefix Ox or OX before the number.

Examples:
G1X100F100
($OW, 12, 0xF000, 0xFF00)
As soon as X axis reaches the position, bits 8-9-10-11 of the word 12 output will be forced at 0 while bits 12-13-14-15 at 1.

G1X100F100
($OW, 128, 16, 16, 1.5)
1,5 milliseconds after char axis X will be positioned, bit 3 of the word 128 output will be forced at 1.

G1X100F100
($OF, 0x01, 0x03)
When X axis is in position, FastOut1 will be forced at 1 and FastOut2 at 0.

17.3 LCK – Lock/unlock WinPLUS tables

This instruction informs the PLC or other applications that the specified table is being edited and is not accessible to them. After the table has been edited it is necessary to give another LCK in order to indicate that the table is available.

Syntax:

(LCK, *table number*, {**0** | **1**}**)**

---

where:

table number Is a number from 1 to 4 that identifies the write protected table. It can be expressed as a numerical value, a local variable or a system variable with the following meaning:
1 tools table
2 tool offsets table
3 origins table
4 user table

0 | **1** Write 1 to indicate that the specified table is write protected. If the table is already reserved by another user, such as PLC or Table Editor, an error message will be displayed:
23/537 PLC table already locked
This error can be managed from program by setting ERR = 1. For more information refer to Appendix C.
Write 0 to indicate that the table is no longer reserved and can be accessed by other users.

Characteristics
Access to table variables is always granted and they need not be write protected. However, it is recommended that the table is locked to make sure that no other user has access to the memory area currently written. Use LCK to unprotect a table only when it has been write protected from the part program. Otherwise, a table that was reserved for other users (such as the PLC) could be accidently released.

Example:
The following example shows how to write variables into the Tools Table:

```
. . .
. . .
ERR = 1                 enables error management from part program
"LOOP"
(LCK,1,1)
(GTO,LOOP,STE=14)       14 = write protected table; waiting for unlocking command
$OFFSNUM(4) = 12
$TDUSER(1)  = 3.45
$TCODE(2)   = 13
$TSUSER (2) = 6
(LCK,1,0)               unprotected tools table
ERR = 0                 disables error management from part program
. . .
. . .
```

18. APPENDIX A: CHARACTERS and COMMANDS LIST

18.1 Table of characters

CHARACTER	DESCRIPTION
0 to 9	Numbers
+	Addition
-	Subtraction
*	Multiplication
/	Division Block delete (optional block)
.	Decimal Point
"	Label or String Identifier
(	Open Parenthesis
)	Close Parenthesis
[	Open Square Bracket
]	Close Square Bracket
{	Open Braces
}	Close Braces
;	Comment Symbol
,	Parameter Separator
=	Assignment Symbol
<	Less than (jump symbol)
>	Great than (jump symbol)
LF (ISO or ASCII line feed) CR (EIA carriage return)	Block End
#	Synchronization
&	A-synchronization
!	Prefix User variable
@	Prefix PLUS variable
$	Technological variables prefix, TCP or I/O
>>	Increase on single operand
A	Axis name Acceleration on profile or on block (associated to F or f)
B b	Axis name Ramp time (deceleration) on profile or on block (associated to F or f) Bevel in cutter diameter compensation
C	Axis name
D d	Axis name Deceleration on profile or on block (associated to F or f) Tool diameter (RQP,RQT)
E	E type parameter

CHARACTER	DESCRIPTION
F f	Axes feedrate in G1-G2-G3 – G12 – G13 – G14 (feed on profile) Global dynamic characteristics set (on profile) in G1, G2, G3, G12, G13, G14
G	Preparatory Code (G00 - G99) Reserved (G100 - G299) Paramacro subroutines (G300 - G999)
H h	Parameters used in PARAMACROS Offset change during continuous move
I	Coordinates of the arc centre in a circular interpolation (G2 G3- G12 – G13) Variable pitch in G33 Depth increment in deep drilling cycles (G83) Coordinate of the third point abscissa in G14 (spatial circular interpolation for three points)
J	Coordinates of the arc centre in a circular interpolation (G2-G3) (ordinate) Min depth increase in (G83)
K	Coordinates of the arc centre in a circular interpolation (G2-G3-G12-G13) Coordinate of the third point in G14 on the third axis (spatial circular interpolation for three points) Reduction factor for I and J in drilling cycle Threading Pitch (G33) Threading Pitch (G84) Helix pitch in helical interpolation Dejerk on profile or on block (associated to F or f)
L l	variables shared between PLC and process Length of tool offset (RQT, RQP) Length 2 of tool offset (RQT, RQP)
M	Auxiliary functions
N	Part program block number
P	Axis name
Q	Axis name
R r	Rapid positioning in cycles G81 - G89 Deviation from the spindle zero (used in multi-start threads) Radius in a circular interpolation G02-G03 – G12 – G13 Tool compensation joint Theoretical hole radius (G73)
S	Spindle speed
T t	Tool and tool offset address Ramp time (acceleration) on profile or on block (associated to F or f) Time needed to complete the move in one block
U u	Axis name Compensation factors in axis 1 (offset)

CHARACTER	DESCRIPTION
V v	Axis name Compensation factors in axis 2 (offset)
W w	Axis name Compensation factors in axis 3 (offset)
X x	Axis name Orthogonal versor component for circumference or semi-circumference in space (G12-G13)
Y y	Axis name Orthogonal versor component for circumference or semi-circumference in space (G12-G13)
Z z	Axis name Orthogonal versor component for circumference or semi-circumference in space (G12-G13)

18.2 G codes

The table below lists the G codes available on OPEN*control* systems.

G CODE	FUNCTION
G00	Rapid axes positioning
G01	Linear interpolation
G02	Circular interpolation CW
G03	Circular interpolation CCW
G04	Dwell at end of step
G09	Deceleration at end of step
G10	Circular interpolation for three points in space
G12	CCW circular interpolation in space
G13	CW circular interpolation in space
G14	Linear interpolation with dynamic rapid parameters
G16	Defined interpolation plane
G17	Circular interpolation and cutter diameter compensation in the XY plane
G18	Circular interpolation and cutter diameter compensation in the ZX plane
G19	Circular interpolation and cutter diameter compensation in the YZ plane
G27	Continuous sequence operation with automatic speed reduction on corners
G28	Continuous sequence operation without speed reduction on corners
G29	Point-to-point mode
G33	Constant or variable pitch thread
G40	Disables cutter diameter compensation
G41	Cutter diameter compensation - tool left
G42	Cutter diameter compensation - tool right
G70	Programming in inches
G71	Programming in millimetres
G79	Programming referred to machine zero
G80	Disables fixed cycles
G81	Drilling cycle

G82	Spot-facing cycle
G83	Deep hole drilling cycle
G84	Tapping cycle
G85	Reaming cycle
G86	Boring cycle
G89	Boring cycle with dwell
G90	Absolute programming
G91	Incremental programming
G92	Axis pre-setting without mirror
G98	Axis pre-setting with mirror
G93	Inverse time (V/D) feedrate programming
G94	Feedrate programming in ipm or mmpm
G95	Feedrate programming in ipr per revolution or mmpr
G96	Constant surface speed in fpm or mpm
G97	Spindle speed programming in rpm
G72	Point probing with probe ball radius compensation
G73	Hole probing with probe ball radius compensation
G74	Probing for theoretical deviation from point without probe ball radius compensation
G99	Deletes G92

18.3 Mathematical and logical functions

&&	||	^^
ABS	AND	ARC
ARS	ART	ATAN2
BCLR	BSET	BTST
COS	EXP10	EXP2
EXPE	EXOR	INT
LG2	LN	LOG
MOD	NEG	NOT
OR	POW	SIN
SQR	TAN	

18.4 Local and system variables modifiable from a program

VARIABLE NAME	FUNCTION
!name	User Variable
@name	PLUS Variable
$I $IW	Input Variable
$O $OW	Output Variable
ACCUR	Calculation accuracy
ARM	Definition of arc normalisation mode

CEND	Continuity at the spline beginning and end
CET	Precision tolerance in circular interpolation
CRV	Curve speed calculation
CURLEN	Minimum length for curve change
CURMODE	Curve change management mode
CURRAP	Length sections ratio to manage the curve change
DAC	Angle (profile axes)for edge detection
DAV	Angle (rotary axes) for edge detection
DLA	Enables/disables look ahead
DPMODE	Management mode of the edge detected with DAC and DAV
DRP	Drilling size in G83
DSB	Disabled barred blocks
DWT	Dwell time
E	Local Variable
ERF	Maximum form error
ERR	Enables/disables the use of system error
FCT	Paramacro variable
FEEDROT	Feed rate programmed on linear axes only
H	Threshold for complete circle
HC	Paramacro string variables
HD	Tool offset number in paramacro programming
HF	Paramacro flag
HFD	Tool offset loaded flag
HFI	Tool number loaded flag
HFS	ASCII tool code loaded flag
HI	Tool number in paramacro programming
HS	ASCII tool code in paramacro programming
IBSIZE	Activate interferences correction
IDMODE	Internal interruptions management mode
JRK	Acceleration mode when RAMP=1
KVGAIN	Speed reduction constant in circular interpolation
L	Numerical system variables for the exchange of data between process and PLUS
LS	System strings variables
MAXLEN	Max length of the generated splines
MBA	Enables/disable auxiliary functions in Multi Block Retrace
MDA	Maximum deviation angle
MOA	Axes motion options
MOM	Manual motions configuration
MSA	Machine allowance definition
ODH	"Online Debug Help"
PLD	WinPLUS double type variables reading
PLR	WinPLUS word type variables reading
PLS	SW variable reading
PMS	Enables/disables auxiliary function S as paramacro
PMT	Enables/disables auxiliary function T as paramacro
PMM	Enables/disables auxiliary function M as paramacro

RAMP	Movement mode
RAPSTOP	Rapid brake enabling
REM	Return to profile mode
SC	System character
SCRLPAR	Disable paramacro scrolling
SHAPERR	Speed reduction threshold in circular interpolation
SN	System number
SNMT	Null movement threshold on spline axes
SSL	Spindle speed limit
STA	Identifier of system task generating the error
STE	Specific error of system task
TAG	Gain for accelerated tapping
TBS	Precisions associated to the spline axes during machining
TBSGO	Precisions associated to the spline axes during rapid movements
TBV	Precision associated to the spline versors during machining
TBVGO	Precision associated to the spline versors during rapid movements
TCPDEF	Default kinematics identifier
TIM	System timing
TKG	Gain for constant speed tapping
TMA	Axis identifier for rigid tapping
TPA	Threshold angle for TPO activation
TPO	Tool path optimisation
TPT	Threshold for TPO
TRP	Tapping return speed
UPA	Probe abscissa update
UPO	Probe ordinate update
VFF	Velocity Feed Forward

18.5 Three-letter codes

CODE	FUNCTION
AXF	Axes to follow
AXO	Axis offset definition
AXS	Axes shared with the logic
AXT	Configure and activate tangential axis
CLD	Call a subroutine for execution
CLS	Call a subroutine for execution
CLT	Call a subroutine for execution
COF	CYCLE OFF command
CON	CYCLE ON command
DAN	Define axis name
UAO	Use absolute origin
UTO	Use temporary origin
UIO	Use incremental origin
RQO	Requalify origin
SOL	Software over travels
DPA	Define protected areas
PAE	Protected area enable
PAD	Protected area disable
MIR	Mirror machining
ROT	Active plane rotation
SCF	Scale factor
AXO	Axis Offset Definition
RQT	Requalifying Tool Offset
RQP	Requalifying Tool Offset
TOU	Declare a tool out of life
DPP	Defining Probing Parameters
FLT	Programming filters on process axis
TTC	Define cutter compensation mode
UVW	Define axis for paraxial compensation
RPT	Open repetition of a set of program blocks
ERP	Close repetition a set of program blocks
DGM	Scroll disable for current paramacro
PTH	Set path for subroutines and paramacros
EPP	Execution of a part of a program
EPB	Execution of a program block
GTO	Branch Command
IF, ELSE, ENDIF	Conditional Execution of parts of a program
DIS	Display a variable
UDA	Dual axis
SDA	Dual axis
XDA	Master / slave axis
xda	Master/slave axis in continuous
DLY	Cause a delay in program execution
DSB	Disable slashed blocks

REL	De-activate a part-program
WOS	Put the system in hold waiting for a signal
GTA	Axes acquisition
GTS	Spindle sharing
SND	Send a synchronization message
WAI	Wait for a synchronization message
EXE	Automatic activation of a part program
ECM	Execution in MDI mode of a block in a specified process
PRO	Definition of default process
RTP	Reading the number of the programmed or active tool
ROP	Reading the number of the programmed or active tool offset
GPS	Reading the process state, sub-state or mode
RAP	Reading the axis coordinates
DIR	Direction of manual movements
FHO	Enable/Disable FEEDHOLD
HOF	HOLD OFF command
HON	HOLD ON command
JOG	Jog step value
RES	RESET command
SAX	Select axis for manual movement
SMD	SMD - Set operating mode
PLD	Read double PLC variables
PLR	Read word PLC variables
PLS	Read PLC SW variables
LCK	Lock / unlock write access to PLC tables

18.6 ASCII codes

The tables that follow show the 256 elements of the extended ASCII character set, together with Their decimal and hexadecimal equivalents.

DEC	HEX	CHARACTER
000	00	BLANK (NULL) (NULL)
001	01	(SOH)
002	02	(STX)
003	03	(ETX)
004	04	(EOT)
005	05	(ENQ)
006	06	(ACH)
007	07	(BEL)
008	08	(BS)
009	09	(HT)
010	0A	(LF)
011	0B	(VT)
012	0C	(FF)
013	0D	(CR)
014	0E	(SO)
015	0F	(SI)

DEC	HEX	CHARACTER
016	10	(DLE)
017	11	(DC1)
018	12	(DC2)
019	13	(DC3)
020	14	(DC4)
021	15	§ (NAC)
022	16	(SYN)
023	17	(ETB)
024	18	(CAN)
025	19	(EM)
026	1A	(SUB)
027	1B	(ESC)
028	1C	(FS)
029	1D	(GS)
030	1E	(RS)
031	1F	(US)

DEC	HEX	CHARACTER
032	20	BLANK (SPACED)
033	21	!
034	22	"
035	23	#
036	24	$
037	25	%
038	26	&
039	27	'
040	28	(
041	29	)
042	2A	*
043	2B	+
044	2C	,
045	2D	-
046	2E	.
047	2F	/

DEC	HEX	CHARACTER
048	30	0
049	31	1
050	32	2
051	33	3
052	34	4
053	35	5
054	36	6
055	37	7
056	38	8
057	39	9
058	3A	:
059	3B	;
060	3C	<
061	3D	=
062	3E	>
063	3F	?

DEC	HEX	CHARACTER
064	40	@
065	41	A
066	42	B
067	43	C
068	44	D
069	45	E
070	46	F
071	47	G
072	48	H
073	49	I
074	4A	J
075	4B	K
076	4C	L
077	4D	M
078	4E	N
079	4F	O

DEC	HEX	CHARACTER
080	50	P
081	51	Q
082	52	R
083	53	S
084	54	T
085	55	U
086	56	V
087	57	W
088	58	X
089	59	Y
090	5A	Z
091	5B	[
092	5C	\
093	5D	]
094	5E	^
095	5F	_

DEC	HEX	CHARACTER
096	60	`
097	61	a
098	62	b
099	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

DEC	HEX	CHARACTER
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	△

DEC	HEX	CHARACTER	DEC	HEX	CHARACTER	DEC	HEX	CHARACTER	DEC	HEX	CHARACTER
128	80	Ç	144	90	É	160	A0	á	176	B0	▒
129	81	ü	145	91	æ	161	A1	í	177	B1	▒
130	82	é	146	92	Æ	162	A2	ó	178	B2	▒
131	83	â	147	93	ô	163	A3	ú	179	B3	│
132	84	ä	148	94	ö	164	A4	ñ	180	B4	┤
133	85	à	149	95	ò	165	A5	Ñ	181	B5	╡
134	86	å	150	96	û	166	A6	a	182	B6	╢
135	87	ç	151	97	ù	167	A7	o	183	B7	╖
136	88	ê	152	98	ÿ	168	A8	¿	184	B8	╕
137	89	ë	153	99	Ö	169	A9	⌐	185	B9	╣
138	8A	è	154	9A	Ü	170	AA	¬	186	BA	║
139	8B	ï	155	9B	¢	171	AB	½	187	BB	╗
140	8C	î	156	9C	£	172	AC	¼	188	BC	╝
141	8D	ì	157	9D	¥	173	AD	¡	189	BD	╜
142	8E	Ä	158	9E	Pt	174	AE	«	190	BE	╛
143	8F	Å	159	9F	f	175	AF	»	191	BF	┐

DEC	HEX	CHARACTER	DEC	HEX	CHARACTER	DEC	HEX	CHARACTER	DEC	HEX	CHARACTER
192	C0	└	208	D0	╨	224	E0	α	240	F0	≡
193	C1	┴	209	D1	╤	225	E1	β	241	F1	±
194	C2	┬	210	D2	╥	226	E2	Γ	242	F2	≥
195	C3	├	211	D3	╙	227	E3	π	243	F3	≤
196	C4	─	212	D4	╘	228	E4	Σ	244	F4	⌠
197	C5	┼	213	D5	╒	229	E5	σ	245	F5	⌡
198	C6	╞	214	D6	╓	230	E6	µ	246	F6	÷
199	C7	╟	215	D7	╫	231	E7	τ	247	F7	≈
200	C8	╚	216	D8	╪	232	E8	ϕ	248	F8	°
201	C9	╔	217	D9	┘	233	E9	θ	249	F9	∙
202	CA	╩	218	DA	┌	234	EA	Ω	250	FA	·
203	CB	╦	219	DB	█	235	EB	δ	251	FB	√
204	CC	╠	220	DC	▄	236	EC	∞	252	FC	n
205	CD	═	221	DD	▌	237	ED	∅	253	FD	2
206	CE	╬	222	DE	▐	238	EE	∈	254	FE	■
207	CF	╧	223	DF	▀	239	EF	∩	255	FF	BLANK 'FF'

Extended ASCII Character Set

19. APPENDIX B: ERROR MESSAGES

19.1 Description of error messages and remedial actions

19.1.1 *Error generated by PLC – Interpolator library*

Code	Description of error messages and remedial actions
22/001	**Too many Interpolators / processes** Internal error: It shows that the system is trying to create too many processes or movement activities. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/003	**Too many Interpolation Activities** Internal error: It shows that indicate the system is creating too many interpolator activities. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/004	**Too many movements** Internal error: It shows that the system is generating more movement than the maximum value of allowed movement configured in ODM. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/005	**Wrong number of axes** Internal error: System has tried to move too many axes (12 max). No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/006	**Illegal movement (not exist)** Internal error: Requested movement is no longer available. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/007	**Illegal movement** Internal error: A process or an interpolator has tried to use a movement it does not own. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/008	**Illegal Interpolator/Process** Internal error: Requested process or interpolator is no longer available. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/009	**Memory allocation error** During boot phase the system has not enough free memory to load configuration. Please contact Customer Support.
22/010	**Command not allowed in point-by-point mode** Internal error: Requested command is not available in block-by-block mode. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/011	**Command not allowed in Continuous** Internal error: Requested command is not available in continuous mode. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/012	**Program Terminated** Internal error: System is trying to perform a move while the program is stopping due to a reset or an emergency. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/013	**Too many interpolators** Internal error: System is creating too many interpolators. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/014	**Command not allowed in continuous abort** Internal error: System is trying to perform a move while it is stopping a continuous movement due to a reset or an emergency. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/015	**Command not allowed** Internal error: System is trying to perform a move while it is performing another movement. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/016	**Wrong Interpolator/Process PosMode parameter** Internal error: System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the configuration defined using ODM tool. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/017	**Wrong Interpolator/Process type parameter** Internal error: System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the configuration defined using ODM tool. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/018	**Wrong Interpolator/Process ErrMode parameter** Internal error: System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the configuration defined using ODM tool. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/019	**Wrong Interpolator/Process EmgMode parameter** Internal error: System is trying to configure a process or an interpolator in an erroneous way. This error does not depend on the configuration defined using ODM tool. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/020	**Interpolator/Process not found** Internal error: System is trying to execute a movement on a non-existent process or interpolator. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/021	**Interpolator/Process already defined** Internal error: System is trying to generate a process or an interpolator that exists already. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/022	**Interpolator/Process still working** Internal error: Impossible to close an interpolator or a process while it still owing move activities. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/023	**Movement is running** Internal error: Impossible to free a movement while it is still working. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/024	**Start denied** Internal error: The request for a move start as been rejected since the interpolator is currently busy. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/025	**Hold denied** Internal error: The requested interpolator cannot be put in hold since it is currently in error, emergency or reset state. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/026	**Exit from Hold denied** Internal error Requested an 'exit' from hold for an interpolator that is not in hold state. No operator generated action or part program should generate this error. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/027	**Axis out of profile (restarting)** Internal error During an exit from Hold at least one axis is not on the position where it was when it entered in hold state. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/028	**Command not allowed for CNC** Internal error CNC tried to execute a command that is not allowed for the requesting process. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/029	**Wrong FeedHold modality** Internal error: Requested FeedHold mode is not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/030	**Unknown command class** Internal error: Requested execution of a command belonging to an unknown class. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/031	**Unknown command** Internal Error: Requested execution of an unknown command. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/032	**RoundOFF only for simple continuous movements** Internal Error: RoundOFF command requested while system is not in continuous mode. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/033	**Command unaccepted as pre-calculation elements are exhausted** Internal Error: Requested execution of a movement in continuous mode while queue is full. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/034	**Unavailable information** Internal Error: Requested info about a non-existing interpolator or process. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/035	**Spindle is not moving** The spindle is not moving and a threading or tapping operation is programmed. These operations require spindle movement. Check for S command programming and for its activation.
22/036	**Spindle without ramps, function not available** Tapping operation needs acceleration ramps (inversion time different from 0) to be configured for the spindle. In this way reversal can be performed at the bottom of the hole. Check spindle configuration using ODM tool.
22/037	**Tapping Length Error** Tapping length is not long enough to let the axes velocity be synchronised with spindle rotation speed. In this condition tapping operation cannot be executed.
22/038	**Tapping Dynamic Error** Tapping axis cannot respect spindle reversal time and tapping velocity since the configured value for acceleration is too small.
22/039	**Spindle axis cannot be programmed** Programmed axes are not a spindle or it is a master / slave spindle.
22/040	**Axis not available, in use by normal movements** The axis is in use by another interpolator or process.
22/041	**Axis not available, in use by hold movements** The axis is in use by another interpolator or process and it is moving it during the hold state.
22/042	**Axes with different clock** Axes belonging to a movement or in acquisition have different interpolation clocks. They cannot be moved together. Check their configuration using ODM tool.
22/043	**Axis not defined** Requested axis is not configured. Check axis ID.
22/044	**Illegal axis Id** Axis identifier is invalid, meaning that its value is out of range.

Code	Description of error messages and remedial actions
22/045	**Gantry axis cannot be programmed** Id of the requested axis identifies a slave gantry axes. Only master gantry axes can be used. Check axis ID and axis configuration using ODM tool.
22/046	**Axis not available** Internal error: Trying to move (during continuous) an axis that does not belong to interpolator or process. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/047	**Axis already defined in the command** Axis name duplicated in a move command. Check command syntax.
22/048	**Axis not available, in use by Master/Slave** Axis requested belongs to a master / slave operation. It will be available once master / slave mode is complete.
22/049	**Request available only from the Axis Owner** Internal error: Requested axes are not associated with interpolator or do not belong to the requesting process. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/050	**Wrong Start condition for movement** The condition for starting a movement is wrong. Check condition code and its associated parameters.
22/051	**Wrong End condition for movement** The condition for ending a movement is wrong. Check condition code and its associated parameters.
22/052	**Wrong Break condition for movement** The condition for interrupting a movement is wrong. Check condition code and its associated parameters.
22/053	**Movement condition required** Please specify at least a start / stop / break condition for the movement.
22/054	**Signal not available from Interpolator** The signal associated with a start / stop / break condition for the movement is wrong or not compatible with the condition itself.
22/055	**Wrong VelMode parameter** Internal error: In a MoveUntil command the requested execution mode is not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/056	**Wrong Start Continuous modality** Internal error: In the start continuous command the requested execution mode is not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/057	**Wrong End Continuous Tolerance management** Internal error: In the end continuous command the requested execution mode is not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/058	**Wrong ChangePositionLoop modality** Internal error: Requested interlocking mode not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/059	**Wrong Ramp mode** Internal error: Requested ramp mode is not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/060	**Movement past Positive Limit** Programmed position on the axis exceeds the positive limit, the command is not executed. Check command and current over travel limits.
22/061	**Movement past Negative Limit** Programmed position on the axis exceeds the negative limit, the command is not executed. Check command and current over travel limits.
22/062	**Positive Limit reached during movement** A movement reached positive limit, movement has been stopped.
22/063	**Negative Limit reached during movement** A movement reached negative limit, movement has been stopped.
22/064	**Axis on Positive Over travel** Axis is on positive over travel. Movement command cannot be executed. Axis has to be moved in manual into operating range.
22/065	**Axis on Negative Over travel** Axis is on negative over travel. Movement command cannot be executed. Axis has to be moved in manual into operating range.
22/066	**Homing required** A homing command should be executed before current command can be executed.
22/067	**Spindle Axis Required** Request command can be executed only on a spindle axis. Check axis type.
22/068	**Spindle not Defined for Interpolator** Requested spindle does not belong to requesting interpolator or process.
22/069	**Gear Value Out of Range** Internal error: Gear value for spindle is out of range. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/070	**Null feed not allowed** Requested a movement with null feed. Check programmed value for feed.
22/071	**JRK value (S ramp) out of range** Programmed value for JRK (selection of characteristics for acceleration / deceleration with S ramps) is out of range.

Code	Description of error messages and remedial actions
22/072	**Variable not available from Interpolator** The variable used in start / stop /break condition of a movement is wrong or not compatible with the condition itself.
22/075	**Wrong Feed Mode** Internal error: Mode used to define work speed (Feed) is not available in the system. Work feed can use the following measurement units: length (mm or in) per minute, time, 1/time. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/076	**Wrong SLL Value** Value for SLL variable (lowest spindle speed in G96) cannot be negative. Check variable value.
22/077	**Wrong SSL Value** Value for SSL variable (highest spindle speed in G96) cannot be negative. Check variable value.
22/078	**SLL Value Greater Than SSL** SSL variable (highest spindle speed in G96) cannot be less than SLL (lowest spindle speed in G96).Check value of both variables.
22/079	**Function not available** Internal error: Tapping command demands a mode not allowed. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/080	**Wrong Origin number** Internal error: The origin requested for activation does not exist. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/081	**Origin not defined** Internal error: Origin requested for activation is not defined. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/082	**Wrong Probing Safety Distance Value** Safety distance for probing cannot be zero or negative. Check probing parameter value.
22/083	**Wrong Probing Approach Distance Value** Approach distance for probing cannot be zero or negative. Check probing parameter value.
22/084	**Wrong Probing Feed Value** Feed for probing cannot be zero or negative. Check probing parameter value.
22/085	**Probing Executed in Approach** Probing executed during approach phase, outside of expected probing area.
22/086	**Probing not Executed** Probing not performed
22/087	**Probing in Execution** Probing still executing

Code	Description of error messages and remedial actions
22/088	**Probing not Required** Probe gives an erroneous state. Please contact Customer Support.
22/089	**Wrong probe Radius** Value for probe radius cannot be negative. Check probe parameters.
22/090	**Interpolation running: request rejected** The command is rejected because there is an interpolation activity running.
22/092	**Wrong Circle Centre Modality (ARM)** Value for ARM variable is invalid. Check ARM value.
22/093	**Wrong Helix step** Value for helix step is not valid. Check helix programming and related K parameter
22/094	**Wrong Rollover Axis required position** Programmed position for rollover axis is invalid. Position has to be less than axis pitch. To perform multiple rotations select incremental programming for rollover axis. Check rollover axis programming section.
22/095	**Wrong Canned Cycle directions** Programmed positions for a canned cycle require a change in cycle direction. Check canned cycle programming section.
22/096	**Wrong Canned Cycle motion lengths** Programmed positions for a canned cycle require lengths that are inconsistent with cycle characteristics. Check canned cycle programming section.
22/097	**Wrong Homing Direction** The direction of the movement programmed for homing is inconsistent with axis configuration. Check, using ODM tool, homing configuration for axis.
22/098	**Request denied Spindle is orienting** Gear change is not allowed when spindle is orienting.
22/102	**Spindle Operation Requires Transducer** Requested operation needs a spindle with transducer. Check function specification and, using ODM, spindle configuration.
22/103	**Wrong Thread Length** Length for thread operation is wrong and threading cannot be performed. Check threading cycle definition and programming.
22/104	**Wrong Thread Pitch Increment** Pitch increment for thread operation is wrong and threading cannot be performed. Check threading cycle definition and programming.
22/105	**Wrong HandWheel Command** Internal error: Requested an invalid command with hand wheel. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/106	**Wrong HandWheel Scale Value** Internal error: Requested an invalid value for handwheel scale. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/107	**Wrong HandWheel Pitch Value** Internal error: Requested an invalid value for handwheel pitch. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/108	**Wrong HandWheel Refresh Tick Value** Internal error: Requested an invalid value for handwheel refresh tick. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/109	**Wrong Spindle Use/Sharing Modality** Sharing modality for spindle is wrong. Check available sharing modalities for spindle.
22/110	**Spindle not Shared** Spindle is not configured as available for sharing, so sharing request is refused.
22/111	**Spindle Request Denied While Interpolator is Running** Request for sharing is accepted only when process or interpolator are not performing other operations.
22/112	**Spindle Already Used in Exclusive Mode** Request for sharing cannot be accepted since another interpolator or process owns t the spindle in exclusive mode.
22/113	**Wrong Axis Use/Sharing Mode** Request axis sharing with wrong mode. Check available axis sharing.
22/114	**Request Denied While Axis is Moving** It is not possible to release the sharing condition of an axis when it is currently moved, in sharing mode, by another interpolator. Switching to exclusive mode is available only when axis is idle.
22/115	**Axis Already Used in Exclusive Mode** Sharing request cannot be accepted since the axis is currently shared with another process.
22/116	**Axis Request Denied While in use** Sharing request is accepted only when requesting interpolator or process are not performing other actions.
22/117	**Request Denied Shared Axis Active** Request for acquiring / releasing an axis can be satisfied when axis is shared.
22/118	**Axis not Shared** Request for sharing can be accepted only when axis owner makes the axis available for sharing.
22/119	**Wrong Jerk Value** Jerk for movement is negative. Jerk value has to be bigger than zero.
22/120	**Wrong Acceleration Value** Acceleration for movement is negative. Acceleration value has to be bigger than zero.
22/121	**Wrong Deceleration Value** Deceleration for movement is negative. Deceleration value has to be bigger than zero.

Code	Description of error messages and remedial actions
22/122	**Wrong Feed Value** Feed for movement is negative. Feed value has to be bigger than zero.
22/123	**Wrong Negative jerk Value** Deceleration for movement is negative. Deceleration value has to be bigger than zero.
22/124	**Wrong Acceleration Time Value** Acceleration Time for movement is negative. Acceleration Time value has to be bigger than zero.
22/125	**Wrong Deceleration Time Value** Deceleration Time for movement is negative. Deceleration Time value has to be bigger than zero.
22/126	**Wrong Dynamic Override Value** A Wrong Dynamic Override Mode has been required. Only % override or velocity override are available.
22/127	**Too many protected areas.** The number of the protected area requested is higher than the number allowed
22/128	**Protected area undefined** Requested the use of an undefined protected area
22/129	**Wrong protected area use command** Unexpected command requested for the use of a protected area
22/130	**Unavailable protected area use command** The request of a protected area use command is not allowed in that moment.
22/131	**Too many axes programmed in the block** The number of axes required in the motion exceeds the number allowed.
22/150	**Positive Limit** Manual movement ended on positive limit.
22/151	**Negative Limit** Manual movement ended on negative limit.
22/152	**Protected area limit reached: end of the movement** The movement stopped at the limit defined by a protected area.
22/201	**Master/Slave setup mismatch** The axis to be set as a slave is already master of the requesting master. Check Master / Slave order and association.
22/202	**Slave axis required** Requested a slave related operation on an axis that is not defined as a slave.
22/203	**Slave axis still linked with a Master** Requested operation cannot be performed since the slave is still linked to its master. To perform the action the master axis has to release the slave
22/204	**Slave axis commands pending** Master cannot release slave since there are operations active on the slave.
22/205	**Wrong following Mode** The requested following mode is not available. Check available modes in Master / Slave section.

Code	Description of error messages and remedial actions
22/206	**Wrong Master/Slave activation or deactivation** The requested Master / Slave activation or deactivation cannot be performed. Activation / deactivation mode is invalid. . Check available modes in Master / Slave section.
22/207	**Reset position for Master differs from Slave axis position** Internal error: Requested Master / Slave synchronisation depending on master position; all slaves should be associated with the same Master / Slave origin. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/208	**Reset position for Master must be 0** Internal error: Requested a Master / Slave synchronisation depending on master position; no slave should be associated to the any Master / Slave origin. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/300	**Illegal Use of Slave Axis** Master / Slave association refused since slave axes already belongs to a dual axis association (UDA / SDA).
22/210	**Illegal Use of Master Axis** Master / Slave association refused since Master axis does not belong to requesting process.
22/211	**Following axis use denied** The error occurs when › The spindle is defined as following axis › The probe uses an axis set as following axis › A following axis is used for a canned cycle › At least one of the axes in circular interpolation (planar or spatial) is defined as following axis
22/212	**Wrong spline starting position** The starting point of a spline must coincide with the final point of the previous spline. This is an internal notification referring to the PLC programing; if displayed on the screen it is advisable to contact the customer support service.
22/213	**Wrong TCP request** Command referred to the unknown TCP. If displayed on the screen it is advisable to contact the customer assistance.
22/214	**The number of TCP axes required is different from those of the kinematic.** The TCP enabling request has a number of axes different from the number defined in the active kinematic. If displayed on the screen it is advisable to contact the customer assistance.
22/215	**The TCP axes id are different from those of the kinematic** The TCP enabling request has been made with an axes ID different from the one defined by the active kinematic. If displayed on the screen it is advisable to contact the customer assistance.

Code	Description of error messages and remedial actions
22/216	**TCP tool direction axis must be a virtual axis** The TCP activation request is made with non-virtual tool direction axes. If displayed on the screen it is advisable to contact the customer assistance service.
22/217	**Homing unavailable for axes referred to TCP** If TCP is enabled, axes cannot be homed.
22/218	**Unavailable homing for virtual axes** Virtual axes cannot be homed.
22/219	**Virtual axis unavailable** Requested the acquisition of a virtual axis within an interpolator, but the axis is not virtual. This is an internal notification referred to the PLC, if displayed on the screen it is advisable to contact the customer assistance service.
22/220	**Unavailable TCP in continuous when in Realtime Tool Compensation mode** If the real-time offset of the length and of the tool diameter is active, (tcp,… cannot be programmed in continuous.
22/221	**Tool and rotary direction axes (TCP) not available at the same time** The movement of the TCP tool direction axes and of the rotary axes cannot be programmed at the same time. Only manual movements are available.
22/222	**A virtual axis CANNOT be used as Slave axis**
22/300	**Process not defined** Internal error: Request for status of a non-existent process. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/301	**LookAhead already started** Internal error: Request start operation on a running LookAhead. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/302	**LookAhead not started** Internal error: Tried to execute a command on a LookAhead that is not running. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/303	**Cycle start denied** Internal error: Requested a Cycle start command when system is in a state that does not allow this command. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/304	**Cycle stop denied** Internal error: Requested a Cycle stop command when system is in a state that does not allows this command. It is an internal error related to PLC programming, if it appears please contact Customer Support.

Code	Description of error messages and remedial actions
22/305	**Point to Point Movement required into continuous** Internal error: Requested a point-point (G29) movement during a continuous movement. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/306	**Function not available for PLC Process** Internal error: Request for status of a non-existent PLC process. It is an internal error related to PLC programming, if it appears please contact Customer Support.
22/307	**Misalignment on axes number** Internal error: Misalignment error in the internal information (number of axes) during operations in UPR. Contact the customer assistance service.
22/308	**Misalignment on axis ID** Internal error: Misalignment error in the internal information (axes Id) during operations in UPR. Contact the customer assistance service.
22/309	**Unavailable function during UPR** An unavailable operation is requires while there is a UPR command running.
22/6001	**Axis homing complete** End of homing operation with info about micro-marker distance.

19.1.2 *Errors generated by Process library*

Code	Description of error messages and remedial actions
23/001	**Syntax Error** Syntax error found in the part program block or in the MDI block
23/002	**Open file NMC_BINARY.DAT error** The machine configuration file generated by ODM has not been found. Recompile the AMP and reboot.
23/003	**Open file LDIRDATA.DAT error** The default directory configuration file generated by ODM has not been found. Recompile the AMP and reboot.
23/004	**Insufficient memory on process** Memory full. Reduce the number of variables configured in AMP, or increase available RAM memory.
23/005	**Process undefined** The command applies to a process that has not been configured, or it is a value smaller than 1 or greater than 24
23/006	**Format error** This error is displayed in the following cases: › Wrong variable format › Invalid IBSIZE values (allowed values are 0,3,4,5) › Wrong variable index › Feedrate (F) = 0 or negative › Wrong variable format › Repeat number is illegal (number of repetitions must be from 1 to 65535) › Format error in assignment, e.g. assignment to strings with different lengths › PLC variables writing/reading error › Character variable format error in DIS code: not specified as CHAR › Protected area not allowed: 0< protected area number <4 › Variable not configured › Wrong parameter for a command
23/007	**Wrong variable identifier** The index identifying the variable in read/write or assign commands is wrong.

Code	Description of error messages and remedial actions
23/008	**Wrong axis name** The name of the axis specified in the command is not correct. This error is displayed in the following cases: › the axis specified in axis data request functions has not been configured › error in reading axis position, because specified axis does not exist › the axis specified in library calls CncCalcAxOfs, CncAxOfsAct, CncCalcToolOfs, CncToolOfsAct, CncUip, CncSelAxi does not exist › the axis programmed during origin management (pre-setting, UIO, UAO, RQO, etc.) or offset pre-setting has not been configured › the axis specified in the triliterals SCF, MIR has not been configured › the definition of the work plan with G17, G18, G19, G16 is wrong, because the number of axes configured is less than 2 or because the axis specified has not been configured › the name of the axis to be selected, homed, shared with logic or blocked has not been found in the table of the axes concerning that process or has not been configured › the axis specified in the triliterals SOL, DPA, AXO, UVW, FLT has not been configured or has been duplicated. › the axis specified in the commands G92, G98, has not been configured.
23/009	**Too many variables in RD/WR** The number of variables in read/write commands exceeds maximum limit.
23/010	**Wrong axes number in the command** The number of axes specified in the command is not correct. Error is displayed to indicate that the number of axes specified in library command CncGetAxShare/CncRelAxShare is higher than the allowed number
23/011	**Error reading value variable** The name of the variable, specified in the library read/write commands, is wrong, or the variable has not been configured
23/012	**Wrong data into ODM configuration file** Wrong data error.
23/013	**Error in activating file to restore TCP** Internal error: Indicates was not possible to enable writing for the file used in restoring TCP configuration. If displayed on the screen, contact the customer service.
23/014	**Error in generating file to restore TCP** Internal error: Indicates the file used in restoring TCP configuration can't be generated. If displayed on the screen, contact the customer service.
23/015	**Invalid value for the variable** Value exceeds the variable valid range, check if the value is correct.
23/016	**Command unknown** Command to be executed is not correct.
23/017	**Command and system status not congruent** Command cannot be accepted while the system is in current status.

Code	Description of error messages and remedial actions
23/018	**Wrong parameter command** Error displayed when: › Error in type of read positions › Erroneous origin number during origin pre-setting › Wrong axis number or selection type during axis selection › Error in versor definition during thickness cutter compensation › Error on command during Multi Block Retrace or Search in Memory
23/019	**Internal error : nc message error** Switch off system, then reboot. If problem persists please contact Customer Support.
23/20	**RCM not available** Search in memory is not available. To make it available, please set the corresponding flag in AMP.
23/21	**Too many virtual axes disabled** Internal error: Shows that too many virtual axes have been enabled/disabled during the active process. If displayed on the screen, contact the customer service.
23/050	**Wrong number of axes for G code** Error displayed when programming: › G04 with no axes. › More than one axis is programmed with canned cycles (G81 → G89) › No axes or too many axes programmed for probing cycles (G72 → G74) › Only one axis is given as the starting point of a protected area
23/051	**Canned cycle parameters missing** Some parameters needed to perform the canned cycle are missing.
23/052	**Missing parameters for G code** Some parameters needed for G code execution are missing
23/053	**Missing J and/or K for G83 cycle** J and/or K parameter for G83 cycle are missing.
23/054	**Missing I and/or J for G83 cycle** I and/or J parameter for G2/G3 cycle are missing.
23/055	**Probing cycle parameters missing** Some parameters for probing cycle are missing.
23/056	**Undefined symbol** Tried to access a variable that is not configured.
23/057	**Expression Overflow** Mathematical expression is too long (more than 40 characters)
23/058	**Function not allowed** Error appears when requested: › activation of an M code of type **program stop** while program already stopped or when the system is in hold. › an M code configured as paramacro and as an M prelude › modification of Master / Slave parameters, but there are no Master / Slave configured
23/059	**Illegal use of slave axis** A SDA / UDA / XDA command tried to define as Slave an axis that is already a Slave.

Code	Description of error messages and remedial actions
23/060	**Operand not allowed in probing cycle** Operand not allowed for G code G72 / G73 / G74
23/061	**K parameter not allowed in G84** K parameter is no accepted when performing a G84 using a spindle without transducer
23/062	**Wrong programming of G2/G3 code** Programmed a G2 / G2 code using both radius (R parameter) and centre coordinates (I and J). Remove R value or centre coordinates.
23/063	**Illegal number of operands** Wrong number of operands for LOA triliteral
23/064	**Illegal number of pseudo axis** Too many pseudo axes programmed into block: maximum number of pseudo axis is 6
23/065	**Illegal number of axes in G33 code** More than two axis programmed in G33 G code.
23/067	**G not allowed** G not allowed in roughing cycle
23/068	**Operand not allowed with G code** At least one operand is not compatible of the type of current movement
23/069	**Block and system status not consistent** MDI or programmed block is inconsistent with system status, e.g. when the request is: › synchronisation and there are blocks queued for execution (e.g. synchronisation while cutter compensation is active) › axis migration (GTA) when canned cycle, cutter compensation, dual axis or master / slave are running
23/070	**G and program state not consistent** G function is not consistent with system status. For instance: › with cutter compensation active (G41-G42) canned cycle cannot be programmed › threading cannot be programmed when cutter compensation or canned cycles are active › function related to interpolation plane change (G16-G17-G18-G19) cannot be programmed when cutter compensation is active
23/071	**Movement type not consistent** Current program status and movement type are not consistent, for instance a threading cycle (G33) cannot be programmed when cutter compensation is active
23/072	**Programming type not consistent** Programming type (absolute, incremental or G79) is not consistent with current programming status
23/073	**Current programming not consistent** Error in updating programming status

Code	Description of error messages and remedial actions
23/074	**G and dynamic mode not consistent** Requested G function is not consistent with current dynamic mode: › functions G72, G73, G74 are not accepted while in continuous (G27, G28) › it is not possible to change from G27 to G28 (and vice-versa) when non-linear ramp (RAMP > 1) is active
23/075	**G41/G42 and part program status not consistent** Cutter compensation is not consistent with current part program status (e.g. during canned or probing cycles)
23/076	**G needs spindle with transducer** G33 and macro cycle FIL need a spindle with transducer
23/077	**G not consistent with feedrate mode** Probing cycles (G72, G73, G74) cannot be programmed when G94 is active
23/078	**Operand and part program status not consistent** At least one operand is not consistent with current program status.
23/079	**PLC operand and part program status not consistent** PLC operand and current part program status are not consistent
23/080	**PLC operand and dynamic mode not consistent** PLC not consistent with current dynamic mode: › M prelude/postlude not compatible with G code G27/G28 (continuous) › T programmed when cutter compensation (G41/G42) is active
23/081	**PLC operand and motion type not consistent** PLC operand non consistent with movement type. For instance: G33 and M postlude
23/082	**Probing cycle operands inhibited** Some operands are not allowed in a probing cycle: › With G73 the following operands are not allowed: I,J,K,R,u,v,w,b,t › With G72 the following operands are not allowed: I,J,K,R,u,v,w,b,t,r
23/083	**Missing third axis for helix** Third axis for helix has not been programmed.
23/084	**"Expedite" function without motion** An M function of type “expedite” has been programmed without any associated movement. M functions of type “expedite” always need an associated movement.
23/085	**Feed not programmed** Missing feed for a canned cycle.
23/086	**Speed not programmed** Missing spindle feed for a canned cycle
23/087	**Read only variable** TIM, STE and STA variable are read only. They cannot be written.
23/091	**Nesting of IF greater than 32** Exceeded maximum nesting of IF command.
23/092	**ELSE not allowed** Found an ELSE command with no matching IF

Code	Description of error messages and remedial actions
23/093	**ENDIF not allowed** Found an ENDIF command with no matching IF
23/094	**Part program record too long (>127 char)** Current part program block is too long
23/095	**Part program access denied** Requested part program cannot be accessed since it is currently used by another user
23/096	**Program already selected** A paramacro, called in MDI, is trying to call itself
23/097	**Part Program name too long (>127 char)** Requested part program name (together with its path) is too long
23/098	**DRIVE not found** Specified logical drive was not configure in ODM or in file browser
23/099	**Path specified not found** Path specified for part program does not exist.
23/100	**File not found** Part program / routine / paramacro not found
23/101	**Illegal file name** Filename provided to load table (LOA command) is not correct
23/102	**File access error** System cannot load table
23/103	**Error during part program file handling**
23/104	**Part program activation failed** Command SPG failed
23/105	**Part program deactivation failed** Part program deactivation failed
23/106	**Part program not selected** Error triggers when no part program has been activated and user: › tries to modify a block in MDI › pushes Cycle Start button, CNC in AUTO or BLK/BLK
23/107	**Tool direction and plane compensation not consistent** During cutter compensation tool direction and plane are inconsistent
23/108	**Duplicated axes in UDA/SDA command** An axis ID/name is duplicated in an UDA/SDA command
23/109	**Use of illegal axis in safety area definition** In a part program defining a safety area the only allowed axis names are X and Y
23/110	**Triliterals not allowed in safety area definition** Part programs defining a safety area cannot contain triliterals
23/111	**M code not allowed** M code type not allowed in continuous
23/112	**Diameter or length duplicated in RQT/RQP** In a block containing triliterals RQT or RQP lengths (or diameters) having same index have been specified. Check if the block does not contain lengths (or diameters) having the same index.
23/113	**G not consistent with Spline status** G code programmed is not consistent with the Spline programming

Code	Description of error messages and remedial actions
23/114	**PLC operands not consistent with Spline status** PLC operand programmed are not consistent with Spline programming
23/150	**Illegal argument for TAN** TAN operator argument is 90 degrees or an odd multiple thereof: the result would be infinite.
23/151	**Illegal argument for SQR** The argument of the SQR operator (square root) is a negative number
23/152	**Too many programmed axes** More than 12 axes have been programmed in the block or the assignment of an external axes during a UDA/SDA/XDA command failed
23/153	**Division by zero** A division by zero has been detected in the expression that calculates an axis dimension (e.g. X10/0)
23/154	**String too long** The max. string length is 127 characters.
23/155	**Label duplicated** This message is displayed when the program is selected or activated. It shows that there are two identical labels in the part program
23/156	**Undefined label** The label programmed in a branch instruction (GTO) or in a call for a subroutine (EPP) does not exist
23/157	**Label too long** This message is displayed when the system reads an SPG block. It indicates that a label having more than 6 characters has been programmed
23/158	**Program table overflow** This message is displayed when the program is selected or activated. It indicates that the number of subroutines programmed overflows the maximum configured in AMP. This parameter can be modified in AMP using ODM tool.
23/159	**Label table overflow** This message is displayed when the program is selected. It indicates that the number of labels programmed overflows the maximum configured in AMP. parameter can be modified in AMP using ODM tool.
23/160	**End of program** End of file marker for: › block skipping › block editing › string search › program execution
23/161	**LF not found (0x0a) at the end of program** End of file (LF) code has not been found. Open program with an editor, go to end of last line and press ENTER.
23/162	**Beginning of program** Program start marker for: › block skipping › string search

Code	Description of error messages and remedial actions
23/163	**Nesting of RPT greater than 5** RPT max. nesting level (5) exceeded
23/164	**Nesting of subroutine greater than 4** Subroutine max. nesting level (4) exceeded
23/165	**Nesting of EPP greater than 5** EPP max. nesting level (5) overflow
23/166	**RPT/ERP cycle open at end of file** This message is displayed when are reached the end of the file, of the paramacro or of the routine execution: › without finding the (ERP) block that closes the programmed (RPT) cycle › without finishing the part program part defined with EPP › without finding the ENDIF block closing the IF block or ELSE in execution
23/167	**ERP without RPT** (ERP) has been programmed without previously programming (RPT)
23/168	**Modal Paramacro already active** A paramacro is programmed when a modal paramacro is already active
23/169	**Paramacro not configured** The programmed paramacro has not been configured in AMP
23/173	**Too many elements in safety area definition** Safety areas should be defined using no more than 10 (closed areas) or 9 (open areas) elements
23/174	**Safety area could not be closed (elements overflow)** An open safety area has been defined using 10 elements, it cannot be closed (too many elements).
23/175	**Active safety area cannot be redefined** Safety areas can be redefined only if not active.
23/176	**Error in limits for 3D safety area** Higher and lower limits for 3D safety area do not belong to the same axis. Check limits definition
23/177	**XMLrules.txt not loaded. XML commands not executed** XML paramacros cannot be executed since no mapping file has been loaded. Provide a valid mapping file (XMLrules.txt)
23/178	**Syntax error in XMLrules.txt** Syntax error in mapping file
23/179	**High Speed Machining not installed** High Speed Machining functions (Spline) are unavailable on the machine.
23/180	**Unaccepted interpolation in HSM machining**
23/181	**High Speed Machining functions disabled by ODM** High Speed Machining functions are disabled in the active AMP. Change enabling flag value in AMP.
23/182	**XML correlation table updating** XML correlation table sharing error: the part program tried to enter the XML correlation table during the uploading. Check the operations timing.

Code	Description of error messages and remedial actions
23/250	**Axis not referenced** This message occurs when: › The programmed axis is not referenced › The offset to be pre-set/re-qualified is associated with a non-referenced axis
23/251	**Undefined DPP for probing cycle** Probing cycle parameters (approach coordinate, safety distance, velocity) are not defined in the DPP block
23/252	**Too many "Expedite" M codes** More than one expedite M code has been programmed in the block
23/253	**Undefined M code** Programmed M code is not configured in AMP. Configure the M in AMP using ODM tool and restart the system
23/254	**Circle not consistent** The circle is not geometrically consistent: radius or final point are not correct
23/255	**Wrong threading parameters (I, K, R)** The programmed threading parameters (I, K and R) are not allowed. Calculate the I parameter with the following formula: $\frac{16\,k}{2(\text{threading distance})}$
23/256	**Helix pitch incongruent** The helix pitch is not geometrically correct
23/257	**Axes of plane needs same scale factor** Plane axes in G02/G03 programming (circle) must have the same scale factor. Change the scale factor with an SCF instruction
23/258	**Geometry incongruent** ISO profile programming is wrong.
23/259	**Profile inconsistent** Profile is no longer continuous after cutter compensation
23/260	**Wrong direction on profile** Offset value in G41-G42 reverses the tool path direction
23/261	**Error disabling cutter compensation** Error exiting from cutter compensation (G40)
23/262	**Transit buffer is full** Error is displayed if: › too many extra plane moves programmed with cutter diameter compensation active (G41-G42) › when interference correction is on (IBSIZE different from 0), too many elements need to be skipped.
23/263	**Too many blocks to unlock** Buffer is full. Message is displayed on the monitor, please call customer assistance.
23/264	**Entry in safety zone** The programmed move enters one of the active safety areas
23/265	**Canned cycle on rotated plane** Canned cycle programmed on rotated plane Disable plane rotation

Code	Description of error messages and remedial actions
23/266	**Canned cycle length data incongruent** The parameters for hole depth, approach dimension, return after machining are not consistent during a canned cycle.
23/267	**Canned cycle data incongruent** The parameters specified in the canned cycle (I, J, K, R) are not allowed. For example: canned cycle K = 0 A G84 or a G86 cycle is being performed with the spindle in non-exclusive status.
23/268	**Canned cycle not allowed**
23/269	**Wrong probing cycle programming** This message appears when: › probing approach distance is null › hole probing is programmed with null radius (for example G73r0E5)
23/272	**Axes not on profile** This message appears when trying to quit CYCLE STOP (HOLD) after a series of jog moves without taking the axes back to the profile. Select JOG RETURN and return the axes to the profile
23/273	**Error in exit HOLD: mode changed** This message occurs when trying to exit from HOLD by setting an operating mode that is different from the one in which the system went on HOLD. Select the right mode and retry.
23/274	**Block not allowed in HOLD** This message occurs when trying to execute an MDI block that is not allowed in HOLD
23/276	**Wrong use of roll-over axis with G90** The programmed coordinate for the axis with rollover in G90 is greater than the roll over pitch configured in AMP
23/277	**Virtual axes programming not allowed** Temporary or virtual axis origin has been programmed or virtual axis has been defined as linear axis for a TCP configuration.
23/278	**Real axes or manual motions not allowed with tool direction active** Zero micro for a virtual axis has been executed
23/279	**Command not allowed during search in memory** Command not allowed during the search in memory
23/280	**Geometric out buffer full** Geometric out buffer is full, programmed profile and cutter dimension generate too many interferences.
23/281	**Axis shared with PLC** Axis shared with PLC cannot be moved in jog mode or returned on profile
23/282	**Safety area is not convex** Defined safety area is not convex. Safety areas must be convex, please check area definition
23/283	**Safety area not defined** Cannot activate requested safety area because it is not defined.
23/284	**Translation of axis not on safety area** Trying to translate a safety area on an axis that does not belong to the area

Code	Description of error messages and remedial actions
23/285	**Axis type not consistent for axis tangential control** At least one of the axis needed by axis tangential control is not correct. Two linear axis and one rotary axis are needed.
23/286	**Automatic return to profile not allowed after MDI in HOLD** Automatic return to profile is not allowed after the execution of an MDI command in HOLD state
23/287	**Radius on concave profile** During cutter compensation radii (r operand) can be programmed on convex angles only.
23/288	**Tool Centre Point functions not enabled by ODM** The Tool Centre Point functions are nor enabled in the active AMP. Change the enabling flag value in AMP.
23/289	**Impossible to set several offsets on an axis** In a block containing an AXO type triliteral, the same axis has been programmed more than once. Correct the block
23/290	**XDA required without master / slave defined** Trying to execute a XDA-xda command without a previous master/slave association. Execute the association using the (XDA,1 command.
23/291	**Real axes programmed during the virtualization** Real axes belonging to an active virtualization (UVA, UVP, UVC) have been programmed. Disable the virtualization before programming the axis.
23/350	**Mode to select out of range** This message occurs when the selected mode is out of range. Allowed modes are in the 1-8 range: 1. MDI 2. AUTO 3. BLOCK by BLOCK 4. CONTINUOUS JOG 5. INCREMENTAL JOG 6. RETURN ON PROFILE *7.* HOMING FILE *8.* HPG (Hand Wheel)
23/351	**Axes number to select out of range** The error triggers in two cases: › The number of axes selected with MIR function is higher than the number of axes available to the process › The number of axes selected for manual moves with library call NC_SELAXI is out of range. The ***allowed range*** is from 1 to the number of axes configured for the process
23/352	**Too many axes selected for manual move** A larger number of axis names than accepted have been inserted in the part program block. Edit the part program block.

Code	Description of error messages and remedial actions
23/353	**Bad select mode for cycle** This error is displayed when CYCLE START is pressed in the following conditions: › a mode other than MDI has been selected during execution of a tool change axis move › system on HOLD, AUTO or BLK/BLK with MBR (multi-block retrace) not configured in AMP › system on HOLD with MBR active and selected mode other than AUTO or BLK/BLK › system in IDLE and ACTIVE_RESET with selected mode other than AUTO or BLK/BLK › system in IDLE with MBR active and selected mode other than AUTO or BLK/BLK › system in HRUN with MBR active and selected mode other than AUTO or BLK/BLK › ACTIVE RESET command in HOLD status with selected mode other than MDI, AUTO or BLK/BLK. **NOTE:** For further information about the machine status (HOLD, MDI, HRUN, etc.) refer to the USER GUIDE.
23/354	**Data length out of range** Error displays if: MDI block length has overflowed. Error on library call for axis sharing activation
23/355	**Operative limit definition wrong** Error in defining the software operating limits with the three-letter mnemonic SOL.
23/356	**Offset length not defined for the axis** This message occurs when trying to pre-set or to re-qualify an offset that is not associated with the specified axis.
23/357	**Tool orientation code wrong** The specified tool orientation code is wrong.
23/358	**Error from servo environment** Internal error: If it appears contact Customer Support
23/359	**Manual movement not executed , no axes configured** Manual movements are not allowed because no axes have been configured
23/360	**Axis not configured for GTA/GTS/TCP** The id programmed in the GTA, GTS or TCP three-letter code is not configured A non-spindle axis has been programmed in the GTS three-letter code
23/362	**Axis or spindle unavailable** Axis programmed for the PLC sharing is not available. Requested axis is unavailable.
23/363	**Axis id duplicated** The axis ID is duplicated in the GTA block

Code	Description of error messages and remedial actions
23/364	**Requested ID identifies a spindle** The ID programmed in the GTA block corresponds to a spindle axis and is not allowed
23/365	**Requested too many tools** Requested more than 64 tools in a multi-tool T address
23/368	**Axis ID out of range** Id of the axis to be acquired / released is out of range (1 → 64)
23/369	**Offset length assigned on multiple axis** Offset length cannot be shared by multiple axis.
23/370	**Offset length out of range** Index for offset length is out of range (1 → 5)
23/371	**Duplicated axis ID** Axis ID is duplicated in PLC command AxisGet
23/400	**Set spindle speed failed** PLC does not accept the variation of spindle speed.
23/401	**New tool request failed** PLC does not accept the programmed T code.
23/402	**M executed failed** PLC does not accept the programmed M code
23/403	**Pseudo axes programming failed** PLC does not accept the programmed pseudo axes.
23/404	**Axis motion inhibited** Axis motion denied by PLC
23/405	**End of move failed** PLC answers with error at the end of the move request
23/406	**Too many blocks without motion in continuous mode** Too many blocks without motion have been programmed in continuous mode
23/450	**Axis on profile** This message indicates that RETURN TO PROFILE has successfully terminated and the axis has returned to the profile.
23/451	**End of automatic return to profile** This message indicates that automatic RETURN TO PROFILE has successfully terminated and all the axis have returned to the profile.
23/452	**End of multi block retrace** This message occurs with multi-block retrace. To retrace a greater number of blocks, alter the configured maximum
23/453	**Stored search end** Stored search execution successfully finished.
23/455	**End of search in memory** Search in memory complete
23/500	**Current process number programmed** The process number specified for synchronisation commands is the current process
23/502	**Circles/lines not defined**
23/503	**Canned cycle not possible with shared spindle** G84 or G86 cycle has been programmed using a shared spindle.

Code	Description of error messages and remedial actions
23/504	**Spindle required with G84 or G86** Programmed canned cycle needs spindle, but no spindle is available for the process
23/505	**Dry Run already activated** Requested Dry Run activation while it was already on
23/506	**Requested process does not exist** A multi-process command tries to communicate with a process that is not defined. Check process number.
23/508	**Queue is full on target process** The process queue (local or remote) that a message was sent to is full.
23/509	**Data sent too long** Data to be transmitted with SND are longer than 576 characters
23/510	**Data loading failed** The type or number of data transmitted with SND is not allowed
23/511	**Message already exists in queue** A SND command has been given before the process cleared the previous message.
23/512	**EXE or ECM failed** This message occurs when: › The status of the process to which the EXE or ECM command is sent does not allow automatic part program execution commands (RUN, HRUN, RUNH, HOLD) or an MDI instruction. › There is a syntax error in the program to which the EXE command is addressed
23/513	**Channel already in use** The channel specified in the triliteral OPN is already referred to a file. Change the channel number or close the channel in use.
23/553	**Wrong UPR programming** › Rotary offsets programmed using UPR 0 or UPR 1. › Axes in the global virtualization active is inconsistent with axes specified for the UPR update. › Axes can't be changed during a "continuous" virtualization › "Continuous" virtualization can't change its type › Parameters virtualization can't be modified in "continuous" when a virtualization is inactive. › An incremental virtualization can't be activated with axes different from the absolute virtualization active.
23/554	**Unaccepted incremental UPR** Trying to program the incremental type Cartesian rotation (UPR) without previously defining an absolute rotation. Before programming the incremental rotation, define the absolute Cartesian tern rotation.
23/555	**Dual axes and polar virtualization not consistent** Dual axes (UDA) and polar virtualization (UVP) functions cannot be applied to the same axes at the same time: a master or a dual slave axis cannot be part of a virtualization at the same time.

Code	Description of error messages and remedial actions
22/556	**Dual axes and UVA virtualization not consistent** Dual axes (UDA) and non-orthogonal axes virtualization functions cannot be applied to the same axes at the same time: a master or a dual slave axis cannot be part of a virtualization at the same time.
23/557	**Invalid minimum value of the radius** During the cylindrical virtualization programming (UVC), the minimum radius required is higher than the present position of the linear axis of the virtualization. Check the programmed radius value or shift the linear axis.
23/558	**Dual axes and cylindrical virtualization not consistent** Dual axes (UDA) and cylindrical virtualization (UVC) functions cannot be applied to the same axes at the same time: a master or a dual slave axis cannot be part of a virtualization at the same time.
23/561	**Slave axes in TCP configuration** One or more axes configured are slave. Slave axes can't be TCP axes.
23/563	**Error in getting/releasing axis** GTA command is not allowed when cutter offset or canned cycle are active
23/562	**Error in the TCP configuration parameters** › The kinematics type does not exist › The rotary axes direction is null Rotary axes not set in the TCP
23/565	**Null module for versors required** › In case of versors kinematics (type 2 kinematics), **i,j,k** versors were not specified or the resulting module is null. › In case of radius compensation by versors, versors orthogonal to the interpolation plane are not specified..
23/566	**Wrong versors programming** › A versors only block has been programmed without any axes motions. › Versors have been programmed on a null length movement
23/568	**Versors p,q,d specified without versors m,n,o** On the block, versors **p,q,d**, have been programmed without **m,n,o** versors.
23/569	**Kinematics ID required to enable TCP** Kinematic ID to enables has not been specified. The value of the kinematics to activate must be specified in TCP or by the TCPDEF variable.
23/570	**Tool direction axis name already in use** The name chosen for the tool direction axis corresponds to a name already in use in the process.
23/571	**Function unavailable** Function required is not implemented by the control. Please, contact the customer service for additional information.
23/572	**Required TCP parameters change with not consistent TCP type** In continuous, it has been asked a TCP parameters change with a TCP type different from the TCP type active. .Make the call with a TCP consistent with the active type.

Code	Description of error messages and remedial actions
23/573	**Virtual axis impossible to be added** Current process already controls the maximum number of axes, therefore the virtual axis can't be enabled. If possible, the number of axes managed by the current process should be reduced.
23/600	**Missing kinematics id** Internal error: In the TCP configuration file created by ODM the kinematics id is missing. It is an internal notification, if displayed on video, please contact the customer service.
23/601	**Index for missing TCP parameter** Internal error: An index is missing in one or more TCP configuration parameters in the configuration file created by ODM. It is an internal notification, if displayed on video, please contact the customer service.
23/602	**TCP name missing** Internal error: One or more TCP parameters is missing in the configuration file created by ODM. It is an internal notification, if displayed on video, please contact the customer service.
23/603	**Unknown TCP parameter** Internal error: One or more parameters in the configuration file created by ODM have not been recognized. It is an internal notification, if displayed on video, please contact the customer service.
23/604	**Wrong name of the tool direction axis** Internal error: Name of one of the axes related to the tool direction is wrong. It is an internal notification, if displayed on video, please contact the customer service.

19.1.3 *Errors generated by geometrical library*

Code	Description of error messages and remedial actions
26/001	**Circles/lines not intersecting** No intersection found for the circle / lines. Check that initial and final point and radius or centre of the circular movement are correct.
26/005	**Parallel lines** The requested contour between two geometrical elements cannot be generated. Contour can belong to an explicit command (r operand) or generated by path optimization during cutter compensation (TPO = 33).
26/006	**Wrong definition of circles/lines** Internal
26/007	**Aligned points** The point measured during the measure of the parameters of an hole (G73) do not define a circle
26/010	**Different radius between initial-final points** Radius of circular movements on starting point differs from the one computed on final point. Check centre and starting and ending point of the circular movement.
26/011	**Circle data incorrect** Normalisation of circular movement failed. . Check centre and starting and ending points of the circular movement. Verify tolerance for variance of circle points (CET) and normalisation mode (ARM).
26/012	**Perpendicular vector not defined** A circumference/ semi-circumference has been programmed in space using G12 (G13) code without defining the vector perpendicular to the plane where the circumference lies. Define this vector using **x**, **y**, and **z**.

20. APPENDIX C: ERROR MANAGEMENT FROM PART PROGRAM

20.1 General

This Appendix explains how the operator can manage errors in order to prevent process machining interruptions.

The following system variables permit to configure the error management mode:

SYSTEM VARIABLE	FUNCTION
ERR	Enables error management from part program
STE	Contains an error, internal to the environment, generated by a previous command. (if automatic error management is active (ERR=1)). *It is a read only system variable.*
STA	Contains the identifier of the system environment that has generated the error (with automatic error management (ERR=1)). *It is a read only system variable.*

20.2 ERR – Enable/disable error management from part program

ERR is a system variable that selects how error are managed.

Syntax

ERR = *value*

where:

value It may be 0 or 1 and can be expressed with a number or a local or system variable.
If it is **0** (default) it disables error management. Errors are displayed and can interrupt program execution.
If it is **1** it enables error management by the part program.

Characteristics:
If ERR=1 the management of errors from the program is enabled.
At power on ERR=0.

The following pages describe:

- The numerical codes identify the various environments (the value of variable STA) and their meanings.
- The programming commands that may generate one of the errors managed by part programs.
- The number of the 23/xxx error that is displayed if ERR=0. (See appendix B).

For the errors specific to the environments different from process environment, see the errors manual.

20.3 Environment identifier numerical codes: STA

Prefix	STA variable value		Environment generating the error
	Decimal	Hexadecimal	
IPC	17	0x11	Operating System
CNS	18	0x12	Console
EMG	19	0x13	Emergency
IO	20	0x14	IO management
PLC	21	0x15	Machine logic (PLC)
GMC	22	0x16	Interpolators-movement
CNC	23	0x17	Process
OSW	24	0x18	OSWire
TBL	25	0x19	Tables
GEO	26	0x1A	Geometric library
LOD	31	0x1F	Loader
SER	32	0x20	SERVO
SEROSW	33	0x21	SERVO OSWIRE

20.4 Errors during probe cycle

Comma	STA	STE	Description	Error displayed if ERR=0
G72,G73,G74	0	0	probe cycle completed correctly	------
	22	1	probing cycle has not taken place	22/86
	22	2	probe has not been released	22/87
	23	3	probe parameters not specified	23/251
	22	4	probe operated during rapid approach cycle	22/85

Example:

Presence of the part is verified by managing errors in the part program:

```
(DPP,30,15,500)
G0 Z50
X80 Y100
ERR = 1
G72  Z0  E1
(GTO,END,STE = 1)
ERR = 0
.....
.....
"END" (DIS, "PART NOT PRESENT")
M...
```

Since G72,G73,G74 modify the value written in the STE variable when ERR=1, it is recommend that STE is used immediately after giving a command that might alter its contents.

20.5 Errors acquiring / releasing axes

Command	STA	STE	Description	Error code displayed if ERR=0
GTA	0	0	execution without errors	------
	23	10	axis ID not configured	23/360
	23	11	axis ID belongs to logic	23/361
	23	12	axis ID belongs to another process	23/362
	23	13	axis ID associated with spindle axis	23/364

Example:

```
;. . . . .
;. . . . .
ERR = 1
;. . . . .
"RETRY"
(GTA,X1,Y2,Z5)          ;Requests acquisition of ID's 1,2,5
(GTO,NEXT,STE<>12)      ;test to see if the axes are still associated with another process
(DLY,0.5)               ;If so the program waits until the axes are released
(GTO,RETRY)
"NEXT"
(GTO,ERROR,STE<>0)      ;If any other error occurs the cycle will be aborted
G1 X10 Y10 F1000        ;After the axes have been acquired the move is executed
Z50
;. . . . .
;. . . . .
;. . . . .
"ERROR"
;. . . . .               ;Error management
```

Error management from part program also provides an easy way of synchronizing two processes when one of them must wait for one or more axes to be available before executing a given process. Since GTA modifies the value written in the STE variable when ERR=1, we recommend that you use STE immediately after giving the command that might change its contents.

20.6 Errors in multiprocess management

Command	STA	STE	STE VARIABLE VALUE	Error code displayed if ERR=0
EXE,ECM	0	0	execution without errors	------
	23	1	EXE or ECM command failed	23/512
	23	3	Wrong process number	23/506

Example:
Verify the correct execution of the synchronous command on process 2 using the error management method:

```
ERR = 1
(ECM,"G1 X100 F1000",P2,S)
(GTO, ERROR,STE <> 0)
ERR = 0
.....
.....
;. . . . .
"ERROR"
;. . . . .                    ; Management of error cases
```

20.7 Error in locking a table

Command:	STE VARIABLE VALUE		Error code displayed if ERR=0
LCK	0	execution without errors	------
	14	Locking failed	23/537

LCK programming example with error management:

```
;. . . . .
;. . . . .
ERR = 1
;. . . . .
"RETRY"
(LCK,1,1)
(GTO,NEXT,STE<>14)          ;Test if table already locked.
(DLY,0.5)                   ; Wait
(GTO,RETRY)
"NEXT"
```

Contacts:
PRIMA ELECTRO S.p.A.

Strada Carignano, 48/2

Moncalieri (TO) - ITALY
Tel. +39 0119899800
Web: www.primaelectro.com
e-mail: sales@primaelectro.com