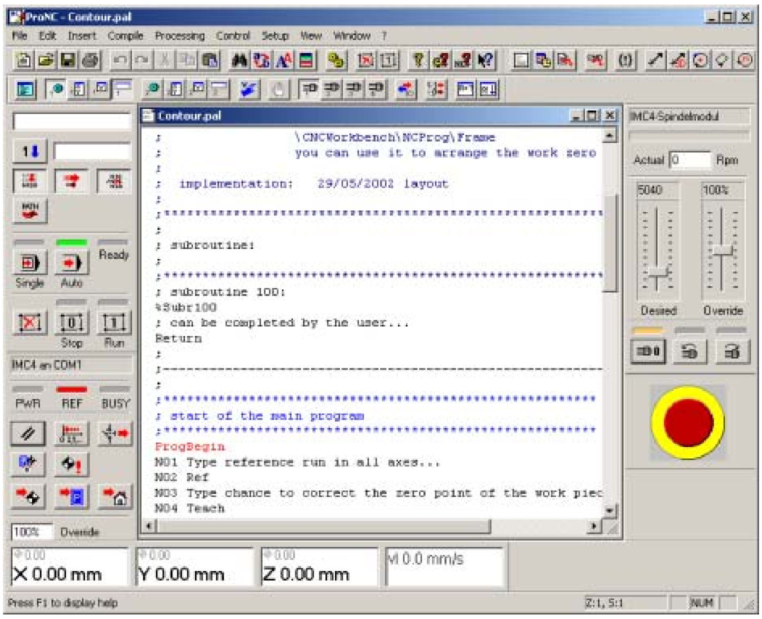
isel automation KG



isel-ProNC

Programming Instruction

isel®

www.isel.com

To the Manual:

In this manual you find same symbols pointing out your attention to important information.

Caution:	**Example:**	**Hint:**	**Information:**
!	☞	✏	✍

In spite of all care printing errors and mistakes cannot be ruled out completely.
Suggestions for improvement and notes on errors are always welcomed.

Producer: ***isel***automation KG
Bürgermeister-Ebert-Straße 40
D-36124 Eichenzell

Tel.: (06659) 981-0
Fax: (06659) 981-777
email: automation@isel.com
http://www.isel.com

Version: 10/2003

Contents

1 Introduction

1 Introduction

Objective of this manual: The documentation for the program-package ProNC should ease the entry into the use of this extensive application. The manual should also help to explain the programming features (motion instructions, input-/output operation, Teach-In, parameter calculation, subprogram technology, arithmetical and Boolean expressions, functions) integrated in ProNC.
The objective is to decrease the time for programming and start up by realizing customer specific tasks at processing (cutting processing, welding-/, water jet cutting-/ burning and sticking technologies) in the handling- / assembly areas.

1.1 Copyright

Copyright: All rights to the manual and programming package ProNC, especially the copyright are owned by

isel-automation KG
Bürgermeister-Ebert-Straße 40
D - 36124 Eichenzell

Copying and transfer the software: All unauthorised copying, distribution or transfer of this software is strictly forbidden and will be prosecuted criminally.

The legal use and distribution of ProNC: The purchase of the installation-CD or – disks including the manuals allows the using comparable to that of a book. With a book, it is not possible for a book to be read at different places by several people at the same time. Similarly, the program-package ProNC may not be used on different controllers (PC-based) at different places by several people simultaneously.

Backup copies: Backup copies of the program-package Pro-NC are permitted. In no case it is allowed to make those copies available to third parties.

Compensation: At violations against the copyright the purchaser engages opposite the owner of the copyright, ***isel***-automation KG, to damage compensation.

Liability for application programs: The program-package ProNC as well as the manual was produced with carefulness. All application programs printed or as example programs on CD, were tested by a corresponding hardware. iselautomation does not assume any liability or guarantee, that this manual, the program-package ProNC or application programs are perfectly or suitable for a particular purpose. For consequence detriments each legal responsibility or liability is ruled out.

!

Suggestions: Since mistakes and errors cannot be ruled out, we are always grateful for suggestions, written notes or opinions.
email: tech-support@isel.com

1.2 Dedication of the program-package ProNC

1.2.1 Short characteristic

Dedication of ProNC: The software product ProNC integrates an operator surface according to the SAA standard and a programming platform for implementation and test/debugging of application programs for CNC controlled machines/plants.
These NC application programs corresponds to the ISO syntax (G-code-programming to DIN 66025) or the PAL syntax (Programming Assembly Language).
Functional extensions were carried out to the standardized syntax of the ISO 1834/DIN 66025.

Support for start-up: Special attention was put to an efficient support to start up during the test-phase. Therefore commands were implemented in ProNC known from a debugger.

- **Display of process- and real-variables in real-time**
- **Manipulation of process- and real-variables in real-time**
- **Activation / deactivation of all input- and output operation as well as of the spindle controlling with help of dummy-functions**
- **Program animation**
- **Teach-In / efficient frame-management and manipulation**
- **Single-step-mode**
- **Program execution to break point**
- **Activation / deactivation of breakpoints**

In automatic mode the program test is supported by occasion of activating break points on any NC-sets (program lines in an application program) as well as the possibility of manipulation of current values of R-variables (data type: floating point).

direct Teach-In: The Teach-In can happen **directly,** if no self-retaining gears exist in the cinematic chain. There the axis with current-free motors will be moved with hand to the desired position. The gained actual position vector (joint coordinates at non Cartesian Kinematics) will be stored in a geometry file (frame file) after a transformation as a cinematic independent data configuration.

indirect Teach-In: When using indirect Teach-In the Tool Center Point (TCP) will be moved to the desired target position / target orientation with help of the mouse or function keys in a complex dialogue-window or with help of the isel-Operating Panel.

Within the hierarchy of the isel-Control-Software under Windows 98 / NT/ 2000 / XP the

operating- and programming surface ProNC applies on a software platform which almost exclusive consists of **D**ynamic **L**ink **L**ibraries (**DLLs**).
These device-driver-DLLs realize primarily motion control modules (MCTL), input-/output-modules (IO), spindle modules (SPN) and tool changer modules (TCH).

Control module: (device-DLLs)

☞

All control module delivered by ***isel***automation KG are independent software products with separate documentations.
ProNC manages in the current implementation:

- **Two** motion control modules (**MCTL) with at most each 6 axes** (motion generation, i.e. interpolation and generation of a velocity profile / slope and motion implementation, i.e. e.g. at DC/AC-servo-drive digital feedback position control)
- **Four input**- and output modules **(IO) with at most 4 input- 4 output ports (each 32 inputs / 32 outputs)**
- **Four** spindle modules (**SPN**)
- **Two** tool change modules **(TCH) with max. 128 tools**

Basic control module draft:

☞

All **control module of a device type** have the same assignation interface and an approximate same functionality. This brings the following advantages for the user:

1. With the investment to the operating- and programming surface ProNC both plants with stepper motors and plants/kinematics with DC/AC-servo motors can be operated and programmed, if the corresponding control module (**MCTL-DLLs**) is provided for the motion control.

2. Plants with max. 4 spindles can be programmed.

3. The created application programs can be exchanged or transmitted between different plants. All plant specific details (e.g. pitch, gear reduction, velocities, acceleration, port addresses and others) are specified in the special initialisation file of each **control module**. It is not necessary to make any changes in the source program.

1.2.2 Principles of programming with ProNC

ProNC was implemented as a component of the isel-control-software for machines / plants with up to **12 axes** (**2 axes systems with each max. 6 axes**).

ProNC is generally executable under Windows 98/Windows NT/Windows 2000/Windows XP. However it is possible that certain control modules (device-DLLs**) can be used only** under **Windows 98** or **Windows NT/2000/XP.**

ProNC is the portation of the hitherto only under MS-DOS running control software Remote, Pro-DIN and Pro-PAL. All user programs, created and used under MS-DOS in NCP-format (from Remote), ISO-format (from Pro-DIN) or PAL-format (from Pro-PAL) are usable furthermore.

ProNC enables both the programming in ISO-/ DIN format and in PAL-format. In section 3 of this manual the syntax is always represented comparatively. That means, that after "ISO: " always follows a NC set/command according to ISO syntax or that after "PAL:" always follows a NC set/command according to PAL syntax.
The technology-oriented syntax of the DIN 66025 (G- and M-instructions) was supplemented

with problem oriented constructions for the structured programming, to parameter calculation as well as to the access on geometry files (geometry file = frame file) and was defined as a flexible, efficient programming standard. This programming standard is described as grammar in section 3 of this manual.
The program package **ProNC** replaces the program packages **Remote**, **Pro-DIN** and **Pro-PAL**. According to the philosophy of these „predecessor programs“ the technological parameters (pitch / gearing, reference velocity, software end switch, switching level etc.) are not defined in the source program in a so called declaration part, but the parameter will be defined in the machine data set / machine parameter file (general in the initialisation file of the motion control module).

Advantage:

Initialisation file of the motion control module

For example the gearing is edited or changed merely once at the configuration of a plant with the help of a dialog windows into the initialisation file of the motion control module, if the available ball screw will be replaced with a spindle with another pitch.
The advantage consists in the fact, that the source programs are always portable, i.e. technological details are always "hidden" in the configuration file (initialisation file) of the motion control module.

In principle, two types of program lines are distinguished in ProNC:

A program line can be::

- a **NC-set** (especially defined in DIN 66025), e.g.: **G1 X100 Y150 Z-50**
- an instruction (not defined in DIN 66025), e.g.: **While R1 > 0.0**

please refer to: Operating Instruction: 5.8.7 Menu Setup - Control

ProNC is based on the experience that with the norms DIN 66025 in Germany or ISO/DIS 6983/1 worldwide it is committed, how numerical controlled machines can be programmed. The ISO syntax is optimised to technological requests. In the ISO-Syntax are used only letters of the Latin alphabet to identification activities and parameters.
The PAL syntax based on the ISO syntax with the characteristic, that the compact G- and M-commands are replaced by mnemonic codes (mnemonics):
In the following example the PAL syntax ***MOVEABS*** corresponds to the ISO syntax G90 G1:

ISO: N10 ***G90 G1*** X100 Y200 Z-50
PAL: N10 ***MOVEABS*** X100 Y200 Z-50

ProNC-program structure:

A **ProNC program** consists of **NC sets** and / or **instructions**. All NC-sets (or short sets) consist of words, frequently also named as commands.
Every word / every command starts with the so-called address letter, followed by a numeric string, with or without sign as well as with or without decimal point.

NC-sets in an ISO-program can contain G-commands (e.g. G1) and / or M-commands (e.g. M3).

NC-sets in a PAL-program can contain mnemonic commands (e.g. MOVEABS or CLW).

Examples for commands	ISO-syntax	PAL-syntax
motion commands (absolute declaration of target)	*G90 G1*	*MOVEABS*
switch on/get up the spindle to target speed	*M3*	*SCLW*

The essential difference gets obvious:

1.A Program line structure by ISO syntax:

Use of G- and M-commands:

Using programming with ISO-syntax motion commands (G-commands), velocities (F-command), miscellaneous commands (M-commands) and other commands can combined and each command type can be written in a program line multiple.

At ISO programming commands are used exclusive with a leading capital letter (address letter).

1.B Program line structure at PAL-syntax:

Use of mnemonic commands:

At programming with Pal syntax mnemonic commands are used exclusive as motion commands and miscellaneous commands.

Advantage of programming in ProNC:

Using programming with ProNC you will get very compact and regular programs. These programs have internationally gained acceptance and proved themselves in the practice (at application of the ISO syntax), primarily at programming of numerically controlled tool machines.

Modality:

Modality means, that a specific value (coordinate, velocity or motion command) is valid in a program context, as long as the value will be defined newly.
A specified coordinate (motion target) is valid as long, as a new coordinate instruction will be made. For programming that means: Within a motion set you have to write only the coordinates, which shall cause a (absolute or relative) **movement** in the concerned set.

Modality at ProNC:

At programming with ISO- or with PAL-syntax **motion commands** (ISO: G-commands, PAL: path commands) and also **coordinate words** (e.g. X, Y, Z, U, V, W, A, B or C) work modal:

The NC-set

ISO: N001 **G90 G1** X100 Y200 Z300
e.g.
PAL: N001 **MOVEABS** X100 Y200 Z300

defines with help of the G-commands **G90 G1** e.g. with help of the PAL-motion command **MOVEABS** a linear interpolation. This definition of the interpolation type is modal e.g. „self holding". So that this interpolation type is also valid in the following set:

ISO: / PAL:
N002 X150 Y250 Z350

It has not be fixed explicitly.

please refer to: Section 3: ProNC language description

Preview chapter 2:

Chapter 2 of this manual contains the most important rules of ISO- e.g. PAL-syntax.

Preview chapter 3:

In **chapter 3 you find the complete** language description of ProNC (**ISO-syntax** compared with **PAL-syntax**). It represents the most extensive chapter of this manual.

Preview chapter 4:

The integration of geometry information in the application program and the access to geometry data (**frames**) during program execution is described in **chapter 4.**

Preview chapter 5:

This chapter describes simple application programs, which can be tested on each plant with at least two axis.

2 Programming with PRONC

2 Programming with PRONC

The constructions in this chapter refer to the application of the ISO-syntax and also to the PAL- syntax.
What is the difference between ISO-syntax and PAL-syntax ?
The difference consists solely in the substitution of G- and M-instructions of ISO-syntax with mnemonic instructions (mnemonic path instruction and mnemonic miscellaneous instructions) at PAL-syntax:

Commands by ...	**ISO-syntax:** *G- and M-commands*	**PAL-syntax:** mnemonic commands
path commands (fix target absolute)	*G90 G1*	*MOVEABS*
command to switch on the spindle	*M3*	*SCLW* *SPINDLE ON*

Hint:

If there is not any equivalent to an ISO-command, it is allowed to use a command in PAL-syntax.
If the program contains a command in ISO-syntax, it must be declared as ISO-program.

ProNC programming:

At ProNC programming all **instructions** in **ISO-syntax** are identical with all **instructions** in **PAL-syntax**.
That means, a FOR loop has always the same syntax, but the NC sets inside a FOR loop have to be defined always either in ISO-syntax or in PAL-syntax.

please refer to: Section 3.2 Instructions

2.1 Structure of the application program

Components of an application program

An application program consists always of a main program none, one or several subprograms. Subprograms will be declared in front of the main program.

2.1.1 Program structure (main program)

Main program:

A main program consists of a sequence of NC sets and/or instructions. The first and the last NC set of the main program are prescribed absolutely.

The follow table shows the simple structure of a main program:

characteristic	**syntactic identifikation by ISO-syntax**	**syntactic identification by PAL-syntax**
first set	marked with the special sign **%** example: **%123**	marked with the mnemonic PROGBEGIN example: **PROGBEGIN**
sequence of sets, forming the real program body	example: **N0 G74** **N1 G1 X100 Y200 Z300** **N2 X200 Y300 Z400**	example: **N0 REF** **N1 MOVEABS X100 Y200 Z300** **N2 X200 Y300 Z400**
last set	marked with the miscellaneous function **M30**	marked with the miscellaneous function **PROGEND**

Table 2.1.1: Structure of a main program

Identification of program beginning:

To indicate the program beginning you have to use the special sign **%** or the mnemonic **PROGBEGIN.** Previous to these special signs subprograms or arbitrarily many comments can be included.

please refer to:
Section 2.1.2 Annotations in the program
Section 2.5 Subprograms

2.1.2 Annotations in the program (comments)

Comments in an application program increase documentation good and relieve so the program test and program maintenance. Four kinds of comments are distinguished in ProNC:

- Comments extending over several lines have to start according to the ISO-syntax with the special sign **(** and they must end with the special sign **)**, according to PAL-syntax you have to use **{** respectively **}.**
- Comments, which shall be separators, have also to start according to the ISO-syntax with the special sign **(** and they must end with the special sign **)**, according to PAL-syntax you have to use **{** respectively **}.**

Comments in round (ISO) respectively curly {PAL} brackets:

Comments, included in round respectively curly brackets, are always filtered by the compiler from the source file and they will not be taken over into the CNC file.
Therefore the CNC file gets more compact.

A comment in round respectively curly brackets can apply to arbitrarily many lines. In ProNC comments can include all signs of ASCII-sign stock (so also the signs % and :) in contrast to the reduction in DIN 66025.

ISO-syntax: **(** this is a comment **)**
PAL-syntax: **{** this is a comment **}**

A comment can extend over several lines. This has a big advantage:

When starting the processing some program sections can be "commented out". That means any long sequence of NC sets will become comments by writing brackets. At processing these sets will be read over as one or more „empty sets" and so they will be ignored.

• Comments, extending over a complete line until the end of line, must begin with a semicolon **;** and must end with the end of line character CR = Carriage Return (ENTER-key code).

Comments over a complete line:

The compiler does **not** filter these comments out of the source file, if the comment filter (compiler option) is switched off. Then you will find these comments in the user program.

The semicolon „**;**" to identify a comment over a complete line must be written in the first column of the comment line.

<u>example:</u>

; this is a comment from column 1 to end of line

• Comments, finishing a NC set, start with a semicolon „**;**" after the last character of the NC set and end with the end of line character CR.

Comments as the end of a NC-set:

Comments as the end of a NC set the compiler will always filter out of the source file.

<u>example:</u>

N10 G1 G91 X100 ; relative motion 100 mm in the X-axis

2.2 Structure of a NC set

2.2.1 Elements of the NC set and variable set length

NC set:

A NC set consists of several commands (also called command words or only words). The first character of a set is <u>always</u> a capital letter. The initial letter of a command/word is also called *address letter*.

Word as synonym for **command:**

In usage of NC programming the synonym command will be often used for word. That means, that path commands can be marked for example G-words as well as G-commands.

Programming in **ISO-syntax G- and M-commands** will be used to define path commands and miscellaneous commands.

Programming in **PAL-Syntax mnemonic commands** will be used to define path commands and miscellaneous commands.

The special capital letter (address letter), introducing every word of the NC set, gives the word a "name":

ISO-syntax:

address letter	word / command	meaning
N	**N**-word = **N**-command	**set number**
G	**G**-word = **G**-command	**path command**
M	**M**-word = **M**-command	**miscellaneous command**
E	**E**-word **=** **E**-command	**rapid feed**
F	**F**-word **=** **F**-command	**processing feed**
T	**T**-word **=** **T**-command	**tool number**
S	**S**-word **=** **S**-command	**spindle speed**

Table 2.2.1: Selection of important words in NC sets (ISO-syntax)

PAL-syntax:

address letter	word / command	meaning
N	**N**-word = **N**-command	**set number**
	mnemonic command, e.g. MOVEABS	**path command**
	mnemonic miscellaneous command, e.g. **SETBIT**	**miscellaneous command**
F	**F**-word **=** **F**-command	**feed**
T	**T**-word **=** **T**-command	**tool number**
S	**S**-word **=** **S**-command	**spindle speed**

Table 2.2.2: Selection of important words (mnemonic commands) in NC sets (PAL-syntax)

Separators: The commands/words of a set are separated with separators. The following separators are allowed in ProNC:

- one or several blank characters
- one or several tabulators
- combination of blank characters and tabulators
- a comment

Length of a NC set: According to the possibility, that the number of words in a set are not dictated, the length of a NC set is variable.

Valid NC sets:

; reference run:
ISO: N1 G74
PAL: N1 REF

; relative path coordinates:
ISO: N2 G1 G91 X100.0 Y200.1 Z300.234 F200.23
PAL: N2 MOVEREL X100.0 Y200.1 Z300.234 F200.23

; absolute path coordinates, spindle speed in [rpm] and spindle on:
ISO: N3 G1 G90 X100.0 S15000 M3
PAL: N3 MOVEABS X100.0 S15000 SCLW

Modality: Viewing the length of a NC set the modality becomes noticeable. That means, all path commands (**ISO:** G-commands, **PAL:** mnemonic path commands), defined in the set n and also valid in the set n+1, you don`t have to define in set n+1 explicitly:

Modality in the NC set: The path command **G1 G90 | MOVEABS** (linear interpolation, absolute path coordinate) is defined in set N001 and will be effective in set N002. Only beginning with the set N003 the use of the path command **G91 | MOVEREL** make the relative path instruction effective.

ISO:
```
N001 G1 G90   X100 Y200
N002          X150 Y250
N003 G1 G91   X10 Y20
```

PAL:
```
N001 MOVEABS X100 Y200
N002         X150 Y250
N003 MOVEREL  X10 Y20
```

please refer to: Section 2.3 Structure of a word

2.2.2 Annotations in the set, comments

A comment is interpreted as separator, if it is enclosed in round brackets (ISO-syntax) respectively curved brackets {PAL-syntax}. Therefore a comment can also be defined between two words.

Comments as separators: valid NC set with comments as separator:

ISO:
N10 G1 X100 Y200 Z300 (velocity) F1000

PAL:
N10 MOVEABS X100 Y200 Z300 {velocity} F1000

2.2.3 Sequence and replay of commands / words in the NC set

The order of the individual words in a set is specified, how described in the following table:

syntax	1	2	3	4	5	6	7
	N-word	**ISO: G-command PAL: mnemonic command**	coordinate-words: **X/Y/Z-U/V/W-A/B/C-word**	**I-word J-word K-word**	**F-word**	**S word**	**ISO: M-command PAL: mnemonic command**
	set-number	path-condition	target coordinates	interpo-lation parameter	feed	spindle speed	miscellaneous function
ISO:	**N**100	**G91 G**2	**X**100	**I**50	**F**75	**S**10000	**M**111
PAL:	**N**100	**CWREL**	**X**100	**I**50	**F**75	**S**10000	**SETB A1.1**

In ProNC it is allowed, to write several path commands (G-commands) and several miscellaneous commands (M-commands) in one set.

2.2.4 Leave out of words

In the norm DIN 66025 the modality is described as follows:

Modality: “A word, which does not change in its effect in several consecutive sets of a user program, has to be defined only once and can be left out in all following sets, for which it shall be valid unchanged."

2.3 Structure of a word

A word according to the DIN- / ISO-norm consists of an address letter, followed by a number (in the DIN norm the name „numeric string“ is used):

Words: G99 is a valid G-word (path command).
N88 is a valid N-word (set number).

GG_100 is an invalid word.
N?88 is an invalid word.

Natural or decimal numbers: The number can be a **natural** or a **decimal** number. There is an absolute assignment of natural respectively decimal numbers to the address letters:
At D-, G-, L-, M-, N-, S- or T-commands a natural number follows always the address letter.
The decimal number can be signed. Do you use a positive number the sign can be left:

+1.0 is identical with **1.0**

2.3.1 Address letters

Address letters: The word is obviously specified in his meaning by address letters.

According to the 26 letters of the Latin alphabet 26 several DIN/ISO words are possible (**sd**: **s**igned **d**igit):

Address-letter	**DIN 66025**	**ProNC**	**assigned number**
A	rotary motion around the X-axis	rotary motion around the X-axis	**decimal number**
B	rotary motion around the Y-axis	rotary motion around the Y-axis	**decimal number**
C	rotary motion around the Z-axis	rotary motion around the Z-axis	**decimal number**
D	tool correction memory	not used	**natural**
E	fast feed	fast feed	**decimal number**
F	processing feed	processing feed	**decimal number**
G	path command	**path command**	**natural**
H	not used	not used	
I	interpolation parameter to X-axis	interpolation parameter to X-axis	**decimal number**
J	interpolation parameter to Y-axis	interpolation parameter to Y-axis	**decimal number**
K	interpolation parameter to Z-axis	interpolation parameter to Z-axis	**decimal number**
L	available	**identification of subprogram**	**natural**
M	miscellaneous command	**miscellaneous command**	**natural**
N	set number	**set number**	**natural**
O	not used	not used	
P	parameter for special calculations	identification of **P-variable**	**natural**
Q	parameter for special calculations	identification of **Q-variable**	**natural**
R	parameter for special calculations	identification of **R-variable**	**natural**
S	spindle speed	**spindle speed**	**natural**

T	tool	tool number	**natural**
U,V,W	second motion parallel to X,Y,Z-axis	reserved to ProNC version for 2 * 9 = 18 axes	**decimal number**
X	motion in direction of X-axis	motion in direction of X-axis	**decimal number**
Y	motion in direction of Y-axis	motion in direction of Y-axis	**decimal number**
Z	motion in direction of Z-axis	motion in direction of Z-axis	**decimal number**

Table 2.3.1: Address letters and its meaning by DIN 66025 and in ProNC

2.3.2 Numeric string with decimal point

In ProNC the oppression both leading and following zeros is permitted. (DIN 66025: numeric strings with explicit decimal point).
So that the demand of DIN 66025 part 1 is fulfilled.

Valid decimal number: X300. is equivalent to the coordinate declaration X300.0
Y.3 is equivalent to the coordinate declaration Y0.3

Compiler: ☞ The compiler (for ISO- or PAL-syntax) writes always decimal numbers with a leading zero into the CNC-file. If a decimal point is defined explicitly the compiler writes three digits after decimal point.

2.3.3 Numeric string without decimal point

Using G-words, L-words, M-words, N-words and P/Q/R-words (variable), only numeric strings without decimal points (**natural numbers**) will be demanded.

Advantage at ProNC: ☞ For all address letters (coordinates, interpolation parameters, feed and others) supported in ProNC is valid: whole numbers (i.e. numeric strings without decimal point) are accepted generally according to DIN 66025. That means, for a coordinate statement X100 you do not have to write X100.0.

2.3.4 Set number: N-word

Set number: ☞ The natural number following the address letter N indicates the set number of a NC set. In ProNC no conditions are made relating to the number. That means, a certain number can appear as often as you like. It is not necessary to number in an ascending order. It is convenient, to write the set numbers at the first program design with a difference from 5 to 10 into the source file. To correct it later, you can insert NC sets into the relevant positions in the file.
In front of instructions to control the program process you do not have to write set numbers.
please refer to:
Section 3.2.3 Assignments to control the program process

Set skip: In the application program any NC set can be suppressed during the processing, if the sign „/" is written in front of the N-word of these set.
The set skip can be activated with the operator panel or with the

function in the Menu processing (display oriented operation).

<u>example:</u>
The following NC set in ISO-syntax will be skipped during the processing, if the **set skip** is activated:

/ N10 S1=1000 M3 ; NC set with optional processing

Program test and set number:

It is pointed out, that the program test happens always line-based. That means, a break-point applies always on a certain program line and never on a certain set number. Because a program line 100 exists only once, but a set number 100 can exist n-times in ISO- or PAL-source program.

please refer to:
Operating Instruction: 5.6.3 Menu Processing - Set skip

2.3.5 Path commands

ISO-syntax: G-commands

The number, following the address letter **G**, is a natural number and is described by DIN 66025 part 1 as index number. The path commands in ProNC (G-words respectively G-commands) will be extensively introduced in section 3.1.1 path commands.

PAL-syntax: mnemonic path-commands

All available mnemonic path commands in ProNC contain a corresponding G-command or a combination of G-commands. The table in part 3.1.1 Path commands explains the comparison.

please refer to: Section 3.1.1 Path commands

2.3.6 Coordinates

The letters **X, Y, Z, A, B, C, U, V** and **W** are reserved as address letter for coordinate words in ProNC. With that nine numerical axes per axis system can be addressed. After one of the listed address letters a whole number or a decimal number can follow.

Axis systems in ProNC:

ProNC can manage two axis systems.

Each axis system can contain maximum 6 numerical axes X,Y, Z, A, B und C in the current version of ProNC.

If you program only one axis system, you have not to differ the coordinate words.
Programming the **two** axis systems
- axis system 1
- axis system 2

in one user program, you must have the possibility to differ between the X-coordinate word of the first axis system and the X-coordinate word of the second axis system. This differentiation happens by indexing of the address letters:
- **Coordinate words in axis system 1:**
  X**decimal number** *or* X1=**decimal number**
  Y**decimal number** *or* Y1=**decimal number**

Z**decimal number** *or* Z1=**decimal number**
A**decimal number** *or* A1=**decimal number**
B**decimal number** *or* B1=**decimal number**
C**decimal number** *or* C1=**decimal number**

- **Coordinate words in axis system 2:**
  X2=**decimal number**
  Y2=**decimal number**
  Z2=**decimal number**
  A2=**decimal number**
  B2=**decimal number**
  C2=**decimal number**

Please note always the following sequence:
1. address letter X, Y, Z, A, B, C
2. index of the axis system (1 or 2)
3. equals sign `"="
4. the decimal number (or an arithmetical term).

The name of axis (allocation of address letters to coordinate words) to the numerical axis in the mechanical system is adapted to the norm **DIN 66025** and **VDI 2861**:

At Tool Machine Controls six translatory (X,Y, Z, U, V, W) and three rotatory axes (A, B, C) are defined.

Tool machine control system: (DIN 66025, DIN 66217)		
translatory axes (linear axes): • main axis:	**X- , Y- and Z-axis build a right-handed coordinate system. The Z-axis is identical with the axis of the spindle. The positive direction of the Z-axis run from the workpiece to the tool.**	
• auxiliary axes:	**U-axis parallel to X-axis** **V-axis parallel to Y-axis** **W-axis parallel to Z-axis**	} reserved for ProNC with 9 axes per axis system
rotatory axes :	**A-axis turns around the X-axis how a right-hand-helix.** **B-axis turns around the Y-axis how a right-hand-helix.** **C-axis turns around the Z-axis how a right-hand-helix.**	

Table 2.3.6: Name of axis by DIN 66025 (Tool machine controls)

The decimal numbers, following the coordinate words immediately, represents absolute values / absolute measurement (path command **G90 | ABS** - self-holding) or relative values / incremental dimension (path command **G91 | REL** - self-holding).
The unit of a translatory axis is **mm** (path command **G71 | METRIC**- self holding) or **INCH** (path command **G70 | INCH** - self-holding).

The unit of a rotatory axis is always **grad**.

2.3.7 Miscellaneous commands: M-word

A number following the address letter **M** is a natural number and is called also as index number according to DIN 66025 part 1. The M-commands (ISO-syntax: M-commands, PAL-syntax: mnemonic commands) realising in ProNC will be described detailed in section 3.1.2.

please refer to: Section 3.1.2 Miscellaneous commands

2.4 Special signs

In accordance with DIN 66025 respectively in addition to above-mentioned all allowed special signs in ProNC are summarized in the following table:

special signs	meaning
% ISO PAL	% natural number : Start of **main program** %L natural number: Start of **subprogram** %SUBR natural number: Start of **subprogram**
ISO: (PAL: {	**start of comments**, if comment shall extend over several lines or comment will be used as separator in NC set
ISO:) PAL: }	**end of comments**, if comment shall extend over several lines or comment will be used as separator in NC set
;	**start of comments** (single-line comment)
CR (end of line)	**end of comment** (single-line comment**)**
ISO: [PAL: (	**start of argument** at functions or bracketing of terms
ISO:] PAL:)	**end of argument** at functions or bracketing of terms
+	sign at decimal numbers or arithmetical operator: **addition**
-	sign at decimal numbers or arithmetical operator: **subtraction**
*	arithmetical operator: **multiplication**
/	arithmetical operator: **division** ***or*** set skip character, if a N-word follows
&	Boolean operation: **AND**

|	Boolean operation: **OR**
^	Boolean operation: **ANTIVALENZ respectively EXCLUSIV OR**: a ^ b = (not a & b) | (a & not b)
<	relational operator: **lower as**
>	relational operator: **greater as**
!=	relational operator: **unequal**
==	relational operator: **equal**
:	character to **selection** of a coordinate component of a Q-variable or of a symbolic frame
/	character for set skip
=	value assignment to coordinate address letters at indexing axis-addressing

Table 2.4: Special characters and its meaning in ProNC

2.5 Subprograms

The subprogram technique in ProNC is realised due to the guideline in DIN 66025.

please refer to: Section 3.1.7 Subprogram technology

subprogram-...	ISO syntax	PAL syntax
-**start** (declaration)	**%L100**	**%SUBR**100
-**end** (declaration)	**M17**	**RETURN**
-**call** (activation)	**L100**	**SUBR**100

Table 2.5: Subprogram declaration and -activation

3 ProNC language description

3 ProNC language description

In the application programs (ISO-source program or PAL-source program, in the following shortly called source program), processed in ProNC, an explicit declaration part is not necessary for constants or variables. It exists only the demand, that in every *source program* the subprograms must be defined in front of the main program(part).

Program text: The program text consists of **program lines**.

To make an explicit reference to the terminology of informatics (data processing), in this documentation will be differed between **program lines**,

● which are typical for programming numerical controlled plants (toll machines, handling systems):
These program lines are **NC sets** with a structure defined for example in DIN 66025 / ISO 6983.

● which are typical for the programming language of data processing:
These program lines are described as **instructions**.

Program lines: Program lines can be **NC-sets** or **instructions**.

Therefore every source program consists of a sequence of NC sets and / or instructions.

NC sets: NC sets correspond in their syntax to the rules of DIN66025.

Instructions: Instructions can be:

- an empty program line, this is an **empty instruction**
- a comment line, this is also an **empty instruction**
- every program line, that is not a NC-set, is an **instruction**

The structure of a set was defined in section 2.2 Structure of a NC set of this documentation.
In the section 3.1 Commands by DIN 66025 all available **NC-sets** and the relevant commands are summarized.

All usable **instructions** in ProNC are described in section 3.2 Instructions.

To a better understanding of the both section 3.1 and 3.2 please read the following statements:

Program text: For all source programs in ProNC the rule is valid: program text can be entered **with any notation (uppercase or lowercase letter).**

There is no difference between the key words

- *EndFor*,
- *ENDFOR* or
- *endfor .*

✍ The compiler realizes an optional pre-processor run. During this run all lowercase letters are converted into uppercase letters (outside any comments). This has the consequence, that also frame names like Park Position, PARKPOSITION and park position are not distinguished.

Within comments arbitrary characters may be used. Comments start either with round or curly opening bracket **(respectively {** and end with the closing round or curly bracket **) respectively }** or start with a semicolon **;** and end with the line end character **CR** (**C**arriage **R**eturn). The special signs und their meaning will be defined in section 2.4 Special signs.

NC-sets:

All NC-sets can start with a set number (N-word). This is also valid for instruction of variables / parameter calculation. To distinction of instructions to control the program process (e.g. FOR-loop) of NC-sets at all loops and branches you must not use any **set numbers** (N-words).

Variable:

The user of ProNC can carry out a very efficient and flexible parameter calculation by the possibility to use variables (section 3.2.3). No complicated names/identifiers or declarations are needed for simple, implicit variables (**P-, Q- or R-variable**), as it is usual at higher programming languages of the EDP. In ProNC a variable starts with the uppercase letter (address letter) P, R or Q, followed by a natural number n:

- P-variable: 0 <= n < 100 -> **process variable**
- Q-variable: 0 <= n < 500 -> **frame variable**
- R-variable: 0 <= n < 1000 -> **real variable**

Valid variable:

P0, P11, P99 are valid P-variables
R1, R222, R999 are valid R-variables
Q2, Q166, Q499 are valid Q-variables

Identifier to define frame names:

In a ProNC application program identifiers are needed to name the elements of the geometry file (frame file). The elements of the geometry file are named **frame**. Therefore an identifier to name a frame is called **frame name**.

Frame name:

A **frame name** consists of a minimum **of 4** characters and a **maximum of 20** characters. It will be demanded, that the first four characters of a frame name must be capital letters. The fifth and all following characters can be numbers, uppercase letters and also the underscore "_" in any order.

Valid / invalid frame name:

valid frame name:
MAXI, ABCD, ABSO, MAXIMUM, MINIMUM, ELVIRA123

invalid frame name:
111, AB1, 12_NORM, _1, N_1, A, AN3_ANTON

Natural number:

Natural numbers are used to define the key number, e. g. at all G-commands and M-commands. They mustn`t have a plus or minus sign.

example:

valid natural number:
100, 200, 300, 1

invalid natural number:

+100, +200, +300, +1_

Hexadecimal numbers:

Natural numbers can also be defined hexadecimal. In this case the prefix 0X or $ must be set in front of the string.
As postfix the character H or h can be used.
At least one and maximum eight signs from following character set must follow the prefix:

- the numbers 1,2,3,4,5,6,7,8,9,0
- the lowercase letters a, b, c, d, e, f
- the capital letters A, B, C, D, E, F

This rule can be described with *the regular term:*

0x([0-9a-fA-F]){1,8}
or
$([0-9a-fA-F]){1,8}
or
([0-9a-fA-F]){1,8} **H**

example:

- **valid hexadecimal numbers:**
  0x1234, 0xaa, 0xAA, $Ff, 12345678H, abcdH

- **invalid hexadecimal numbers:**
  0a1234, xaa, 0yAA, 0x_Ff, 0x123456789, 0xabxycd

please refer to: Section 3.2.2.2 Functions

Binary numbers:

A natural number can be written as binary number. The identification of the numeric string **bbbbbbbb** as binary number is defined with the letter B.
bbbbbbbbB b =[0,1]

example:

- 10101010B binary notation for the natural number 170
- 00000011B binary notation for the natural number 3

Decimal numbers:

Decimal numbers to define coordinates, velocities, constants (direct values for the assignment to R-variables) or arguments of functions can be indicated in three different ways:

- as decimal number with whole and broken part, e.g. **3.142** oder **0.142**
- as decimal number without whole and with broken part, e.g. **.142**
- as decimal number with whole and without broken part, e.g. **3.**

Key words (tokens):

Like in every programming language, keywords are also available in ProNC, which define certain syntactic constructions in their structure. These keywords (frequently also described as tokens in the usage of computer science) are summarized next:

☞

- instructions to control the program processing:
- **FOR, ENDFOR**
- **WHILE, ENDWHILE**
- **DO, ENDDO**
- **REPEAT, UNTIL**
- **IF, ELSE, ENDIF**
- **SWITCH, CASE, ENDCASE, DEFAULT, ENDSWITCH**

- for trigonometric functions:
- **SIN, COS, TAN**
- **ASIN, ACOS, ATAN**

- for real functions
- **FABS**
- **SQR, SQRT**
- **FLOOR**
- **EXP**
- **LOG, LN**
- **POW**

- for a waiting period
- **TIME / DELAY**

- for the circle number Pi
- **Pi**

Arithmetical und Boolean terms:

Both arithmetical and Boolean terms are used at the parameter calculation. A term is general a number, a variable or a combination of variables and / or of numbers. Depending on the operation is an arithmetical or a Boolean operation it will be called arithmetical or Boolean term.

Instructions to control the program processing:

Using instructions to control the program processing (FOR-loop, WHILE-loop, DO/REPEAT-loop) respectively using a program branching (IF-construction, SWITCH-construction) **conditions** are tested. Conditions are comparisons between arithmetical terms or Boolean terms. A condition has always a so-called truth value:

If the condition is filled, the truth value is 1 (TRUE) . If the condition is not filled, the truth value is 0 (FALSE) .

Condition:

In the syntax notation the condition is written with lower-case letters. That means, please write at the place of the grammatical construction a syntactic faultless notation representing a condition.

Syntax notation for a condition:

In the syntax-notation

IF **condition**
...
ELSE
...
ENDIF

condition is the word for a valid notation of a condition. The condition can be for example:

R1 > R2

Then a syntactic correct program text would be e. g.:

IF **R1 > R2**
 N10 G1 G91 X100
ELSE
 N20 G1 G91 X-100
ENDIF

Instruction:

Instructions to control a program processing contain the word **instructions** (written in lowercase letters) in the syntax notation. This word **instructions** stands as an abbreviation for:

- **empty instruction**
  (empty program line or comment line)

or
- **NC set**

or
- **sequence of NC sets**

or
- **instruction**

or
- **sequence of instructions**

please refer to:
Section 3.2.3 Assignments to control the program process

Nested depth:

It gets obvious, that an instruction for the control of the program flow (e.g. FOR loop) can contain instructions again.
Because this instruction can be a FOR loop again, in ProNC a nesting of instructions is possible. To limit the administration effort of this nestings, the so-called nested depth is limited on **five**.

In the following sections a uniform structure is used for the description of all NC sets (words / commands within NC sets) or description of all instructions:

NC set:

Command by ISO-syntax	**Summary for the NC set**	**Command by PAL-syntax**

Instruction:

Name of the instruction	**Summary for the instruction**

☐ Syntax: The syntax defines, how the construction (WORD / COMMAND or INSTRUCTION) must be written in the application program text. It is noted, which parameters, e.g. coordinates, variables or identifiers are permitted within the construction.

Hint to the notation in the syntax-field:

Notation	**Meaning**
[construction**]?**	the construction indicated in square brackets is optional, i.e. it can be programmed once or left out
[construction**]***	non, one or several repetitions of the defined construction
[construction**]+**	one or several repetitions of the defined construction
[construction**]{m,n}**	minimal **m** and maximum **n** repetitions of the construction

☐ Declaration: The purpose, the task, the characteristics and / or the application of the construction are explained as text.

☐ Example: The purpose, the task, the characteristics and / or the application of the construction are explained with examples.

☐ Reference: It will be referred to a reference to related constructions.

3.1 Commands by DIN 66025 in the NC set

3.1.1 Path commands

Fast velocity

The fast velocity will be defined in the initialization file of the motion module (*isel*-Motion Control MCTL) or will be set with the command FASTVEL in the application program (modal effect).

With fast velocity primarily positioning movements are programmed. A positioning movement is e. g. a movement to the work piece zero point before a processing or the movement to the park position after a processing.

Processing velocity:

The processing velocity is defined in the initialization file of the motion module (*isel*-Motion Control MCTL) or it is set up in the source program with help of the F-command.

✍

With processing velocity technological movements are primarily programmed, e. g. the milling of an edge, the welding of a seam or the drilling of a hole. All these movements have one community: a motion segment (a straight line or a circle) or a trajectory is driven.

Interpolation plane:

The statement of the interpolation plane is only useful at Cartesian systems, because only at Cartesian systems the motion module carries out a circle command.

The interpolation plane fixes, in which plane the next circle is driven: X Y plane or X Z plane or Y Z plane.
The specification of the interpolation plane has no influence for the linear interpolation at Cartesian kinematics (straight commands **G0** | **FASTABS** or **G1** | **MOVEABS** or **G10** | **FASTFRAME** or **G11** | **MOVEFRAME**), because this interpolation is always a 3D interpolation.

Zero point shift:

A zero point shift during the technological processing (milling, drill, stick, weld and others) serves primarily to fix the zero point of the work piece coordinate system opposite the zero point of the machine coordinate system.

✍

The zero point shift is used at handling systems to open a so-called local coordinate system, e.g. the reference system of an image recognition system, in the global coordinate system of the handling system.

3.1.1.1 Overview Path commands in ProNC

ISO-command	Meaning	PAL-command
G0	Motion with **fast** velocity	**FASTABS** **FASTREL**
G1	**Linear interpolation** at Cartesian Kinematics **S-PTP-motion** at non Cartesian Kinematics	**MOVEABS** **MOVEREL**
G2	**Circle interpolation** clockwise at Cartesian Kinematics	**CWABS** **CWREL**
G3	**Circle interpolation** counter clockwise at Cartesian Kinematics	**CCWABS** **CCWREL**
G4	**Dwell / Wait / Delay**	**TIME** **DELAY**
G10 **G11**	Motion with **fast velocity** in connection with a frame variable Q0 ... Q499 Motion with **processing velocity** in connection with a frame variable Q0 ... Q499	**FASTFRAME** **MOVEFRAME**
G12	**Helix** clockwise	**CWHLXABS** **CWHLXREL**
G13	**Helix** counter clockwise	**CCWHLXABS** **CCWHLXREL**
G17 **G18** **G19**	Definition of the interpolation plane (**X-Y-plane**) Definition of the interpolation plane (**X-Z-plane**) Definition of the interpolation plane (**Y-Z-plane**)	**PLANE XY** **PLANE XZ** **PLANE YZ**
G53 **G54** **G55** **G56**	Zero point shift **deactivate** Zero point shift 1 **activate** Zero point shift 2 **activate** **Set** the work piece zero point on the current position	**WPCLEAR** **WPREG1** **WPREG2** **WPZERO**
G60 **G64**	Switch off explicit path mode (path end) Switch on explicit path mode (path start)	**PATHEND** **PATH**
G70 **G71**	Definition of measure for translatory axis: **inch** Definition of measure for translatory axis: **mm**	**INCH** **METRIC**
G74	**Reference run**	**REF**
G75	**Teach-In**: The window „current geometry file: ...“ can activated during the automatic mode	**TEACH**
G80	Define parameter of a drilling cycle	**DrillDef**

G81 **G82** **G83** **G84**	**Simple drilling** **Drilling** with **dwell** **Drilling in** operating mode **countersick** **Drilling** in operating mode **break chip**	**DrillN** **DrillT** **DrillD** **DrillB**
G90 **G91**	Coordinate statements are absolute statements (**absolute dimension**) Coordinate statements are incremental statements (**incremental dimension**)	**ABS** **REL**
G92 **G93**	**Set memory** (work piece zero-point register 1) **Set memory** (work piece zero-point register 2)	**WPREG1WRITE** **WPREG2WRITE**
G98	**Parameter input** for technological variable (R-variable)	**PARAMETER**
G99	**Text output** into the status line	**TYPE**

Table 3.1.1: Path commands in ProNC (Overview)

3.1.1.2 Positioning with fast velocity

G0-command	Motion with **fast velocity**	**FASTABS-command** **FASTREL-command**

☐ Syntax:

G-code	PAL
[set number]?	[set number]?
[further command:	[further command:
G17, G18, G19, G70, G71]*	PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC]*
G0	**FASTABS** or **FASTREL**
[target-coordinates]{1,6}	[target-coordinates]{1,6}
[F-command]?	[F-command]?
[S-command]?	[S-command]?
[M-command]*	[miscellaneous command]*

☐ Explanation:

Cartesian Kinematic:
positioning motion with fast velocity:

- the fast velocity is defined in the initialisation file of the motion module or by the command **FASTVEL**
- at least one coordinate statement must be available in the NC set
- at most six coordinate statements may be available in the NC set
- if an absolute dimension is adjusted (**G90 | ABS**) the target coordinates refer to the current zero point of the work piece coordinate system
- if incremental dimension (**G91 | REL**) is adjusted, the target coordinates refer to the current start point
- the unit of the target position (X, Y, Z) is millimetre [mm], for rotatory axes (A, B, C) grad [°]

☐ Example:

Cartesian Kinematics:
; absolute motion to the target point with the coordinates
; (100mm, 200mm, 300mm) with fast velocity:
ISO: N200 **G00** G90 X100.0 Y200.0 Z300.0
PAL: N200 **FASTABS** X100.0 Y200.0 Z300.0

Cartesian Kinematics:
; relative motion of the X-axis **about** 10 mm, of the Y-axis **about**
; 20 mm and the Z-axis about 30 mm, viewing from the current start point
; with fast velocity:
ISO: N200 **G00** G91 X10.0 Y20.0 Z30.0
PAL: N200 **FASTREL** X10.0 Y20.0 Z30.0

non-Cartesian Kinematics:
; absolute motion to the target point with the values:
; C-axis: 100 grad Z-axis: 180 mm
; B-axis: 45.0 grad A-axis: -45.0 grad
; with fast velocity:
ISO: N100 **G00** G90 C100.0 Z180.0 B45.0 A-45.0
PAL: N100 **FASTABS** C100.0 Z180.0 B45.0 A-45.0

☐ Reference:

G-code	PAL
G1, G10, **G11,** G70, G71, G90, G91	**MOVEABS, FASTFRAME,** **MOVEFRAME**, INCH, METRIC, ABS, REL

3.1.1.3 Linear interpolation

G1-command	**Linear interpolation** at Cartesian Kinematics **S-PTP-motion** at non Cartesian Kinematics	**MOVEABS-command** **MOVEREL-command**

☐ Syntax:	[set number]? [further command: G17, G18, G19, G70, G71]* **G1** [target coordinates]{1,6} [F-command]? [S-command]? [M-command]*	[set number]? [further command: PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC]* **MOVEABS** *or* **MOVEREL** [target coordinates]{1,6} [F-command]? [S-command]? [miscellaneous command]*

☐ Explanation:

Cartesian Kinematics: Linear interpolation with processing velocity
non-Cartesian Kinematics: Positioning motion with processing velocity

- the processing velocity can be defined with help of a F-command in the current NC-set or the processing velocity, defined in the previous NC-set, is valid
- the fast velocity is defined in the initialisation file of the motion module or by the command **FASTVEL**
- at least one coordinate statement must be available in the NC set
- at most six coordinate statements may be available in the NC set
- if an absolute dimension is adjusted (**G90 | ABS**) the target coordinates refer to the current zero point of the workpiece coordinate system
- if incremental dimension (**G91 | REL**) is adjusted, the target coordinates refer to the current start point
- the unit of the target position (X, Y, Z) is millimetre [mm], for rotatory axes (A, B, C) grad [°]

☐ Example:

Cartesian Kinematics (XYZ):
; straight in the space to the absolute target point with the
; coordinates (100 mm, 200 mm, 300 mm) with
; processing velocity:

ISO: N100 **G1** G90 X100.0 Y200.0 Z300.0
PAL: N100 **MOVEABS** X100.0 Y200.0 Z300.0

Cartesian Kinematics (XYZ):
; straight in the space to the absolute target point with the coordinates
; X-IST + 10 mm, Y-IST + 20 mm, Z-IST – 30 mm
; with processing velocity:

ISO: N200 **G1** G91 X10.0 Y20.0 Z-30.0
PAL: N200 **MOVEREL** X10.0 Y20.0 Z-30.0

non-Cartesian Kinematics:
; absolute motion to the target point with the values:
; C-axis: 100 grd Z-axis: 180 mm
; B-axis: 45.0 grd A-axis: -45.0 grd
; with fast velocity:

ISO: N100 **G01** G90 C100.0 Z180.0 B45.0 A-45.0
PAL: N100 **MOVEABS** C100.0 Z180.0 B45.0 A-45.0

☐ Reference:	**G0, G10,** **G11,** G70, G71, G90, G91	**FASTABS, FASTFRAME,** **MOVEFRAME**, INCH, METRIC, ABS, REL

3.1.1.4 Circular interpolation clockwise

G2-command	**Circular interpolation cw** (clockwise) at Cartesian Kinematics	**CWABS-command** **CWREL-command**

☐ Syntax:

[set number]?	[set number]?
[further command:	[further command:
G17, G18, G19, G70, G71]*	PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC]*
G2	**CWABS** or **CWREL**
[target coordinates]{1,3}	[target coordinates]{1,3}
[center coordinates]{1,3}	[center coordinates]{1,3}
[F-command]?	[F-command]?
[S-command]?	[S-command]?
[M-command]*	[miscellaneous command]*

☐ Explanation:

Cartesian Kinematics:
circle / arc of a circle in the active interpolation plane clockwise with definition of the center coordinates

● this command can only be used for Cartesian plants
● at least one target position value and the corresponding center coordinate have to be defined:

X -> I, Y -> J, Z -> K

● the definition of target coordinates can be absolute (G90 | ABS) or relative (G91 | REL)
● the definition of center coordinates are always specified relatively to the start point
● the unit of the target position is millimetre [mm]
● the direction of rotation is defined so, that the third coordinate runs always from positive to negative, if you look on the interpolation plane

<u>Hint:</u>
The **X-Y-plane** as interpolation plane is selected with the command G17 | PLANE XY ; now please look into negative **Z-direction** on a "phantom-clock" in this plane, that direction of rotation agrees with the direction of rotation of the circle.

☐ Example:

; Semicircle clockwise in the X-Y-plane:
; start point:
(X_start,Y_start)=(**0**,**0**)
; endpoint:
(X_end,Y_end)=(**100**,0)

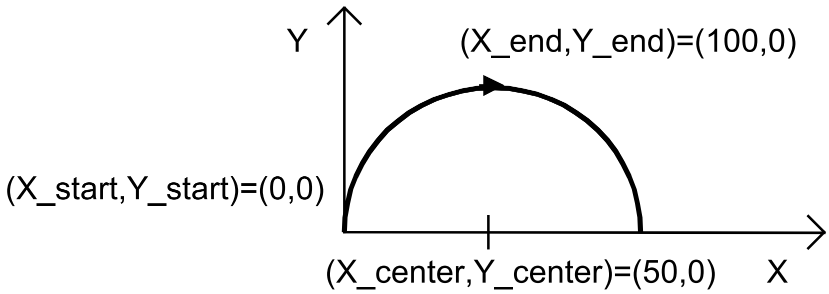
; processing velocity: **50** mm/sec:

ISO:	N10 G17	; define the interpolation plane
	N20 G0 G90 X**0** Y**0**	; move to start point
	N30 **G2** X**100** I50 F**50**	; drive circle
PAL:	N10 PLANE XY	; define interpolation plane
	N20 FASTABS X**0** Y**0**	; move to start point
	N30 **CWABS** X**100** I50 F**50**	; drive circle

Hint:
the center coordinates (X_center**,** Y_center) result always by addition of the I- respectively J-values to the start values of the circle (X_A, Y_A):

X_center := X_A + I
Y_center := Y_A + J

Consequently the **I**- , J- and K-coordinates are always relative statements.

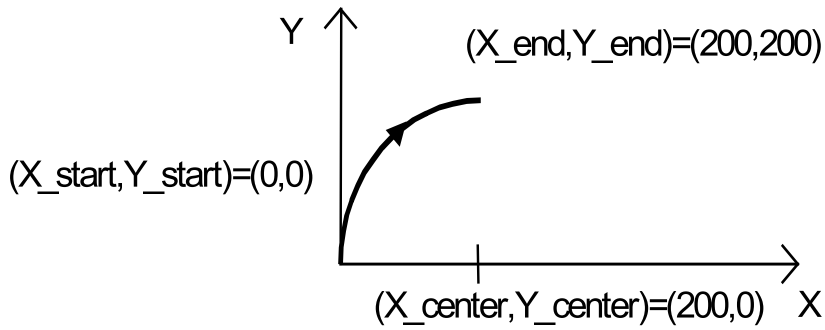


☐ Example:

; **Circle arc** clockwise in the X-Y-plane:
; start point:
(X_start,Y_start)=(**0**,**0**)
; end point:
(X_end,Y_end)=(200,**200**)

; processing velocity **75** mm/sec:

ISO: N10 G17 ; define the interpolation plane
N20 G0 G90 X**0** Y**0** ; move to the start point
N30 **G2** X**200** Y**200** I**200** J**0** F**75** ; drive circle
PAL: N10 PLANE XY ; drive the interpolation plane
N20 FASTABS X**0** Y**0** ; move to start point
N30 **CWABS** X**200** Y**200** I**200** J0 F**75** ; drive circle



☐ Reference:

G3, G17, G18, G19, G90, G91	**CCWABS**, PLANE XY, PLANE XZ, PLANE YZ, ABS, REL

3.1.1.5 Circular interpolation counter clockwise

G3-command	**Circular interpolation ccw** (counter clockwise) at Cartesian Kinematics	**CCWABS-command** **CCWREL-command**
☐ Syntax:	[set number]? [further command: G17, G18, G19, G70, G71]* **G3** [target coordinates]{1,3} [center coordinates]{1,3} [F-command]? [S-command]? [M-command]*	[set number]? [further command: PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC]* **CCWABS** or **CCWREL** [target coordinates]{1,3} [center coordinates]{1,3} [F-command]? [S-command]? [miscellaneous command]*

☐ Explanation:

Cartesian Kinematics:
circle / arc of a circle in the active interpolation plane clockwise with definition of the center coordinates

- this command can only be used for Cartesian plants
- at least one target position value and the corresponding center coordinate have to be defined:

X -> I, Y -> J, Z -> K

- the definition of target coordinates can be absolute (G90 | ABS) or relative (G91 | REL)
- the definition of center coordinates are always specified relatively to the start point
- the unit of the target position is millimetre [mm]
- the direction of rotation is defined so, that the third coordinate runs always from positive to negative, if you look on the interpolation plane

<u>Hint:</u>
The **X-Y-plane** as interpolation plane is selected with the command G17 | PLANE XY ; now please look into negative **Z-direction** on a "phantom-clock" in this plane, that direction of rotation agrees with the direction of rotation of the circle.

☐ Example:

; **Quarter circle** counterclockwise in the XY-plane:
; startpoint:
(X_start**,**Y_start)=(**600**,**0**)
; endpoint:
(X_end,Y_end)=(**300**,**300**)

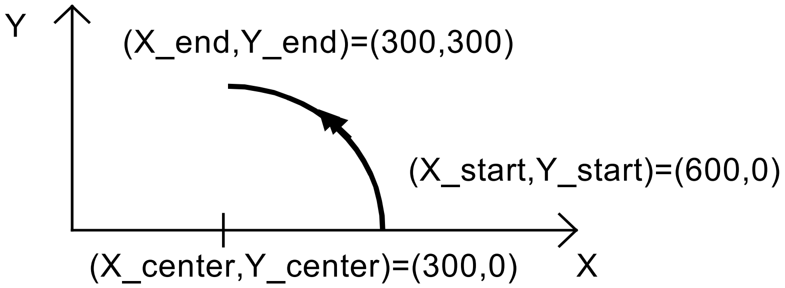
; processing velocity **66** mm/sec:

ISO:
N10 G17 G90
N20 G0 **X600** **Y0** ; move to start point
N30 **G3** X**300** Y**300** I-*300* F**66** ; drive circle

PAL:
N10 PLANE XY ABS
N20 FASTABS **X600** **Y0** ; move to start point
N30 **CCWABS** X**300** Y**300** I-*300* F**66** ; drive circle

Hint:
The absolute coordinates of the circle center in the following drawing result out of the addition of the specified **I**-coordinate value –300 in the set N30 to the start value of the X-coordinate: 600 -300 = 300.

Because the center coordinate Y_center = 0 does not change opposite the start value Y_start = 0 , the definition of the J-position value in the NC-set can escape.



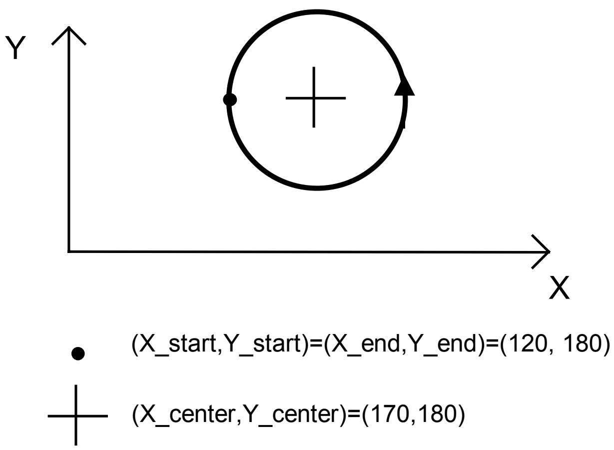
☐ Example:

; Circle counterclockwise in the X-Y-plane:
; startpoint: (X_start,Y_start)=(**120**,**180**)
; radius. 50 mm
; endpoint: (X_end,Y_end)=(**120**,**180**)

; processing velocity **110** mm/sec:

ISO:
N10 G17 G90
N20 G0 X**120** Y**180**
N30 **G3** X**120** Y**180** I**50** J0 F**11**

PAL:
N10 PLANE XY ABS
N20 FASTABS X**120** Y**180**
N30 **CCWABS** X**120** Y**180** I**50** J0 F**11**



☐ Reference:

G2, G17, G18, G19, G90, G91	**CWABS**, PLANE XY, PLANE XZ, PLANE YZ, ABS, REL

3.1.1.6 Dwell time

G4-command	**Dwell time**	**TIME-command** **DELAY-command**

☐ Syntax:	[set number]? **G4** dwell time	[set number]? **TIME** dwell time **DELAY** dwell time

☐ Explanation: **Definition of a dwell time in an application program**

- **dwell time** is a natural number
- the smallest naming unit is 1 millisecond
- the range of values of **dwell time** is the data type unsigned long (32 Bit); that means, the maximum dwell time can be (2 to the 32 - 1) * 0,001 sec

☐ Example: ; 1000 msec = wait 1 sec:

ISO: N10 **G4** 1000
PAL: N10 **TIME** 1000

; the dwell time is determined by the current contents of the R-variable:

ISO: N20 **G4** R1
PAL: N20 **TIME** R1

3.1.1.7 Fast velocity with statement of frame

G10-command	Motion with fast velocity in combination with a frame variable **Q0 ... Q499 or with an** indexing Q-variable **or** a frame name	**FASTFRAME-command**

☐ Syntax:	[set number]? [further command: G17, G18, G19, G70, G71]* **G10 q_variable** *or* **G10 Q r_variable** ***or*** **G10 frame_name** [S-command]? [M-command]*	[set number]? [further command: PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC]* **FASTFRAME q_variable** *or* **FASTFRAME Q r_variable** ***or*** **FASTFRAME frame_name** [S-command]? [miscellaneous command]*

☐ Explanation:

Positioning motion with fast velocity, without explicit target coordinates, but with a frame variable (Q-variable) or an indexing Q-variable or a frame name.

- the target statement is always absolute
- the fast velocity is defined in the initialisation file of the motion module or by the command **FASTVEL**
- the frame variables must initialised in the initialisation part of the application program

ATTENTION

Past a G10-command | FASTFRAME-command the absolute measure is always active, even if ahead of a G10-command | FASTFRAME-command a relative measure (incremental measure) was defined by a G91-command | REL-command.

☐ Example:

; the Q-variable Q1 is initialised:
N10 Q1 = START

; positioning motion in fast velocity to the position, which is actually stored
; in the Q-variable Q1:

ISO: N20 G10 Q1
PAL: N20 **FASTFRAME** Q1

; indexing of Q-variable:
ISO: N100 G10 QR5 ; synchron-PTP-motion to the
; Q-target point, that index is just in R5
PAL: N100 **FASTFRAME** QR5 ;synchron-PTP-motion to the
; Q-target point, that index is just in R5

; direct statement of the frame name in the command:
ISO: N20 G10 PARK_POSITION
PAL: N20 **FASTFRAME** PARK_POSITION

☐ Reference:

G11	**MOVEFRAME**

Section 3.2.1.2: Q-variable
Section 3.2.2.4: Assignments

3.1.1.8 Processing velocity with statement of frame

G11-command	Motion with **processing velocity** in combination with a frame variable **Q0 ... Q499 or with an** indexing Q-variable **or** a frame name	**MOVEFRAME**
☐ Syntax:	[set number]? [further command: G17, G18, G19, G70, G71]* **G11 q_variable** *or* **G11 Q r_variable** *er* **G11 frame_name** [F-command]? [S-command]? [M-command]*	[set number]? [further command: PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC]* **MOVEFRAME q_variable** *or* **MOVEFRAME Q r_variable** ***or*** **MOVEFRAME frame_name** [F-command]? [S-command]? [miscellaneous command]*

☐ Explanation:

Positioning motion with processing velocity, without explicit target coordinates, but with a fame-variable (Q-variable) or an indexing Q-variable or a frame name.

- the target statement is always absolute
- the processing velocity can be defined with F-commands or VEL-commands in the actual NC set or the processing velocity, defined in the previous set, is valid
- the frame variables must initialised in the initialisation part of the application program

ATTENTION

Past a G11-command | MOVEFRAME the absolute measure is always active, even if ahead of a G11-command | MOVEFRAME a relative measure (incremental measure) was defined by a G91-command | REL-command.

☐ Example:

;the Q-variable Q2 is initialised:
;N10 Q2 = ENDE

; positioning motion with defined processing velocity to the position, which is
; actually stored in the Q-variable Q2:

ISO: N20 G11 Q2 F100.1
PAL: N20 **MOVEFRAME** Q2 F100.1

; indexing of Q-variables:

ISO: N100 G11 QR6 ; synchron-PTP-motion to the
; Q-target point, that index is just in R6

PAL: N100 **MOVEFRAME** QR6 ; synchron-PTP-motion to the
; Q-target point, that index is just in R6

; direct statement of the frame name in the command::

ISO: N20 G11 PARK_POSITION
PAL: N20 MOVE**FRAME** PARK_POSITION

☐ Reference:

G10	**FASTFRAME**

Section 3.2.1.2: Q-variable
Section 3.2.2.4: Assignments

3.1.1.9 Helix clockwise

G12-command	Helix interpolation CW (clockwise) at Cartesian Kinematics	CWHLXABS-command CWHLXREL-command

☐ Syntax:

[set number]?	[set number]?
[further command:	[further command:
G70, G71]*	INCH, METRIC]*
G12	**CWHLXABS** or **CWHLXREL**
rotation angle W	rotation angle W
[target coordinates]{1,3}	[target coordinates]{1,3}
[center-coordinates]{1,3}	[center-coordinates]{1,3}
[F-command]?	[F-command]?
[S-command]?	[S-command]?
[M-command]*	[miscellaneous command]*

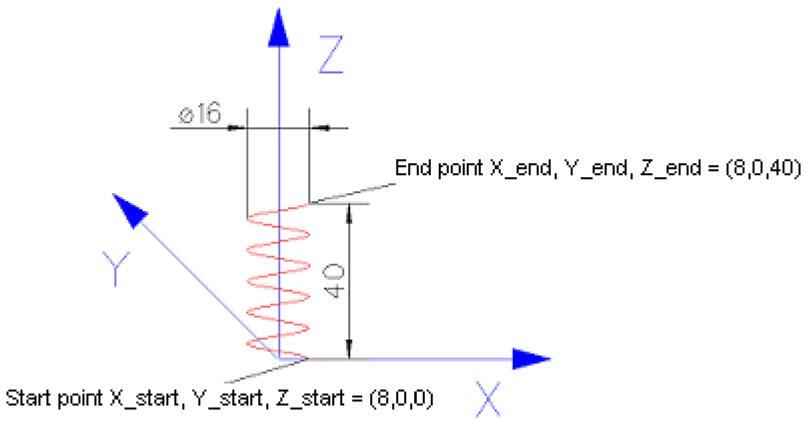
☐ Explanation:

Helix motion to an end point (target coordinates), around a radius center (center coordinates) with the angle of rotation W clockwise.

- target coordinates statements can be made absolute (G90 | ABS) or relative (G91 | REL)
- center point coordinate statements are always relative according to the start point
- the statement of the rotation angle defines the number of rotations; 360° defines 1 rotation

☐ Example:

; Helix drive with a whole angle of 1800 grad
; (it corresponds to 5 full circles) with a radius = 8 mm




```
ISO:  N10 G17                     ; fix interpolation plane
      N20 G0 G90 X8 Y0 Z0         ; run to start point
      N30 G12 W1800 X8 Z40 I-8 J0 ; drive helix

PAL:  N10 PLANE XY                ; fix interpolation plane
      N20 FASTABS X8 Y0 Z0        ; run to start point
      N30 CWHLXABS W1800 X8 Z40 I-8 J0  ; drive helix
```

☐ Reference: **G13**, G2, G3 | **CCWHLXABS**, CWABS, CCWABS

3.1.1.10 Helix counter clockwise

G13-command	**Helix interpolation CCW** (counter clockwise) at Cartesian Kinematics	**CCWHLXABS-command** **CCWHLXREL-command**

☐ Syntax:

[set number]? [further command: G17, G18, G19, G70, G71]* **G13** rotation angle W [target coordinates]{1,3} [center-coordinates]{1,3} [F-command]? [S-command]? [M-command]*	[set number]? [further command: INCH, METRIC]* **CCWHLXABS** or **CCWHLXREL** rotation angle W [target coordinates]{1,3} [center-coordinates]{1,3} [F-command]? [S-command]? [miscellaneous command]*

☐ Explanation:

Helix motion to an end point (target coordinates), around a radius center (center coordinates) with the angle of rotation W counter clockwise.

- target coordinates statements can be made absolute (G90 | ABS) or relative (G91 | REL)
- center point coordinate statements are always relative according to the start point
- the statement of the rotation angle defines the number of rotations; 360° defines 1 rotation

☐ Example:

; thread milling in the pre-drilled hole
; radius = 5 mm, helix with 10 full circle

ISO:

N10 G17 ; fix interpolation plane
N20 G0 G90 **X5** Y**0** Z**-10** ; run to start point
N30 **G13** W**3600** X**5** Y**0** Z**0** I**-5** J**0** ; drive helix

PAL:

N10 PLANE XY ; fix interpolation plane
N20 FASTABS **X5** Y**0** Z**-10** ; run to start point
N30 **CCWHLXABS** W**3600** X**5** Y**0** Z**0** I**-5** J**0** ; drive helix

☐ Reference:

G12, G2, G3	**CWHLXABS**, CWABS, CCWABS

3.1.1.11 All motion commands

all motion commands	**programmable abort** of motions in automatic mode

☐ Syntax:

An input can be programmed with definition of the port and the bit number in any program line, which causes a motion of the axes in the mechanical system (all **G0**-, **G1**-, **G2/G3**-, **G10-** and **G11-commands** | **FASTABS-, MOVEABS-, CWABS/CCWABS-, FASTFRAME-** and **MOVEFRAME-commands**).
If during the motion to the programmed target point a low high flank or high low flank of the corresponding input is carried out, the motion will be aborted.

☐ Explanation:

ProNC has the ability to abort motions in automatic mode, if a programmed binary input is activated and to continue with the command, following in the application program.
This functionality can be used if ProNC instructs a motion control (MCTL) for servo plants (numerical axes with DC-/AC-servomotors, isel-Servo-Controller CV with slot card UPMV4/12 or isel-CAN-Controller). That means, the programmable abort of motions in automatic mode is not usable for following axes with stepper motor:

- plants with Controller C116-4 / C142-4
- all plants of CPM-line (CPM 2018, CPM 3020, CPM 4030)
- plants of GFM-line (GFM 4433)

☐ Example:

; the motion to the target point X=100mm, Y=200mm
; is aborted, if the binary input E1.1
; is activated during the motion (low-high-edge):

ISO: N10 G1 X100 Y200 E1.1
PAL: N10 MOVEABS X100 Y200 E1.1

; the motion to the target position in Q5
; is aborted, if the binary input E4.7
; is activated during the motion (low-high-edge):

ISO: N10 G10 Q5 NOT E4.7
PAL: N10 FASTFRAME Q5 NOT E4.7

3.1.1.12 Definition of interpolation plane

G17-command	Definition of interpolation plane **(X-Y-plane)**	**PLANE XY-command**
G18-command	Definition of interpolation plane **(X-Z-plane)**	**PLANE XZ-command**
G19-command	Definition of interpolation plane **(Y-Z-plane)**	**PLANE YZ-command**

☐ Syntax:

[set number]?	[set number]?
G17 *oder* **G18** *oder* **G19**	**PLANE XY** *or* **PLANE XZ** *or* **PLANE YZ**
[further command: G53, G54, G55, G56, G70, G71, G90, G91]*	[further command: WPCLEAR, WPREG1, WPREG2, WPZERO, INCH, METRIC, ABS, REL]*
[F-command]?	[F-command]?
[S-command]?	[S-command]?
[M-command]*	[miscellaneous command]*

☐ Explanation:

select interpolation plane:

G17 | PLANE XY: the X-Y-plane is selected

● all previous circle commands (G2 or **G3** | CWABS or **CCWABS**) refer to the X-Y-plane

● the G17-command | PLANE XY-command is default in every application program; that means, this command does not have to be programmed

G18 | PLANE XZ: the X-Z-plane is selected

● all previous circle commands (G2 or **G3** | CWABS or **CCWABS**) refer to the X-Z-plane

G19 | PLANE YZ: the Y-Z-plane is selected

● all previous circle commands (G2 or **G3** | CWABS or **CCWABS**) refer to the Y-Z-plane

☐ Example:

; the X-Y-plane is selected:

ISO: N10 G17
PAL: N10 PLANE XY

; the X-Z-plane is selected:
ISO: N10 G18
PAL: N10 PLANE XZ

; the Y-Z-plane is selected:
ISO: N10 G19
PAL: N10 PLANE YZ

☐ Reference:

G2, G3	**CWABS, CCWABS**

3.1.1.13 Set up zero point

G53-command **G54/G55-command** **G56-command**	Zero point shift **deactivate** Zero point shift 1 / 2 **activate** **Set up** work piece zero point on the actual position	**WPCLEAR-command** **WPREG1/WPREG2-command** **WPZERO-command**
☐ Syntax:	[set number]? **G53** *or* **G54** *or* **G55** *or* **G56** [further command: G17, G18, G19, G70, G71, G90, G91]* [F-command]? [S-command]? [M-command]*	[set number]? **WPCLEAR** *or* **WPREG1** *or* **WPREG2** *or* **WPZERO** [further command: PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC, ABS, REL]* [F-command]? [S-command]? [miscellaneous command]*

☐ Explanation:

G53 | WPCLEAR: zero point shift **deactivate**

G54 | WPREG1 – **zero point shift 1 activate:**
the actual zero point of the workpiece coordinate system is shifted absolutely about the values in the zero point register **1** opposite the zero point of the machine coordinate system

G55 | WPREG2 - **zero point shift 2 activate:**
the actual zero point of the workpiece coordinate system is shifted absolutely about the values in the zero point register **2** opposite the zero point of the machine coordinate system

G56 | WPZERO: a new zero point is **set up** on the **actual position**

☐ Example:

; the shift of the work piece zero point, defined with the command G54 | WPREG1 or G55 | WPREG2 is cancelled or the zero point, installed with the command G56 | WPZERO , will be deleted

ISO: N10 G53
PAL: N10 WPCLEAR

; install a new work piece zero point with help of the zero point register **1**:
ISO: N20 G54
PAL: N20 WPREG1

; install a new work piece zero point with help of the zero point register **2**:
ISO: N30 G55
PAL: N30 WPREG2

; install a new work piece zero point on the actual position:
ISO: N40 G56
PAL: N40 WPZERO

☐ Reference: **G92, G93** | **WPREG1WRITE, WPREG2WRITE**

3.1.1.14 Path motion

G60-command **G64-command**	**Path motion switch off,** **Path motion switch on**	**PATHEND-command** **PATH-command**
☐ Syntax:	[set number]? **G64** *or* **G60**	[set number]? **PATH** *or* **PATHEND**

☐ Explanation:

● The user has two possibilities to realize the continuous path mode (path motion):

1. Possibility:
If the configured motion control has the ability of an online path mode, an activated button in the dialog window Processing causes the wanted path mode.
That means, all successive motion segments (G1, G2, G3, G11 | MOVEABS, MOVEREL, CWABS, CWREL, CCWABS, CCWREL, MOVEFRAME)
will be summarized to a continuous path.

2. Possibility
Are some successive motion segments to be driven in a user program and the button "Path mode" is not activated, the motion segments have to be bracketed with the commands G64/G60 | PATH/PATHEND.

● The calculation of the velocity profile about all motion segments which shall be summarized to a path, carries out by the motion control during the processing of the user program in real time (look ahead); thereby the variable concept is usable complete, because the values of R-variable will be always processed correctly.

The summary of motion segments to a path is carried out by "bracketing" with the commands G64 | PATH (marking the start of a trajectory driving with path velocity) and G60 | PATHEND (marking the end of a trajectory driving with path velocity).

● All programmed motion segments between G64 | PATH and G60 | PATHEND are summarized to a current path. The command G64 | PATH introduces the path motion. A programmed F-command defines the path velocity for the whole path segment, several F-commands in several segments causes several path velocities during a „connected“ path motion.

● The G60-command | PATHEND-command defines the end of a path (trajectory) in the source program.

☐ Example:

```
; the target points, stored in the Q-variable Q1 to Q4 are summarized to a
; path:
%L200
; subprogram to demonstration of path mode (CP):
;   -> milling with path velocity of 10 mm/sec:
; path mode switch on:
```

ISO:

```
N5 G99 path mode switch on ...
N1 G64
N10 G11 Q1 F10.0
N20 G11 Q2
N30 G11 Q3
N40 G11 Q4
N50 G60
N60 G99 path mode switch off ...
N70 M17
```

PAL:

```
N5 TYPE path mode switch on ...
N1 PATH
N10 MOVEFRAME Q1 F10.0
N20 MOVEFRAME Q2
N30 MOVEFRAME Q3
N40 MOVEFRAME Q4
N50 PATHEND
N60 TYPE path mode switch off ...
N70 RETURN
```

☐ Hint: Path mode is possible at the motion control for IMS6-Controller, the Servo-Card UPMV 4/12 respectively CAN-Controller.

3.1.1.15 Definition of measure

G70-command	Definition of **measure** for **translatory axes**: **inch**	**INCH-command**
G71-command	Definition of **measure** for **translatory axes**: **mm**	**METRIC-command**

☐ Syntax:

[set number]?	[set number]?
G70 *or* **G71**	**INCH** *or* **METRIC**
[further command: G17, G18, G19, G53, G54, G55, G90, G91]*	[further command: PLANE XY, PLANE XZ, PLANE YZ, WPCLEAR, WPREG1, WPREG2, ABS, REL]*
[F-command]?	[F-command]?
[S-command]?	[S-command]?
[M-command]*	[miscellaneous command]*

☐ Explanation:

G70 | INCH:
The **measure inch** is assigned to all coordinate statements for linear axes.

G71 | METRIC:
The **measure mm** is assigned to all coordinate statements for linear axes.

☐ Example:

Cartesian Kinematics (XYZ):
; straight line in space to the absolute target point with the
; coordinates (100 inch, 200 inch, 300 inch) with
; processing velocity:

ISO: N100 **G70** G1 X100.0 Y200.0 Z300.0
PAL: N100 **INCH** MOVEABS X100.0 Y200.0 Z300.0

Cartesian Kinematics (XYZ):
; straight line in space to the target point with the coordinates
; X-IST + 10mm, Y-IST + 20 mm, Z-IST - 30mm
; with processing velocity:
ISO: N200 G91 **G71** G1 X10.0 Y20.0 Z-30.0
PAL: N200 MOVEREL **METRIC** X10.0 Y20.0 Z-30.0

non-Cartesian Cinematic with 3 rotatory axes (e. g. Scara Robot):
; absolute motion to the target point with the values:
; foot turning: 100 grad vertical motion: 180 **inch**
; elbow joint: 45.0 grad hand turning: -45.0 grad
; with fast velocity:

ISO: N100 **G70** G0 C100.0 Z180.0 B45.0 A-45.0
PAL: N100 **INCH** FASTABS C100.0 Z180.0 B45.0 A-45.0

☐ Reference:

G0, G1, G2, G3	FASTABS, MOVEABS, CWABS, CCWABS

3.1.1.16 Reference run

G74-command	**Reference run**	**REF-command**

☐ Syntax:	[set number]? [further command: G17, G18, G19, G70, G71, G90, G91]* **G74** [**address letter**]? [M-command]*	[set number]? [further command: PLANE XY, PLANE XZ, PLANE YZ, INCH, METRIC, ABS, REL]* **REF** [**address letter**]? [miscellaneous command]*

☐ Explanation:

Implementation of a reference run:
- **address letter = {X, Y, Z, A, B, C}** for axis system 1
- **address letter = {X2, Y2, Z2, A2, B2, C2}** for axis system 2

(that means, the address letter is an element of the defined quantity)

- using the G74-command | REF-command without argument, all axes are synchronized in the order:

  Z-axis ➜ Y-axis ➜ X-axis ➜ A-axis ➜ B-axis ➜ C-axis.

- after a reference run the motion control module is reset, that means, a possible defined zero point was deleted and all initialization settings are valid (e.g. processing / fast velocity).

☐ Example:

; reference run for all axis with the velocity defined in the initialization file
; of the motion control module

ISO:	N10 **G74**
PAL:	N10 **REF**

; reference run with just one axis:

ISO:	N10 G74 X	; reference run of X-axis
	N20 G74 C	; reference run of C-axis
PAL:	N10 REF X	; reference run of X-axis
	N20 REF C	; reference run of C-axis

☐ Reference:	G70, G71, G90, G91	INCH, METRIC, ABS, REL

3.1.1.17 Teach

G75-command	programmable correction of axis positions	TEACH-command

☐ Syntax:

[set number]?	[set number]?
G75	**TEACH**

☐ Explanation:

With the command G75 | TEACH the window **actual geometry file: ...** can be activated during the run time of the user program.

With this function corrections of axis positions / Teach-In (a new input or update of geometry information of the current geometry file) can be done online without leaving the automatic mode.

The window **actual geometry file: ...** is left with OK or CANCEL, after that the user program will be continued directly with the command / instruction, following G75 | TEACH.
The current **geometry file** is located in the directory **CNCWorkbench\NCProg\Frame** and has the **same file name** like the current ISO respectively PAL user program.
Please pay attention to the characteristic, that just the **file extension** (ISO: name.iso | PAL: name.pal) is replaced with the typical extension **fra** for a geometry file.
That means, if you process the PAL user program ABC.PAL, you can also use the geometry file ABC.FRA.
If you process with several user programs with one geometry file, please use the standard geometry file stdframe.fra in the directory \CNCWorkbench\Bin.

☐ Example:

;Teach-In

ISO: N10 **G75**
PAL: N10 **TEACH**

☐ Reference:

Operating Instruction:
5.7.3.9 Menu Control - Manual movement
5.7.3.10 Menu Control - Setup machine positions
2.2.2. The geometry file

3.1.1.18 Drilling cycle define

G80	Definition of a drilling cycle	DRILLDEF

☐ Syntax:

[set number]?	[set number]?
G80	**DRILLDEF**

CY: **Drilling cycle**
1 = single drilling
2 = countersick
3 = break chip

PL: **Plane**
0 = XY
1 = XZ
2 = YZ

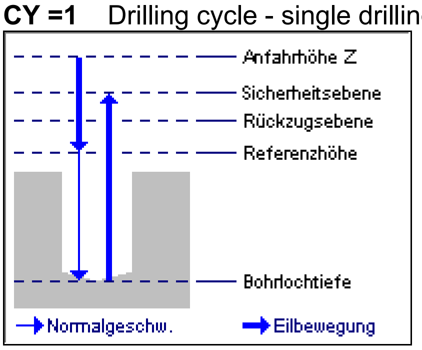
DI: **Direction**
0 = standard
1 = inverse

RF:	reference height	(mm)
DE:	depth	(mm)
TI:	time	(s)
VE:	processing velocity	(mm/s)
VF:	fast velocity	(mm/s)
FI:	first increment drill depth	(mm)
OT:	further increment drill depth	(mm)
IC:	decrease of increment drill depth	(mm)
RE:	increment retreat	(mm)
LE:	retreat plane out of drill hole	(mm)
SE:	security height	(mm)

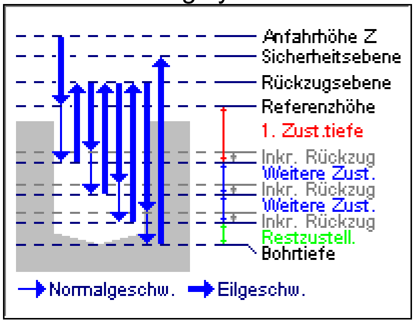
☐ Explanation:

● all drilling parameters for the drilling command **DRILL** are defined
● drilling parameter are modal, that means the parameter are valid so long as they will be set again to another value
● at the beginning of the program standard parameters can be defined, single parameters can be modified immediately before the DRILL command

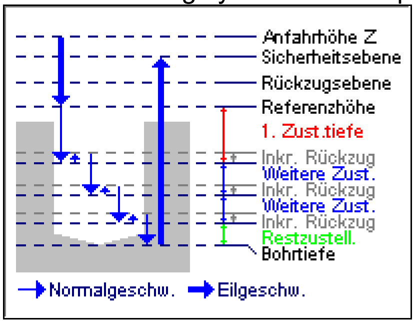
CY =1 Drilling cycle - single drilling



☐ Explanation: **CY = 2** Drilling cycle - clear out



☐ Explanation: **CY = 3** Drilling cycle - break chip



☐ Example:

; CY: drilling cycle = 1 (single drilling)
; PL: plane = 0 (XY)
; DI: direction = 0 (Standard)
; RF: reference height = 1 mm
; DE: depth = 4 mm
; TI: delay after reaching the depth = 20 s
; VE: velocity = 3 mm/s
; VF: fast velocity = 20 mm/s
; FI: first incremental depth = 4 mm
; OT: further increment drill depth = 10 mm
; IC: decrease of increment drill depth = 7 mm
; RE: increment retreat = 4 mm
; LE: retreat plane out of drill hole = 3 mm
; SE: security height = 5 mm

ISO: N10 **G80** CY1 PL0 DI0 RF1 DE4 TI20 VE3 VF20 FI4 OT10 IC7 RE4 LE3 SE5

PAL: N10 **DrillDef** CY1 PL0 DI0 RF1 DE4 TI20 VE3 VF20 FI4 OT10 IC7 RE4 LE3 SE5

☐ Reference:

G81, G82, G83, G84	DrillN, DrillT, DrillD, DrillB

3.1.1.19 Start drilling cycle

G81 **G82** **G83** **G84**	**Single drilling** **Drilling with dwell** **Drillling** (mode **countersick)** **Drilling** (mode **break chip)**	**DrillN** **DrillT** **DrillD** **DrillB**

☐ Syntax:

[set number]?	[set number]?
G81 *or* **G82** *or* **G83** *or* **G84**	**DrillN** *or* **DrillT** *or* **DrillD** *or* **DrillB**
coordinates {x,y}	coordinates {x,y}

ISO:
G81: Single Drilling
G82: Drilling with delay
G83: Drilling mode countersick
G84: Drilling with break chip

PAL:
DrillN: Single Drilling
DrillT: Drilling with dwell
DrillD: Drilling mode countersick
DrillB: Drilling with break chip

☐ Explanation:

- start of drilling cycle
- the required parameters for the drilling process have to be defined in the command **G80 | DrillDef** .
- according to the identification in the DRILL command (**G81, G82, G83, G84 | DrillN, DRILLT, DRILLD, DRILLB**) the values are used out of the definition with **G80 | DrillDef**

☐ Example:

; single drilling at position X=20, Y=100

ISO: N10 G81 X20 Y100
PAL: N10 **DRILLN** X20 Y100

☐ Reference:

G80	DrillDef

3.1.1.20 Coordinate statement

G90-command **G91-command**	**Coordinate statements** are absolute statements (**absolute measure**) **Coordinate statements** are relative statements (**incremental measure**)	**ABS-command** **REL-command**

☐ Syntax:

[set number]? [further command: G17, G18, G19, G53, G54, G55, G70, G71]* **G90** *or* **G91** [F-command]? [S-command]? [M-command]*	[set number]? [further command: PLANE XY, PLANE XZ, PLANE YZ, WPCLEAR, WPREG1, WPREG2, INCH, METRIC]* **ABS** *or* **REL** [F-command]? [S-command]? [miscellaneous command]*

☐ Explanation:

G90 | ABS: all target coordinates are absolute statements (absolute measure)

G91 | REL: all target coordinates are relative statements (incremental measure)

☐ Example:

Cartesian Kinematics (XYZ):
; straight line in space to the absolute target point with the
; coordinates (100 mm, 200 mm, 300 mm) with
; current velocity:

ISO: N100 G01 **G90** X100.0 Y200.0 Z300.0
PAL: N100 **MOVEABS** X100.0 Y200.0 Z300.0

Cartesian Kinematics (XYZ):
; straight line in space to the target point with the coordinates
; X-START + 10mm, Y-START + 20 mm, Z-START - 30mm
; with current velocity:

ISO: N200 G01 **G91** X10.0 Y20.0 Z-30.0
PAL: N200 **MOVEREL** X10.0 Y20.0 Z-30.0

☐ Reference:

G0, G1, G2, G3	FASTABS, MOVEABS, CWABS, CCWABS

3.1.1.21 Set memory

G92-command	**Set memory** (work piece zero point register 1)	**WPREG1WRITE-command**
G93-command	**Set memory** (work piece zero point register 2)	**WPREG2WRITE-command**

☐ Syntax:

[set number]?	[set number]?
[further command:	[further command:
G17, G18, G19,	PLANE XY, PLANE XZ, PLANE YZ,
G70, G71, G90, G91]*	INCH, METRIC, ABS, REL]*
G92 *or*	**WPREG1WRITE** *or*
G93	**WPREG2WRITE**
[target coordinates]{1,6} *or*	[target coordinates]{1,6} *or*
Frame-name	Frame-name
[F-command]?	[F-command]?
[S-command]?	[S-command]?
[M-command]*	[miscellaneous command]*

☐ Explanation:

G92| WPREG1WRITE: set zero point register 1

with an ensuing G54-command | WPREG1-command a new zero point shift can be activated; the absolute values for each coordinate are written into the so-called zero-point-register 1 and this shift is delivered with the following G54-command | WPREG1-command to the motion control

G93| WPREG2WRITE: set zero point register 2

with an ensuing G55-command | WPREG2-command a new zero point shift can be activated; the absolute values for each coordinate are written into the so-called zero-point-register 2 and this shift is delivered with the following G55-command | WPREG2-command to the motion control

☐ Example:

; set zero point shift register 1 (the coordinates of the zero point are directly defined in the command):

ISO: N10 **G92** X100 Y200 Z300
PAL: N10 **WPREG1WRITE** X100 Y200 Z300

; zero point shift activate:
ISO: N20 G54
PAL: N20 WPREG1

; set zero point shift register 2 (the coordinates of the zero point are defined
; in a frame with a frame name in the current geometry file):

ISO: N10 **G93 NULLPUNKT1** ; load register 2
N20 G55 ; activate zero point

PAL: N10 **WPREG2WRITE NULLPUNKT1** ; load register 2
N20 WPREG2 ; activate zero point

☐ Reference:

G53, G54, G55, G56	WPCLEAR, WPREG1, WPREG2, WPZERO

3.1.1.22 Manipulation of technology variables

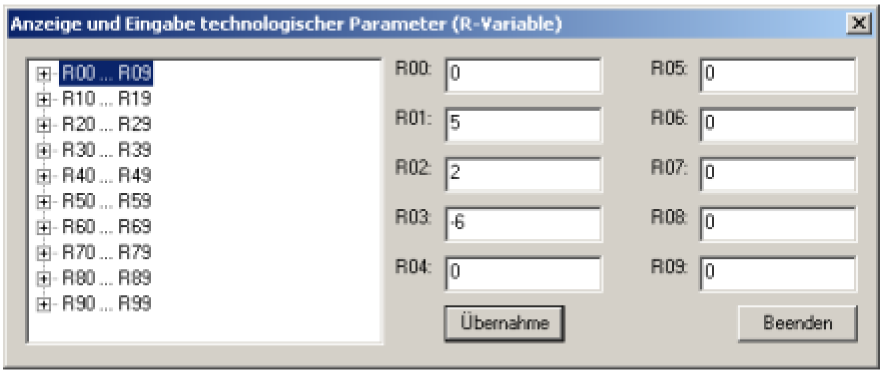
G98-command	**Command to manipulation of technology variables (R-variable)**	**PARAMETER-command**
☐ Syntax:	[set number]? **G98**	[set number]? **PARAMETER**

☐ Explanation:

● if an input or an update of R-variables (technology variables) shall be carried out in a user program in a certain constellation, then this necessary interaction can be programmed with a G98 command | PARAMETER command

● the following three activities follows in temporal order during the program execution, when the commands G98 | PARAMETER will be carried out:
❶ interrupt of program processing
❷ activation of dialog box to display the current values respectively to change values of technology variables
❸ return to the interpreter mode and program continuation

The G98-command | PARAMETER-command can also be used, to wait for a keyboard input, when the user file is processing. If the operator presses the ESC key during the program processing the program will be continued after the activated G98-command | PARAMETER-command.



☐ Example:

ISO:

```
IF R1 == 100
  N5 G98     ; input of new technology values
ENDIF

; wait on the ESC key on he keyboard:
N100 G98
```

PAL:

```
IF R1 == 100
  N5 PARAMETER     ; input of new technology values
ENDIF

; wait on the ESC key on he keyboard:
N100 PARAMETER
```

☐ Reference: Section 3.1.2.16 Dialog field to assign a value to a R-variable

3.1.1.23 Text output

G99-command	Text output into the status line	TYPE-command

☐ Syntax:

	G99	TYPE
	[set number]?	[set number]?
	G99 text	**TYPE text**

☐ Explanation:

text: arbitrary ASCII-text, at most 70 signs long

● the text, following the command G99 | TYPE will be provided in the status line (this is the screen line above the Windows Task bar) as program information during the program processing
Old text or program information will be overwritten.

☐ Example:

; output of an operator request:
IF E1.1
; the test, if a clamping device is closed, was positive:
; no text output !

ISO:
```
ELSE
  G99 please close the clamp device !
ENDIF
```

PAL:
```
ELSE
  TYPE please close the clamp device !
ENDIF
```

; text output, that an edge is milling:

ISO:
```
N100 M3       ; spindle on
N110 G99      ; edge is milling ...
N120 G1 X100  ; the milling process
```

PAL:
```
N100 SCLW      ; spindle on
N110 TYPE      ; edge is milling ...
N120 MOVEABS X100  ; the milling process
```

3.1.2 Miscellaneous commands

In the following table all miscellaneous commands, used in ProNC, are summarized:

Miscellaneous command	Meaning	Miscellaneous command
M0	Programmed program interruption (abort)	ABORT
M1	Programmed program interruption (stop)	QUIT
M3	Spindle switch on (clockwise)	SCLW Spindle Cw
M4	Spindle switch on (counter clockwise)	SCCLW Spindle Ccw
M5	Spindle switch off	SOFF Spindle off
M8/M9	Coolant on/off	Coolant on/off
M10/M11	Workpiece clamp on/off	WpClamp on/off
	Pump on/off	Pump on/off
	Lamp on/off	Lamp on/off
	Periphery option 1 on/off Periphery option 2 on/off	Poption 1 on/off Poption 2 on/off
	Hand mode off/on	HOFF/HON
	Test-mode off/on	TOFF/TON
M17	Return from subprogram	RETURN
M30	Program end	PROGEND
	Get input/output	GetPort GetP GetBit
Mpby	Set output	SetPort SetP SetBit SetAnalog SetPWM
	Get actual value	PosA.n GetDate GetTime GetValue

Table 3.1.2: Miscellaneous functions in ProNC (overview)

3.1.2.1 Program interruption

M0-command **M1-command**	**Programmed program interruption (abort)** **Programmed program interruption (stop)**	**ABORT-command** **QUIT-command**

☐ Syntax:

[set number]?	[set number]?
M0 *or* **M1**	**ABORT** *or* **QUIT**

☐ Explanation:

M0 | ABORT: Program processing is aborted
M1 | QUIT: Program processing is paused

● using the M0-command | ABORT-command the program processing will be aborted in every case after an operator receipt
● using the M1-command | QUIT-command the program processing can be aborted after an operator input (ESC key on the keyboard) or it will be continued (CR key on the keyboard)

☐ Example:

; if the binary input E1.1 is set, the actual user program shall be aborted:

ISO:
```
IF E1.1
  M0        ; unconditional program abort
ENDIF
```
PAL:
```
IF E1.1
  ABORT  ; unconditional program abort
ENDIF
```

; if the binary input E2.2 is set and the variable R1 has the value 100
; the actual user program can be aborted or continued:

ISO:
```
IF E2.2
  IF R1 == 100
    M1  ; program pause with the possibility to continue
  ENDIF
ENDIF
```
PAL:
```
IF E2.2
  IF R1 == 100
    QUIT  ; program pause with the possibility to continue
  ENDIF
ENDIF
```

3.1.2.2 Program beginning, program end

% **M30-command**	**Program beginning** **Program end**	**ProgBegin** **ProgEnd**

☐ Syntax: **%** *or* **M30** | **ProgBegin** *or* **ProgEnd**

☐ Hint: The program beginning marks always the entry of the main program.
In front of the main program subprograms can be declared. These subprograms can be called in the main program.

☐ Explanation: **% | ProgBegin: program beginning**
M30 | ProgEnd: program end

☐ Reference: Section 3.1.7 Subprogram technology

3.1.2.3 Spindle commands

M3-command	**Spindle switch on (clockwise clw)**	**SCLW-command**
M4-command	**Spindle switch on (counter clockwise cclw)**	**SCCLW-command**
M5-command	**Spindle switch off**	**SOFF-command**

☐ Syntax:	[set number]? [further command: [coordinates]{0,6} [F-command]? [S-command]? **M3** or **M4** or **M5**	[set number]? [further command: [coordinates]{0,6} [F-command]? [S-command]? **SCLW** or **SCCLW** or **SOFF**

☐ Explanation:

M3 | SCLW: Spindle switch on
- clw - clockwise

M4 | SCCLW: Spindle switch on
- cclw – counter clockwise

- the number of revolutions of the spindle is defined with the S-command
- although the M command is located at the end of the NC set, the spindle turn on is started before an axis motion starts
- using the commands M3 | SCLW and M4 | SCCLW a defined turn on period of the working spindle is waited, if a starting delay/run-up period was defined in the initialization file of the selected spindle.DLL (SETUP dialog)

M5 | SOFF: Spindle switch off

- using the command M5 | SOFF a defined turn off of the working spindle is waited, if a turn off delay was defined in the initialization file of the selected spindle.DLL (SETUP dialog)

please refer to:
Section 3.1.5 S-command

☐ ***Hint:*** *Equal to the described syntax you can also use the following notation:*

	Spindle command	**Spindle**

☐ Syntax: [set number]?

Spindle CW, RPM, [CCW], [RPS], [ON], [OFF], [TIME milliseconds]

☐ Explanation: **Spindle CW: Spindle on** clockwise

Spindle CCW: Spindle on counter clockwise

Spindle ON: Spindle on in the last declared mode (cw or ccw)

Spindle OFF: Spindle off

Parameter RPM: Define spindle speed in revolutions per minute
Parameter RPS: Define spindle speed in revolutions per second

A delay in the program for the turn on /off of the spindle to rated speed can be defined with the parameter TIME.

☐ Example: ; spindle switch on clockwise with spindle speed 5.000 revolutions/min,
; wait for 5 seconds
N10 **Spindle** CW RPM**5000** TIME 5000

3.1.2.4 Coolant

M8-command/ **M9-command**	**Coolant on** **Coolant off**	**COOLANT ON** **COOLANT OFF**

☐ Syntax:

[set number]? [further command: [coordinates]{0,6} [F-command]? [S-command]? **M8** *or* **M9**	[set number]? [further command: [coordinates]{0,6} [F-command]? [S-command]? **COOLANT ON** *or* **COOLANT OFF**

☐ Explanation:

M8 | COOLANT ON: Coolant **on**
M9 | COOLANT OFF: Coolant **off**

- although the M command is located at the end of the NC set, the command is started before an axis motion starts

☐ Hint:

The assignment, which binary output the coolant switches on or off, is carried out in the dialog Setup - Control - I/O-modules - Extended settings - - Peripherals.

please refer to:
Operating Instruction: 5.8.7.3 Menu Control - Input-/ Output module

3.1.2.5 Workpiece clamp

M10 **M11**	**Work piece clamp on** **Work piece clamp off**	**WPCLAMP ON** **WPCLAMP OFF**

☐ Syntax:

[set number]? **M10** **M11**	[set number]? **WPCLAMP ON** **WPCLAMP OFF**

☐ Explanation:

activate / deactivate a work piece clamping equipment

M10 | WPCLAMP ON: work piece **clamp on**
M11 | WPCLAMP OFF: work piece **clamp off**

☐ Hint:

The assignment, which binary output the coolant switches on or off, is carried out in the dialog Setup - Control - I/O-modules - Extended settings - - Peripherals.

please refer to:
Operating Instruction: 5.8.7.3 Menu Control - Input-/ Output module

3.1.2.6 Pump

	Pump on **Pump off**	**PUMP ON** **PUMP OFF**

☐ Syntax: [set number]?

PUMP ON
PUMP OFF

☐ Explanation: switch on / switch off of a vacuum equipment or a hydraulic pump

PUMP ON: Pump on
PUMP OFF: Pump off

☐ Hint: The assignment, which binary output the pump switches on or off, is carried out in the dialog Setup - Control - I/O-modules - Extended settings -- Peripherals.

please refer to:
Operating Instruction: 5.8.7.3 Menu Control - Input-/ Output module

3.1.2.7 Lamp

	Lamp on **Lamp off**	**LAMP ON** **LAMP OFF**

☐ Syntax: [set number]?

LAMP ON
LAMP OFF

☐ Explanation: switch on / off a lamp / illumination

LAMP ON: lamp **on**
LAMP OFF: lamp off

☐ Hint: The assignment, which binary output the lamp switches on or off, is carried out in the dialog Setup - Control - I/O-modules - Extended settings -- Peripherals.

please refer to:
Operating Instruction: 5.8.7.3 Menu Control - Input-/ Output module

3.1.2.8 Periphery option

	Periphery option1 on / off **Periphery option2 on / off**	**POPTION1 ON / OFF** **POPTION2 ON / OFF**

☐ Syntax:

	[set number]?
	POPTION1 ON/POPTION1 OFF **POPTION2 ON/POPTION2 OFF**

☐ Explanation**:** switch on / switch off an optional device (possibility to connect a user specific hardware to a corresponding output port)

POPTION1 ON: Periphery device 1 **on**
POPTION1 OFF: Periphery device 1 **off**

POPTION2 ON: Periphery device 2 **on**
POPTION2 OFF: Periphery device 2 **off**

☐ Hint: The assignment, which binary output the optional device switches on or off, is carried out in the dialog Setup - Control - I/O-modules - Extended settings -- Peripherals**.**

please refer to:
Operating Instruction: 5.8.7.3 Menu Control - Input-/ Output module

3.1.2.9 Hand-/Test-Mode

	Hand mode switch off /switch on **Hand mode switch off /switch on**	**HOFF/HON-command** **TOFF/TON-command**

☐Explanation: The hand mode respectively the test mode can be switched on or switched off in the user program.
The switched on hand mode causes, that the motor amplifier will be switched current free.

The test mode is to be switched on by program, if e. g. a 4th axis (A-axis) shall carry out a reference run without the existence of a reference switch.

☐ Hint: Not all motor amplifiers/motion controls offer the possibility, to switch on or switch off a hand mode or a test mode.

☐ Example:

✏ **PAL:** **HOFF**: Hand mode switch off
HON: Hand mode switch on
TOFF: Test mode switch off
TON: Test mode switch on

3.1.2.10 Get inputs / outputs

	Get inputs- /outputs	**GetPort / GetP**

☐ Syntax:

	[set number]?
	r_variable=GetPort Ep/ **r_variable=GetP Ep** **r_variable=GetPort Ap /** **r_variable=GetP Ap**

☐ Explanation: Read the value either of a logical input port p (parameter Ep) or the current value of an output port p (parameter Ap).

☐ Example:

```
N10  R11=GetPort E1   ;read the input port 1
N15  R12=GetPort A1   ;read the current value of the output port 1
```
or
```
N10  R11=GetP E1      ;read the input port 1
N15  R12=GetP A1      ;read the current value of the output port 1
```

3.1.2.11 Set outputs

Mpby-command	**Set output port** **(y=1| SETB) set respectively** **(y=0| RESB) reset**	**SetPort / SetP** **SetBit** **ResBit**

☐ Syntax:	[set number]?	[set number]?
	[Mpby]+	**SetPort Ap**=constant **SetP Ap**=constant **SetBit Ap.b=y** **SetBit Ap.b ; set always** **ResBit Ap.b ; reset always**

☐ Explanation: **Mpby:** general M-command to set / reset binary outputs:

● the letter **p** represents the number 1, 2 ... 8;
this number defines the corresponding output port;
● an output port includes always the eight output bits 1 to 8
● the letter **b** represents the number 1, 2 ... 8,
this number defines the corresponding bit in the output port;
● the output ports correspond to the P-variables with an even index:
.

ISO:
M1by refers to output port 1 = P0
M2by refers to output port 2 = P2
M3by refers to output port 3 = P4
M4by refers to output port 4 = P6
M5by refers to output port 5 = P8
M6by refers to output port 6 = P10
M7by refers to output port 7 = P12
M8by refers to output port 8 = P14

● the letter **y** represents a number 0 or 1,
this number defines, if the bit selected with **b**, shall be set (y = 1) or shall be reset (y = 0)

PAL:
SetBit / ResBit A1.b refers to the output port 1 = P0
SetBit / ResBit A2.b refers to the output port 2 = P2
SetBit / ResBit A3.b refers to the output port 3 = P4
SetBit / ResBit A4.b refers to the output port 4 = P6
SetBit / ResBit A5.b refers to the output port 5 = P8
SetBit / ResBit A6.b refers to the output port 6 = P10
SetBit / ResBit A7.b refers to the output port 7 = P12
SetBit / ResBit A8.b refers to the output port 8 = P14

please refer to:
Section 3.2.1.1: P-variable

☐ Example:

; set the output port 1 - binary notation:
SetPort A1=11110000B
; set the output port 1 - hexadecimal notation:
SetPort A1=0xF0
or
SetPort A1= $F0
or
SetPort A1=F0H

please refer to:
Section 3: ProNC language description

☐ Example:

ISO:

; set bit **5** in the output port 1:
N10 M1**5**1

; reset bit **7** in the output byte 2:
N20 M2**7**0

PAL:

; set bit **5** in the output byte 1:
N10 SetBit A1.5; *or:*
N10 SetBit A1.5=1

; reset bit **7** in the output byte 2:
N20 ResBit A2.7; *or:*
N20 SetBit A2.7=0

3.1.2.12 Set Analog-/PWM-output

	Set Analog output, **Set PWM output**	**SetAnalog** **SetPWM**

☐ Syntax:

[set number]?

SetAnalog Ak = voltage value [mV]
SetPWM Ak **=** pulse width [%]

☐ Explanation:

- Set the value of the analog output with the submitted channel number. The unit of the analog value is millivolt. The chosen analog output must be declared in the extended IO administration.

- Set the value of the PWM signal with the submitted channel number k [1 ...4]. The unit of the pulse width is percent. The chosen PWM channel must be declared in the extended IO administration.

☐ Example:

```
; set the analog output 1 to 2,5 volt
N10  SetAnalog A1=2500

; set the pulse width on PWM output 1 to 50%
N10  SetPWM A1=50     ;pulse duty ratio 50%
```

3.1.2.13 Current axis position

	Current axis in **axis system 1 or 2**	**POSn.A**

☐ Syntax:

[set number]?

r_variable = POSn.A

n=[1,2]
A=[X, Y, Z, A, B, C]

☐ Explanation:

Function to query the current position of the axis **A** of the axis system **n**. The axes systems 1 and 2 can be indexed. The letter A can be replaced with the axis marker X, Y, Z, A, B, C.

☐ Example:

```
R11=POS1.X    ;R11 contains the position X in the axis system 1
R12=POS1.Y    ;R12 contains the position Y in the axis system 1
R21=POS2.X    ;R21 contains the position X in the axis system 2
```

3.1.2.14 Current system time

	Current system time	**GetTime**

☐ Syntax:

	[set number]?
	r_variable0=GetTime r_variable1 r_variable2 r_variable3

☐ Explanation: Query the current system time. After the call of this function these values are available in the R-variables:

r_variable1: = hour
r_variable2: = minute
r_variable3:= second

☐ Example: N10 R0=GetTime R11 R12 R13; R11=hour, R12=minute, R13=second

☐ Hint: Only if after finishing the command GetTime R0 owns the value 0, the values in R11 (hour), R12(minute) and R13 (second) are correctly.

3.1.2.15 Current date

	Current date	**GetDate**

☐ Syntax:

	[set number]?
	r_variable0=GetDate r_variable1 r_variable2 r_variable3

☐ Explanation: Query the current date. After the call of this function these values are available in the R-variables:

r_variable1: = year
r_variable2: = month
r_variable3: = day

☐ Example: N10 R0=GetDate R1 R2 R3; R1=year, R2=month, R3=day

☐ Hint: Only if after finishing the command GetDate R0 owns the value 0, the values in R1 (year), R2(month) and R3 (day) are correctly.

3.1.2.16 Dialog field to assign a value to a R-variable

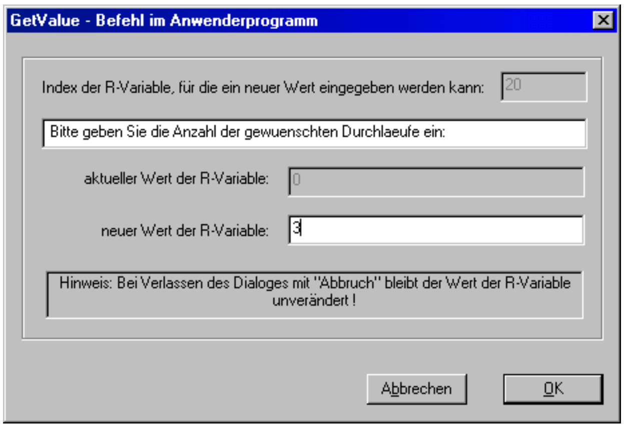
	Dialog field to assign a value to a R-variable	**GetValue**

☐ Syntax: [set number]?

r_variable=GetValue "<message text>"

☐ Explanation: Display of a dialog field: The user can enter a value, this value is assigned to a R-variable.

R20=GetValue "Please enter the number of the desired repetitions"



3.1.3 FastVel-command

	Fast velocity in mm/sec	FASTVEL-command
☐ Syntax:	[set number]? [further command]* [coordinates]{0-6} FastVel velocity [S-command]? [M-command]*	[set number]? [further command]* [coordinates]{0-6} **Fastvel velocity** [S-command]? [miscellaneous command]*

☐ Explanation:

- **velocity** is a decimal number
- with this command the fast velocity will be defined
- the unit of the fast velocity is always mm / sec at Cartesian plants; if the first axis at the plant is a rotary axis the unit is grad / sec
- at plants with rotary axes and at least with one linear axis the adjustment velocity of the rotary axes will be calculated by the interpolator according to the "Leading velocity" of the linear axis.

☐ Example: ; adjust fast velocity 100 mm/sec:

ISO: N10 G0 X100 Y200 Z300 **FASTVEL 100.0**
PAL: N10 FASTABS X100 Y200 Z300 **FASTVEL100.0**

☐ Reference:	G0	FASTABS

3.1.4 F-command

F-command	**Processing velocity in mm/sec**	**VEL-command**

☐ Syntax:

[set number]? [further command]* [coordinates]{0-6} **F velocity** [S-command]? [M-command]*	[set number]? [further command]* [coordinates]{0-6} **VEL velocity** [S-command]? [miscellaneous command]*

☐ Explanation:

- **velocity** is a decimal number
- with this command the processing velocity will be defined
- the unit of the processing velocity is always mm / sec at Cartesian plants
- plants with rotary axes and at least with one linear axis the adjustment velocity of the rotary axes will be calculated by the interpolator according to the "Leading velocity“ of the linear axis

☐ Example:

; adjust processing velocity 100 mm/sec:

ISO: N10 G1 X100 Y200 Z300 **F100.0**
PAL: N10 MOVEABS X100 Y200 Z300 **VEL100.0**

☐ Reference:

G1, G2, G3 G11, G12, G13	MOVEABS, CWABS, CCWABS MOVEFRAME, CWHLXABS, CCWHLXABS

3.1.5 S-command

S-command	**Spindle speed define in revolutions /min**

☐ Syntax:	[set number]? [further command]* [coordinates]{0-6} [F-command]? **S speed** [M-command]*	[set number]? [further command]* [coordinates]{0-6} [F-command]? **S speed** [miscellaneous command]*

☐ Explanation:

- the spindle speed is defined with this command: **spindle speed** is a decimal number and has the unit revolutions per minute
- the direction of rotation of the spindle **clw** (clockwise) is defined with the M-command M3 | SCLW.
- the direction of rotation of the spindle **cclw** (counter clockwise) is defined with the M-command M4 | SCCLW.

☐ Reference:	M3, M4, M5	SCLW, SCCLW, SOFF

3.1.6 Tool change

T-command **T1-command** **T2-command**	**Tool change**	**GetTool** **GetTool TC1** **GetTool TC2**

☐ Syntax:

[set number]?	[set number]?
T tool number **T1=tool place** **T2 =tool place**	**GetTool tool number** **GetTool TC1 tool place** **GetTool TC2 tool place**

☐ Explanation:

- **tool number** {1-128}
- **tool place** {1-16}
- per tool changer (max. two) max.128 tools are configurable for at most 16 tool places

Using a tool number without the parameter TC1 respectively TC2 the tool with the number, defined in the tool administration, will be changed.

Using the tool place and the further parameters TC1 or TC2 to define the tool changer the tool will be fetched, deciding on the tool place in the corresponding tool changer.
The tool administration will be avoided, if you use this command.

☐ Example:

; tool with the tool number 4, defined in the tool administration, is fetched

ISO: N10 T 4
PAL: N10 GetTool 4

; tool from tool place 4 in the tool changer 1 is fetched

ISO: N10 T1=4
PAL: N10 GetTool TC1 4

3.1.7 Subprogram technology

Subprogram technology: With help of subprogram technology it enables to the user to create and to test compact and easily comprehensible application programs successfully .

The following boundary conditions have to be taken into account using the subprogram technology in ProNC:

Declaration force of subprograms: Subprograms have to be first declared in the program text (to declare) before they are called in the main program. The reason for this is, that the compiler checks, if forbidden subprogram calls are found in the application program. A forbidden subprogram call happens, when the corresponding subprogram was not yet declared.

Subprograms can be nested . So It is possible, that in a subprogram another subprogram is called.

Maximum nested depth: The maximum nested depth is defined with 10. The restriction on 10 is meaningful with safety and results only from the fact, that the stack to the return addresses is limited to exactly this value.

3.1.7.1 Declaration subprogram

%L declaration	**Subprogram declaration**	**%SUBR declaration:**

☐ Syntax: **%L subprogram_number** | **%SUBR subprogram_number**

☐ Explanation:

Subprogram declaration:

- a subprogram is declared (that means it is "agreed" or "announced")
- **subprogram_number** is a natural number
- the special sign % and the address letter **L | Key word SUBR** must stand in front of the identification of a subprogram declaration
- the number of the subprogram (declared) serves for the clear identification and may be found in a source program only once, (however, a declared subprogram can be arbitrarily often called in a user program)
- each declared subprogram has to be completed with the command **M17 | RETURN**

☐ Example:

ISO:

```
; the subprogram with the number 11 is declared
; and is completed with the according M17-command:
%L11

; set bit 1 in the output byte A1:
N10 M111

; wait 1 sec:
```

```
N20 G4 1000

; reset bit 1 in the output byte A1:
N30 M110

; finish the subprogram:
N40 M17
```

PAL:

```
; the subprogram with the number 11 is declared
; and is completed with the according RETURN-command:
%SUBR11

; set bit 1 in the output byte A1:
N10 SETB A1.1

; wait 1 sec:
N20 TIME 1000

; reset bit 1 in the output byte A1:
N30 RESB A1.1

; finish the subprogram:
N40 RETURN
```

☐ Reference:

Subprogram call: **L**	Subprogram call: **SUBR**

3.1.7.2 Subprogram call

L-command	**Subprogram call (direct)** **Subprogram call (indexed)**	**SUBR-command**
☐ Syntax:	**direct subprogram call:** [set number]? **L subprogram_number** **indexed subprogram call:** [set number]? **L r_variable**	**direct subprogram call:** [set number]? **SUBR subprogram_number** **indexed subprogram call:** [set number]? **SUBR_ r_variable**

☐ Explanation:

● a subprogram must be declared ("agreed") in front of its call (its activation)

● the number of the subprogram serves for the clear identification, a declared subprogram can be called in the (main) program in arbitrarily many places

direct subprogram call:
● a subprogram is called directly as follows: After an optional set number the address letter **L | Key word SUBR**, followed by a natural number, is programmed in the main program or in another subprogram.

indexed subprogram call:
● a subprogram is called **indexed**, if after an optional set number the address letter **L | Key word SUBR**, followed by a R-variable, will be programmed in the user main program or in another subprogram
● indexed subprogram call increases the flexibility of the programming considerable, because just that subprogram is activated, which number agrees with the current content of the corresponding R-variable

☐ Example:

ISO:

```
; the subprogram with the number 5 is declared:
%L5                 ; subprogram to grip
N10 M8              ; close gripper
N20 G4 1000         ; wait 1 sec until closed
N30 M17             ; return

; the subprogram with the number 6 is declared:
%L6                 ; subprogram to clamp off
N10 M9              ; open gripper
N20 G4 2000         ; wait 2 sec, until opened
N30 M17             ; return

%100                ; start of the main program
N10 ...
N20 ...
N30 L5              ; activate subprogram 5
N40 ...
N50 L6              ; activate subprogram 6
N60 ...

; example to indexed subprogram call:
; the three subprograms %L10, %L11 und %L12 are declared:
%L10
N100 R1 = 1  R2 = 2  R3 = 3
N200 M17
```

```
%L11
N100 R1 = 5  R2 = 6  R3 = 7
N200 M17
%L12
N100 R1 = 10 R2 = 11 R3 = 12
N200 M17
;
; in the main program a parameter input is made (with keyboard):
N5  G98
; then the R-variable R1 has the value of the wished
; subprogram number:
N10 LR1   ; the subprogram is activated, which
          ; subprogram number agrees just now with the
          ; current content of R-variable R1
```

Example: **PAL:**

```
; the subprogram with the number 5 is declared:
%SUBR5                  ; subprogram to grip
N10 GCLOSE              ; close gripper
N20 TIME 1000           ; wait 1 sec until closed
N30 RETURN              ; return

; the subprogram with the number 6 is declared:
%SUBR6                  ; subprogram to clamp off
N10 GOPEN               ; open gripper
N20 TIME 2000           ; wait 2 sec until opened
N30 RETURN              ; return

%100                    ; start of the main program
N10 ...
N20 ...
N30 SUBR5               ; activate subprogram SUBR5
N40 ...
N50 SUBR6               ; activate subprogram SUBR6
N60 ...

; example to indexed subprogram call:
; the three subprograms %SUBR10, %SUBR11 and %SUBR12  are
; declared:
%SUBR10
N100 R1 = 1  R2 = 2  R3 = 3
N200 RETURN
%SUBR11
N100 R1 = 5  R2 = 6  R3 = 7
N200 RETURN
%SUBR12
N100 R1 = 10 R2 = 11 R3 = 12
N200 RETURN
;
; in the main program a parameter input is made (with key board):
N5  PARAMETER
; then the R-variable R1 has the value of the wished subprogram
; number
N10 SUBR_R1   ; this subprogram is activated, which
              ; subprogram number agrees just now with the
              ; current content of the  R-variable R1
```

☐ Reference:

Subprogram declaration: **%L**	Subprogram declaration: **%SUBR**

3.2 Instructions: Syntactic extensions to DIN 66025

3.2.1 Variables

In ProNC variables have an elementary importance for the possibility, to generate flexible application programs. Variables represent the basis for the parameter calculation.

Variable: A variable must be in the position to be *named* in the program text. That means in an application program a variable is represented by a *name*.

In ProNC very simple names are chosen for the available variables: a natural number **n** follows a capital letter **P**, **Q** or **R**:
0 <= n < 100 at P, 0 <= n < 500 at Q, 0 <= n < 1000 at R.

No declaration force for variables: ✍

In ProNC variables *don't* have to be declared explicitly. Therefore this isn't necessary because always one hundred P-variables, five hundred Q-variables and one thousand R-variables are available in every source program (the variables are declared implicitly). The assignment of data types to certain variables, how it is usual at programming languages in the EDP, you don`t need. The data type defines always the range of values of a variable. In ProNC fixed data types are assigned to the available variables:

Variable in ProNC	fixed assigned data type
P-variable	**natural number:** memory volume: 8 bit = 1 byte range of values: 0 to 255
R-variable	**floating-point number:** memory volume: 8 byte = 64 bit range of values: 11 bit exponent, 53 bit mantissa
Q-variable	structure of 12 floating-point numbers (X, Y, Z, A, B, C for axis system 1, X2, Y2, Z2, A2, B2, C2 for axis system 2)

<u>Table 3.2.1:</u> Variables in ProNC and their range of values

During the run time of a user program a variable is realized always by a memory position and a memory content.

Memory position: *Position*: Where is the current value of the variable stored physically ?

Memory content: *Value:* Which value does the variable just represent ?

Therefore the value of a variable can be changed at any time (during the runtime of the user program) or it can be assigned to another variable of the same type (valid for P- and R-variables).

Runtime of the user program: The run time is exactly the time, while the application program is processed.

R-variables: ☞

In ProNC the R-variables are the most important variables. With their help it is possible, to carry out calculations and to store the results of the calculations. To carry out decisions R-variables can be compared with each other or with constants.
ProNC owns a basic quality: All available R-variables (R0 to R999) can "submit" their current values as parameters to coordinates or F commands. So an indirect value declaration is possible.

please refer to:
Section 3.2.2 Parameter calculation

☐ Example: ; NC set with direct statement of values:

ISO:	N10 G1 G90 X100 Y200 Z-50
PAL:	N10 MOVEABS X100 Y200 Z-50

✏ ; NC set with indirect statement of values with help of R-variables:

ISO:	N10 G1 G90 XR1 YR2 ZR3
PAL:	N10 MOVEABS XR1 YR2 ZR3

3.2.1.1 P-variables

P-variables	

☐ Syntax: **P-variable_index**

☐ Exlanation: **P**rocess **variables P**0 to **P**99:

- **0 <= variable_index < 100**
- Process variables have a value range 0 to 255 (0x00 to 0xFF)
- a P variable can be combined with help of Boolean operations with other P variables or with constants (natural numbers from **0 to 255** or hexadecimal numbers from **0x00 to 0xFF**)
- the process variables with odd index (P1, P3, P5 to P15) represent at any time the current input ports I1, I2, I3 to I8:
  P1 = I1, P3 = I2, P5 = I3, P7 = I4, P9 = I5, P11 = I6, P13 = I7, P15 = I8

Please note:
P index for input port := I index * 2 - 1
with I index from 1 ... 8

- the process variables with even index (P0, P2, P4 to P14) represent at any time the current output ports O1, O2, O3 to O8:
  P0 = O1, P2 = O2, P4 = O3, P6 = O4, P8 = O5, P10 = O6, P12 = O7, P14 = O8

Please note:
P index for output port := O index * 2 - 2
with O index from 1 ... 8

☐ Example:

P**33** (P-variables with index **33**)
P**66** (P-variables with index **66**)
P**99** (P-variables with index **99**)

```
P0=11110000B    ; write output port O1
P2=0xF0         ; write output port O2
```

☐ Reference:

Section 3.2.1.3: R-variable
Section 3.2.2.3: Boolean expression

3.2.1.2 Q-variables

Q-variables	

☐ Syntax: **Q-variable_index** *or* **QRvariable_index**

☐ Explanation: **Q-variables** own a variable index:

- **0 <= variable_index < 500**
- **Q-variables are initialised with frames, every frame has a name.**

- **Q-variables** represents a structure, consisting of maximum 12 floating-point numbers for the elements X ... C in the axis system 1 respectively X2 ... C2 in the axis system 2

- with the following construction can be accessed to the structure elements (the coordinate values of the axes):

Q-variable_index : axis_address letter

Each component of a Q-variable can be isolated with help of the axis identifier (address letter):

Q-variable_index : X describes **the X coordinate in the axis system 1**
Q-variable_index : Y describes **the Y coordinate in the axis system 1**
Q-variable_index : Z describes **the Z coordinate in the axis system 1**
Q-variable_index : A describes **the A coordinate in the axis system 1**
Q-variable_index : B describes **the B coordinate in the axis system 1**
Q-variable_index : C describes **the C coordinate in the axis system 1**

Q-variable_index : X2 describes **the X2 coordinate in axis system 2**
Q-variable_index : Y2 describes **the Y2 coordinate in axis system 2**
Q-variable_index : Z2 describes **the Z2 coordinate in axis system 2**
Q-variable_index : A2 describes **the A2 coordinate in axis system 2**
Q-variable_index : B2 describes **the B2 coordinate in axis system 2**
Q-variable_index : C2 describes **the C2 coordinate in axis system 2**

	Q1	
the coordinate values for the axis system 1 →	Q1: X Q1: Y Q1: Z Q1: A Q1: B Q1: C	← The structure of Q-variables contains 12 components, each 6 for the axis system 1 respectively 6 for the axis system 2
the coordinate values for the axis system 2 →	Q1: X2 Q1: Y2 Q1: Z2 Q1: A2 Q1: B2 Q1: C2	

● coordinate values, found out by Teach-In, can be stored into the frame structure

● a Q-variable can be initialised with help of an assignment for example: Q1=TARGET_POINT; TARGET_POINT is the name of a frame in the current geometry file

You can access indirectly to the coordinate values of Q-variable. That means, the index of the wished Q-variable can be defined with help of a R-variable.

☐ Example:

Q**33** (the Q-variable with the index **33**)
Q**66** (the Q-variable with the index **66**)
Q**99** (the Q-variable with the index **99**)

; the value of the X coordinate within the structure of the
; Q-variable Q15 is assigned to the R-variable R14
; (valid for axis system 1):
N100 R14 = Q15:X
N200 R15=Q15:X2; for axis system 2

important detail:
; the value of the Y coordinate within the structure of the
; Q-variable, which index corresponds to the current value of R88
; is transferred to the R-variable R14:
N200 R14 = QR88:Y
To the fore going example the following <u>explanation</u>:
<u>Hypothesis</u>: R88 has just the value 5.
Result: R14 obtains the value, which is in the Q-variable Q5 at the position of the Y-coordinate.

☐ Reference:

Section 3.2.1.3: R-variable
Section 3.2.2.4: Assignments
Operating Instruction: Menu 2.2.2.2 Structure of geometry file

3.2.1.3 R-variables

R-variables	

☐ Syntax: **R-variable_index** *or* **RRvariable_index**

☐ Explanation: **R**eal-**Variables R**0 to **R**999:

● **0 <= variable_index < 1000**

● real variables own the value range of a floating-point number (double precision: 64 bit)

☞ ● in ProNC all 1000 available R-variables R0 to R999 are arranged as array

● a R-variable can be combined with help of arithmetical operators with other R-variables or with constants (decimal numbers)

● R-variables enables the data transfer to a coordinate, to a F- or a S-command

● R-variables can be assigned with a component of a Q-variable (X, Y, Z, A; B, C respectively X2, Y2, Z2, A2, B2, C2)

● you can access to R-variables indirectly, if you use as index of a R-variable a further R-variable e. g. RR6

● the R-variables R101 to R106 represent at any time the current actual position of the axes 1 to 6 in axis system 1:

1. axis : R101, 2. axis : R102, 3. axis : R103
4. axis : R104, 5. axis : R105, 6. axis : R106

respectively in axis system 2:

1. axis : R201, 2. axis : R202, 3. axis : R203
4. axis : R204, 5. axis : R205, 6. axis : R206

Please make sure, that in your user program those R-variables (R101 to R106 or R201 to R206) aren't written.

☐ Example: **indexing of R-variables:**

R**33** (the R-variable with the index **33**)
R**66** (the R-variable with the index **66**)
R**99** (the R-variable with the index **99**)

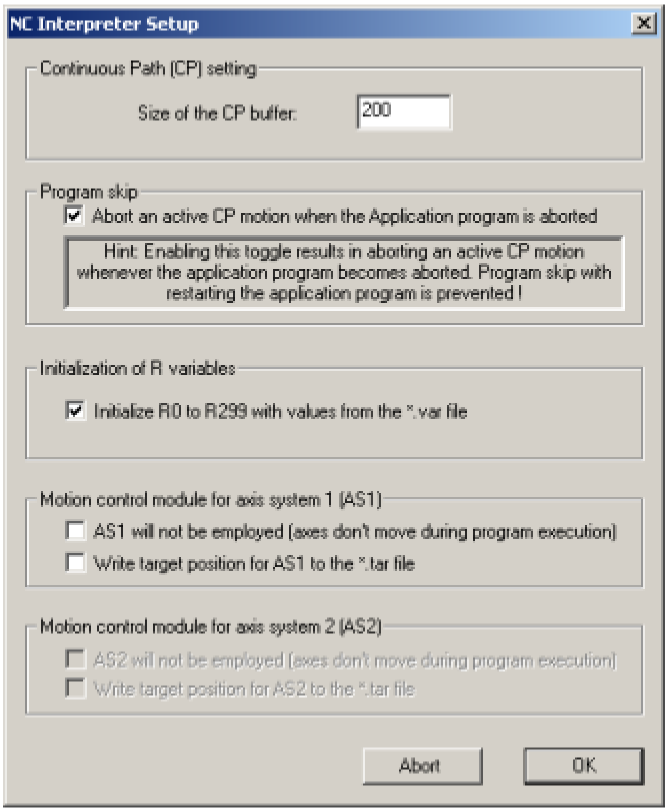
indirect access to R-variable:

```
N01 R1 = 1.11 R2 = 2.22 R3 = 3.33 R4 = 4.44 R5 = 5.55
...
N10 R10 = 4            ; the wished index to R10
N20 R11 = RR10         ; R11 is initialised with the value 4.44
N30 R10 = 5
N40 R11 = RR10         ; R11 is initialised with the value 5.55
```

Initialisation of R-variables with the values from the variables file:

ProNC has the ability to write **the current values of the R-variables R0 to R299** into the special **variable file "*.var"** if the concrete **user program** is **exited** or **aborted**.
That means after the abort of the user program example.pal the current values of the R-variables R0 to R299 are written into the file example.var (in the directory /CNCWorkbench/NCProg/Dest).
At the **next start** these variables will be read again out of the **variable file**, converted and stored **index-right** in the working memory.

If you want to start your user program **without the initialisation** of the variables R0 to R99 with the values from the file *.var, you can switch off the corresponding control small box in the dialog NC Interpreter setup.



You can activate this dialog about the Menu **Setup - Interpreter**.

☐ Reference:

Section 3.2.1.1: P-variable
Section 3.2.1.2: Q-variable
Section 3.2.1.4: Data transfer R-variable to coordinate
Section 3.2.2: Parameter calculation

3.2.1.4 Data transfer R-variable to coordinate

Data transfer	

☐ Syntax: **address letter R variable_index**

☐ Explanation:

- **address letter = {X, Y, Z, A, B, C}**
that means, the address letter is an element of the specified quantity

- **0 <= variable_index < 1000**

☞

- an indirect value declaration at a coordinate word is possible, if instead of a decimal number a R-variable is indicated

- the current value of the R-variable is assigned to the according coordinate

☐ Example:

ISO:
```
N10 R1 = 100
; the target point has the  coordinate X=100mm, Y=200mm:
N20 G1 XR1 Y200

; the R-variable is calculated new:
N30 R1 = R1 + 50

; the target point has now the coordinate X=150mm,Y=200mm:
N40 G1 XR1 Y200
```

PAL:
```
N10 R1 = 100
; the target point has the  coordinate X=100mm, Y=200mm:
N20 MOVEABS XR1 Y200

; the R-variable is calculated new:
N30 R1 = R1 + 50

; the target point has now the coordinate X=150mm,Y=200mm:
N40 MOVEABS XR1 Y200
```

☐ Reference:

G0, G1, G2, G3	FASTABS, MOVEABS, CWABS, CCWABS

Section 3.1.4 F-command

3.2.2 Parameter calculation

3.2.2.1 Arithmetical expressions

arith_expression	**arithmetical expressions**

☐ Syntax:

1. **arith_expression arith_operator arith_expression** *or*
2. **[arith_expression]** *or*
3. **function[arith_expression]** *or*
4. **r_variable** *or*
5. **real_number**
6. **symb_constant**

☐ Explanation:

● an **arithmetical expression** can be combined with help of arithmetical operators with a second arithmetical expression

● the result of this operation is an arithmetical expression again. The value of this expression can be assigned to a **R-variable**.

✍ ● **arith_operator** is a **sign** from the quantity **{+,-,*,/, MODULO},** so that the arithmetical operations addition, subtraction, multiplication, division and modulo division are possible

● to determine the priority, arithmetical expressions can be clipped; if an arithmetical expression has no brackets, the rule is valid "point before line"

● an arithmetical expression can be an argument of a function (trigonometric and real functions, please refer to the next section)

● an arithmetical expression can be a R-variable

● a **real_number** is a decimal number

● a **symb_constant** can be **Pi** = 3.142

● a **symb_constant** is a symbolic constant and can be a **word** out of the quantity **{IDOK, IDCANCEL, IDABORT, IDRETRY, IDIGNORE, IDYES, IDNO}**

The following values are assigned to these symbolic constants:

symb_constant	**value**
IDOK	1
IDCANCEL	2
IDABORT	3
IDRETRY	4
IDIGNORE	5
IDYES	6
IDNO	7

Example: ; 1. a complex arithmetical expression stands on the right side of the assignment; the value of the arithmetical expression will be assigned to the variable R10:

```
N10 R10 = PI / 4 * R11 * R11

; 2. using the bracket:

N20 R12 = [R13 + R14] * R16

; 3. an arithmetical expression as argument of the SIN-function:
N30 R13 = SIN [ 2.0 * R14 ]

; 4. R-variable results from the R-variable with
; sign inversion: this is an assignment

N40 R99 = 0.0-R98

; 5. R-variable results from a decimal number: this is an assignment:

N50 R33 = 123.456

; 6. Modulo division
N10 R2 = 7
N15 R3 = 3
N20 R1 = R2 MODULO R3
; after executing of the program line with the number 20
; R1 has the value 1
```

The Modulo division ...

```
N10 R1 = R2 MODULO 2.0
```

is often used to test the evenness of a R-variable. If R2 is even, the integer rest in R1 is always 0 after doing the modulo division and it can be evaluated correspondingly:

```
IF R1 == 0
  ; R2 was even ...
ELSE
  ; R2 was odd
ENDIF
```

☐ Reference: Section 3.2.2.2 Functions
Section 3.2.2.3 Boolean expression
Section 3.2.2.4 Assignments
Section 3.2.3.3 Selection instruction
Section 3.2.4.1 Request of an operator dialog

3.2.2.2 Functions

Functions	**Functions to calculation of R-variable**

☐ Syntax: **function_name [arith_expression]**

☐ Explanation:

• a function owns always an argument; this argument is placed into a square bracket (at PAL syntax you can also use round brackets)

• the function name defines the concrete function more nearly: trigonometric and real functions are distinguished

	<u>Trigonometrical functions</u>

the arguments of trigonometric functions have to be always indicated into radian measure (rad):

α (radiant) = α (Grad) * Pi / 180.0

SIN

SIN (arith_expression)

Operator for the trigonometric function SINUS

☐ Example:

```
R11=PI/2
R12=SIN(R11)   ;result 1.0
R13=SIN(2*R11)   ;result 0
```

ASIN

ASIN (arith_expression)

Operator for the cyclometric function ARCUSSINUS. The ASIN function is the mathematical inversion of the SINUS function.

☐ Example:

Example.: Calculation of the angle between the opposite leg and hypotenuse:

```
R11=ASIN((R22-R21)/(R32-R31))
```

COS

COS (arith_expression)

Operator for the trigonometric function COSINUS.

☐ Example:

```
R11=PI/2
R12=COS(R11)   ;result 0
R13=COS(2*R11)   ;result -1.0
```

ACOS **ACOS** (arith_expression)

Operator for the cyclometric function ARCUSCOSINUS. The ACOS function is the mathematical inversion of the COSINUS function.

☐ Example: Calculation of the angle between adjacent leg and hypotenuse:

```
R11=ACOS((R25-R24)/(R32-R31))
```

TAN **TAN** (arith_expression)

ATAN Operator for the trigonometrical function TANGENS.

ATAN (arith_expression)

Operator for the cyclometric function ARCUSTANGENS. The ATAN function is the mathematical inversion of the TANGENS function.

☐ Example:

```
N10 R1 = Pi / 4          ; a 45° angle
N20 R2 = TAN[ R1 ]     ; result: R2 gets the value 1.0
N30 R3 = ATAN[ R2 ]   ; result: R3 gets the value 0.7854 = Pi / 4
```

	Real functions

FABS **FABS** (arith_expression)

Function for the calculation of the absolute value of a signed numeric expression.

☐ Example:

```
R2=FABS(R1)   result: amount of  R1
R2=FABS(-5.0)  result: 5
```

SQRT **SQRT** (arith_expression)

Function for the calculation of the square root of a numerical expression.

☐ Example:

```
R2=SQRT(R1)   result: the root out of R1
R2=SQRT(9.0)   result: 3

; Pythagorean theorem:
; R10=3, R11=4
R12 = SQRT [ R10 * R10 + R11 * R11]; result: R12 contains the value 5.0
```

FLOOR **FLOOR** (arith_expression)

Function to round off the argument value. The function calculates the next integer number, which is smaller or equal to the defined arithmetical expression.

☐ Example:

```
R2=FLOOR(2.5)   result: 2
R2=FLOOR(3.9999)   result: 3

N05 R2 = 5.89
N10 R1 = FLOOR[ R2 ]
; after processing the program line with the number 10
; R1 has the value 5.0
```

EXP

EXP (arith_expression)

EXP calculates the exponential function with the base e.

☐ Example:

```
R22=Exp(R21)  ;R-variable as argument
R22=Exp(0)    ;result is 1
R22=Exp(1)    ;result is 2.718282
```

LN

LN (arith_expression)

LN calculates the natural logarithm of the argument.

☐ Example:

```
R22=Ln(R21)    ;R-variable as argument
R22=Ln(1)      ;result is 0
R22=Ln(2.718282)      ;result is 1
```

LOG

LOG (arith_expression)

LOG calculates the decade logarithm (base 10) of the argument.

☐ Example:

```
R22=Log(R21)  ;R-variable as argument
R22=Log(1)    ;result is 0
R22=Log(10)   ;result is 1
```

SQR

SQR (arith_expression)

Function to square the argument.

☐ Example:

```
R22=Sqr(R21)  ;R-variable as argument
R22=Sqr(2)    ;result is 4
```

POW

Operator to calculate a potency function. The operator POW is used, how it is shown in the following example:

result = base **POW** exponent

☐ Example:

```
R22=R23 POW R24      ;with R-variable
R22=R23 POW 3        ;constant as exponent
R22=3 POW R23        ;constant base
R3=R1 POW R2         ; result:  R3=2^3 = 8
```

☐ Reference: Section 3.2.2.1: Arithmetical expressions
Section 3.2.2.3: Boolean expression
Section 3.2.2.4: Assignments

3.2.2.3 Boolean expressions

bool_expression	**Boolean expression**

□ Syntax:

1. **bool_expression bool_operator bool_expression** *or*
2. **[bool_expression]** *or*
3. **p_variable** *or*
4. **constant**

□ Explanation:

● a **Boolean expression** can be combined with help of Boolean operators with a second Boolean expression

● the result of this operation is a Boolean expression again, the value of this expression is assigned to a **P-variable** with help of an assignment

● **bool_operator** is a **sign** from the quantity **{&, | , ^},** so that the Boolean operations **AND**, **OR** respectively **EXCLUSIV OR** (ANTIVALENZ) are available

● to determine the **priority**, Boolean expressions can be clipped

● in the simplest case a Boolean expression is a P-variable or a hexa-decimal number

● a bit wise negation of a P-variable is possible with the tilde ~

□ Example:

; 1. on the right side of the assignment is a complex
; Boolean expression, which value is assigned to the variable **P20**:
N10 **P20** = P11 & 0x11 | P12

; 2. use of the bracket:
N20 P12 = [P13 | P14] & P16

; 3. P-variable results from a P-variable: this is an assignment
; to negation the P-variable:
N40 P99 = ~P98

; 4. P-variable results from a constant: this is an assignment:
N50 P33 = 0xAB ; declaration of the constant hexadecimal
N51 P34 = 10101011B ; declaration of the constant binary

□ Reference:

Section 3.2.2.1: Arithmetical expressions
Section 3.2.2.2: Functions
Section 3.2.2.4: Assignments

3.2.2.4 Assignments

Assignment	**Assignment of values / variables to variables**

☐ Syntax:

1. **r_variable = arith_expression** *or*
2. **r_variable = q_variable : address letter** *or*
3. **r_variable = frame_name: adress letter** *or*
4. **r_variable = p_variable** *or*
5. **r_variable = MessageBox** *or*
6. **r_variable = GetValue** *or*
7. **r_variable = Posn.A**
8. **r_variable = USER** *or* **USERBAT** *or* **USEREXE** *or* **USERDLL**
9. **p_variable = bool_expression** *or*
10. **q_variable = frame_name**

● an assignment can be named as equation
● the both sides of the equation are connected by the sign **=**
● the left side of the equation is always a variable
● 1. the result of an arithmetical expression is assigned to a R-variable
● 2. a component of a Q-variable is assigned to a R variable; this component is chosen by an address letter **addressletter={X,Y,Z,A,B,C}** for **axis system 1** respectively **{X2,Y2,Z2,A2,B2,C2}** for **axis system 2**
● 3. a component of a FRAME is assigned to a R-variable
● 4. a **R-variable** is assigned to a P-variable; so that it will be possible, e. g. to calculate with an input byte
● 5. the result of an operating dialog is assigned to a **R-variable**
● 6. a value, entered by an operator within an operating dialog, is assigned to a **R-variable**
● 7. the current value of an axis position is assigned to a **R-variable**
● 8. the return code of the current USERDLL function is assigned to a R-variable
● 9. the result of a Boolean expression is assigned to a **P-variable**
● 10. a chosen element of the geometry file (this element is a data structure of 12 float-point numbers), marked by the name **frame_name,** is assigned to the Q-variable; therefore the Q-variable is initialised

● **frame_name** consists of maximum 20 signs (capital letters or numbers), the first four signs have to be capital letters)

☐ Example:

; 1. the result of the arithmetical expression on the right side of the equation is assigned to the variable **R7**:
N10 **R7** = R8 * R9 + PI * R10

; 2. the value of the **Y c**omponent of the Q-variable **Q2** is assigned to the
; R-variable **R22**:
N20 **R22** = **Q2**:**Y**
N21 **R23 = Q2 : Y2** **;** Y component from axis system 2 to R23

; 3. the value of the **Z** component of the frame **PARK_POSITION** is assigned to the R-variable **R23**:
N25 **R23** = **PARK_POSITION: Z**

; 4. the value of the P-variable **P1** is assigned to the R-variable **R10**:
N30 **R10** = **P1**

; 5. the value according to the result of the operating dialog
; is stored in the R-variable **R30**:
N35 **R30** = MessageBox Question YesNo "text"

; 6. the operator defines the number of the wished output repetitions
; and this value is stored in the variable **R35**
N40 **R35** = GetValue "Please enter the number of the output repetitions"

; 7. the current axis position of the axis Y in the axis system 1
; is stored in **R40**
N45 **R40** = POS1.Y ; for axis system 1
N46 R40 = POS2.Y ; for axis system 2

; 8. call of an User-Batch-file (DOS) with three parameters, return code
; is stored in **R50**
N50 **R50** = USER [R101 R102 R103]

; 9. the result of the Boolean expression on the right side
; of the equation is assigned to the P-variable **P22**:
N40 **P22** = P1 & P3 & P5 & P7 & P9

; 10. the Q-variable **Q4 is initialized**:
N50 **Q4** = PALETTE_1

☐ Reference:

Section 3.2.2.1: Arithmetical expressions
Section 3.2.2.3: Boolean expression
Section 3.2.2.4: Assignments
Section 3.2.4.1: Request of an operator dialog
Section 3.2.4.2: Activation of several user programs

3.2.3 Assignments to control the program process

3.2.3.1 Conditions

Conditions	**Conditions to test**

☐ Syntax:

1. **arith_expression comparision_operator arith_expression** *or*
2. **bool_expression** *or*
3. **NOT bool_expression** *or*
4. **input_bit** *or*
5. **output_bit**

☐ Explanation:

● a condition comes true, if the comparison comes true (1.) *or*
a Boolean expression consists the truth value TRUE (2.) *or*
a NOT Boolean expression consists the truth value FALSE (3.) *or*
an input bit is set (4.) *or*
an output bit is set (5.)

● **arith_expression** is an *arithmetical expression* according to the syntax instruction in section 3.2.2.1 Arithmetical expressions

● **bool_expression** is a *Boolean expression* according to the syntax instruction in section 3.2.2.3 Boolean expression

● **NOT bool_expression** is the key word **NOT** together with a *Boolean Expression*

● **relational_operator = {<, >, !=, ==, <=, >=}** is one of the following character sequences with the meaning:
less then: **<**
greater then: **>**
unequal: **!=**
equal: **==**
less or equal: **<=**
greater or equal: **>=**

● **input_bit** is an expression, how it is known from the PLC programming according to the instruction list:
E byte_number.bit_number

● **output_bit** is an expression, how it is known from the SPS programming according to the instruction list:
A byte_number.bit_number

with: 0 < **byte_number** < **9**
and: 0 < **bit_number** < **9**

☐ Example:

● syntactic right comparison (less then):
R1 < R2

● syntactic right comparison (greater then):
R3 > R4

● syntactic right comparison (unequal):
R5 != 100

● syntactic right comparison (equal):
R6 == 123.456

● syntactic right comparison (greater or equal):
R7 >= 200

● syntactic right comparison (less then):
R8 <= 2 * PI

● Boolean expressions as condition:
P1 & 0x01
P3 ^ 0x55

● syntactic right input bit:
E1.5

● syntactic right output bit:
A2.8

☐ Reference:

Section 3.2.2.2: Functions
Section 3.2.2.3: Boolean expression

3.2.3.2 Branch

IF-construction	**conditional branch**

☐ Syntax:

IF construction
instructions
ELSE
instructions
ENDIF

☐ Explanation:

● program branch:
1. if the condition comes true, the instructions **instructions** will be executed till the key word **ELSE**

2. if the condition does not come true, the instructions **instructions** between the key words **ELSE** and the key word **ENDIF** will be executed

● if in the second case no instructions have to be executed, the keyword **ELSE** can be left out

☐ Example:

if the value of the variable R1 is greater then the value of the variable R2, the subprogram with the number 100 is called, in the other case the subprogram with the number 200 is called:

ISO:

```
IF R1 > R99
N10 L100
ELSE
N20 L200
ENDIF
```

PAL:

```
IF R1 > R99
N10 SUBR100
ELSE
N20 SUBR200
ENDIF
```

☐ Reference: Section 3.2.3.1: Conditions

3.2.3.3 Selection instruction

SWITCH-construction	**Selection instruction**

☐ Syntax:

```
SWITCH arith_expression
  CASE arith_expression:
     instructions
     ENDCASE
  CASE arith_expression:
     instructions
     ENDCASE
  DEFAULT
     instructions
     ENDCASE
ENDSWITCH
```

☐ Explanation:

The selection instruction acquires the user the comfort of high level programming language regarding to the structure block "branch" and it completes the IF-construction efficient.
The **arith_expression,** following the Token **SWITCH,** represents in the moment of program running a value (real number).
This value is compared with the current value of the following **CASE**-section. At agreement the instruction block **instructions** is carried out, which is a component of the **CASE** section, which value after the **CASE** Token agrees with the value after the **SWITCH**-Token.
The instruction block to be done is completed with the Token ENDCASE.
If there isn`t no agreement of the values, the block instructions after the token **DEFAULT** is executed.
The **DEFAULT** section within the **SWITCH-**construction is optional.

☐ Example:

```
N1  R1=P1  ;the current input port E1 is transferred as
           ;value (0<=value<256) to the variable R1
SWITCH R1
  CASE 1:
    R2=5     ;the value 5.0 is assigned to the variable R2
    ENDCASE

  CASE 2:
    R2=6     ;the value 6.0 is assigned to the variable R2
    ENDCASE

  CASE 3:
    R2=7     ;the value 7.0 is assigned to the variable R2
    ENDCASE

  DEFAULT
    R2=10    ;the value 10.0 is assigned to the variable R2
    ENDCASE
ENDSWITCH
```

☐ Reference: Section 3.2.4.1: Request of an operator dialog

3.2.3.4 Counting loop

FOR-loop	**Counting loop with a counting variable**

☐ Syntax:

FOR r_variable = startvalue, endvalue, stepvalue
instructions
ENDFOR

☐ Explanation:

● **r_variable** is a R-variable R0 to R999

● the start value **startvalue** is assigned to the R-variable

● the assignments until the key word ENDFOR are executed as long as the R-variable is less then or equal the end value **endvalue**

● the R-variable is increased about the step value **stepvalue**, if the last instruction in front of the key word **ENDFOR** is carried out

● maximum five FOR-loops can be programmed interleaved

● maximum 62 FOR-loops can be programmed in a source program behind each other

● from the both previous items follows, that maximum 5 x 62 = 310 FOR-loops can be programmed in a source program

☐ Example:

```
; start of the counting loop
FOR R0=0,100,1
; the coordinate value for the center point coordinate
  N10 R8= 50.0 + R1 / 2.0
; the coordinate value for the target coordinate
  N20 R7 = 100.0 + R1
; semicircle with value transfer of the variable R7 to the
; target coordinate X and of the variable R8 to the center point coordinate I
ISO:   N30 G2 XR7 IR8
PAL:   N30 CWABS XR7 IR8
ENDFOR
```

☐ Reference:

Section 3.2.1: Variables
Section 3.2.2: Parameter calculation

3.2.3.5 Loop with test at start

WHILE-loop	**Loop with test of a condition at start**

☐ Syntax:

WHILE condition
  instructions
ENDWHILE

☐ Explanation:

- the instructions **instructions** are executed as long as the condition **condition** is true
- it won't be executed any instruction, if the condition is false at the first test
- maximum five WHILE-loops can be programmed interleaved
- maximum 62 WHILE-loops can be programmed in a source program behind each other
- from the both previous items follows, that maximum 5 x 62 = 310 WHILE-loops can be programmed in a source program

Please notice these bit mask:

Mask to selection of bit 1 out of the byte: 0x01
Mask to selection of bit 2 out of the byte: 0x02
Mask to selection of bit 3 out of the byte: 0x04
Mask to selection of bit 4 out of the byte: 0x08

Mask to selection of bit 5 out of the byte: 0x10
Mask to selection of bit 6 out of the byte: 0x20
Mask to selection of bit 7 out of the byte: 0x40
Mask to selection of bit 8 out of the byte: 0x80

P-variable with mask or input bit:

P1 & 0x01 corresponds to E1.1	; the input 1 in the input port 1
P1 & 0x02 corresponds to E1.2	; the input 2 in the input port 1
P1 & 0x04 corresponds to E1.3	; the input 3 in the input port 1
P1 & 0x08 corresponds to E1.4	; the input 4 in the input port 1
P1 & 0x10 corresponds to E1.5	; the input 5 in the input port 1
P1 & 0x20 corresponds to E1.6	; the input 6 in the input port 1
P1 & 0x40 corresponds to E1.7	; the input 7 in the input port 1
P1 & 0x80 corresponds to E1.8	; the input 8 in the input port 1

☐ Example:

ISO:

```
; as condition is tested, if  bit 8 (input 8) is set
; in input port 2:
; Hint: E2.8 is equivalent to P3 & 0x80

WHILE E2.8
; straight line to the target point 100mm, 200mm, -300mm:
  N10 G1 X100 Y200 Z-300
;  wait 1 sec:
N20 TIME 1000
; straight line to target point 0mm, 0mm, 0mm:
N30 G1 X0 Y0 Z0
ENDWHILE
```

```
; the empty WHILE-loop is carried out as long as
; the bit 4 is set in input port 1:
; synchronisation with a binary input and wait
; for the high-low-flank:
WHILE E1.4
; empty loop body
ENDWHILE

; the empty WHILE-loop is carried out as long as
; one of the bits 4 or 5 or both are set in the input port 2:
; synchronisation with two binary inputs and wait, until
; both inputs E2.4 and E2.5 have the value 0:
WHILE P3 & 0x18
; empty loop body
ENDWHILE
```

PAL:

```
; as condition is tested, if  bit 8 (input 8) is set
; in input port 2:
; Hint: E2.8 is equivalent to P3 & 0x80

WHILE E2.8
; straight line to the target point 100mm, 200mm, -300mm:
  N10 MOVEABS X100 Y200 Z-300
;  wait 1 sec:
N20 TIME 1000
; straight line to target point 0mm, 0mm, 0mm:
N30 MOVEABS X0 Y0 Z0
ENDWHILE

; the empty WHILE-loop is carried out as long as
; the bit 4 is set in input port 1:
; synchronisation with a binary input and wait
; for the high-low-flank:
WHILE E1.4
; empty loop body
ENDWHILE

; the empty WHILE-loop is carried out as long as
; one of the bits 4 or 5 or both are set in the input port 2:
; synchronisation with two binary inputs and wait, until
; one of the both inputs E2.4 and E2.5 have the value 0:
WHILE P3 & 0x18
; empty loop body
ENDWHILE
```

☐ Reference: Section 3.2.1: Variables
Section 3.2.2: Parameter calculation
Section 3.2.3.2: Branch
Section 3.2.3.6: Loop with test at end

3.2.3.6 Loop with test at the end

DO-loop	**Loop with test of a condition at the end**

☐ Syntax:

DO
instructions
ENDDO condition

OR

☐ Syntax:

REPEAT
instructions
UNTIL condition

☐ Explanation:

● the instructions **instructions** are executed at least once

● the condition will be tested, if the key word **ENDDO | UNTIL** is reached

● if the condition **condition** is true, the DO-loop is finished and it will be continued with the next instruction respectively the next NC set in the program

● maximum five **DO | REPEAT**-loops can be programmed interleaved

● maximum 62 **DO | REPEAT**-loops can be programmed in a source program behind each other

● from the both previous items follows, that maximum 5 x 62 = 310 **DO | REPEAT**-loops can be programmed in a source program

☐ Example:

ISO:

```
DO
; straight line to the target point 100mm, 200mm, -300mm:
N10 G1 X100 Y200 Z-300
; wait 1 sec:
N20 G4 1000
; straight line to the target point 0mm, 0mm, 0mm:
N30 G1 X0 Y0 Z0
; it is tested as condition, if bit 3 (the input 3)
; is set in input byte 5 (input port 5):
; if  E5.3 is logical true, the DO-loop is finished:
ENDDO E5.3

; the empty DO-loop is carried out as long as
; the bit 7 in the input port 2 is low:
; synchronisation with a binary input and with wait
; for the low-high-flank:
; hint: E2.7  is equivalent to P3 & 0x40
DO
; empty loop body
ENDDO E2.7

; the empty DO-loop is carried out as long as,
; one of the bits 4 or 5 or both are set in the input port 2:
```

```
; synchronisation with two binary inputs and with wait,
; until one of the both inputs E2.4 or E2.5 has the value 1:
DO
; empty loop body
ENDDO P3 & 0x18

PAL:  DO
; straight line to the target point 100mm, 200mm, -300mm:
N10 MOVEABS X100 Y200 Z-300
; wait 1 sec:
N20 TIME 1000
; straight line to the target point 0mm, 0mm, 0mm:
N30 MOVEABS X0 Y0 Z0
; it is tested as condition, if bit 3 (the input 3)
; is set in input byte 5 (input port 5):
; if  E5.3 is logical true, the DO-loop is finished:
ENDDO E5.3

; the empty DO-loop is carried out as long as,
; the bit 7 in the input port 2 is low:
; synchronisation with a binary input and with wait
; for the low-high-flank:
; hint: E2.7  is equivalent to P3 & 0x40
DO
; empty loop body
ENDDO E2.7

; the empty DO-loop is carried out as long as
; one of the bits 4 or 5 or both are set in the input port 2:
; synchronisation with two binary inputs and with wait,
; until one of the both inputs E2.4 or E2.5 has the value 1:
DO
; empty loop body
ENDDO P3 & 0x18
```

☐ Reference:
Section 3.2.1: Variables
Section 3.2.2: Parameter calculation
Section 3.2.3.2: Branch
Section 3.2.3.5: Loop with test at start

3.2.4 Instructions to communication with extern devices

3 .2.4.1 Request for an operator dialog

MessageBox	**Request for an operator dialog**

☐ Syntax:

r_variable = MessageBox [icon] [button] "text**"**
[icon] {**INFO** *or* **WARNING** *or* **QUESTION** *or* **ERROR**}
[button] {**OK** *or* **OKCANCEL** *or* **YESNO** *or* **YESNOCANCEL** *or* **RETRYCANCEL** *or* **ABORTRETRYIGNORE**}

☐ Explanation:

- a dialog box with a message appears on the screen
- the program is interrupted as long as an operator input will happen
- according to the decision of the operator the return parameter is stored in the R-variable
- the user can determine itself symbols, buttons and text of errors within the default
- if the "text" to be given out in the dialog box contains several lines for each line break must be written a \n
- an evaluation of the R-variable can be done with the **SWITCH**-instruction

☐ Example:

```
; after decision of the operator to continue (YES)
; or to finish (NO) the process
; the return code IDYES or IDNO is stored in R1
N10 R1 = MessageBox ERROR YESNO  "Temperature limit is reached: \n\n\n continuation ?"
;
; evaluation of the return code
SWITCH R1
    Case IDYES:
      Type Messagebox finished with YES . . .
      EndCase
    Case IDNO:
       Type Messagebox finished with NO . . .
       EndCase
ENDSWITCH
```

☐ Reference:

Section 3.2.2.1: Arithmetical expressions
Section 3.2.2.4: Assignments
Section 3.2.3.3: Selection instruction

3.2.4.2 Activation of several user programs

USER|USERBAT **USEREXE** **USERDLL**	**Activation of user programs in a DOS-BATCH-File,** **as Windows-EXE or** **as function of a Windows-DLL**

☐ Syntax:

Rx = USER[Ra Rb Rc ... Ri]

- x, a ... i for arbitrary indices of the R-variable or the axis position
- maximum 9 parameters (Ra to Ri) can be submitted
- x can have the value 0 ... 999

Rx = USEREXE name_exe_file
Rx = USERDLL name_dll_file name_dll_function

☐ Explanation:

- It is possible in ProNC, to activate user specific programs during the runtime of the ISO/PAL user program (DOS-BATCH-Files, EXE-Files or DLL-functions).

To this purpose the files user.bat, name.exe, name.dll, to be called, must be installed in the directory \CNCWorkbench\BIN. These files the user can create arbitrary respectively he can use already existing files according to his concrete task.

The declaration of the Bat-, Exe- or DLL-file in the user program happens always with the name of the file without extension.

☞ Hint:
In ProNC it is very simple to integrate protocol and/or printer functions in the running user program with the activation of Batch files.

1. R101 to R106 respectively POS1.X to POS1.C are the current values of the axes 1...6 in the axis system 1
2. R201 to R206 respectively POS2.X to POS2.C are the current values of the axes 1...6 in the axis system 2

The possible applications for the ability to activate own user files in the ISO/PAL application program are for example:
- generation of tracing files and protocols
- print of arbitrary program data
- parameter setting of extern devices how controller for laser- or welding processes with serial interface
- communication with stand alone controllers
- operation of OEM-Hardware (e. g. DAC) in the master PC
- realization of a remote diagnostics (Modem or Ethernet)
- integration of user specific dialogs respectively visualisation

☐ Example:

❶; activation of the user-batch-file user.bat
; with delivery off the three parameters (actual value of axes X, Y and Z):
N100 R1 = UserBat[POS1.X POS1Y POS1Z]

the batch-file user.bat has the following content e. g.:
@echo off
echo logging actual values>prn
echo X=%1 Y=%2 Z=%3>prn

❷; activation of the User-Exe Wordpad
N200 R0=UserExe Wordpad

❸; the function function_name of the DLL
; dll_name, creating by a user,
; shall be started
; this user_DLL must be stored in the BIN-directory
; of the current ProNC installation
; (e. g. C:\CNCWorkbench\Bin)
; the user_DLL has to provide at least the function
; function_name

; Activation of a user_DLL
N300 R2=UserDLL dll_name function_name

☐ Hint: *Hint for **C-Programmer** to implementation of the /a User-DLL:*

```
// defines:
#define ERR_NOERROR          (DWORD)0
#define USER_DLLFUNC         __declspec(dllexport) _stdcall

// types:
typedef struct r_var
{
        int nInit;
        double dValue;
} typ_r_var;

// prototyping:
DWORD USER_DLLFUNC FUNKTION_NAME(LPVOID parameter1,
LPVOID parameter2);

// function body:
DWORD USER_DLLFUNC FUNKTION_NAME(LPVOID parameter1,
LPVOID parameter2)
//-----------------------------------------------------------------------------------------------
// function code for a User DLL function ...
// in:
//   parameter1: pointer to the name of the current variable file *.var
//   parameter2: pointer to the R-variables
// out:
//   return parameter
//-----------------------------------------------------------------------------------------------
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState());

        DWORD  dwRetc;
        int          nResponse;
        typ_r_var  MyCurrentRvariableR0;
        typ_r_var *pR_variable = (typ_r_var *)parameter2;
// Your user source code in C / C++ ...

        // Hint:
        //   You have access to all 1000 R-variables, which are available in
        //ProNC for the user programming.

        // The R-variable R0 is read how follows:
        mycurrentRvariableR0.dValue  = pR_variable[0].dValue;

        // The R-variable R0 is written with the value 123.456:
        pR_variable[0].dValue = 123.456;

        return(ERR_NOERROR);
}
```

☐ Hint: If you want to create your user-DLL with a PASCAL programming environment or with LabView®, we enjoy you to help.
Please contact the customer support tech-support@isel.com sales / service, sales / technical support.

4 Synchronisation to the motion end, integration of Teach In

4.1 Synchronisation to the motion end

Programming of a motion command as a "motion kickoff":

With the software system ProNC it is possible to program a parallelism of movement and program interpretation. To this purpose it is required, to define syntactically a "motion kickoff" in the user program.
The **"motion kickoff"** is programmed with a **lozenge #** . The lozenge must lead the motion command. This **motion kickoff** causes, that the so marked motion command is delivered to the motion control, the motion is started, but the **end of the motion is not awaited** as otherwise usually:

```
N10 #MOVEABS X100 Y200;"motion kickoff" with special character #
While (InMotion)                ; synchronisation to the motion end
  If (POS1.X > 40) and (POS1.X < 60)
   SetBit A1.1=1
  Else
   SetBit A1.1=0
  EndIf
EndWhile
; continue here, if the target position is reached
; X=100, Y=200
```

The synchronisation to the motion end is realized within the loop (e. g. WHILE-loop) following the set N10. The special condition **InMotion** is true as long as the **initiated** motion to the target point [**X100, Y200**] is active. **During** this **motion** e. g. the actual axis values (POS1.X for X, POS1.Y for Y, POS1.Z for Z, ... POS1.C for C) **can be controlled**, **to set or reset certain outputs.**

During an axis motion the movement can be affected.

To this the following example:

☐ Example:

```
MoveAbs X50.0                ; start position in the X-axis
#MoveRel X1000.0 F5.0        ; motion kickoff
While (InMotion)
 IF E1.1
   MOverride 50              ; override to 50%
   SetPort A1=11110000B
 EndIf
 If E1.2
   MOverride 140             ; override to 140%
   SetPort A1=00001111B
 EndIf
 If E1.3
   MStop                     ; stop the motion
 EndIf
 If E1.4
   MStart                    ; continue the motion
 EndIf
 If E1.5
   MAbort                    ; abort the motion
 EndIf
EndWhile
```

The following commands to the motion influence are available:

MOverride value	; set motion override to value ; (0 ≤ value ≤ 140)
MStop	; stop the motion
MStart	; continue the motion
MAbort	; abort the motion

☐ **Hint:** The above-mentioned commands are equal for ISO and PAL.

4.2 Integration Teach-In

Integration of *Teach In:* The integration of *Teach-In* on the level of the user program (ISO- or PAL-user program) is made by:

- the programming of the command Teach
- the using of frame variables (Q-variable)
- the declaration of frame names to definition the motion targets of motion commands
  - FASTFRAME
  - MOVEFRAME

Frame-variable initialise :

Example:
Q1 = **START_MILLING**

Explanation:
In the current geometry file **name.fra** which belongs to the user program **name**.cnc, a frame structure with the name **START_MILLING** is searched. If such a structure exists, the values X-coordinate, Y-coordinate, Z-coordinate, A-coordinate, B-coordinate, C-coordinate) stored in the geometry file will be assigned to the Q-variable **Q1**. The purpose of this

construction (initialisation of the Q-variable) is: The access to the geometry information (coordinate values) within the Q-variables is done much faster then to a named structure within the geometry file.

please refer to:
Operating Instruction: Menu 2.2.2 The geometry file

Access to geometry information in frame-variables:

The access to the stored geometry information (coordinate values of all axes), stored after the initialisation of the frame variable in this variable, can occurs as follows:
Isolation of a certain component of a frame variable:

☞

Example:
R1 = Q1:X
Explanation:
The value of the **X**-coordinate within the geometry variable **Q1 is assigned** to the R-variable **R1**.

Example:
R2 = Q1:Y
Explanation:
The value of the **Y**-coordinate within the geometry variable **Q1 is assigned** to the R-variable **R2**.

Example:
R3 = Q1:Z
Explanation:
The value of the **Z**-coordinate within the geometry variable **Q1 is assigned** to the R-variable **R3**.

Example:
R4 = Q1:A
Explanation:
The value of the **A**-coordinate within the geometry variable **Q1 is assigned** to the R-variable **R4**.

Example:
R5 = Q1:B
Explanation:
The value of the **B**-coordinate within the geometry variable **Q1 is assigned** to the R-variable **R5**.

Example:
R6 = Q1:C
Explanation:
The value of the **C**-coordinate within the geometry variable **Q1 is assigned** to the R-variable **R6**.

Target-setting with help of geometry information in frame variables or direct declaration of the frame name:

The access to the stored geometry information, stored in the frame variable after its initialisation, can occur as follows:

☐ Example:

; designation of a Q-variable as target-setting of a motion command:

ISO: N100 G11 **Q1**
PAL: N100 **MOVEFRAME Q1**

Explanation:
It will be executed an absolute motion G11| MOVEFRAME of all axes to the coordinate values, which are stored within the Q-variable:
That means:
The value of the X-coordinate within the Q-variable **Q1** determines the target position of the X-axis.
The value of the Y-coordinate within the Q-variable **Q1** determines the target position of the Y-axis.
The value of the Z-coordinate within the Q-variable **Q1** determines the target position of the Z-axis.
The value of the A-coordinate within the Q-variable **Q1** determines the target position of the A-axis.

Alternative to the declaration of a Q-variable in the command G10 |FASTFRAME respectively G11 | MOVEFRAME the name of the wished frames can be defined directly:

Example:

ISO: N100 G11 **START_MILLING**
PAL: N100 **MOVEFRAME START_MILLING**

Explanation:
It will be executed an absolute motion G11 | MOVEFRAME of all axes to the coordinate values, which are stored in the frame with the name **START_MILLING**:
That means:
The value of the X-coordinate within the frame with the name **START_MILLING** determines the target position of the X-axis.
The value of the Y-coordinate within the frame with the name **START_MILLING** determines the target position of the Y-axis.
The value of the Z-coordinate within the frame with the name **START_MILLING** determines the target position of the Z-axis.

(appropriate also at the axes A, B and C)

Geometry file - frame structure:

The initialisation of a Q-variable presupposes, that the information was stored in a geometry file. These information represents during the Teach-In current positions / orientation(s) of a plant. The coordinates are stored into a fixed structure, the frame structure. Each geometry file can contain arbitrarily many frame structures. Each of these structures gets a name (consisting of maximum 20 signs). Within a geometry file a frame name may appear only once.

4.3 Example for a user program with integration Teach-In

Next the user program (in PAL syntax) **mfp_p.pal** (**m**y **f**irst **p**rogram) is listed. This simple user program demonstrates self-documenting the integration of Teach-In in an application program .

mfp_p.pal ☞ The described PAL source program **mfp_p.pal** is provided in the directory

\CNCWorkbench\NCProg\PAL\Sample

after the installation.
The analog ISO source program you will find in the directory \CNCWorkbench\NCProg\ISO\Sample as mfp_p.iso.

```
;=====================================================================
; Anwenderprogramm für ProNC:
;
; User program for ProNC:
;
; mfp_p.pal: my first program (PAL Syntax)
;=====================================================================
; no subprogram declaration required ...
;
;---------------------------------------------------------------------
; Start of the main program:
ProgBegin
N10 Ref XYZ   ; reference run in all axes
N20 Type the teached position can be still corrected ...
N30 Teach
;
; FOR-loop: 200 times
For R0=1,200,1
;
; approach to park position with fast velocity:
  N100 FastFrame PARK_POSITION
; switch on spindle 1 (clockwise), 8000 r.p.m.
  N110 Sclw 1 S1=8000
; approach to start position to milling with rapid velocity
; (spindle turn on):
  N120 FastFrame START_MILLING
; milling process with processing velocity of 5 mm per second:
  N130 MoveFrame END_MILLING F5.0
; switch off spindle:
  N140 Soff
; wait in the current position 1 sec = 1000 msec:
  N150 Time 1000
; test, is a reference run to carry out after 100 cycles:
  If R0 == 100
    N200 Type reference run in all axes ...
    N210 REF
; end of the IF-construction
  EndIf
; end off the counting loop
EndFor
;---------------------------------------------------------------------
; program end
ProgEnd
;=====================================================================
```

5 Selected solutions with ProNC

5.1 isel-XYZ-plants / several Cartesian Kinematics

5.1.1 Learning

The following user program (ISO: learning.iso / PAL: learning.pal) is well suitable, to get to know important commands of ProNC e. g. linear- and circular interpolation, path conditions like absolute measurement and relative measurement (incremental measure), interpolation plane for the circular interpolation, dwell time, subprogram technique and others.

Work piece zero point with Teach-In: A successful processing of the program **learning.iso / learning.pal** presupposes, that the workpiece zero point is assumed with Teach-In in the associated geometry file „learning.fra“.

WPZP = **W**ork **P**iece **Z**ero **P**oint: The geometry file **learning.fra** contains only the frame structure with the name **WPZP: W**ork **P**iece **Z**ero **P**oint.

Learning.iso

☞ The according ISO source program **learning.iso** is stored in the directory

\CNCWorkbench\NCProg\ISO\Sample

after installation.

Learning.pal

☞ The according PAL source program **learning.pal** is stored in the directory

\CNCWorkbench\NCProg\PAL\Sample

after installation.

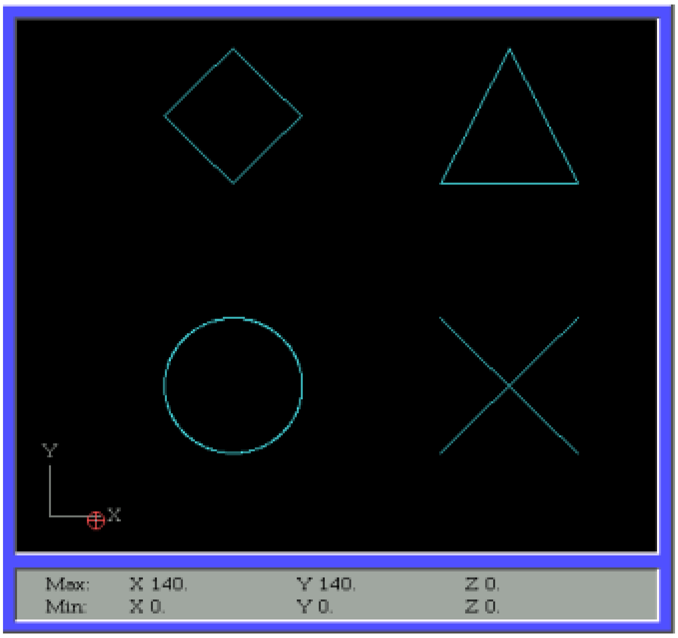
5.1.2 Figures

Processing the following program

ISO: Figures.iso
PAL: Figures.pal

at an ***isel***-XYZ-plant the following milling contour results (referred to the XY-plane):

Up left: square, standing on an edge
Up right: equilateral triangle
down left: circle
down right: cross (height = circle diameter)



Figures.iso The according ISO source program **figures.iso** is stored in the directory

☞ **\CNCWorkbench\NCProg\ISO\Sample**

after installation.

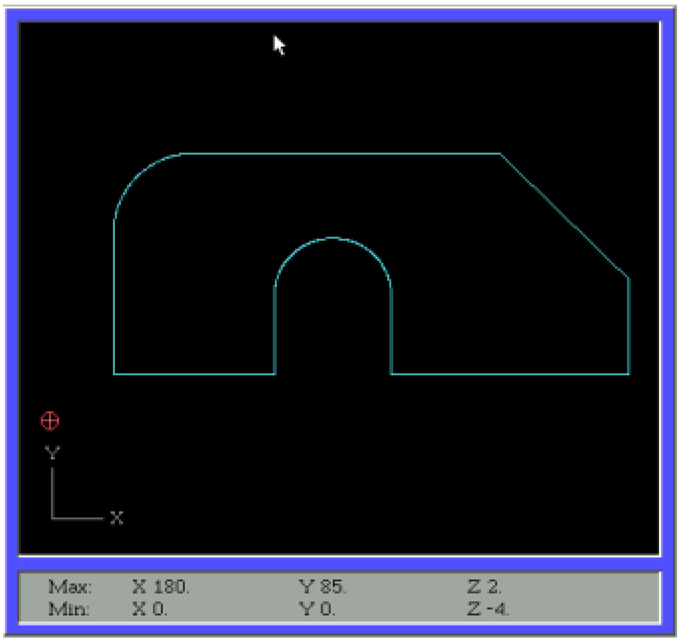
Figures.pal The according PAL source program **figures.pal** is stored in the directory

☞ **\CNCWorkbench\NCProg\PAL\Sample**

after installation.

5.1.3 Milling of a simple contour

Processing the following program with an ***isel***-XYZ-plant produces the following milling contour:



Contour.iso The according ISO source program **contour.iso** is stored in the directory

\CNCWorkbench\NCProg\ISO\Sample

after installation.

Contour.pal The according PAL source program **contour.pal** is stored in the directory

\CNCWorkbench\NCProg\PAL\Sample

after installation.

5.1.4 Drilling

In the following drilling program the use of nested FOR-loops as well as the subprogram technology will be demonstrated.

Use of FOR-loops: R-variables have to be used as loop counter.
The loop counter **R10** can deliver its current value to the X-coordinate as target value:

```
For R10=10,100,10   ; counter loop for the X-coordinate
;
;   approach the X-coordinate according to the value R10
    N40 FASTABS XR10
...
```

Determine the zero point: For a correct working of the program at your plant you have to determine the zero point of the work piece, that will be drilled and you have to update it in the next set:

N20 FASTABS X80 Y70 Z-30 ; approach the new work piece zero point

More effective is surely a procedure according to the program „learning.pal“ in the previous capital. There the work piece zero point WPZP was teached and was assumed into the Q-variable **Q1** during the run time of the user program. After approaching the teached point with the NC set **MOVEFRAME** **Q1**, a new zero point can be activated with the WPZERO-command.
You should carry out and translate newly the above-mentioned program modification in your source program **"drilling.iso"/"drilling.pal"**. Is the zero point submitted with Teach-In in the new defined geometry file „drilling.fra“, the user program „**drilling.cnc**“ can be started:

Drilling.iso

The according ISO source program **drilling.iso** is stored in the directory

\CNCWorkbench\NCProg\ISO\Sample

after installation.

Drilling.pal

The according PAL source program **drilling.pal** is stored in the directory

\CNCWorkbench\NCProg\PAL\Sample

after installation.

5.1.5 Milling of pockets

The following example program to mill pockets bases on the use of the variable concept and the parameter calculation with R-variable. The use of nested FOR-loops as well as the subprogram technique will be demonstrated.

Pockets.iso

The according ISO source program **pockets.iso** is stored in the directory

\CNCWorkbench\NCProg\ISO\Sample

after installation.

Pockets.pal

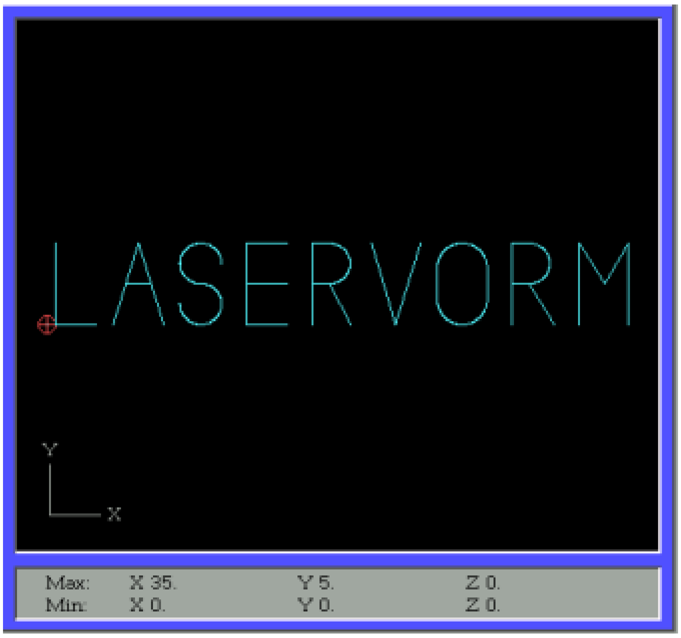
The according PAL source program **pockets.pal** is stored in the directory

\CNCWorkbench\NCProg\PAL\Sample

after installation.

5.1.6 Engraving script with Laser

The following script results when processing the following program with an ***isel***-XYZ-plant:



Script.iso

The according ISO source program **script.iso** is stored in the directory

\CNCWorkbench\NCProg\ISO\Sample

after installation.

Script.pal

The according PAL source program **script.pal** is stored in the directory

\CNCWorkbench\NCProg\PAL\Sample

after installation.

5.1.7 Welding

The following programme realizes an automatic welding of a car body frame.
Four seams are welded.

Teach-In for welding seams:

Eight teached points are required for four welding seams:

```
N05 Q1 = SEAM_PPOINT1_START      ; seam 1 – start point
N10 Q2 = SEAM_POINT1_END         ; seam 1 – end point
N15 Q3 = SEAM_PPOINT2_START      ; seam 2 – start point
N20 Q4 = SEAM_POINT2_END         ; seam 2 – end point
N25 Q5 = SEAM_PPOINT3_START      ; seam 3 – start point
N30 Q6 = SEAM_POINT3_END         ; seam 3 – end point
```

N35 Q7 = SEAM_PPOINT4_START ; seam 4 – start point
N40 Q8 = SEAM_POINT4_END ; seam 4 – end point

To transition from end point of seam 1 to the start point of seam 2 an intermediate point is required:

N45 Q12= ONE_TO_TWO ; transition seam 1 to seam 2

To transition from end point of seam 3 to the start point of seam 4 an intermediate point is required:

N50 Q34= THREE_TO_FOUR ; transition seam 3 to seam 4

Migmag.iso

The according ISO source program migmag**.iso** is stored in the directory

\CNCWorkbench\NCProg\ISO\Sample

after installation.

Migmag.pal

The according PAL source program **migmag.pal** is stored in the directory

\CNCWorkbench\NCProg\PAL\Sample

after installation.

6 Summary

6 Summary

ProNC: The software package ***ProNC*** is an operating and programming system for CNC plants / CNC systems for processing and handling applications. ***ProNC*** integrates a mouse sensitive operator surface according to the SAA-standard and a programming platform to design, start up and test of ISO- resp. PAL-user programs.

Operator-dialog: The operator dialog will be realized with several pull-down-menus with a functional clear operating structure. Plain text error messages and error information and the direct context-dependent branch to the online-help supports the using of the programming system effectively. Operating- and runtime errors will be reported as plain text in Windows style.

isel*-control software:** ***ProNC is executable on PCs running the operating system Win98 / Win2000 / WinNT4.x / WinXP. The software was implemented as further development of the systems **Remote / Pro-DIN** resp. **Pro-PAL**. There the technology-oriented syntax of DIN 66025 resp. of isel-NCP-format was completed with problem-oriented constructions for structured programming, parameter calculation as well as for access to geometry files. This is defined as flexible, efficient programming language (ISO- or PAL syntax).

ISO- / PAL-compiler: The possibility of nested use of constructions to control the programming run as well as the subprogram technique requires a compiler with tasks of the syntactical analysis of the source program and the generation of the CNC user program as input language for the CNC interpreter. The ISO- / PAL compiler is provided as an independent DLL and offers the user an extensive support to correct syntactical errors.

Automatic mode: In automatic mode the program test will be efficiently supported with the possibility of activation of break points in arbitrary NC-sets as well as the manipulation of current values of R-variables (data type: floating-point).

Teach-In: The Teach-In can be made directly, if no self-arresting gears are available in the cinematic chain. There the corresponding axes will be moved with current-free motors with hand to the wished position. The taken joint vector will be stored in the geometry file.
Using indirect Teach-In the tool will be moved to the desired target position with help of a complex dialog box and usage of function keys.

Software hierarchy Within the hierarchy of the *isel*-control-software the operating- and programming surface ***ProNC*** applies on a plane of device-DLLs for

- motion control **(Motion Control)**
- input- and output **(Input / Output)**
- spindle control **(Spindle)**
- tool changer control **(Tool Changer)** and others.

With this hierarchy concept it will be possible, to apply the program package for a great range of isel-control Hardware (e. g. IMC4-/C116/C142-, IMS6/IML4-Controller, UPMV4/12-controls respectively CVC496-Controller with CANopen-Interface), if the special device-DLLs, offered for the special Hardware, are compatible in their functions.

Glossary

button

A graphic image, that can be clicked with the mouse to initiate some action e. g. start of a dialog field

CNC user file

The CNC user file will be created out of a syntactic errorless ISO- or PAL-source file. The CNC user file is the input file for the CNC interpreter.

EDP

Electronic Data Processing

FRAME

Data structure, coordinate system, certain matrix

IO

Input/Output

MCTL

Motion control

PLC

Programmable Logic Controller

SAA-Standard

System Application Architecture

SPN

Spindle module

S-PTP

synchronous Point-to-Point

TCH

Tool Changer

Index

L

M

N

O

P

Q

R

S