

PAL-PC 2.0

Programming Instruction

General

The Manual:

In this manual, you will find various symbols that will draw your attention to some important information.

Caution:



Example:



Hint:



Information:



© **isel**automation KG 2004
All Rights Reserved.

Despite all care taken, printing errors and mistakes cannot be ruled out entirely.
Suggestions for improvement or comments on errors are always welcome.

No part of this publication may be copied or distributed, transmitted, transcribed, or stored in a retrieval system, without the expressed written permission of iselautomation KG.
All information is supplied without liability. Changes may be made at any given time without prior notice.

Producer: **isel**automation KG
Buergermeister-Ebert-Straße 40
36124 Eichenzell
Germany

Phone: +49 (0) 66 59 98 10
Fax: + 49 (0) 66 59 98 17 76
E-mail: automation@isel.com
<http://www.isel.com>

Version: 06/2004

Contents

CONTENTS	3
1 INTRODUCTION	4
1.1 PRODUCT.....	4
1.2 SUPPORTED CONTROLLERS	5
1.3 FUNCTIONS OF PAL-PC	5
2 LANGUAGE ELEMENTS OF PAL-PC	6
2.1 OVERVIEW.....	6
2.2 CONSTRUCTION OF A PAL-PC PROGRAM	6
2.3 THE EDITOR	7
3 COMMANDS PAL-PC	9
3.1 COMMANDS OF THE DECLARATION BLOCK.....	9
3.1.1 <i>Overview of commands of the declaration block</i>	9
3.1.2 <i>The command #axis</i>	10
3.1.3 <i>The command #steps</i>	11
3.1.4 <i>The command #units</i>	12
3.1.5 <i>The command #elev</i>	13
3.1.6 <i>The command #define</i>	14
3.1.7 <i>The command #redefine</i>	17
3.1.8 <i>The command #start</i>	18
3.1.9 <i>The command #ref_speed</i>	19
3.1.10 <i>The command #include</i>	20
3.1.11 <i>The command Comment</i>	21
3.1.12 <i>The command #GN</i>	22
3.1.13 <i>The command #input</i>	23
3.2 THE STATEMENT PART.....	24
3.2.1 <i>Overview commands of the statement part</i>	24
3.2.2 <i>The command label</i>	25
3.2.3 <i>The command move</i>	26
3.2.4 <i>The command moveto</i>	28
3.2.5 <i>The command movep</i>	30
3.2.6 <i>The command send</i>	31
3.2.7 <i>The command wait</i>	32
3.2.8 <i>The command loop</i>	34
3.2.9 <i>The command port and pulse</i>	35
3.2.10 <i>The command time and delay</i>	36
3.2.11 <i>The command reference</i>	37
3.2.12 <i>The command tell</i>	38
3.2.13 <i>The command stop</i>	39
3.2.14 <i>The command line</i>	40
3.1.15 <i>The command repeat ... until</i>	41
3.2.16 <i>The command goto</i>	42
3.2.17 <i>The command null</i>	43
3.2.18 <i>The command on_key</i>	44
3.2.19 <i>The command on_port</i>	45
3.2.20 <i>The command set_port</i>	46
3.2.21 <i>The command 3D linear interpolation</i>	47
3.2.22 <i>The command circle</i>	48
INDEX	50

1 Introduction

1.1 Product

Product description: **PAL-PC** is a programming system of isel's interface card series which is used as a solution to problems encountered with regard to simple process controllers. With PAL-PC, a maximum of 3 axes can be controlled. PAL-PC can be run **directly** (in DNC mode) or in **memory mode** (CNC mode). Therefore, applications are possible in the **stand-alone mode** as well as in conjunction with the **controller PC**.
By using the CNC extension of the IMC4 controller, the machines of the CPM-series/GFM 4433 can also be used independently.
PAL-PC in Windows is the **successive version** of PAL-PC in DOS. It includes the **entire** range of functions that the DOS version contains.

The graphical user interface is designed in such a way that the most important functions of the program can be accessed by buttons in the toolbar.

PAL-PC contains an **integrated editor and compiler**. Usual editor functions such as "search", "substitute", "copy" and "paste", as well as formatting functions for colour and font design, allow a comfortable and quick development of a program, even a bug-free application program in translation.

The hardware-(based)-option "battery backup" guarantees durable storage of the program even after shutdown of the system. With a **memory card**, the translated application program can be saved and directly reloaded into the memory of the controller.

Pal-PC runs under the operating systems of Windows 98, Windows 2000 and Windows XP.

DNC and CNC mode : In the **DNC mode**, the transmission of the application program to the controller is done command-wise/in a segmental manner with direct execution. The program is only bootable in this mode with a connected controller PC (direct mode).



In the **CNC mode**, upon transmitting (downloading) the application program to the controller (end controller), the program will be saved onto the interface card. It is directly bootable from the controller, as the case may be, or from the machine (memory mode or stand-alone mode). A PC is only required for program creation, testing and download.

1.2 Supported Controllers

Controllers: The following controllers are supported:

- Interface card V5.c
- Compact system EP10905
- Single-axis controlling IT116**Fehler! Textmarke nicht definiert.**
- 4-axis stepping motor controller CSI 464

Machines: The following machines are compatible with PAL-PC:

- All machines of the CPM-family (IMC4 with CNC extension)
- GFM 4433 (IMC4 with CNC extension)

1.3 Functions of PAL-PC

Administration with PAL-PC:

- 1-axis system with a maximum of 4 axes (X, Y, Z, A)
- Communication with other computer systems over a serial interface
- Direct execution (DNC mode) and memory mode (CNC mode)
- Memory card for storing the CNC program onto an external storage medium
- Battery backup to save the program after shut down of the system
- Transmission rates from 2,400 baud to 19,200 baud
- Start and processing of the program without controller PC possible
- Teach/reference speed adjustable
- Teach-in
- Set workpiece zero-point
- Reference point driving

Program functions:

- Integrated editor for programming
- Compiler for translating the application program
- Instructions for relative and absolute positioning
- 2D interpolation which can be switched to 3D interpolation
- Analysis of the in- and output signals for process controlling
- Loops for repetition of the instruction part, unconditional and conditional branching, and time delay

2 Language elements of PAL-PC

2.1 Overview

Overview:

PAL-PC is a programming language that uses **strictly defined tokens** in connection with instruction-specific **parameters**. In general, “tokens” are written in lower case and can contain letters, numbers and underscored symbols.

Self-defined symbols should be written in capital, so that there will not be any conflict with the standard expressions (instruction set) of PAL-PC. The program PAL-PC distinguishes between the controller commands and the saveable instructions. The controller commands are in the declaration part of the program and are marked at their outset with the hash symbol #. The saveable instructions contain the executable commands. They are transmitted to the interface card, respectively to the connected controller, and then saved. If the option “battery backup” is installed, the saved instructions will still be available after a system shutdown. After switching on the machine, this saved program can be recalled by pressing the start button on the machine without any connection to a superior computer. In PAL-PC, the commands for the declaration part and the instruction part are predefined.

refer to:

3.1.1 Overview of commands of the declaration blocpart on page 9

refer to:

3.2.1 Overview commands of the statement part on page 24

One should avoid using the specified command notation for one’s own definitions, otherwise the original functions will no longer be available.

2.2 Construction of a PAL-PC Program

Basics:



A PAL-PC program consists of the optional declaration part, which contains pre-adjustments of the connected mechanics, and further parameters which deliver important data for the commands in the subsequent instruction part of the program. The commands in the declaration part consist of a keyword with the hash symbol (#) as a prefix, and the parameter definitions.

The instruction part with the saveable commands generally begins with the command **Fehler! Textmarke nicht definiert.#input**.

These instructions consist of the command word followed by the command specific parameters. The instruction part contains, among other things, the move commands, loops, direct jumps and keypad enquiries, which are required in the application program in order to fulfil the desired automated application flow.

Comments can be inserted into both programs parts for better transparency of the application program. In order to indicate the end of the program, the command **stop** is used; the command **#start** implies that a block of data handed over to the controller is commenced after the transmission.

Sentence structure:

A sentence in PAL-PC generally consists of **one** command word followed by its associated parameters. In this case the command line does not have to be compulsively ended with a semicolon.

The usage of more than one command word in a sentence is also allowed, but then there has to be a semicolon at each end of the command declaration. A sentence may have a label as a prefix which contains, in distinction from the command word, a **colon**.

An explanation detailing which parameters have to follow the command word, or if the command only fulfils a switch function into a certain mode, can be found in Chapter 3: "Commands of PAL-PC".

Also refer to:

3.2.4 Command moveto (command with parameter block) on page 28

3.2.21 Command set3d on/set3d off (command with switch function) on page 47

Hint:

Simplifications were made to some commands after consideration of the complexity of the syntax. Those simplifications should lead to a better understanding of the instructions and a more feasible parameter input. All existing syntax regulations are still valid in order to provide for a complete compatibility of PAL-PC under DOS.

refer to: 3.2.20 The command set_port on page 46

2.3 The Editor

Source file:

With the program system PAL-PC, the user now can create his own programs (e.g. programs for process controlling or editing programs) following the discovery of an algorithm in accordance with the technological concept implemented in order to deal with the problem.

The algorithm is transferred into the program text structure (**sequence, loop, branching**). This kind of programming is called **textual programming**.

The source file is created as a text file by the integrated editor of the PAL-PC. Each source file has the extension *.txt.

In addition, a text file created outside of PAL-PC (e.g. with WordPad) can be loaded and processed with PAL-PC.

Upon **creation** of the program text, certain **conditions** have to be kept.

These **rules are to be defined**, in all languages, as **grammar**.

The **grammar defines as well syntax** ("which sequence of words produces a valid sentence?") as **semantics** ("what is the meaning of this sentence?") of the programming language.

The valid grammar is described in detail and with examples in Chapter 3 of this handbook.

2 Language elements of PAL-PC

User file:



The user file is generated after a compilation and is saved as a file with the extension *.out in the same directory as the source file.

The user file is only generated if, during compilation, no syntactical errors were found. A corresponding display and an error protocol will point out the syntactical mistakes.

An example of a bug-free syntactical source program can be found in the following sequence:

Example:



```
*****
/ File:          test
/
/ created:       06/09/2003, iselautomation KG
/
/ *****
/ Declaration part
#axis x;         {choice of axis: x-axis}
#reference x;    {reference start point: x-axis}
#units mm;      {set dimension unit; mm is default}
#input;

/ Begin instruction part, following commands are transmitted to
/ the controller and saved

repeat ; infinite loop ...
  reference x;

  repeat
    set_port A1,1=1; / set Bit1 at output port A1
    delay 1;
    set_port A1,1=0; / delete Bit1 at output port A1
  until 2;

  moverel 100(500); {x-axis 100 mm proceed with 500 Hz}
  moverel -100(9000); {x-axis -100 mm proceed with 9,000 Hz}
  delay 10;          {wait 1 sec}

until 0;           {end of loop}

stop.              {end of program}
#start;            {start execution}
```


3 Commands PAL-PC

3.1 Commands of the declaration block

Pre-settings for the program: The declaration part serves to inform the compiler of the pre-settings which are necessary for the treatment. These important parameters are, among other things, the number of attached axes (max. 3 axes), working and reference speed of the axes, spindle upward gradient, and a definition of the used unit for all instructions.

The declaration block can be ruled null and void, however in such an event, the pre-settings then become effective.

Default values can be found in the detailed description of the individual instructions.

3.1.1 Overview of commands of the declaration block

<u>Command</u>	<u>Content</u>
#axis	Allocation of the attached axes (Chapter 3.1.2 on page 10)
#define	Define text replacement (Chapter 3.1.6 on page 14)
#elev	Define spindle gradient (Chapter 3.1.5 on page 13)
#GN	Define device number (Chapter 3.1.12 on page 22)
#include	Insert file (Chapter 3.1.10 on page 20)
#input	Set memory mode (Chapter 3.1.13 on page 23)
#redefine	Redefine text replacement (Chapter 3.1.7 on page 17)
#ref_speed	Define reference speed (Chapter 3.1.9 on page 19)
#start	Initiate execution (Chapter 3.1.8 on page 18)
#steps	Indicate number of steps/revolution (Chapter 3.1.3 on page 11)
#units	Define unit of measurement (Chapter 3.1.4 on page 12)
{.} or /	Insert comments (Chapter 3.1.11 on page 21)

3.1.2 The command #axis

#axis	Choice of axis
-------	----------------

Syntax: #axis [axis];

Explanation: axes {x, y, z}

These instructions must initiate the program.
By handing over the number of axes, the processor card is initialized once again.
In doing so, the data memory is deleted and reset for memory optimization according to the number of axes.

The default setting is the axes configuration x, y and z.

Example: #axis x; {only x-axis is connected}
#axis xz; {x- and z-axes are connected}



Hint: The axes can be defined by the operator under **Menu Settings - General ...**.
The number of defined axes should agree with the number of targeted axes.



For example, the separate choice of the x-axis can lead to undefined procedure movements with respect to three attached axes of the IMC4.

Caution: The command "set number of axes" deletes all existing data in the RAM even if, through the use of the integrated option "memory back-up", the data were stored after omission of power supply in the RAM memory of the processor card.



3.1.3 The command #steps

#steps	Number of steps/revolution
--------	----------------------------

Syntax: #steps [number of steps X],[number of steps Y],[number of steps Z];

Explanation: [number of steps] Number of steps/revolution of the connected axes

For each axis separated by commas, the number of steps/revolution of the engine of the respective axis is indicated.

The number of steps per revolution indicates the necessary conversion factor to the compiler, so that the latter will be able to transform the working unit in stepping motor steps.

Example: Example 1:



If the engine of the x-axis has 400 steps/revolution and the pertinent spindle has an upward gradient of 4 mm, then the following allocation results occur:

Generally accepted formula:

$$\frac{\text{motorsteps/revdution}}{\text{spindleincrease(mm)}} = \text{steps/millimetre}$$

$$\frac{400\text{steps}}{4\text{ mm}} = 100\text{steps/mm}$$

Results: **#steps 100;**

In the event of the use of a transmission stepping motor, the reduction is to be multiplied by the number of the motor steps/revolution.

Example 2:

The engine of the x-axis has 400 steps/revolution and a 1:9-transmission, and the pertinent spindle upward gradient amounts to 4 mm:

Generally accepted formula:

$$\frac{\text{motor steps/revolution x reduction}}{\text{spindle increase}} = \text{steps/millimetre}$$

$$\frac{400\text{ steps x }9}{4\text{ mm}} = 900\text{ steps/mm}$$

Result: **#steps 900;**

The instruction must be used after the choice of axes since, if no axis has been chosen beforehand, the number of axes is initialized to xyz.

3.1.4 The command #units

#units	Define unit of measurement
--------	----------------------------

Syntax: #units [unit];

Explanation: Indication of the unit for procedure movements.
The following are allowed:

- #units mm;
- #units cm;
- #units inch;
- #units inch/10;
- #units inch/20;
- #units inch;
- #units inch/10;
- #units inch/20;

Where the working unit is not indicated, "mm" is accepted as the default unit. Inch/10 and inch/20 are the common units for conductor boards, since these are drilled into the inch-grid.

3.1.5 The command #elev

#elev	Define spindle gradient
-------	-------------------------

Syntax: #elev [gradient x],[gradient y],[gradient z];

Explanation: [gradient] Specifies the spindle gradient of the x-, y- and z-axes

If #units is not differently agreed upon, then the default unit is mm. In order to make a correct conversion of the working unit into stepping motor steps, the spindle gradient of the attached axes must be handed over.

Example: Example 1:



A plant has 4 mm spindles in the x- and y-axes, as well as a 2.5 mm spindle in the z-axis.

The associated instruction is then defined as:

#elev 4, 4, 2.5;

Example 2:

At a plant with a 2 mm spindle in the x-axis and a 4 mm spindle in the y-axis, the instruction is defined as (please note that the z-axis is missing):

#elev 2, 4;

If no spindle gradient instruction is used, a spindle gradient of 4 mm is accepted for all attached axes.

3.1.6 The command #define

#define	Define text replacement
----------------	-------------------------

Syntax: `#define [(string) (statement)\;...\;(statement)];`

Explanation: [string] Indicates the defined name of the following text/statement. For generation of the name, all letters, numbers and the sign "_" are allowed.
The name always has to start with a letter. If a separator (blank or tabulator) is recognized, this is interpreted as the end of the name.

[statement] Indicates the desired texts/instructions for the defined name. If several statements, which are terminated with a semicolon, are defined in one text replacement, this has to be written in front of each semicolon, which does not, in turn, terminate the text replacement, or the so-called escape character "\".

Caution: Points to be considered: The selected definition and defined jump address should not be identical; also, the latter may not be contained as a string in a jump address.



If, for definitional purposes in a rational way of writing, only one symbol is used, please use special characters such as &, §, \$ or as well as letters enclosed within parentheses, such as (V) and (S).

Attention should also be paid to the large and lower cases, since this represents a distinguishing feature as well.

You are allowed, in PAL-PC, to define text replacement symbols. Text replacement means that you only have to define the used text or one or more statement sequences once; thereafter, you can use them in the text under this name. Text replacement can be very simple (refer to example 1) or can also contain a complex statement block (refer to example 2) with loops and several individual instructions.

Text replacement of a complex statement block has an analogue function as a sub-routine.

The entire instruction must be terminated with a semicolon.

Example: Example 1: Definition of simple statements



```
#define () (3000);
#define & 0(21),0(21);
#define drill 20(1000),-20(9000);
```

The first definition selects for the round parentheses () without indication of a speed of 3,000 Hz.

The symbol & indicates that no movement of the z-axis has been assigned. Depending on at which position in the instruction this definition is inserted, this can stand for, for example, the indication of the z-coordinate in the move instruction. The minimum statement for the speed is 21.

refer to 3.2.3 The command move on page 26

The third definition **drill** is used instead of the z-coordinate in order to be able to adjust the drilling depth constantly for the entire program.
Using the above definitions the following program section can be created:

```
/move x and y each 20 mm
move 20(),20(),&;
/ move x= 2mm, y= 5 mm, drill a hole with a depth of 20 mm
move 2(),5(5000),drill;
```

During translation, the compiler would interpret the above instruction sequence as the following statements:

```
move 20(3000),20(3000),0(21),0(21);
move 2(3000),5(5000),20(1000),-20(9000);
```

Example 2: Definition of statement blocks

Since the instruction "text replacement" needs a semicolon as its end-character, the occurrence of a semicolon in the text must be particularly marked. Therefore a special symbol is inserted directly in front of the semicolon, the so-called "escape" character. An example of this is the definition of a 14-pin IC-socket:

```
#define DIL14
  repeat
    move 1(),0(),drill\;
  until 7\;
  move 1(),3(),&\;
  repeat
    move 1(),0(),drill\;
  until 7;
```

Please note that there is no "escape character" before the semicolon in the last line of the instruction. This semicolon terminates the definition.

The symbol defined above falls back to the symbols **&**, **()** and **drill** as defined in example 1. These symbols must be specified before the definition of **DIL14**, otherwise a compiler run error will surface thereafter.

Now you can use the defined symbol **DIL14** as follows in your own programs:

3 Commands PAL-PC

```
...  
move 20(),30(),&;           {move to start point}  
Repeat                       {two sockets ...}  
  DIL14;                     {drill DIL-14}  
  move 1(),20(),&;           {start point next socket}  
until 2;                      {and next socket}
```

Constraint:



Note that the compiler differentiates between large and lower cases. The use of the symbol "dil14" would therefore result in an error.
The maximum number of definitions which can be stored amounts to 500.
The length of a definition may amount to a maximum of 250 symbols. The length of the line, in which the definition is replaced, may not exceed 255 symbols.


3.1.7 The command `#redefine`

#redefine	Redefinition
------------------	--------------

Syntax: `#redefine *[(string) (statement)\;...\;(statement)];`

Explanation: [string] Indicates the definition name of the following text/ statement. For the generation of the name, all letters, numbers and the sign "_" are allowed. The name always has to start with a letter. If a separator (blank or tabulator) is recognized, this is interpreted as the end of the name.

[statement] Indicates the desired texts/instructions for the defined name. If several statements which are terminated with a semicolon, are defined in one text replacement, this has to be written in front of each semicolon, which does not subsequently terminate the text replacement, or the so-called escape character "\".

Hint:  Points to be considered: The selected definition and defined jump address should not be identical; also, the latter may not be contained as a string in a jump address. If, for definitional purposes in a rational way of writing, only one symbol is used, please use special characters such as &, §, \$ or as well as letters enclosed within parentheses, such as (V) and (S). Attention should also be paid to the large and lower cases, since this represents a distinguishing feature as well.

In order to change definitions of speed or other declarations in an NC-program, the instruction "#define" cannot be used repeatedly. The function "#redefine*" replaces a definition already taking place.

Usually this function is used for the assignment of processing and entrance speeds for the teach-in if an NC-program with more than two speeds is to be worked.

Example: Changes to, for example, the speed definition would not be suitable for the following program construction:



```
#define () (2000);
```

.....

```
#define () (3000);
```

This method of writing can lead to unexpected results being obtained. This is correct:

```
#define ()(2000);
```

```
#redefine *() (3000);
```

The function "#redefine" replaces a definition already taking place. A new speed (3,000 Hz) is assigned to the text () and used from this point onwards.

3 Commands PAL-PC

3.1.8 The command #start

#start	Initiate execution
--------	--------------------

Syntax: #start;

Explanation: After termination of the data field, the transferred data field can be immediately initialised with the command "start execution". If the instruction is used, the data field has to be terminated with the **stop** command. If no data field was transferred, the instruction starts an existing data field in the interface card.

3.1.9 The command `#ref_speed`

<code>#ref_speed</code>	Definition of reference speed
-------------------------	-------------------------------

Syntax: `#ref_speed` [speedr X],[speedr Y],[speedr Z];

Explanation: The speeds for the reference driving of the axes are set with this instruction. If no information is handed over with regard to reference speed, the execution takes place with a default value of 800 Hz. A changed value remains after switching off if the option "battery backup" is inserted.

Please also refer to **Menu Settings - Control Parameters**.

The reference speed must lie in the range of the permissible speeds (21 to 3,000 Hz).

Example: Adjusting the reference speed to 2000 Hz for the axes x and y, and to 800 Hz for the z-Axis:



`#ref_speed` 2000,2000,800;

3.1.10 The command #include

#include	Insert file
----------	-------------

Syntax: #include <[file name]>;

Explanation: The possibility exists of using program parts in several programs with PAL-PC. For that purpose, the more frequently needed program part is created in its own file or by inserting a file into an existing program. At the position in which this instruction sequence in the program is needed, the instruction **#include** can be used.

Example: The compiler expects a file name after this instruction, optionally with prefixed indication of drive assembly and/or indication of its path.



#include <C:\User\editor\reference.txt>;

This instruction would insert the file "reference.txt" from drive "C:" into the submenu "User\editor". The file name in the instruction must be included either in **pointed parentheses** or **quotation marks**.

If the indicated file does not exist, the compiler displays an appropriate message.

At the position of the instruction "#include", all instructions will be inserted in such a way into the indicated file, such that these instructions are located in the file which will be translated subsequently.

3.1.11 The command Comment

<pre>{ } or /</pre>	Insert comments
---------------------	-----------------

Syntax: { [comment] } or / [comment]

Explanation: Comments can occur everywhere in the program. You should use comments to fix important information within your program sequence.

There are two certified ways of commenting:

- Comments which are initiated by the opening curved clip "{" and comments terminated by the closing curved clip "}"
- Comments which start with the symbol "/" and are terminated by the end-of-line symbol CR = Carriage Return (enter key code)

A comment contains text and can stand apart or independently of an instruction within an instruction sequence.

Example: #define VEL (3000); {VEL represents the speed 3,000 Hz}



```
/ move the x- and y-axis 2 mm with 100 Hz
move 2(100),2(100)
```

3 Commands PAL-PC

3.1.12 The command #GN

#GN	Set device number
-----	-------------------

Syntax: #GN [device no.];

Explanation: This instruction defines which device number possesses the addressed interface card.
The interface card must be adjusted to the device number used.

Example: #GN 1 {interface card with device number 1 is programmed}



#GN 2 {interface card with device number 2 is programmed}

3.1.13 The command *#input*

#input	Set memory mode
---------------	-----------------

Syntax: **#input**

Explanation: All the instructions that follow the command **#input**, are stored in the internal data memory.
The execution of the saved data field is initiated by pressing the start button at the interface card/controller/machine, by choosing the instruction “start” in the **Menu Transfer**, or by including the instruction **#start in the user program**.

3.2 The statement part

Storable statements: The statement part contains all the instructions which are transferred to the interface card and saved there. It contains instructions detailing moves, process controlling (management and analysis of synchronisation characters, in- and output signals), loops and direct branches, as well as instructions for reference driving and definition of the zero point.

3.2.1 Overview commands of the statement part

<u>Command</u>	<u>Content</u>
circle_ccw / circle_cw	Circle interpolation (Chapter 3.1.22 on page 48)
goto	Branching (Chapter 3.2.16 on page 42)
label	Set branch destination (Chapter 3.2.2 on page 25)
line	Set level of interpolation (Chapter 3.2.14 on page 40)
loop	Loop (Chapter 3.2.8 on page 34)
move/moverel	Relative movement (Chapter 3.2.3 on page 26)
movep	Movement until impulse (Chapter 3.2.5 on page 30)
moveto/moveabs	Absolute movement (Chapter 3.2.4 on page 28)
null	Set zero point (Chapter 3.2.17 on page 43)
on_key	Keyboard query (Chapter 3.2.18 on page 44)
on_port	Read input-port (Chapter 3.2.19 on page 45)
port and pulse	Impulse input (Chapter 3.2.9 on page 35)
reference	Reference drive (Chapter 3.2.11 on page 37)
repeat ... until	Repetition (Chapter 3.2.15 on page 41)
send	Send sync character (Chapter 3.2.6 on page 31)
set_port	Set output port (Chapter 3.2.20 on page 46)
set3d on / set3d off	3D linear interpolation (Chapter 3.2.21 on page 47)
stop	End of program (Chapter 3.2.13 on page 39)
tell	Output of control characters (Chapter 3.2.12 on page 38)
time and delay	Time delay (Chapter 3.2.10 on page 36)
wait	Wait for sync character (Chapter 3.2.7 on page 32)

3.2.2 The command label

Label	Set branch destination
--------------	------------------------

Syntax: [label]:

Explanation: A label is a word which can contain letters, numbers and the underscore character. The first character of the label has to be a letter. A colon has to come after the label in order to mark the end of the label. The label presents a branch address.

The mixing of large and lower cases is allowed; however, a consistent style of writing is essential for the clear allocation of the branch addresses.

Some instructions of the interface card permit branches. Those branches can be indicated individually and consist of a whole positive or negative number. Depending on the algebraic sign, the program branches out forwards or backwards, as per the indicated number of instructions.

"Goto -5" creates a repeated sequence of five consecutive instructions in the program. Such instructions can contain errors, since one can easily mis-estimate with further jumps, and insertions immediately require a correction of these branch statements.

Bearing this in mind, PAL-PC is permitted to use branch destinations (so-called labels) in the program text.

Example: Some examples of acceptable labels:



```
BEGIN:
prog_drill:
begin_secondly:
```

The compiler differentiates between the label "BEGIN" and "begin" because in PAL-PC, large and lower case is different.

Some examples of incorrect labels:

```
124:           a number is not accepted as a label
1.Subroutine: contains an incorrect character and starts with a number
PROG MILL:    contains a space (invalid character)
```

In the instructions which use branch destinations, the use of labels is described in the respective instruction.

3.2.3 The command move

move moverel	Relative movement
-------------------------------	-------------------

Syntax: **move** [X(X_v),[Y(Y_v),[Z₁(Z_{v1}),[Z₂(Z_{v2})];
 or
 moverel [X(X_v),[Y(Y_v),[Z₁(Z_{v1}),[Z₂(Z_{v2})];

- | | |
|------------------------------------|---|
| [X(X _v) | Destination coordinate x relative to the current starting point.
Indication of speed for x-axis |
| [Y(Y _v) | Destination coordinate y relative to the current starting point.
Indication of speed for y-axis |
| [Z ₁ (Z _{v1}) | 1. Destination coordinate z relative to the current starting point.
Indication of speed for z-axis |
| [Z ₂ (Z _{v2}) | 2. Destination coordinate z relative to the starting point and
Indication of speed for z-axis |

Explanation: The indication of the moves of the axes (x, y, z) takes place in the selected measurement unit (default unit of measurement being the millimetre [mm]), and the conversion into stepping motor steps is done on the basis of the mechanical parameters defined in the declaration part.
 When using fractional numbers for the indication of the coordinates, a point must always be used instead of a comma.
 The speed is indicated by a whole number in Hertz (Hz); the minimum value is 21.

For each axis, measurements of motion and speed are indicated. The speed is noted in round parentheses before the motion size. The data for the individual axes are separated by commas.

For each attached axis a pair of parameters will be entered (destination coordinate, speed).
 For the z-axis two pairs of parameters are expected, since during phases of operation, the procedure to lower and raise the tool occurs frequently.

Example:

/ move x-axis 2 mm with a speed of 2,000 Hz
move 2(2000);

/ move x-axis 2 mm with 2,000 Hz, y-axis 2 mm and 3,000 Hz
move 2(2000),2(3000);

/ move x-axis 20 mm with 900 Hz, z-axis 30 mm in positive and 30 mm in
 / negative direction with 1,000 Hz
moverel 20(900),30(1000),-30(1000);

/ move x-axis 2 mm and y-axis with 100 Hz
 / z-axis 2.8 mm in positive direction and 2 mm in negative direction with 100 Hz
move 2(100),2(100),2.8(200),-2(100);

The motion sequence always takes the following form: first the attached x-/y-axes are moved and interpolated, then the first z-coordinate is moved. Thereafter, the second z-coordinate is executed.

The interpolation of the axes is freely selectable (refer to command **line**).

For axes which should not execute a movement, the motion size can be set to zero. Thereby the speed must be set to a correct value (valid speed values are between 21 and 20,000 Hz).

Please note that neither PAL-PC nor the interface card can verify whether the movement falls outside of the permissible range of the attached mechanics.

3.2.4 The command *moveto*

moveto moveabs	Absolute movement
---------------------------------	-------------------

Syntax: **moveto** [X(X_v)],[Y(Y_v)],[Z₁(Z_{v1})],[Z₂(Z_{v2})];
 or
 moveabs [X(X_v)],[Y(Y_v)],[Z₁(Z_{v1})],[Z₂(Z_{v2})];

[X(X_v)] Destination coordinate x, indication of speed for x-axis

[Y(Y_v)] Destination coordinate y, indication of speed for y-axis

[Z₁(Z_{v1})] Destination coordinate z₁, indication of speed for z-axis

[Z₂(Z_{v2})] Destination coordinate z₂, indication of speed for z-axis

Explanation: The indication of the movement of the axes (x, y, z) takes place within the selected measurement unit (default unit of measurement being the millimetre [mm]), the conversion into stepping motor steps is done on the basis of the mechanical parameters defined in the declaration part.
 When using fractional numbers for the indication of the coordinates, a point must always be used instead of a comma.
 The speed is indicated by a whole number in Hertz (Hz); the minimum value is 21.

The command **moveto/moveabs** causes a linear movement of the axes to the indicated destination coordinates with the defined speeds, this corresponds to a path instruction with absolute specification; i.e. the destination coordinates refer to the set zero point of the workpiece coordinate system.

For compatibility reasons in terms of the relative positioning instruction for the z-axis, two pairs of numbers are expected. Since in this instance, the destination coordinates were defined, the z₂-value always has to be zero and is ignored.

Example:

/ move x-actual +2 mm with a speed of 2,000 Hz
moveto 2(2000);

/ move x-actual +2 mm with 2,000 Hz, y-actual +2 mm and 3,000 Hz
moveabs 2(2000),2(3000);

/ move x-actual +20 mm with 900 Hz, z-actual +30 mm in positive and 30 mm in
 / negative direction with 1,000 Hz
moveto 20(900),30(1000),-30(1000);

/ move x-actual +2 mm and y-actual +2mm with 100 Hz
 / z-actual +2.8 mm in a positive direction and 2 mm in a negative direction with
 100 Hz
moveabs 2(100),2(100),2.8(200),-2(100);

The motion sequence always takes the following form: first the attached x-/y-axes are moved and interpolated, then the z-coordinate is moved. The interpolation of the axes is freely selectable (refer to command **line**).

For axes which should not execute a movement, the motion size can be set to zero. Thereby the speed must be set to a correct value (valid speed values are between 21 and 20,000 Hz).

Hint:

The instruction can only be used after the number of axes has been set. Please note that neither PAL-PC nor the interface card can verify whether the movement falls outside of the permissible range of the attached mechanics.

3.2.5 The command *movep*

movep	Movement until impuls
--------------	-----------------------

Syntax: **movep** [X(X_v)],[Y(Y_v)],[Z₁(Z_{v1})],[Z₂(Z_{v2})],[A(A_v)];

[X(X_v)] Destination coordinate x relative to the current starting point, indication of speed for x-axis

[Y(Y_v)] Destination coordinate y relative to the current starting point, indication of speed for y-axis

[Z₁(Z_{v1})] 1. Destination coordinate z relative to the current starting point, indication of speed for z-axis

[Z₂(Z_{v2})] 2. Destination coordinate z relative to the current starting point, indication of speed for z-axis

Explanation: The measurement unit of the destination position (x, y, z) is the millimetre [mm] (default value).
 When using fractional numbers for the indication of the coordinates, a point is always to be used instead of a comma.
 The speed is indicated by a whole number in Hertz (Hz); the minimum value is 21.

The instruction "movement until impulse" behaves exactly like the instruction "movement relatively", but if an impulse occurs at the impulse entry point (e.g. a stop impulse due to pressing the stop button), the movement is terminated and the next instruction is executed.
 If during the movement no impulse is received, the movement takes place within the indicated distance and the program continues with the next or subsequent instruction.
 The occurring impulse must have a minimum width of around 20 µsec.
 The maximum width of the impulse may not exceed 100 µsec.
 It should be considered when using this instruction that the impulse rests against the stop button input of the interface card.
 If an impulse occurs and the interface card executes a normal positioning, this movement is stopped and the next instruction is executed.
 If problems with the resting pulse width should arise, one recommends the insertion, after the *movep*-command, of a time or delay instruction with a waiting period > = 1 (100 msec).

Example:

```
#axis x; {connected axis = x-axis }
#input   ; save the following instructions
movep 20(1000); {move the x-axis 20 mm until impulse occurs}
reference x;     {reference movement of the x-axis}
```



3.2.6 The command *send*

send	Send sync character
-------------	---------------------

Syntax: **send** [number];

Explanation: [number] Number between 33 and 126 (ASCII character)

In order to make a process synchronisation for the interface card with a second interface card or a superordinate computer possible, a sync character can be sent within reach of a certain point in the data field.

The sync character must lie in the range of 33 and 126; the character 64 (@) should not be used.

This character set contains, among other things, all numbers as well as large and small case letters.

The synchronise characters used can be assigned names at the beginning of the program with the instruction **#define**, which are defined by the instruction **send** instead of the number.

The choice of meaningful names improves the transparency of the program and facilitates the process of synchronisation.

Example: / the sync character 90 is the text `drilling_is_ready` assigned
#define `drilling_is_ready` 90;



.....
send `drilling_is_ready`;

You can define the declarations for the two synchronising devices by using the instruction **#include** in one shared file together.

Example of the communication function: To show/visualise the instructions `send/wait`, you can test the following program with the help of the communication window:



```
#axis x;
repeat
  move 20(800);
  send 90;
  wait 65;
  move -20(800);
until 5;
stop
```

After translating and transferring the program to the controller, please start the program from the communication window (**Menu** Transfer - Terminal ...) with the shortcut (Shift + F1).

The x-axis is moved 20 mm forward and sends, as a request, a "Z" ("Z" corresponds to the character 90). Now the program waits for the character 65. Enter the character by pressing the Alt-key and typing 65 on the numeric keyboard.

The character A appears in the communication window; after this input the program is continued until the next wait-instruction in the program loop.

3.2.7 The command wait

wait	Waiting for sync character
-------------	----------------------------

Syntax: **wait** [number], [offset];

[number] Number between 33 and 126 (ASCII-character)

[offset] Number, which indicates, how many lines are to be jumped forwards or backwards
or
 Label (jump address) to which it should be branched and continued with.

Explanation: Analogous to the command last discussed, "waiting for sync character" is also used in order to make possible a process synchronisation of the interface card with a second interface card or a superordinate computer.

The interface card can accomplish several actions during the execution of the instruction. These actions cover the following possibilities:

The character itself is received:

The interface card then works on the next instruction saved in the data field

The character +1 is received:

The interface card branches (relatively) to the point which is defined in the instruction.

The character 127 is received:

The interface card releases a reset and waits for instructions from the computer.

Example: Example 1:



wait 90; {wait for character 90 and process after receipt of this, the next instruction in the saved CNC program}

wait 90,-5; {upon receipt of the character 90, go to the next instruction in the program; upon receipt of the character 91, go back 5 lines in the program}

wait 80,anfang; {upon receipt of the character 81, go to the label "anfang"}

Example 2:

A superordinate computer examines the parts and communicates to the interface card, after its request, whether or not the examination was successful. If the part is correct, TEIL_OK+1 is handed over; the interface card then puts the part into the working machine; otherwise it sends the superordinate computer the sequence TEIL_OK+0, and the interface card puts the part aside and proceeds with the next part.

Following the illustrated sample, very complex systems can be developed and controlled.

label1:

move ...	{fetch part}
send TEIL_DA;	{signals ready for examination}
wait TEIL_OK, go_on;	{wait for release}
move ...	{put part aside (defect part)}
goto label1;	{try next part}

go_on:

move ...	{put part into machine}
move ...	{in starting position}
goto label1;	{branch again to label1}

3.2.8 The command loop

loop	Loop
-------------	------

Syntax: **loop** [quantity] **times** [label];

Explanation: [quantity] Number between 0 and 32767. 0 denotes an infinite loop.
 [label] Marks the beginning of the loop and is addressed with each loop cycle according to the number of repetitions.

Loops serve to summarize the consecutively occurring similar operational and/or motion sequences. Therefore, the available storage space is used more effectively.

Example: Example 1:



```
loop_1:
    move ...                ;movement instruction
;repeat all instructions starting from label "loop_1" five times
    loop 5 times loop_ 1;
```

Example 2:

```
loop_ continuous:
    move ...                ; movement instruction
; continuous loop: repeat all instructions starting from label "loop_ continuous"
; continuously
    loop 0 times loop_ continuous;
```

For the creation of a loop, the instructions "repeat" and "until" can also be used.

3.2.9 The command port and pulse

port pulse	Impulse input/impulse output
---------------	------------------------------

Syntax: port [state]
 pulse [state]

Explanation: [state] Indicates the desired condition of the exit.

<u>Syntax</u>	<u>State</u>
port on;	Port output on
port off;	Port output off
pulse out;	Give 50 msec impulse
pulse in;	Wait for impulse
pulse sync out;	Send pulse, wait for acknowledgement
pulse sync in;	Wait for impulse, send acknowledgement

If the hardware option "impulse output" is attached to the interface card, then the card can be addressed by using these instructions. Once again, the individual options are briefly specified here; the compiler understands the command "port" as well the command "pulse".

The impulse output can also be used for the synchronisation of two devices (the last two options). One of the major tasks of the impulse control is it to wait for the start button during the operation of a data field. This can be necessary if, at a certain position, a manual interference into the process cycle becomes essential, and where the interface card is only allowed to continue working after completion of the interference. Since the start button is attached to the impulse input, the instruction "pulse" can be used.

Example: pulse in; {wait for acknowledgement of the start button}



3.2.10 The command *time* and *delay*

time delay	Time delay
-----------------------------	------------

Syntax: **time** [time];
 delay [time];

Explanation: [time] Indicates the waiting time. The indication of this time takes place within a tenth of a second.

A time delay causes the interface card to wait for the specified waiting time. The waiting time cannot be cancelled by pressing the stop button.

Example: Example:



time 500; {wait 50 seconds}
delay 20; {wait 2 seconds}

The maximum waiting time amounts to 3276.7 seconds.
The instructions must be completed with a semicolon.

3.2.11 The command reference

reference	Reference drive of the axes
------------------	-----------------------------

Syntax: **reference** [axes];

Explanation: Axes {x, y, z}

PAL-PC expects, after the reference instruction, an indication of which axes for which a reference drive is to be released. Here one can indicate each axis which is contained in the axis choice instruction.

Example:



Example 1:

reference xy; {accomplishes a set to zero position of the x- and y-axis}

reference x; {accomplishes a set to zero position of the x-axis}

reference xyz; {accomplishes a set to zero position of all three axes}

It has to be considered that the axes are moved in the order zyx, i.e. first the reference drive of the z-axis, then the reference drive of the y-axis, and, finally, the reference drive of the x-axis. If this behaviour is not desired, two reference instructions must be used (refer to Example 2).

Example 2:

reference x; {accomplishes a set to zero position of the x-axis}

reference y; {accomplishes a set to zero position of the y-axis}

Hint:



This is to be referenced from a speed derived from the default value, please agree on the reference speed in the declaration part of the program.

refer to:

3.1.9 The command #ref_speed on page 19

3.2.12 The command *tell*


tell	Output of control characters
-------------	------------------------------

Syntax: `tell [GN] [options];`

Explanation: [GN] Indicates the device number of interface cards which can be addressed
 [options] Indicates the instructions for the interface card which can be addressed (refer to examples).

The output of control characters belongs to the group of instructions of the process synchronisation. The instruction results in the output of up to four characters during its execution. Primarily, the instruction is meant to release a start or reference at a second attached interface card. It can, however, also be used arbitrarily for other purposes.

Example:

	<code>tell 0 start;</code>	<code>{start device 0}</code>
	<code>tell 0 start,wait;</code>	<code>{start device 0, wait for end}</code>
	<code>tell 0 reference xyz;</code>	<code>{activate reference drive at device 0}</code>
	<code>tell 0 reference,wait xyz;</code>	<code>{activate reference drive at device 0, wait for end}</code>

The execution of the instructions may best be tested by the function communication.

3.2.13 The command stop

stop.	Marks end of program
--------------	----------------------

Syntax: **stop.**

Explanation: This command marks the end of a program.

Example: Example:



```
#axis x;                            {choose x-axis}
#units mm;                        {unit of measurement = millimetre}
#input
reference x;                        {set to zero position of the x-axis}
move 100(8000);                    {drive 100 mm in x-direction}
stop.                                {mark end of program}
```

This command must be contained within each program.

3.2.14 The command line


line	Set level of interpolation
-------------	----------------------------

Syntax: line [axis];

Explanation: Axes {x, y, z}

Indicates the axes which are to be interpolated:
In the standard setting of the interface card, the attached x-/y-axes are interpolated against each other (straight-lined moves to the destination point).
With the choice of the interpolation level, however, the possibility exists of defining every other level configuration as the main level.

Example:



```
line xy; {interpolate x and y}
line xz; {interpolate x and z}
line yz; {interpolate y and z}
```

A setting of the interpolation level remains active within an NC program until another "line" instruction is used. Generally, with the start of a program, "line xy" is adjusted.

The allocation of the interpolation level does not have any influence on the reference sequence; i.e., the reference drive is still done in the order "zyx".

3.1.15 The command *repeat ... until*


repeat ... until	Repetition
-----------------------------	------------

Syntax: **repeat**
 until [quantity];

Explanation: [quantity] Indicates the number of times a section is repeated. If the value "0" is used as the quantity, a continuous loop develops.

The instructions "repeat" and "until" are used to repeat sections within a program. "Repeat" marks the starting point of the repetition, and "until", the end.

Example:



```

repeat {mark begin of loop}
         move 5(800) {relative movement}
         delay 20    {wait 2 seconds...}

until 7; {end of loop}

```

After "repeat" there is no need for a semicolon because the instruction contains no parameters. "Repeat" and "until" can be used inclusively, i.e. one repetition can contain another. The maximum number of instructions which are allowed between "repeat" and "until" is limited by the maximum memory space available on the interface card.

The command "until" has to be terminated with a semicolon.

3.2.16 The command goto

goto	Branching
-------------	-----------

Syntax: goto [destination];

Explanation: [destination] Number of lines which are leaped over in the program process
or
 Label (jump destination), marks the program line from which the process should be continued

The number can be positive or negative, according to whether it should be branched forward or backward.

For the definition of "label" please refer to:

3.2.2 The command label on page 25

The command "goto" enacts the continuation of the program, starting from the indicated jump destination.

Example:

```

goto 5;      {leap over the next five instructions}
goto -5;     {branch five instructions backward}
goto anfang; {branch back to label "anfang"}
goto ende;   {branch forward to label "ende"}
  
```



3.2.17 The command null

null	Set zero point
------	----------------

Syntax: null [axes];

Explanation: Axes {x, y, z}

Definition of a zero point on the actual position of the referenced axes.

The workpiece zero point can again be put on the mechanical zero point of the system by a reference drive.

Example: #define () 800;
reference xyz;



```
/ move to position x = 20 mm, y = 30 mm, z = 15mm
moveto 20( ),30( ),15( ),0( );
/ set workpiece zero point at this position
null xyz;
```

```
/ move to position x = 10 mm, y = 20 mm, z = 20mm relative to
/ workpiece zero point or x = 30mm, y = 50mm, z = 35mm relative to
/ machine zero point
moveto 10( ),20( ),20( ),0( );
```

3.2.18 The command *on_key*

on_key	Keyboard query
---------------	----------------

Syntax: **on_key** [key no.], [label];

Explanation: [key no.] Indicates the respective keyboard number.
 [label] Indicates the label to which the keyboard number should be branched after the button is depressed..

This instruction applies at the time of the connection of an "isel" program selection unit. At the user's disposal, this acts as a programmable keyboard (there is no possibility of programming the interface card by the program selection unit).

The isel-program selection unit features a keyboard consisting of 12 keys and is connected to the interface card via the serial interface.

Example:



```
#axis x;
#units mm;
#elev 4;
Begin:
    on_key 1, do_reference;
    on_key 2, do_move;
    on_key 3, ende;
goto anfang;

do_reference:
    reference x;
goto anfang;

do_move:
    move 5(8000);
goto anfang;

End:
stop.
```

If the key F1 of the program selection unit is pressed, the program branches out according to the subroutine do_reference, and the interface card executes the instruction reference drive.

If the key F2 of the program selection unit is pressed, the program branches out according to the subroutine do_move, and the interface card executes the instruction move.

If the key F3 of the program selection unit is pressed, the program branches out for the subroutine "Ende", and the interface card executes the instruction stop.

3.2.19 The command `on_port`

<code>on_port</code>	Read input port
----------------------	-----------------

Syntax: `on_port [ADDRESS], [BIT NO.]=[VALUE],[OFFSET];`

Explanation:

[ADDRESS] Specification of the input port E1 or E2 (replaces the numerical values 65531 and 65532 from the DOS version)

[BIT NO.] Bit-by-bit reading BIT NO. = [1,...,8]
Byte-by-byte reading BIT NO. = 0 or 128

[VALUE] Bit-by-bit reading □VALUE = 0 or 1
Byte-by-byte reading □Request for the bit pattern at the input port

[OFFSET] Numerical value or label, which causes a branching forward or backward in the program sequence

The input port is tested for the desired bit or bit pattern. If the condition is fulfilled, a branching is accomplished.

The interface card reads an input port and is branched at a true condition.

Example: 1. Case: Reading bit-by-bit



Command	Qualifier	Branch
<code>on_port E1,2=0,3;</code>	Bit 2 = off	3 lines forward
<code>on_port E1,8=1,-2;</code>	Bit 8 = on	2 lines backward

2. Case: Reading byte-by-byte

Command	Qualifier	Branch
<code>on_port E1,0=10,3;</code>	Dual 00001010	3 lines forward
<code>on_port E1,0=0,-2;</code>	Dual 00000000	2 lines backward
<code>on_port E1,0=205,-4;</code>	Dual 11001101	4 lines backward

3.2.20 The command `set_port`

<code>set_port</code>	Set output port
-----------------------	-----------------

Syntax: `set_port [ADDRESS], [BIT NO.]=[VALUE];`

Explanation:

[ADDRESS] Specification of the output port A1 or A2 (replaces the numerical values 65531 and 65532 from the DOS version)

[BIT NO.] Numerical value, which is used for the case differentiation between “set bit-by-bit” or “set byte-by-byte” of the output pattern.

Set bit-by-bit BIT NO. = [1,...,8]
Set byte-by-byte BIT NO. = 0 or 128

[VALUE] Set bit-by-bit →VALUE= 0 or 1

Set byte-by-byte →Number between 0 and 255 which can optically be defined at the output port as bit pattern (dual number), as well as in a technical circuit.

A desired output pattern or defined output bit is set at the interface card.

Example: 1. Case: Set bit-by-bit



Command	Output port	Bit	State
<code>set_port A1,5=0;</code>	A1	5	off
<code>set_port A1,4=1;</code>	A1	4	on
<code>set_port A2,4=0;</code>	A2	4	off
<code>set_port A2,1=1;</code>	A2	1	on

2. Case: Set byte-by-byte

Command	Output port	Dual
<code>set_port A1,0=10;</code>	A1	00001010
<code>set_port A1,0=27;</code>	A1	00011011
<code>set_port A2,0=205;</code>	A2	11001101
<code>set_port A2,0=255;</code>	A2	11111111
<code>set_port A2,0=0;</code>	A2	00000000

3.2.21 The command 3D linear interpolation

set3d on set3d off	3D interpolation
-----------------------	------------------

Syntax: **set3d on**
 set3d off

Explanation: Switches the three-dimensional (spatial) interpolation on or off.

The instruction works modally, which means that all the instructions set3d (i.e. "move/moverel", "moveto/moveabs") are processed three-dimensionally until this mode is switched off again with the instruction "set3d off".

The indication of z₂-parameters in these procedures is ignored.

The value of the x-axis is consulted as an indication of the speed of the interpolation.

The maximum speed of a 3D interpolation amounts to 10,000 Hz.

Example:



```
#axis xyz;
reference xyz;
set3d on;            {switch to 3D interpolation}
/ movement of x-, y-, z-Axis simultaneously
move 10(700),15(700),3(400),0(30);
set3d off;            {switch off 3D interpolation}
```

Hint:



The introduction of a reference drive resets the program automatically to a 2.5 dimensional interpolation.

The correct treatment of a 3D interpolation assumes, as a reference level, an xy-level.

Also refer to 3.2.14 The command line on page 40

3.2.22 The command circle

circle_cw circle_ccw	Circle interpolation in clockwise direction Circle interpolation in counter clockwise direction
-------------------------	--

Syntax: circle_cw [r(v)], [angle1], [angle2]
circle_ccw [r(v)], [angle1], [angle2]

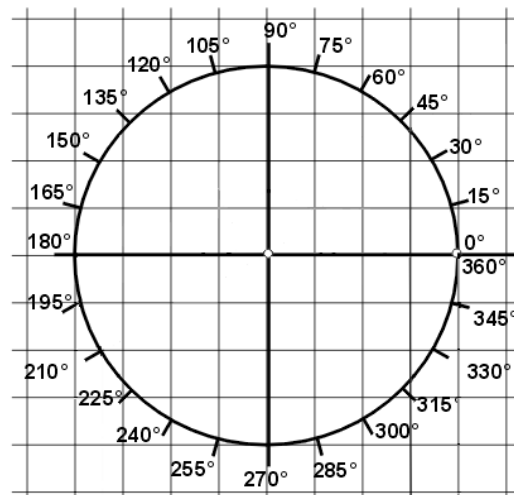
- cw Direction of rotation is clockwise
- ccw Direction of rotation is counter clockwise

- [r(v)] r = radius
 v = speed of operation

- [angle1] Starting angle of the circular path in [degrees]

- [angle2] Final angle of the circular path in [degrees]

Explanation: A circular motion will be carried out, depending on the radius defined and on the starting and end point (defined over starting and final angle). The direction of rotation is determined by the parameters cw (clockwise) in a clockwise direction (mathematically negative direction of rotation) and ccw (counter clockwise) in an counter clockwise direction (mathematically positive direction of rotation) and is to be seen from the starting point. With the circle instruction, at maximum, a complete circle (difference between starting and final angles = 360 degrees) can be driven. The following illustrated circle is the basis of the definition of the circle segment which can be driven.



Example:



1. Movement of different circles or circle sections in a positive direction (in a counter clockwise direction). The diameter of the radius amounts to 20 mm and the movement speed, 5,000 Hz.

/ Complete circle, beginning with 0 degrees and ending with 360 degrees
circle_ccw 20(5000),0,360

/ Circle segment with an angle = 45 degrees
/ beginning with 0 degrees and ending with 45 degrees
circle_ccw 20(5000),0,45

/ Circle segment with an angle = 150 degrees
/ beginning with 60 degrees and ending with 210 degrees
circle_ccw 20(5000),60,210

/ Complete circle, beginning with 225 degrees and ending with 225 degrees
circle_ccw 20(5000),225,585

2. Movement of different circles or circle sections in a negative direction (in a clockwise direction). The diameter of the radius amounts to 20 mm and the movement speed, 5,000 Hz. Here, it is to be noted that the starting angle is always larger than the final angle. This is arrived at by the addition from 360 degrees to the starting angle.

/ Complete circle, beginning with 0 degrees and ending with 360 degrees
circle_cw 20(5000),360,0

/ Circle segment with an angle = 305 degrees
/ beginning with 0 degrees and ending with 45 degrees
circle_cw 20(5000),360,45

/ Circle segment with an angle = 150 degrees
/ beginning with 60 degrees and ending with 270 degrees
circle_cw 20(5000),420,270

/ Complete circle, beginning with 225 degrees and ending with 225 degrees
circle_cw 20(5000),585,225

Index

3	
3D interpolation.....	47
A	
axis.....	10
B	
battery backup.....	4
branch designation.....	25
branching.....	42
C	
ccw.....	48
choice of axis.....	10
circle_ccw.....	48
circle_cw.....	48
CNC mode.....	4
comments.....	21
CSI 464.....	5
cw.....	48
D	
declaration part.....	6
define.....	14
delay.....	36
DNC mode.....	4
E	
elev.....	13
end of program.....	39
F	
final angle.....	48
G	
GN.....	22, 38
goto.....	42
gradient.....	13
I	
impuls input.....	35
impuls output.....	35
include.....	20
input.....	23
insert file.....	20
Interface card.....	5
K	
keyboard query.....	44
L	
label.....	25
level of interpolation.....	40
line.....	40
loop.....	34
M	
measurement.....	12
memory card.....	5
memory mode.....	23
move.....	26
moveabs.....	28
movement until impuls.....	30
movep.....	30
moverel.....	26
moveto.....	28
N	
null.....	43
number of steps.....	11
O	
on_key.....	44
on_port.....	45
output of control characters.....	38
P	
port.....	35
pulse.....	35
R	
read input port.....	45
redefinition.....	17
redefine.....	17
ref_speed.....	19
reference.....	37
reference drive of axes.....	37
repeat.....	41
repetition.....	41
S	
send.....	31
serial interface.....	5
set output port.....	46
set_port.....	46
set3d off.....	47, 48
set3d on.....	47
stand-alone mode.....	4
start.....	18
starting angle.....	48
steps.....	11
stop.....	39
Sync character.....	31
T	
tell.....	38
text replacement.....	14
time.....	36
time delay.....	36
transmission rate.....	5

<i>U</i>		
units.....	12	
until.....	41	
<i>W</i>		
wait.....	32	
		waiting for sync character..... 32
		<i>Z</i>
		zero point..... 43