Intermec

Programmer's Reference Manual

Intermec Fingerprint® v8.20

Intermec Technologies Corporation

Corporate Headquarters
6001 36th Ave. W.
Everett, WA 98203
U.S.A.
www.intermec.com

Contents

1 Introduction

This chapter gives a quick introduction to the Intermec Fingerprint v8.20 programming language and describes the most important differences between this version and the previously documented version (Fingerprint v8.10).

It also lists the various files included in the Fingerprint v8.20 firmware package.

Description

Intermec Fingerprint v8.xx is a BASIC-inspired, printer-resident programming language that has been developed for use with the Intermec EasyCoder PF2/4i and EasyCoder PM4i series of direct thermal and thermal transfer printers.

The Intermec Fingerprint firmware is an easy-to-use intelligent programming tool for label formatting and printer customizing, which allows you to design your own label formats and write your own printer application software.

You may easily create a printer program by yourself that exactly fulfils your own unique requirements. Improvements or changes due to new demands can be implemented quickly and without vast expenses.

Intermec Fingerprint also contains an easy-to-use slave protocol, called Intermec Direct Protocol. It allows layouts and variable data to be downloaded from the host and combined into labels, tickets and tags with a minimum of programming. Intermec Direct Protocol also includes a versatile error handler and a flexible counter function.

This Reference Manual contains detailed information on all programming instructions in the Intermec Fingerprint programming language in alphabetical order. It also contains other program-related information that is common for all Intermec Fingerprint-compatible printer models from Intermec.

All information needed by the operator, like how to run the printer, how to load the media or ribbon supply and how to maintain the printer, can be found in the User's Guide for the printer model in question.

In the User's Guide, you will also find information on installation, setup, print resolution, media specifications, relations between printhead and media, and other technical information, which is specific for each printer model.

Note: Version 8.00 and later versions of Intermec Fingerprint presently only apply to the EasyCoder PF2/4i and EasyCoder PM4i-series printers. Current and phased-out printer models not supporting Intermec Fingerprint v8.xx, but restricted to Intermec Fingerprint v6.xx or v7.xx, are:

Fingerprint v6.XX:

EasyCoder 101	EasyCoder 101 E	EasyCoder 101 SA
EasyCoder 201 IIS	EasyCoder 201 II E	EasyCoder 201 II SA
EasyCoder 401	EasyCoder 401 LinerLess	
EasyCoder 501 S	EasyCoder 501 E	EasyCoder 501 SA
EasyCoder 501 Ticketing	EasyCoder 501 LinerLess	
EasyCoder 601 S	EasyCoder 601 E	

Fingerprint v7.XX:

EasyCoder F2	EasyCoder F4	EasyCoder F4 Compact Industrial
EasyCoder 501 XP	EasyCoder 601 XP	

There are also a number of non Fingerprint-compatible printers in the Intermec EasyCoder product range, such as EasyCoder C4 and EasyCoder E4.

News in Intermec Fingerprint v8.20

General Improvements

- Support for the Intermec Readiness Indicator (IRI).
- Improved EasyLAN support including revised home pages.
- 8 new bar codes supported:
  MicroPDF417, RSS-14, RSS-14 Truncated, RSS-14 Stacked, RSS-14 Stacked Omnidirectional, RSS-14 Limited, RSS-14 Expanded, RSS-14 Expanded Stacked.
- QR Code Model 2 supported.
- Provision for resetting the printer to factory default using the printer's keyboard
- Provision for testing the label stop sensor (LSS) in the Setup Mode.
- Support for the Bidirectional Direct Protocol.

New Instructions

- ERR$ (introduced in Fingerprint v8.10 but not previously documented)
- REPRINT ON/OFF (Direct Protocol only)
- SYSHEALTH
- SYSHEALTH($)

Modified Instructions

- ERROR
  More errors supported in the Direct Protocol.
- LED ON/OFF
  Support for the blue Intermec Readiness Indicator and a new version of the console pcb with bi-color LED in the center position.
- SYSVAR
  New parameter added (SYSVAR (48) for enabling/disabling Bidirectional Direct Protocol.

New Error Codes

1086	Secret not found
1087	Paper Jam
1710	Power supply Generic Error
1711	Power supply Pending
1712	Power supply Status OK
1713	Power supply Power Fail
1714	Power supply Over Volt V24
1715	Power supply Under Volt V24
1716	Power supply Over Volt VSTM
1717	Power supply Under Volt VSTM
1718	Power supply Over Temperature
1719	Power supply Error
1821	Disc quota exceeded

File System with Directories

Two parts of the printer's memory support the use of directories, namely the read-only memory (`rom`) and the read/write permanent storage memory (`c`). Directories cannot be used in any other parts of the memory or in CompactFlash memory cards (`card1`).

The slash letter (`/`) is used as a divisor between directories and files, that is, the path `"/c/DIR1/DIR2/FILE"` refers to a file or directory named `FILE` in the directory `DIR2`, which in its turn is located in the directory `DIR1` in the root of the device `/c` (the printer's permanent memory). The maximum length of a path is 255 characters.

The "old" device names (`c:`, `rom:`, `tmp:`, and so on) are now aliases ("shortcuts") to the new directories (`/c/`, `/rom/`, `/tmp/`, and so on). The file STDIO on `c:` (`/c`) can thus be accessed using either `c:STDIO` or `/c/STDIO`. Writing `c:` is equivalent to writing `/c/`.

The philosophy in the design of the different commands and output formats is to be as backwards-compatible as possible, whilst giving the user access to the new features–directories. Examples of this are:

- FILES give a size of 0 for directories to minimize impact on applications that parse the output.
- FILENAME$ only report files to minimize impact on applications that use FILENAME$ to get file listings.

To relieve the user from always having to use the entire path when referring to a directory above the current one, each directory (including the root directories) contains a "parent directory". This parent directory is called "`..`". It refers to the directory's parent directory. It is listed by `FILES,A`.

Each directory also has a reference to itself ("`.`"), that is, `"/c/./DIR1/./../FILE"` refers to `"/c/FILE"` (or, using the legacy format, to `"c:FILE"`).

Example:

`CHDIR "/c/DIR1/DIR2"`	Changes the directory
`COPY "../DIR3/FILE", "FILE"`	Copies /c/DIR1/DIR3/FILE to /c/DIR1/DIR2/FILE
`CHDIR ".."`	Go up to "/c/DIR1"
`CHDIR "../"`	Go up to /c. Note that a trailing slash (/) may be used.

Note: A file or directory name may contain all printable characters except ":" (colon) and "/" (slash). Only `/c` (`c:`) supports creating and removing directories.

Auxiliary Files in Intermec Fingerprint v8.20

The Intermec Fingerprint v8.20 firmware contains the following files. Some files in "/c/" only appear after default settings have been changed.

In device "/rom/"

.FONTALIAS		Creates reference fonts
.coms		System file
.profile		System file
.setup.saved		Default setup values
.uartx		System file
.ubipfr1.bin		Standard fonts
wi_firm		Firmware for EasyLAN Wireless
CHESS2X2.1		Standard image for test labels
CHESS4X4.1		Standard image for test labels
DIAMONDS.1		Standard image for test labels
ERRHAND.PRG		Error Handler
FILELIST.PRG		List the lines of a file
GLOBE.1		Standard image for test labels
LBLSHTXT.PRG		Intermec Shell auxiliary file
LINE_AXP.PRG		Intermec Shell Line Analyzer
LSHOPXP1.SUB		Intermec Shell auxiliary file
MKAUTO.PRG		Create a startup (autoexec) file
PUP.BAT		Intermec Shell Startup file
SHELLXP.PRG		Intermec Shell startup program
WINXP.PRG		Intermec Shell auxiliary file
default.html		EasyLAN home page
home.htmf		EasyLAN home page
htmlhead.htmf		EasyLAN homepage
images/		EasyLAN home page
	itclogo1.gif	EasyLAN home page
monitor		System file
nav.html		EasyLAN home page
passwd		Default password file
restrictions		Default restriction file
secure/		EasyLAN home page
	configj.js	EasyLAN home page
	configtree.html	EasyLAN home page
	empty.htm	EasyLAN home page
	ftie4style.css	EasyLAN home page
	ftiens4.js	EasyLAN home page
	ftv2blank.gif	EasyLAN home page
	ftv2doc.gif	EasyLAN home page
	ftv2folderclosed.gif	EasyLAN home page
	ftv2folderopen.gif	EasyLAN home page
	ftv2lastnode.gif	EasyLAN home page
	ftv2link.gif	EasyLAN home page
	ftv2mlastnode.gif	EasyLAN home page
	ftv2mnode.gif	EasyLAN home page
	ftv2node.gif	EasyLAN home page
	ftv2plastnode.gif	EasyLAN home page
	ftv2pnode.gif	EasyLAN home page
	ftv2root.gif	EasyLAN home page

	ftv2vertline.gif	EasyLAN home page
	general.html	EasyLAN home page
	mail.html	EasyLAN home page
	main.html	EasyLAN home page
	snmp.html	EasyLAN home page
	tcpip.html	EasyLAN home page
	view.html	EasyLAN home page
	wlan.html	EasyLAN home page
support.htmf		EasyLAN home page

In device "/c/"

.setup.saved		Current setup values (prt section)
.setup.snmp		Current setup values (alerts section)
boot/		
	.domain	Current wireless region setting
	rmmcz	Kernel file
	resu	Kernel file
	psa	Kernel file
	.secrets	Secrets storage (used in TRANFER NET)
	passwd	Password storage
STDIO		Intermec Shell auxiliary file
APPLICATION		Intermec Shell auxiliary file
ADMIN/		
	restrictions	Restrictions storage
	.mibconf	Current setup values (lan1 section)
	.wlanconf	Current setup values (wlan section)
	WEP	WEP key storage (EasyLAN Wireless)

To read the contents of these files, run the FILELIST.PRG program or COPY the file in question to the serial port "uart1:".

2 Program Instructions

This chapter explains all program instructions in alphabetic order, lists their syntaxes and input parameters, and gives some examples how to use the instructions in simple programs.

Syntax

In the syntax descriptions which follow, certain punctuation marks are used to indicate various types of data. They must not be included in the program.

[ ]	indicate that the enclosed entry is optional.
|	indicates alternatives on either side of the bar.
< >	indicate grouping.
.....	indicate repetition of the same type of data.
↔	indicates a compulsory space character between two keywords.
"	is a quotation mark (ASCII 34 dec).
↵	indicates a carriage return or linefeed on the host

Uppercase letters indicate keywords, which must be entered exactly as listed, with the exception that lowercase letters also are allowed unless otherwise stated.

The following abbreviations are used:

<scon>	string constant
<ncon>	numeric constant
<sexp>	string expression
<nexp>	numeric expression
<svar>	string variable
<nvar>	numeric variable
<stmt>	statement
<line label>	line label

ABS

Purpose

Function returning the absolute value of a numeric expression.

Syntax

ABS(<nexp>)

<nexp>	is a numeric expression, from which the absolute value will be returned.

Remarks

The absolute value of a number is always positive or zero. Note that the expression must be enclosed within parentheses.

Examples

```
PRINT ABS(20-25)
5

PRINT ABS(25-20)
5

PRINT ABS(5-5)
0

PRINT ABS(20*-5)
100
```

ACTLEN

Purpose

Function returning the length of the most recently executed PRINT-FEED, FORMFEED, or TESTFEED statement.

Syntax

ACTLEN

Remarks

The length of the most recently executed paper feed operation, resulting from a PRINTFEED, FORMFEED, or TESTFEED statement, will be returned as a number of dots. Due to technical reasons concerning the stepper motor control and label gap detection, a small deviation from the expected result may occur.

Example

In this example, a 12 dots/mm printer is loaded with 90 mm (1080 dots) long labels separated by a 3 mm (36 dots) gap. Start- and stopadjust setup values are both set to 0:

```
10   FORMFEED
20   PRINT ACTLEN
RUN
```

yields:

```
1121
```

The deviation from the expected result (1116) is normal and should have no practical consequences (less than 1 mm).

ALIGN (AN)

Purpose

Statement specifying which part (anchor point) of a text field, bar code field, image field, line, or box will be positioned at the insertion point.

Syntax

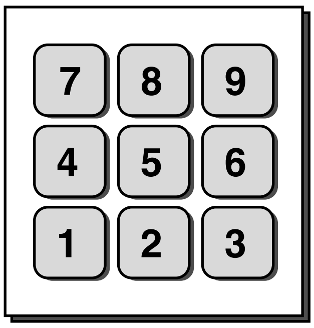
ALIGN|AN<nexp>

<nexp>	is the anchor point of the object (1–9).
Default value:	1
Reset to default by:	PRINTFEED execution

Remarks

Each text, bar code, or image field has nine possible anchor points, whereas lines and boxes have three. One of these points must be selected, or the default value (1) will be used. The selected anchor point decides the position of the object in relation to the insertion point, which is decided by the nearest preceding PRPOS statement. Furthermore, the field will be rotated around the anchor point according to the nearest preceding DIR statement.

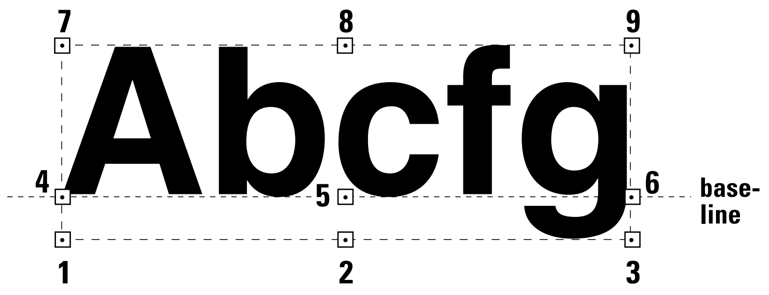
The nine anchor points of a text, bar code, or image field are located in the same manner as, for example, the numeric keys on a computer keyboard:



Lines and boxes have three anchor points only: left, center, and right.

The anchor points for the various types of field are illustrated below.

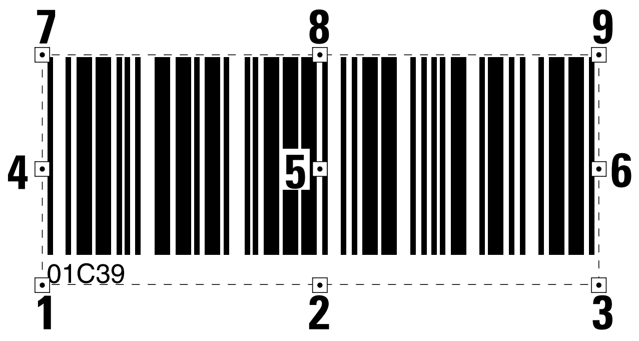
Text field:



A text field makes up an imaginary box limited in regard of width by the length of the text, and in regard of height by the matrix size of the selected font. In text fields, the anchor points 4, 5, and 6 are situated on the baseline, as opposed to bar code fields and image fields.

ALIGN (AN), cont.

Bar Code Field:



A bar code field makes up an imaginary box sufficiently large to accommodate the bar code interpretation, regardless if it will be printed or not (provided that the selected type of bar code may include an interpretation at all).

However, for EAN and UPC codes, the box is restricted in regard of width by the size of the bar pattern, not by the interpretation. This implies that the first digit of the bar code interpretation will be outside the imaginary box:

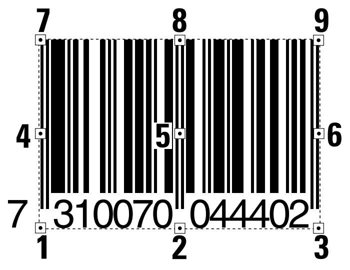
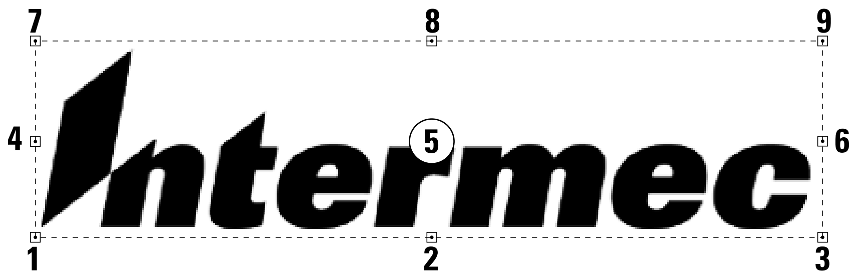


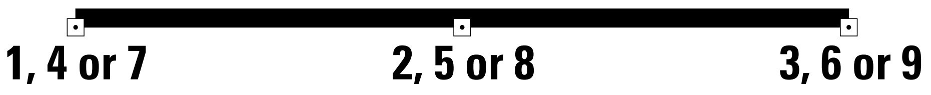
Image field:



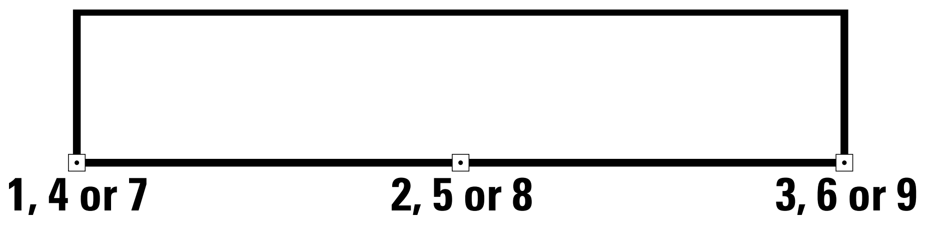
The size of an image field is decided when the field is created. Note that an image field consists of the entire area of the original image, even possible white or transparent background.

ALIGN (AN), cont.

Line:



Box:



The anchor points are situated at the lower side of the line or box in relation to how text is printed in the selected direction. Lines and boxes have only three anchor points, each of which can be specified by means of three different numbers.

A special case is multi-line text fields in a box. The fields can be aligned in nine positions in relation to the box, whereas the box itself only has three anchor points, as described above. Refer to the PRBOX statement for more information on alignment of multi-line text fields.

Example

Printing of a label with a single line of text being aligned left on the baseline:

```
10   PRPOS 30,250
20   DIR 1
30   ALIGN 4
40   FONT "Swiss 721 BT"
50   PRTXT "Hello!"
60   PRINTFEED
RUN
```

The text “Hello!” will be positioned with the baseline aligned left to the insertion point specified by the coordinates X=30; Y=250 in line 10.

ASC

Purpose

Function returning the decimal ASCII value of the first character in a string expression.

Syntax

ASC(<sexp>)

<sexp>	is a string expression, from which the ASCII decimal value of the first character will be returned.

Remarks

ASC is the inverse function of CHR$. The decimal ASCII value will be given according to the selected character set (see NASC statement).

Examples

```
10    A$="GOOD MORNING"
20    PRINT ASC(A$)
RUN
```

yields:

```
71
```

```
10    B$="123456"
20    C% = ASC(B$)
30    PRINT C%
RUN
```

yields:

```
49
```

BARADJUST

Purpose

Statement for enabling/disabling automatic adjustment of bar code position in order to avoid faulty printhead dots.

Syntax

BARADJUST<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is the maximum left offset in dots.
<nexp$_2$>	is the maximum right offset in dots.
Default:	0,0 (BARADJUST disabled)

Remarks

Under unfortunate circumstances, a printer may have to be run for some time with a faulty printhead, before a replacement printhead can be installed. Single faulty dots will produce very thin ”white” lines along the media. This may be tolerable for text, graphics, and vertical (ladder) bar codes, but for horizontal bar codes (picket fence), this condition is likely to render the bar code unreadable.

If the bar code is moved slightly to the left or right, the trace of a faulty dot may come between the bars of the bar code and the symptom is remedied for the time being.

The BARADJUST statement allows the Intermec Fingerprint firmware to automatically readjust the bar code position within certain limits, when a faulty dot is detected (see HEAD function) and marked as faulty (see SET FAULTY DOT statement). The maximum deviation from the original position, as specified by the PRPOS statement, can be set up separately for the directions left and right. Setting both parameters to 0 (zero) will disable BARADJUST.

The BARADJUST statement does not work with:

• Vertically printed bar codes (ladder style)

• Stacked bar codes (for example Code 16K)

• Bar codes with horizontal lines (for example DUN-14/16)

• EAN/UPC-codes (interpretation not repositioned)

Examples

Enabling BARADJUST within 10 dots to the left and 5 dots to the right of the original position for a specific bar code, then disabling it:

```
10   BARADJUST 10,5
20   PRPOS 30,100
30   BARSET "CODE39",2,1,3,120
40   BARFONT ON
50   PRBAR "ABC"
60   BARADJUST 0,0
70   PRINTFEED
```

BARCODENAME$

Purpose

Function returning the names of the bar code generators stored in the printer's temporary memory ("tmp:").

Syntax

BARCODENAME$(<nexp>)

<nexp>	the result of the expression should be either false or true, where... False (0) indicates first bar code. True (≠0) indicates next bar code.

Remarks

BARCODENAME$(0) produces the first bar code name in alphabetical order. BARCODENAME$(≠0) produces next name. Can be repeated as long as there are any bar code names left.

Example

Use a program like this to list the names of all bar codes in "tmp:". Note that bar codes with dynamic downloading will not appear before they have been called by a BARSET or BARTYPE statement.

```
10   A$ = BARCODENAME$ (0)
20   IF A$ = "" THEN END
30   PRINT A$
40   A$ = BARCODENAME$ (-1)
50   GOTO 20
RUN
```

yields for example:

```
CODABAR
CODE11
CODE16K
CODE39
CODE39A
CODE39C
CODE49
CODE93
CODE128
DUN
EAN8
EAN13
EAN128
ADDON5
C2OF5IND
C2OF5INDC
INT2OF5
INT2OF5C
```

etc, etc.

BARFONT (BF)

Purpose

Statement specifying fonts for the printing of bar code interpretation.

Syntax

BARFONT|BF[#<ncon>,]<sexp$_1$>[,<nexp$_1$>[,<nexp$_2$>[,<nexp$_3$> [,<nexp$_4$> [,<nexp$_5$>[,<nexp$_6$>]]]]][ON]

#<ncon>	is, optionally, the start parameter in the syntax above.
<sexp$_1$>	is the name of the font selected for bar code interpretations.
<nexp$_1$>	is the height in points of the font.
<nexp$_2$>	is the clockwise slant in degrees (0-90°).
<nexp$_3$>	is the distance in dots between bar code and bar font.
<nexp$_4$>	is the magnification in regard of height (1-4).
<nexp$_5$>	is the magnification in regard of width (1-4).
<nexp$_6$>	is the width enlargement in percent relative the height (1-1000). Default: 100. Does not work with bitmap fonts.
ON	optionally enables the printing of bar code interpretation.
Reset to default by:	PRINTFEED execution.

Remarks

Start Parameter:
The start parameter specifies which parameter in the syntax above should be the first parameter in the statement. Thereby you may bypass some of the initial parameters.
Default value: #1

Font Name:
This parameter corresponds to the FONT statement, but will only affect bar code interpretation. Double-byte fonts cannot be used.
Default : Swiss 721 BT

Font Size:
This parameter corresponds to the FONT statement, but will only affect bar code interpretation. The size is specified in points. (1 point = 1/72 inch 0.352 mm.)
Default : 12

Font Slant:
This parameter corresponds to the FONT statement, but will only affect bar code interpretation. Slanting increases clockwise. Values greater that 65-70° will be unreadable.
Default : 0

Vertical Offset:
The distance between the bottom of the bar code pattern and the top of the character cell is given as a number of dots.
Default value: 6

BARFONT (BF), cont.

Magnification:
Two parameters allows you to specify the magnification separately in regard of height and width (corresponding to MAG statement). Note that if a MAG statement is executed after a BARFONT statement, the size of the barfont will be affected by the MAG statement.
Default value for both parameters: 1

Width:
A scaleable font can enlarged in regard of width relative height. The value is given as percent (1-1000). This means that if the value is 100, there is no change in the appearance of the characters, whereas if the value is given as for example 50 or 200, the width will be half the height or double the height respectively. When using this parameter, all parameters in the syntax must be included in the statement, (name, height, slant, and width).

Enabling Interpretation Printing:
The printing of bar code interpretation can enabled by a trailing ON, which corresponds to a BARFONT ON statement.

Exceptions:
Note that in all EAN and UPC bar codes, the interpretation is an integrated part of the code. Such an interpretation is not affected by a BARFONT statement, but will be printed in according to specification, provided that interpretation printing has been enabled by a BARFONT ON statement.

Certain bar codes, like Code 16K, cannot contain any interpretation at all. In such a case, the selected barfont will be ignored.

Example

Programming a Code 39 bar code, selecting the same barfont for all directions, and enabling the printing of the bar code interpretation can be done this way:

```
10   PRPOS 30,400
20   DIR 1
30   ALIGN 7
40   BARSET "CODE39",2,1,3,120
50   BARFONT "Swiss 721 BT",10,8,5,1,1,100 ON
60   PRBAR "ABC"
70   PRINTFEED
80   END
```

BARFONT ON/OFF (BF ON/OFF)

Purpose

Statement enabling or disabling the printing of bar code interpretation.

Syntax

BARFONT|BF↔ON|OFF

Default:	BARFONT OFF
Reset to default by:	PRINTFEED execution

Remarks

Usually, you start your program by selecting a suitable bar code interpretation font, see BARFONT. Then use BARFONT ON and BARFONT OFF statements to control whether to print the interpretation or not, depending on application.

BARFONT ON can be replaced by a BARFONT statement appended by a trailing ON, see BARFONT statement.

Example

Programming a Code 39 bar code, selecting a barfont for each direction and enabling the printing of the bar code interpretation. Compare with the example for BARFONT statement:

```
10    PRPOS 30,400
20    DIR 1
30    ALIGN 7
40    BARSET "CODE39",2,1,3,120
50    BARFONT "Swiss 721 BT",10,8,5,1,1
60    BARFONT ON
70    PRBAR "ABC"
80    PRINTFEED
90    END
```

BARHEIGHT (BH)

Purpose

Statement specifying the height of a bar code.

Syntax

BARHEIGHT|BH<nexp>

<nexp>	is the height of the bars in the bar code expressed in number of dots.
Default value:	100 dots.
Reset to default by:	PRINTFEED execution.

Remarks

The barheight specifies the height of the bars, that make up the code. In bar codes consisting of several elements on top of each other, for example Code 16K, the barheight specifies the height of one element. The height is not affected by BARMAG statements.

BARHEIGHT can be replaced by a parameter in the BARSET statement.

Example

Programming a Code 39 bar code, selecting a barfont for all directions and enabling the printing of the bar code interpretation:

```
10    PRPOS 30,400
20    DIR 1
30    ALIGN 7
40    BARTYPE "CODE39"
50    BARRATIO 2,1
60    BARHEIGHT 120
70    BARMAG 3
80    BARFONT "Swiss 721 BT"ON
90    PRBAR "ABC"
100   PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.

BARMAG (BM)

Purpose

Statement specifying the magnification in regard of width of the bars in a bar code.

Syntax

BARMAG|BM<nexp>

<nexp>	is the magnification in regard of width of the bars, which make up the bar code.
Allowed input:	Depends on type of bar code.
Default value:	2
Reset to default by:	PRINTFEED execution.

Remarks

The magnification only affects the bar code ratio (see BARRATIO), not the height of the bars (see BARHEIGHT). For example, by default the BARRATIO is 3:1 and the BARMAG is 2, which means that the wide bars will be 6 dots wide and the narrow bars will be 2 dots wide (2 × 3:1 = 6:2).

The magnification also affects the interpretation in EAN and UPC bar codes, since the interpretation is an integrated part of the EAN/UPC code.

BARMAG can be replaced by a parameter in the BARSET statement.

Example

Programming a Code 39 bar code, selecting a barfont for all directions and enabling the printing of the bar code interpretation:

```
10    PRPOS 30,400
20    DIR 1
30    ALIGN 7
40    BARTYPE "CODE39"
50    BARRATIO 2,1
60    BARHEIGHT 120
70    BARMAG 3
80    BARFONT "Swiss 721 BT" ON
90    PRBAR "ABC"
100   PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.

BARRATIO (BR)

Purpose

Statement specifying the ratio between the wide and the narrow bars in a bar code.

Syntax

BARRATIO|BR<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is the thickness of the wide bars relative to the narrow bars.
<nexp$_2$>	is the thickness of the narrow bars relative to the wide bars.
Default value:	3:1
Reset to default by:	PRINTFEED execution.

Remarks

This statement specifies the ratio between the wide and the narrow bars in a bar code in relative terms. To decide the width of the bars in number of dots, the ratio must be multiplied by the BARMAG value.

Example:
The default BARRATIO is 3:1 and the default BARMAG is 2.
(3:1) × 2 = 6:2
that is, the wide bars are 6 dots wide and the narrow bars are 2 dots wide.

Note that certain bar codes have a fixed ratio, for example EAN and UPC codes. In those cases, any BARRATIO statement will be ignored. Refer to Chapter 5, “Bar Codes” later in this manual.

BARRATIO can be replaced by two parameters in the BARSET statement.

Example

Programming a Code 39 bar code, selecting a barfont for all directions and enabling the printing of the bar code interpretation:

```
10   PRPOS 30,400
20   DIR 1
30   ALIGN 7
40   BARTYPE "CODE39"
50   BARRATIO 2,1
60   BARHEIGHT 120
70   BARMAG 3
80   BARFONT "Swiss 721 BT"ON
90   PRBAR "ABC"
100  PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.

BARSET

Purpose

Statement specifying a bar code and setting additional parameters to complex bar codes.

Syntax

BARSET[#<ncon>,][<sexp>[,<nexp$_1$>[,<nexp$_2$>[,<nexp$_3$>[,<nexp$_4$>[,<nexp$_5$>[,<nexp$_6$>[,<nexp$_7$>[,<nexp$_8$>[,<nexp$_9$>[,<nexp$_{10}$>]]]]]]]]]]]

#<ncon>		is the the start parameter in the syntax above.
<sexp>	(#1)	is the barcode type.
<nexp$_1$>	(#2)	is the ratio of the large bars.
<nexp$_2$>	(#3)	is the ratio of the small bars.
<nexp$_3$>	(#4)	is the enlargement.
<nexp$_4$>	(#5)	is the height of the code in dots (or model in QR Code).
<nexp$_5$>	(#6)	is the security level according to bar code specification.
<nexp$_6$>	(#7)	is the aspect height ratio.
<nexp$_7$>	(#8)	is the aspect width ratio.
<nexp$_8$>	(#9)	is the number of rows in the bar code.
<nexp$_9$>	(#10)	is the number of columns in the bar code.
<nexp$_{10}$>	(#11)	is a truncate flag according to bar code specifications
Reset to default by:		PRINTFEED execution.

Remarks

This statement can replace the statements BARHEIGHT, BARRATIO, BARTYPE, and BARMAG. Although being primarily intended for some complex bar codes such as PDF417, it can be used for any type of bar code if non-relevant parameters are left out (for example <nexp$_5$> to <nexp$_{10}$>).

Note: The parameter descriptions under the syntax do not apply to all bar codes that can be specified using BARSET. Some bar codes have other meanings or use certain parameters only as place holders. See Chapter 5 “Bar Codes” for more information.

Start Parameter:
Start parameter specifies which parameter in the syntax above should be the first optional parameter (#1-#11). Thereby you may bypass some of the initial parameters, for example ratio and enlargement.
Default value: #1

#1 Bar Code Type:
The bar code type parameter corresponds to the BARTYPE statement.
Default bar code: "INT2OF5"

#2 and #3 Bar Code Ratio:
The two ratio parameters correspond to the BARRATIO statement.
Default value: 3:1

#4 Enlargement:
The enlargement parameter corresponds to the BARMAG statement.
Default value: 2

BARSET, cont.

#5 Bar Code Height:
The height parameter corresponds to the BARHEIGHT statement. In CR Code, this parameter is used to specify model (1 or 2).
Default value: 100 dots

#6 Security Level:
The security level is only used in some complex bar codes and should be used according to the specifications of the bar code in question.
Default value: 2

#7 and #8 Aspect Ratios:
The aspect height ratio and aspect width ratio is used for complex bar codes to define the relation between height and width of the pattern. This method of defining the bar code size has lower priority than rows and colomns, see below. Refer to the bar code specifications for allowed input.
Default values:
3 for aspect ratio height
1 for aspect ratio width.

#9 and #10 Rows and Columns:
The rows in bar code and columns in bar code parameters have priority over the aspect height ratio and aspect width ratio, but have the same purpose. Refer to the specifications of the bar code for allowed input.
Default value: 0

#11 Truncate Flag:
The truncate flag is used in some complex bar codes to omit parts of the code pattern. Refer to the specifications of the bar code for allowed input.
Default value: 0

Examples

This example shows how a BARSET statement is used to specify a Code 39 bar code (compare for example with the example for BARTYPE stmt):

```
10   PRPOS 30,400
20   DIR 1
30   ALIGN 7
40   BARSET "CODE39",2,1,3,120
50   BARFONT "Swiss 721 BT",10,8,5,1,1 ON
60   PRBAR "ABC"
70   PRINTFEED
```

This example shows how BARSET is used in two different ways to create a QR Code Model 2 with element size 4 and security code M:

```
BARSET "QRCODE",1,1,4,2,2
```

or

```
BARSET #4,"QRCODE",4,201,2
```

BARSET, cont.

This example shows PDF417 in GM label as per ANSI B-14, with following data:

Data Identifier/Separator	Data	Field name
[)>{RS}		Message Header
06{GS}		Format Header
P	12345678	Part Number
{GS}		Group Separator
Q	160	Quantity
{GS}		
1J	UN123456789A2B4C6D8E	License Plate
{GS}		
20L	LA6-987	Material Handling Code
{GS}		
21L	54321 ZES	Plant/Dock Code
{GS}		
K	GM1234	PO Number
{GS}		
15K	G1155	Kanban Number
{GS}		
B	KLT3214	Container Type
{GS}		
7Q	10GT	Gross Weight
{RS}		Record Separator
{EOT}		Message Trailer

```
10    PRPOS 16,1180
20    DIR 4
30    ALIGN 9
40    BARSET "PDF417",1,1,2,6,5,1,2,0,5,0
50    PRBAR "[)>"+CHR$(30)+"06"+CHR$(29)+
      "P12345678"+CHR$(29)+"Q160"+CHR$(29)+
      "1JUN123456789A2B4C6D8E"+CHR$(29)+
      "20LA6-987" +CHR$(29)+"21L54321 ZES"
      +CHR$(29)+"KGM1234"+CHR$(29)+"15KG1155"
      +CHR$(29)+"BKLT3214"+CHR$(29)+"7Q10GT"
      +CHR$(30)+CHR$(4)
60    PRINTFEED
RUN
```

BARTYPE (BT)

Purpose Statement specifying the type of bar code.

Syntax

BARTYPE|BT<sexp>

<sexp>	specifies the type of bar code.
Allowed input:	Valid bar type name.
Default value:	"INT2OF5"
Reset to default by:	PRINTFEED execution.

Remarks The selected bar code type must exist in the printer's memory and be entered in the form of a string expression. Please refer to Chapter 5, "Bar Codes" later in this manual for a list of the bar codes that are included in the Intermec Fingerprint firmware and their respective designations.

BARTYPE can be replaced by a parameter in the BARSET statement.

Example Programming a Code 39 bar code, selecting a barfont for all directions, and enabling the printing of the bar code interpretation:

```
10    PRPOS 30,400
20    DIR 1
30    ALIGN 7
40    BARTYPE "CODE39"
50    BARRATIO 2,1
60    BARHEIGHT 120
70    BARMAG 3
80    BARFONT "Swiss 721 BT" ON
90    PRBAR "ABC"
100   PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.

BEEP

Purpose

Statement ordering the printer to emit a beep.

Syntax

BEEP

Remarks

This statement makes the printer's built-in buzzer sound at 800 Hz for 1/4 of a second. If a different frequency and/or duration is desired, use a SOUND statement instead.

Example

In this example, a beep is emitted when an error occurs:

```
10    ON ERROR GOTO 1000
.....
.....
.....
1000  BEEP
1010  RESUME NEXT
```

BREAK

Purpose

Statement specifying a break interrupt character separately for the keyboard and each serial communication channel.

Syntax

BREAK<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is one of the following devices: 0 = "console:" 1 = "uart1:" 2 = "uart2:" 3 = "uart3:"
<nexp$_2$>	is the decimal ASCII value for the break interrupt character.
Default:	Comm. channels: ASCII 03 decimal Console: ASCII 158 decimal (<Shift> + <Pause>)

Remarks

The execution of a program can be interrupted using a method specified by the BREAK statement. In addition, the printing of a batch of labels can also be interrupted and resumed by pressing the <Pause> or the <Print> key on the printer's front panel.

To issue a break interrupt, by default, hold down the <Shift> key and press the <Pause> key. Together these keys will produce the ASCII character 158 decimal (128 + 30).

It is possible to remap the keyboard, which may affect the keys used for break interrupt. Please refer to the variable KEYBMAP$.

Another method is to transmit the character ASCII 03 decimal (default) to the printer on one of the serial communication channels. The execution will be interrupted regardless of any INPUT waiting (that is, INPUT [#], LINE INPUT [#], and INPUT$).

The BREAK statement allows you to specify other ways of interrupting the execution, for example by pressing another combination of keys on the printer's keyboard or transmitting another ASCII character from the host.

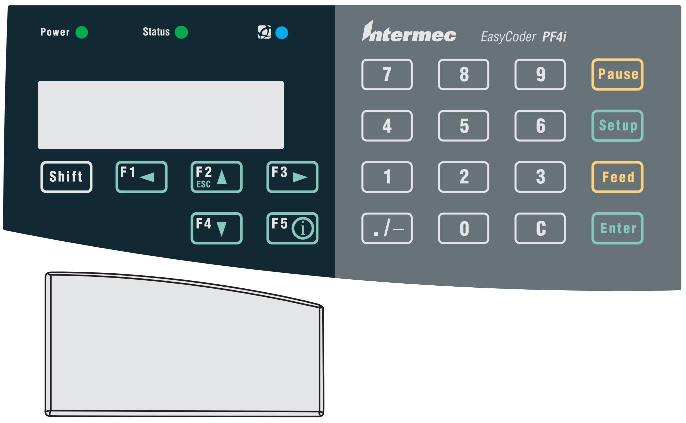
A specified break interrupt character is saved in the temporary memory until the printer is restarted or REBOOTed, which may be confusing for example when switching between programs. To change a break interrupt character, specify a new one for the same device using a BREAK statement and to remove it from memory, use a BREAK OFF statement.

The use of break interrupt is enabled or disabled separately for each device by BREAK ON or BREAK OFF statements. By default, break interrupt on the "console:" is enabled, while break interrupt on any of the communication channels is disabled.

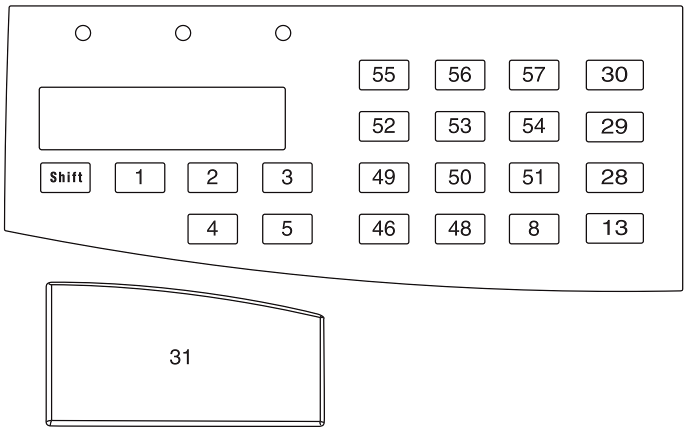
It is strongly recommended to include some facility for issuing a break interrupt from the host computer in startup (autoexec) files. If not, you may find yourself with an erroneous program running in a loop without being able to break it!

BREAK, cont.

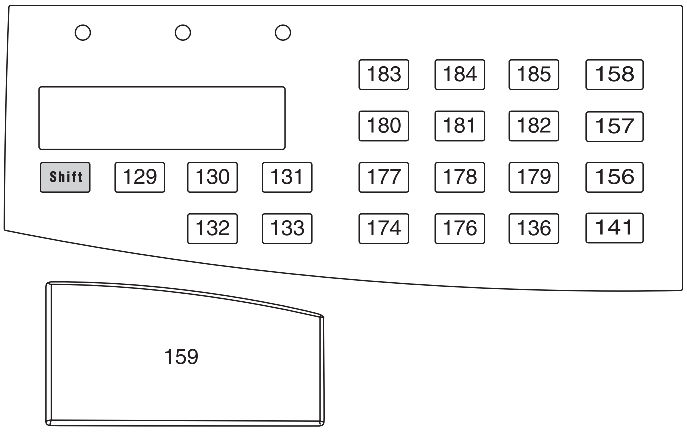
EasyCoder PF-series



Actual keyboard appearance



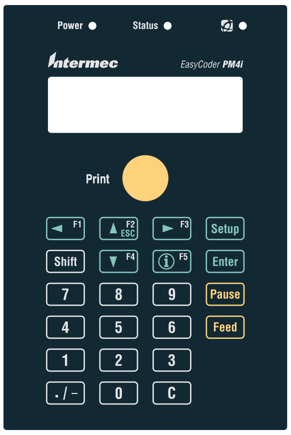
Unshifted keys ASCII values



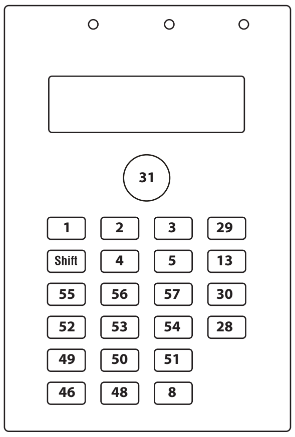
Shifted keys ASCII values

BREAK, cont.

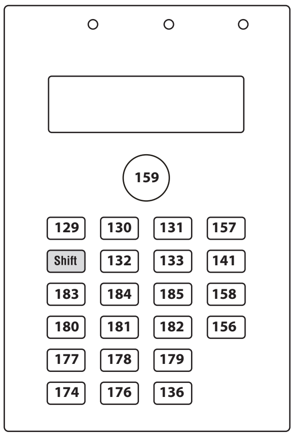
EasyCoder PM4i



Actual keyboard appearance



Unshifted keys; ASCII values



Shifted keys; ASCII values

Examples

In this example, the ASCII character 127 decimal is selected and enabled as BREAK character on the communication channel "uart1:":

```
10    BREAK 1,127
20    BREAK 1 ON
.....
.....
.....
```

In next example, BREAK characters are specified for both the keyboard ("console:") and the serial communication channel "uart1:". The loop can be interrupted either by pressing the key usually marked “F1” on the printer’s keyboard, or by typing an uppercase X on the keyboard of the host:

```
10    BREAK 0,1:BREAK 1,88
20    BREAK 0 ON:BREAK 1 ON
30    GOTO 30
RUN
```

Reset BREAK to default by turning the printer off and on.

BREAK ON/OFF

Purpose

Statement enabling or disabling break interrupt separately for the keyboard and each serial communication channel.

Syntax

BREAK<nexp>ON|OFF

<nexp>	is one of the following devices:	
	0 = "console:"	
	1 = "uart1:"	
	2 = "uart2:"	
	3 = "uart3:"	
Default:	Comm. ports:	Disabled
	Console:	Enabled

Remarks

The use of the break interrupt specified by a BREAK statement can be enabled or disabled separately for each serial communication channel or for the printer's built-in keyboard by BREAK ON or BREAK OFF statements. By default, break interrupt is enabled from the printer's keyboard and disabled from all communication channels.

BREAK OFF deletes any existing break interrupt character stored in the printer's temporary memory for the specified device.

Example

In this example, the ASCII character 127 decimal is selected and enabled as BREAK character on the communication channel "uart1:". At the same time, BREAK from the printer's keyboard is disabled.

```
10   BREAK 1,127
20   BREAK 1 ON:BREAK 0 OFF
.....
.....
.....
```

BUSY

Purpose

Statement ordering a busy signal, for example XOFF, CTS/RTS, or PE, to be transmitted from the printer on the specified communication channel.

Syntax

BUSY[<nexp>]

<nexp>	optionally specifies the channel as: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:"

Remarks

Communication protocol usually contain some "busy" signal, which tells the host computer that the printer, for some reason, is unable to receive data.

The BUSY statement allows you to order a busy signal to be transmitted on the specified communication channel. If no channel is specified, the signal will be transmitted on the standard OUT communication channel, see SETSTDIO statement.

To allow the printer to receive more data, use a READY statement.

For the optional "centronics:" communication channel, BUSY/READY control the PE (paper end) signal on pin 12 according to an error-trapping routine (BUSY = PE high).

Example

You may, for example, want to prevent the printer from receiving more data on "uart2:" during the process of printing a label (running this example requires an optional interface board to be fitted):

```
10   FONT "Swiss 721 BT"
20   PRTXT "HELLO!"
30   BUSY2
40   PRINTFEED
50   READY2
RUN
```

CHDIR

Purpose

Statement specifying the current directory.

Syntax

CHDIR<scon>

<scon>	specifies the current directory (see DEVICES)
Default:	"/c"

Remarks

By default, the printer's permanent memory ("/c") is the current directory, which means the directory that is used if the Intermec Fingerprint instruction does not contain any reference to a directory, for example FILES. This implies that to access the temporary memory ("tmp:"), the storage part of the RTC/Dallas key circuit ("storage:"), or an optional memory card ("/rom" or "card1:"), you must include such a reference in your instructions, for example FILES "/rom".

The CHDIR statement allows you to appoint another directory than "/c" as the current directory. Obviously, this implies that you must specify the permanent memory ("/c") whenever you want to access it.

Example

In this example, the current directory is changed to "card1:", all files in "card1:" are listed, and finally the current directory is changed back to "/c". (This example is only included to illustrate the principles of changing the current directory. It is more efficient to use FILES "card1:" to read its contents.)

```
10   CHDIR"card1:"
20   FILES
30   CHDIR"/c"
RUN
```

CHECKSUM

Purpose

Statement calculating the checksum of a range of program lines in connection with the transfer of programs.

Syntax

CHECKSUM(<nexp$_1$>,<nexp$_2$)

<nexp$_1$>	is the number of the first line in a range of program lines.
<nexp$_2$>	is the number of the last line in a range of program lines.

Remarks

The checksum is calculated from parts of the internal code using an advanced algorithm. Therefore, it is recommended to let the printer calculate the checksum before the transfer of a program. After the transfer is completed, let the receiving printer do the same calculation and compare the checksums.

Example

In this example, the checksum is calculated of all program lines between line 10 and line 2000 in the program "DEMO.PRG".

```
NEW
LOAD "DEMO.PRG"
PRINT CHECKSUM(10,2000)
```

yields:

```
60095
```

CHR$

Purpose

Function returning the readable character from a decimal ASCII code.

Syntax

CHR$(<nexp>)

<nexp>	is the decimal ASCII code to be converted to a readable character.

Remarks

This function is useful for entering characters that cannot be produced from the keyboard of the host, for example non-printable characters ASCII 0-31 dec. Only integers between 0 and 255 are allowed. Input less than 0 or larger than 255 will result in an error condition (Error 41, "Parameter out of range)."

Example

The decimal ASCII code for "A" is 65 and for "B" is 66.

```
10   A$ = CHR$(65)
20   B$ = CHR$(40+26)
30   PRINT A$
40   PRINT B$
RUN
```

yields:

```
A
B
```

CLEANFEED

Purpose

Statement running the printer's feed mechanism.

Syntax

CLEANFEED<nexp>

<nexp>	is the feed length expressed as a positive or negative number of dots.

Remarks

The CLEANFEED statement activates the stepper motor that drives the printer's platen roller (the rubber roller beneath the printhead). In case of thermal transfer printers, it also often drives the ribbon mechanism. The motor will run regardless of possible error conditions, for example if the printhead is lifted or not, or if there is no ribbon or media supply left.

Thus, the CLEANFEED statement is suitable for cleaning and for the loading of transfer ribbon.

A positive CLEANFEED value makes the stepper motor rotate the rollers forward, that is as when feeding out a label.

A negative CLEANFEED value makes the stepper motor rotate the rollers backwards, that is as when pulling back a label.

The execution of a CLEANFEED statement, as opposed to TESTFEED, does not affect the adjustment of the label stop sensor or black mark sensor, regardless what type of media or other objects that passes the sensor.

Note that the CLEANFEED statement, as opposed to FORMFEED, always must be specified in regard of feed length.

Example

In order to pull a cleaning card back and forth under the printhead three times, three 1200 dots long positive CLEANFEEDs and then the same amount of negative CLEANFEEDs are performed:

```
10   FOR A%=1 TO 3
20   CLEANFEED 1200
30   CLEANFEED -1200
40   NEXT
RUN
```

CLEAR

Purpose

Statement clearing strings, variables, and arrays in order to free memory space.

Syntax

CLEAR

Remarks

The CLEAR statement empties all strings, sets all variables to zero, and resets all arrays to their default values. As a result, more free memory space becomes available.

Example

In this example, more free memory space is obtained after the strings have been emptied by means of a CLEAR statement:

```
10    A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20    B$ = "abcdefghijklmnopqrstuvwxyz"
30    FOR I%=0 TO 3:FOR J%=0 TO 3:FOR K%=0 TO 20
40    C$(I%,J%)=C$(I%,J%)+A$
50    NEXT K%:NEXT J%:NEXT I%
60    PRINT "String A before:    ";A$
70    PRINT "String B before:    ";B$
80    PRINT "Free memory before: ";FRE(1)
90    CLEAR
100   PRINT "String A after:     ";A$
110   PRINT "String B after:     ";B$
120   PRINT "Free memory after:  ";FRE(1)
RUN
```

yields:

```
String A before:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
String B before:    abcdefghijklmnopqrstuvwxyz
Free memory before: 1867368
String A after:
String B after:
Free memory after:  1876200

Ok
```

CLIP

Purpose

Statement for enabling/disabling the printing of partial fields.

Syntax

CLIP [BARCODE [HEIGHT|INFORMATION|X|Y]][ON|OFF]

BARCODE	toggles between partial bar code fields enable/disable.
BARCODE HEIGHT	clips the height of the bar so the bar code will fit inside the print window. A one-dimensional bar code may still be readable.
BARCODE INFORMATION	clips the bar code lengthwise, so some bars will be missing, making the bar code unreadable.
BARCODE X	clips the part of the bar code that is outside the X-dimension of the print window.
BARCODE Y	clips the part of the bar code that is outside the Y-dimension of the print window.
ON	enables use of partial text, image, line, and box fields.
OFF	disables use of partial text, image, line, and box fields.

Remarks

Partial fields means that the firmware will accept and print text, bar code, image, lines, and box fields even if they extend outside the print window as specified by the printer's setup in regard of X-Start, Width, and Length. Even negative PRPOS values are allowed. However, all parts the fields outside the print window will be excluded from the printout, that is they will be clipped at the borders of the print window.

There are two main cases:

CLIP BARCODE [HEIGHT|INFORMATION|X|Y] is used for bar code fields only. (Note that some bar codes, like Maxicode, consist of images and should in this context be regarded as image fields.)

CLIP ON|OFF is only used for text, image, line, and box fields.

When the use of partial fields is disabled, the Error 1003, "Field out of label" will result if any field extends outside the print window.

Note the difference between the physical size of the label and the size of the print window specified by the printer's setup. It is the latter that decides were the fields will be clipped.

Example

In this example, only the last part of the text will be printed:

```
10   CLIP ON
20   PRPOS -100,30
30   PRTXT "INTERMEC PRINTER"
40   PRINTFEED
RUN
```

CLL

Purpose

Statement for partial or complete clearing of the print image buffer.

Syntax

CLL [<nexp>]

<nexp>	optionally specifies the field from which the print image buffer should be cleared.

Remarks

The print image buffer is used to store the printable image after processing awaiting the printing to be executed. The buffer can be cleared, partially or completely, by the use of a CLL statement:

- **CLL<nexp>** partially clears the buffer from a specified field to the end of the program. The field is specified by a FIELDNO function.

  Partial clearing is useful in connection with print repetition. To avoid superfluous reprocessing, one or several fields can be erased from the buffer and be replaced by other information, while the remaining parts of the label are retained in the buffer.

  Note that there must be no changes in the layout between the PRINTFEED and the CLL statements, or else the layout will be lost. Also note that partial clearing always starts from the end, which means that the fields which are executed last are cleared first.

- **CLL** (without any field number) clears the buffer completely.

  When certain error conditions have occurred, it is useful to be able to clear the print image buffer without having to print a faulty label. Should the error be attended to, without the image buffer being cleared, there is a risk that the correct image will be printed on top of the erroneous one on the same label. It is therefore advisable to include a CLL statement in your error-handling subroutines, when you are working with more complicated programs, in which all implications may be difficult to grasp.

CLL, cont.

Examples

Partial clearing:
Two labels are printed, each with two lines of text. After the first label is printed, the last line is cleared from the print image buffer and a new line is added in its place on the second label:

```
10    PRPOS 100,300
20    FONT "Swiss 721 BT"
30    PRTXT "HAPPY"
40    A%=FIELDNO
50    PRPOS 100,250
60    PRTXT "NEW YEAR!"
70    PRINTFEED
80    CLL A%
90    PRPOS 100,250
100   PRTXT "BIRTHDAY!"
110   PRINTFEED
RUN
```

Complete clearing:
In this example, the print image buffer will be cleared completely if Error 1030, “Character missing in chosen font” occurs.

```
10    ON ERROR GOTO 1000
.....
.....
.....
1000  IF ERR=1030 GOSUB 1100
1010  RESUME NEXT
.....
....
1100  CLL
1110  PRINT "CHARACTER MISSING"
1120  RETURN
```

CLOSE

Purpose

Statement closing one or several files and/or devices for input/output.

Syntax

CLOSE[[#] <nexp> [, [#] <nexp>...]]

#	optionally indicates that whatever follows is a number.
<nexp>	is the number assigned to a file or device when it was OPENed.

Remarks

This statement revokes OPEN. Only files or devices, which already have been OPENed, can be CLOSEd.

A CLOSE statement for a file or device OPENed for sequential output entails that the data in the buffer will be written to the file/device in question automatically before the channel is closed.

When a file OPENed for random access is CLOSEd, all its FIELD definitions will be lost.

END, NEW, and RUN will also close all open files and devices.

Examples

This statement closes all open files and devices:

```
200   CLOSE
```

A number of files or devices (No. 1-4) can be closed simultaneously using any of the following types of statement:

```
200   CLOSE 1,2,3,4
```

or

```
200   CLOSE #1,#2,#3,#4
```

or

```
200   CLOSE 1,2,#3,4
```

COM ERROR ON/OFF

Purpose

Statement enabling/disabling error handling on the specified communication channel.

Syntax

COM↔ERROR<nexp>ON|OFF

<nexp>	is one of the following communication channels: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:"
Default:	COM ERROR OFF on all channels.

Remarks

This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COMSTAT, and COMBUF$.

Each character received is checked for the following errors:
- Received break
- Framing error
- Parity Error
- Overrun error

If any such communication error occurs and COM ERROR is ON for the channel in question, the reception will be interrupted. This condition can be read by means of a COMSTAT function, but you cannot read exactly what type of error has occurred. COM ERROR OFF disables this type of error-handling for the specified channel.

COM ERROR ON cannot be used with USB (communication channel #6).

Example

In this example, a message will appear on the screen when the reception is interrupted by any of four COMSET conditions being fulfilled:

```
10    COM ERROR 1 ON
20    A$="Max. number of char. received"
30    B$="End char. received"
40    C$="Attn. string received"
50    D$="Communication error"
60    COMSET 1, "A",CHR$(90),"#","BREAK",20
70    ON COMSET 1 GOSUB 1000
80    COMSET 1 ON
90    IF QDATA$="" THEN GOTO 90
100   END
1000  QDATA$=COMBUF$(1)
1010  IF COMSTAT(1) AND 2 THEN PRINT A$
1020  IF COMSTAT(1) AND 4 THEN PRINT B$
1030  IF COMSTAT(1) AND 8 THEN PRINT C$
1040  IF COMSTAT(1) AND 32 THEN PRINT D$
1050  PRINT QDATA$:RETURN
```

COMBUF$

Purpose

Function reading the data in the buffer of the communication channel specified by a COMSET statement.

Syntax

COMBUF$(<nexp>)

<nexp>	is one of the following communication channels: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:"

Remarks

This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMSTAT. Using COMBUF$, the buffer can be read and the content be used in your program.

When the communication has been interrupted by any of the three conditions "end character", "attention string", or "max. no. of char." (see COMSET), you may use an ON COMSET GOSUB subroutine and assign the data from the buffer to a variable as illustrated in the example below.

Note that COMBUF$ filters out any incoming ASCII 00 dec. characters (NUL) by default. Filtering can be enabled/disabled using SYSVAR(44).

Example

In this example, the data from the buffer is assigned to the string variable A$ and printed on the screen:

```
1     REM Exit program with #STOP&
10    COMSET1,"#","&","ZYX","=",50
20    ON COMSET 1 GOSUB 2000
30    COMSET 1 ON
40    IF A$ <> "STOP" THEN GOTO 40
50    COMSET 1 OFF
.....
.....
1000  END
2000  A$= COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```

COMSET

Purpose

Statement setting the parameters for background reception of data to the buffer of a specified communication channel (see COMBUF$).

Syntax

COMSET<nexp$_1$>,<sexp$_1$>,<sexp$_2$>,<sexp$_3$>,<sexp$_4$>,<nexp$_2$>

<nexp$_1$>	is one of the following communication channels: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:"
<sexp$_1$>	specifies the start of the message string (max. 12 char).
<sexp$_2$>	specifies the end of the message string (max. 12 char).
<sexp$_3$>	specifies characters to be ignored (max. 42 char).
<sexp$_4$>	specifies the attention string (max. 12 char).
<nexp$_2$>	specifies the max. number of characters to be received. Enter a value ≥1. (If <nexp$_2$> = 0, the first character will be lost.)

Remarks

Data can be received by a buffer on each of the communication channels without interfering with the running of the current program. At an appropriate moment, the program can fetch the data in the buffer and use them according to your instructions. Such background reception has priority over any ON KEY GOSUB statement.

Related instructions are COMSTAT, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$.

The communication channels are explained in connection with the DEVICES statement.

The start and end strings are character sequences which tells the printer when to start or stop receiving data. Max. 12 characters, may be "".

It is possible to make the printer ignore certain characters. Such characters are specified in a string, where the order of the individual characters does not matter. Max. 42 characters, may be "".

The attention string interrupts the reception. Max. 12 characters, may be "".

The length of the afore-mentioned COMSET strings are checked before they are copied into the internal structure. If any of these strings are too long, Error 26, "Parameter too large" will occur.

When the printer has received the specified maximum number of characters, without previously having encountered any end string or attention string, the transmission will be interrupted. The maximum number of characters also decides how much of the memory will be allocated to the buffer.

COMSET, cont.

The reception of data to the buffer can be interrupted by four conditions:

- If an end string being encountered.
- If an attention string being encountered.
- If the maximum number of characters being received.
- If error-handling is enabled for the communication channel in question (see COM ERROR ON/OFF) and an communication error occurs. This condition can be checked by a COMSTAT function.

Any interruption will have a similar effect as a COMSET OFF statement, that is close the reception, but the buffer will not be emptied and can still be read by a COMBUF$ function. After the reception has been interrupted, an ON COMSET GOSUB statement can be issued to control what will happen next.

COMSET does not support auto-hunting (see SETSTDIO).

Example

This example shows how "uart1:" is opened for background communication. Any record starting with the character # and ending with the character & will be received. The characters Z, Y and X will be ignored. The character = will stop the reception. Max. 50 characters are allowed.

```
1     REM Exit program with #STOP&
10    COMSET1,"#","&","ZYX","=",50
20    ON COMSET 1 GOSUB 2000
30    COMSET 1 ON
40    IF A$ <> "STOP" THEN GOTO 40
50    COMSET 1 OFF
.....
.....
1000  END
2000  A$= COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```

COMSET OFF

Purpose

Statement turning off background data reception and emptying the buffer of the specified communication channel.

Syntax

COMSET<nexp>OFF

<nexp>	is one of the following communication channels: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:"

Remarks

This statement is closely related to COMSET, ON COMSET GOSUB, COMSTAT, COMSET ON, COM ERROR ON/OFF, and COMBUF$.

The COMSET OFF statement closes the reception and empties the buffer of the specified communication channel.

Example

In this example, the COMSET OFF statement is used to close "uart1:" for background reception and empty the buffer:

```
1     REM Exit program with #STOP&
10    COMSET1,"#","&","ZYX","=",50
20    ON COMSET 1 GOSUB 2000
30    COMSET 1 ON
40    IF A$ <> "STOP" THEN GOTO 40
50    COMSET 1 OFF
.....
.....
1000  END
2000  A$= COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```

COMSET ON

Purpose

Statement emptying the buffer and turning on background data reception on the specified communication channel.

Syntax

COMSET<nexp>ON

<nexp>	is one of the following communication channels: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:"

Remarks

This statement is closely related to COMSET, ON COMSET GOSUB, COMSTAT, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. It allows you to open any of the communication channels for background data reception with an empty buffer, provided the communication parameter for the channel has already been set up by a COMSET statement.

When the reception has been interrupted by the reception of an end character, an attention string or the max. number of characters, the buffer can be emptied and the reception reopened by issuing a new COMSET ON statement.

Example

In this example, the COMSET ON statement on line 30 is used to open "uart1:" for background reception. After the buffer has been read, it is emptied and the reception is reopened by a new COMSET ON statement in the subroutine on line 2020:

```
1     REM Exit program with #STOP&
10    COMSET1,"#","&","ZYX","=",50
20    ON COMSET 1 GOSUB 2000
30    COMSET 1 ON
40    IF A$ <> "STOP" THEN GOTO 40
50    COMSET 1 OFF
.....
.....
1000  END
2000  A$= COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```

COMSTAT

Purpose

Function reading the status of the buffer of a communication channel.

Syntax

COMSTAT(<nexp>)

<nexp>	is one of the following communication channels: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:"

Remarks

This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. It allows you to find out if the buffer is able to receive background data, or—if not—what condition has caused the interruption.

The buffer's status is indicated by a numeric expression, which is the sum of the values given by the following conditions:

Copy of hardware handshake bit (not on "net1:" or "usb1:")	0 or 1
Interruption: Max. number of characters received	2
Interruption: End character received	4
Interruption: Attention string received	8
Interruption: Communication error (not on "net1:" or "usb1:")	32

Example

A message will appear on the screen when the reception is interrupted by any of four COMSET conditions being fulfilled:

```
10    COM ERROR 1 ON
20    A$="Max. number of char. received"
30    B$="End char. received"
40    C$="Attn. string received"
50    D$="Communication error"
60    COMSET 1, "A",CHR$(90),"#","BREAK",20
70    ON COMSET 1 GOSUB 1000
80    COMSET 1 ON
90    IF QDATA$="" THEN GOTO 90
100   END
1000  QDATA$=COMBUF$(1)
1010  IF COMSTAT(1) AND 2 THEN PRINT A$
1020  IF COMSTAT(1) AND 4 THEN PRINT B$
1030  IF COMSTAT(1) AND 8 THEN PRINT C$
1040  IF COMSTAT(1) AND 32 THEN PRINT D$
1050  PRINT QDATA$
1060  RETURN
```

CONT

Purpose

Statement for resuming execution of a program that has been interrupted by means of a STOP, BREAK, or DBBREAK statement.

Syntax

CONT

Remarks

The CONT statement may be used to resume program execution after a STOP, BREAK, or DBBREAK statement has been executed. Execution continues at the point where the break happened with the STDIO settings restored.

CONT is usually used in conjunction with DBBREAK or STOP for debugging. When execution is stopped, you can examine or change the values of variables using direct mode statements. You may then use CONT to resume execution. CONT is invalid if the program has been editied during the break.

It is also possible to resume execution at a specified program line using a GOTO statement in the immediate mode.

Example

```
10      A%=100
20      B%=50
30      IF A%=B% THEN GOTO QQQ ELSE STOP
40      GOTO 30
50      QQQ:PRINT "Equal"

Ok
RUN
Break in line 30

Ok
PRINT A%
100

Ok
PRINT B%
50

Ok
B%=100

Ok
CONT
Equal

Ok
```

COPY

Purpose

Statement for copying files.

Syntax

COPY<sexp$_1$>[,<sexp$_2$>]

<sexp$_1$>	is the name and optionally directory of the original file.
<sexp$_2$>	is, optionally, a new name and/or directory for the copy.

Remarks

This statement allows you to copy a file to another name and/or directory as an alternative to LOADing the file in question and then SAVEing it.

If no directory is specified for the original and/or copy, the current directory will be used by default (see CHDIR statement). By default, the current directory is "/c", which is the printer's permanent memory. If the file is to be copied from or to another directory than the current one, the file name must contain a directory reference.

A file cannot be copied to the same name in the same directory.

In addition to copying files to the printer's permanent or temporary memory or a DOS-formatted memory card, a file can also be copied to an output device such as the printer's display or a serial communication channel. Copying a program to the standard OUT channel has the same effect as LOADing and LISTing it.

Note that bitmap fonts and images are not files and therefore cannot be copied.

Examples

In the following examples, "/c" is the current directory.
Copying a file from "card1:" to the current directory without changing the file name:

```
COPY "card1:LABEL1.PRG"
```

Copying a file from "card1:" to the current directory and changing the file name:

```
COPY "card1:FILELIST.PRG","COPYTEST.PRG"
```

Copying a file from "/c" to a directory other than the current one without changing the file name:

```
COPY "/c/FILELIST.PRG","card1:FILELIST.PRG"
```

Copying a file in the current directory to a new name within the same directory:

```
COPY "LABEL1.PRG","LABEL2.PRG"
```

Copying a file in the current directory to serial channel "uart1:":

```
COPY "LABEL1.PRG","uart1:"
```

COUNT&

Purpose Statement for creating a counter (Intermec Direct Protocol only).

Syntax **COUNT& <sexp$_1$>,<nexp$_1$>,<sexp$_2$>**

<sexp$_1$>	is the type of counter parameter to be set:	
	START (start value)	
	WIDTH	(minimum number of digits)
	COPY	(number of copies before update)
	INC	(increment/decrement at update)
	STOP	(stop value)
	RESTART	(restart counting at this value)
<nexp$_1$>	is the counter reference number (integers only)	
<sexp$_2$>	is the parameter value	

Remarks This instruction can only be used in the Intermec Direct Protocol.

The counters can be used in text and bar code fields and are global, which means that they are not connected to any special label or layout, but will be updated at every execution of PRINTFEED statements where the counter in question is used.

Counters are designated using positive integers, for example 1, 2, or 3. When used for printing, they are referred to by “CNT<ncon>$” variables, where <ncon> is the number of the counter as specified by COUNT&, for example CNT5$. A counter variable without a matching counter will be regarded as a common string variable.

The value of the start, stop, and restart parameters decide the type of counter (alpha or numeric). If different types of counter are specified in these parameters, the last entered parameter decides the type. Alpha counters count A-Z whereas numeric counters use numbers without any practical limit.

Counters are not saved in the printer’s memory, but will have to be recreated after each power up. Therefore, it may be wise to save the COUNT& statements as a file in the host.

START:
Decides the first value to be printed. If a single letter is entered (A-Z), the counter will become an alpha counter, and if one or several digits are entered the counter will be numeric. Numeric values can be positive or negative. Negative values are indicated by a leading minus sign.
Default: 1 (numeric) or A (alpha)

COUNT&, cont.

WIDTH:
This parameter can only be used in numeric counters and decides the minimum number of digits to be printed. If the counter value contains a lesser number of digits, leading zero (0) characters will be added until the specified number of digits is obtained. If the number of digits in the counter value is equal to or larger than specified in the width parameter, the value will be printed in its entity.
Default: 1 (no leading zeros)

COPY:
Decides how many copies (labels etc.) will be printed before the counter is updated according to the INC parameter.
Default: 1

INC:
Decides by which value the counter should be incremented or decremented when it is updated. In case of decrementation, the value should contain a leading minus sign.
Default: 1

STOP:
Decides the value after which the counter should start all over again at the value specified by the RESTART parameter. If a single letter is entered (A-Z), the counter will become an alpha counter, and if one or several digits are entered the counter will be numeric. When a counter is decremented, a stop value less than the start value must be given, since the default stop value will never be reached.
Default: 2,147,483,647 (numeric) or Z (alpha)

RESTART:
Decides at which value the counter should start all over again after having exceeded the STOP parameter value. If a single letter is entered (A-Z), the counter will become an alpha counter, and if one or several digits are entered the counter will be numeric.
Default: 1 (numeric) or A (alpha)

Examples

In this example, a counter is created. It will start at number 100 and be updated by a value of 50 after every second label until the value 1000 is reached. Then the counter will start again at the value 200. All values will be expressed as 4-digit numbers with leading zeros.

```
COUNT& "START",1,"100" ↵
COUNT& "WIDTH",1,"4" ↵
COUNT& "COPY",1,"2" ↵
COUNT& "INC",1,"50" ↵
COUNT& "STOP",1,"1000" ↵
COUNT& "RESTART",1,"200" ↵
```

CSUM

Purpose

Statement calculating the checksum of an array of strings.

Syntax

CSUM<ncon>,<svar>,<nvar>

<ncon>	is the type of checksum calculation: 1: Longitudinal Redundancy Check (LRC) XOR in each character in each string array[0][0] xor array[0][1] ... array[n][n] 2: Diagonal Redundancy Check (DRC) right rotation, then XOR on each character in each string rot(array[0][0] xor array[0][1] 3: Longitudinal Redundancy Check (LRC) Strip string of DLE (0x10) before doing the LRC
<svar>	if <ncon> =1 or 2: The array of strings of which the checksum is to be calculated. if <ncon> = 3: Checksum string.
<nvar>	is the variable in which the result will be presented.

Remarks

These types of checksum calculation can only be used for string arrays, not for numeric arrays. In case of CSUM 3,<svar>,<nvar>, the resulting variable will be the indata for next CSUM calculation, unless the variable is reset.

Example

In this example, the DRC checksum of an array of strings is calculated:

```
10   ARRAY$(0)="ALPHA"
20   ARRAY$(1)="BETA"
30   ARRAY$(2)="GAMMA"
40   ARRAY$(3)="DELTA"
50   CSUM 2,ARRAY$,B%
60   PRINT B% :REM DRC CHECKSUM
RUN
```

yields:

```
252
```

CURDIR$

Purpose

Function returning the current directory as the printer stores it.

Syntax

CURDIR$

Example

```
CHDIR "/c/DIR1/DIR2"
PRINT CURDIR$
```

yields:

```
/c/DIR1/DIR2
```

CUT

Purpose

Statement activating an optional cutter.

Syntax

CUT

Remarks

Obviously, this statement only works with printers fitted with a cutter. A cutter is normally used to cut non-adhesive paper strip or to cut through the liner between self-adhesive labels.

When a PRINTFEED statement is executed, the printer feeds out a certain amount of the media according to the printer's setup in regard of startadjust and stopadjust, as explained in its User's Guide . Then the cutter can be activated by a CUT statement.

Example

This program orders the printer to print a text and then cut off the media:

```
10   PRPOS 250,250
20   DIR 1
30   ALIGN 4
40   FONT "Swiss 721 BT"
50   PRTXT "Hello everybody!"
60   PRINTFEED
70   CUT
RUN
```

CUT ON/OFF

Purpose

Statement enabling or disabling automatic cutting after PRINTFEED execution and optionally adjusting the media feed before and after the cutting.

Syntax

CUT [<nexp>] ON|CUT OFF

<nexp>	is optionally the length of media to be fed out before cutting and pulled back after cutting.

Default: CUT OFF

Remarks

This statement makes it possible to enable or disable automatic execution of a CUT operation directly after the execution of each PRINTFEED statement. If any extra media feed in connection with the cutting operation is required, use startadjust and stopadjust setup or specify the desired length of media to be fed out before the cutting is performed and pulled back afterwards in the CUT ON statement.

The amount of media feed (<nexp>) will not automatically be reset to 0 (zero) by an CUT OFF statement, but must be manually be specified as 0 (CUT 0 ON:CUT OFF). However, a reboot resets it to 0.

Example

This program enables automatic cutting and orders the printer to print a text and feed out an extra amount of strip before cutting the media. The media is then pulled back the same distance:

```
10   CUT 280 ON
20   PRPOS 250,250
30   DIR 1
40   ALIGN 4
50   FONT "Swiss 721 BT"
60   PRTXT "Hello everybody!"
70   PRINTFEED
RUN
```

DATE$

Purpose

Variable for setting or returning the current date.

Syntax

Setting the date:	**DATE$=<sexp>**
<sexp>	sets the current date by a 6-digit number specifying Year, Month and Day (YYMMDD).

Returning the date:	**<svar>=DATE$[(<sexp>)]**
<svar>	returns the current date according to the printer's calendar.
<sexp>	is an optional flag "F", indicating that the date will be returned according to the format specified by FORMAT DATE$.

Remarks

This variable works best if a real-time clock circuit (RTC) is fitted on the printer's CPU board. The RTC is battery backed-up and will keep record of the time even if the power is turned off or lost.

If no RTC is installed, the internal clock will be used. After startup, an error will occur when trying to **read** the date or time before the internal clock has been manually **set** by means of either a DATE$ or a TIME$ variable. If only the date is set, the internal clock starts at 00:00:00 and if only the time is set, the internal clock starts at Jan 01, 1980. After setting the internal clock, you can use the DATE$ and TIME$ variables the same way as when an RTC is fitted, until a power off or REBOOT causes the date and time values to be lost.

Date is always entered and, by default, returned in the order YYMMDD, where:

YY	=	Year	Last two digits	(for example 2003 = 03)
MM	=	Month	Two digits	(01-12)
DD	=	Day	Two digits	(01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

The format for how the printer will return dates can be changed by means of a FORMAT DATE$ statement and returned by DATE$("F").

Example

Setting the date and then returning the date in two different formats:

```
10      DATE$ = "031201"                         (sets date)
20      FORMAT DATE$ "DD/MM/YYYY"         (sets date format)
30      PRINT DATE$                 (returns unformatted date)
40      PRINT DATE$("F")              (returns formatted date)
RUN
                                                      yields:
031201
01/12/2003
```

DATEADD$

Purpose

Function returning a new date after a number of days have been added to, or subtracted from, the current date or optionally a specified date.

Syntax

DATEADD$([<sexp$_1$>,]<nexp>[,<sexp$_2$>])

<sexp$_1$>	is any date given according to the DATE$ format, which a certain number of days should be added to or subtracted from.
<nexp>	is the number of days to be added to (or subtracted from) the current date or optionally the date specified by <sexp$_1$>.
<sexp$_2$>	is an optional flag "F", indicating that the date will be returned according to the format specified by FORMAT DATE$.

Remarks

The original date (<sexp$_1$>) should be entered according to the syntax for the DATE$ variable, that is in the order YYMMDD, where:

YY	=	Year	Last two digits	(for example 2003 = 03)
MM	=	Month	Two digits	(01-12)
DD	=	Day	Two digits	(01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

The number of days to be added or subtracted should be specified as a positive or negative numeric expression respectively.

If no "F" flag is included in the DATEADD$ function, the result will be returned according to the DATE$ format, see above.

If the DATEADD$ function includes an "F" flag, the result will be returned in the format specified by FORMAT DATE$.

Example

```
10    DATE$ = "031201"
20    A%=15
30    B%=-10
40    FORMAT DATE$ "DD/MM/YY"
50    PRINT DATEADD$("031201",A%)
60    PRINT DATEADD$("031201",A%,"F")
70    PRINT DATEADD$(B%,"F")
RUN
```

yields:

```
031216
16/12/03
21/11/03
```

DATEDIFF

Purpose

Function returning the difference between two dates as a number of days.

Syntax

DATEDIFF(<sexp$_1$>,<sexp$_2$>)

<sexp$_1$>	is one of two dates (date 1).
<sexp$_2$>	is the other of two dates (date 2).

Remarks

To get the result as a positive numeric value, the two dates, for which the difference is to be calculated, should be entered with the earlier of the dates (date 1) first and the later of the dates (date 2) last, see the first example below.

If the later date (date 2) is entered first, the resulting value will be negative, see the second example below.

Both dates should be entered according to the syntax for the DATE$ variable, that is in the order YYMMDD, where:

YY	=	Year	Last two digits	(for example 2003 = 03)
MM	=	Month	Two digits	(01-12)
DD	=	Day	Two digits	(01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

Examples

Calculation of the difference in days between the dates October 1, 2003 and November 30, 2003:

```
10   A%=DATEDIFF("031001","031130")
20   PRINT A%
RUN
```

yields:

```
60
```

If the later date is entered first, the result will be negative:

```
10   A%=DATEDIFF("031130","031001")
20   PRINT A%
RUN
```

yields:

```
-60
```

DBBREAK

Purpose

Statement for adding or deleting a breakpoint for the Fingerprint Debugger.

Syntax

DBBREAK<nexp>|<sexp>[ON|OFF]

<nexp>	is the line number where the debugger will break and also the name of the breakpoint.
<sexp>	is the line label where the debugger will break and also the name of the breakpoint.
ON	adds the specified breakpoint (default).
OFF	deletes the specified breakpoint.

Remarks

The execution of a program will break at each program line, that has been specified as a breakpoint, and the message “break in line nnn” will be transmitted on the Debug STDOUT port. If a CONT statement is issued, the execution will continue at next line, whereas if RUN is issued, the execution will start again from the first program line.

The line number or line label does not to have to exist when a breakpoint is added, but if a non-existing breakpoint is deleted an error will occur (Error 39 or 70).

There is no error given if a breakpoint is added more than once. When a breakpoint is deleted, all breakpoints with the same name are deleted at the same time. There will only be one break for each line even if there are more than one breakpoint on that line.

When a NEW statement is issued, all breakpoints will be deleted.

If a breakpoint is set on a line with a call to a FOR or WHILE loop, there will only be one break on that line (the first time it is executed).

Related instructions are DBBREAK OFF, DBEND, DBSTDIO, and DBSTEP.

Example

```
10   PRINT "A"
20   PRINT "B"
30   PRINT "C"
DBBREAK 20 ON
RUN
```

yields:

```
A
Break in line 20
```

DBBREAK OFF

Purpose

Statement for deleting all breakpoints for the Fingerprint Debugger.

Syntax

DBBREAK OFF

Remarks

This statement is similar to DBBREAK<nexp>|<sexp>OFF but deletes all breakpoints instead of just one breakpoint at the time.

Related instructions are DBBREAK, DBEND, DBSTDIO, and DBSTEP.

DBEND

Purpose

Statement for terminating the Fingerprint Debugger.

Syntax

DBEND

Remarks

This statement is used for termianting the Fingerprint Debugger prematurely and restore the STDIO settings as they were before the Debugger was started.

Related instructions are DBBREAK, DBBREAK OFF, DBSTDIO, and DBSTEP.

DBSTDIO

Purpose

Statement for selecting the standard IN/OUT channel for the Fingerprint Debugger.

Syntax

DBSTDIO <nexp$_1$>,<nexp$_2$>[,<sexp$_1$>,<sexp$_2$>]

DBSTDIO [<nexp$_1$>,<nexp$_2$>,]<sexp$_1$>,<sexp$_2$>

<nexp$_1$>	is the desired Debug STDIN channel:
	0 = "console:"
	1 = "uart1:" (default)
	2 = "uart2:"
	3 = "uart3:"
	4 = "centronics:"
	5 = "net1:"
	6 = "usb1:"
<nexp$_2$>	is the desired Debug STDOUT channel:
	0 = "console:"
	1 = "uart1:" (default)
	2 = "uart2:"
	3 = "uart3:"
	5 = "net1:"
	6 = "usb1:"
<sexp$_1$>	Preamble (default: empty string)
<sexp$_2$>	Postamble (default: empty string)

Remarks

The maximum size of the preamble and postamble strings is 12 characters.

Related instructions are DBBREAK, DBBREAK OFF, DBEND, and DBSTEP.

Example

This statement selects "uart2:" as Debug STDIO channel. Preamble is specified as “in" and postamble as “out":

```
DBSTDIO 2,2,"in","out"
```

DBSTEP

Purpose

Statement for specifying the interval between breaks for the the Fingerprint Debugger and execute the program accordingly.

Syntax

DBSTEP<ncon>

<ncon>	is the number of lines to be executed before break. Default: 1 line.

Remarks

If <ncon> is omitted, one line will be executed, but if <ncon> is specified as 0, nothing at all will happen.

DBSTEP cannot be used in execution mode (yields Error 78).

When DBSTEP is used on the last line in a program, the line will be executed but there will be no break.

If DBSTEP is used in a program with a FOR or WHILE loop, there will only be one break on the line which is calling for the FOR or WHILE loop (the first time it is executed).

Related instructions are DBBREAK, DBBREAK OFF, DBEND, and DBSTDIO.

Example

```
10   PRINT "11"
20   PRINT "22"
30   PRINT "33"
40   PRINT "44"
50   PRINT "55"
60   PRINT "66"
70   PRINT "77"
80   PRINT "88"
90   PRINT "99"
DBSTEP 4
11
22
33
44
Break in line 50
Ok
DBSTEP
55
Break in line 60
Ok
DBSTEP 2
66
77
Break in line 80
CONT
88
99
Ok
```

DELETE

Purpose

Statement deleting one or several consecutive program lines from the printer's working memory.

Syntax

DELETE<$ncon_1$>[-<$ncon_2$>]

<$ncon_1$>	is the line, or the first line in a range of lines, to be deleted.
<$ncon_2$>	is (optionally) the last line in a range of program lines to be deleted.

Remarks

This statement can only be used for editing the current program in the Immediate Mode and cannot be included as a part of the program execution.

Examples

`DELETE 50`	deletes line 50 from the program.
`DELETE 50-100`	deletes line 50 thru 100 from the program.
`DELETE 50-`	deletes all lines from line 50 to the end of the program.
`DELETE -50`	deletes all lines from the start of the program to line 50.

DELETEPFSVAR

Purpose

Statement for deleting variables saved at power failure.

Syntax

DELETEPFSVAR<sexp>

<sexp>	is the name of the variable to be deleted.

Remarks

Related instructions are SETPFSVAR, GETPFSVAR, and LISTPFSVAR.

Examples

```
DELETEPFSVAR "QCPS%"
DELETEPFSVAR "QS$"
```

DEVICES

Purpose

Statement for returning the names of all devices on the standard OUT channel.

Syntax

DEVICES

Remarks

All devices available to the user in the Intermec Fingerprint firmware will be listed, regardless if they are installed or not. There are also a number of devices for internal use only. The list below indicates if and how the device can be OPENed (see OPEN statement). If you try to OPEN a device, which is not fitted or is disconnected, the message "Error in file name" will be printed to the standard OUT channel (see SETSTDIO). Note that all names of devices are lowercase and most are appended by a colon (:).

Device	Explanation	Can be OPENed for...
c: (= "/c/")	Printer's permanent memory	Input/Output/Random
card1:	CompactFlash memory card	Input/Output/Random
centronics:	Parallel communication port	Input
console:	Printer's display and/or keyboard	Input/Output
dll:	Special applications only	–
finisher:	The finisher interface	Input/Output
lock:	Electronic keys	Input
net1:	EasyLAN	Input/Output
par:	Special applications only	–
rom: (= "/rom/")	Kernel and Read-only memory card	Input
rs485:	RS-485 communication	Input/Output
storage:	Electronic keys	Input/Output/Random
tmp:	Printer's temporary memory	Input/Output/Random
uart1:	Serial communication port	Input/Output
uart2:	Serial communication port	Input/Output
uart3:	Serial communication port	Input/Output
usb1:	Serial communication port	Input/Output
wand:	Data from Code 128 bar code	Input

c: or **/c/** is the printer's permanent read/write memory (Flash SIMMs). It supports file system with directories and will retain its content when the power is switched off. For compatibility with programs created in previous versions of Intermec Fingerprint, the designation "ram:" will also be accepted.

card1: is a read/write DOS-formatted CompactFlash memory card inserted in the printer's memory card adapter.

centronics: is the Centronics parallel port. Three different types can be selected by means of SYSVAR(25).

console: is the printer's display and keyboard. The keyboard can be used for input only and the display for output only.

DEVICES, cont.

dll: is used for special applications only.

finisher: is the device controlling the finisher interface, where for instance a cutter can be connected.

lock: is an electronic key items that has been specified as locks by means of special software. An electronic key may contain several key items with different properties (counter, lock, or storage). The device name calls all key items with the corresponding properties. Each key item has a 4-character name, usually appended by a delimiter (?) and a 4-character password. Also see OPEN statement.

net1: is the communication channel for an EasyLAN interface board.

par: is used for special applications only.

rom: or **/rom/** is both the read-only kernel sectors in the Boot-Bank flash SIMM, and any resource files on a CompactFlash memory card inserted in the printer's memory card adapter. It supports file system with directories.

rs485: is used in connection with RS-485 point-to-point or multidrop communication to specify that the RS-485 protocol is used and to specify the protocol address of the unit, for example "rs485:23".

storage: is all electronic key items in the printer that has been specified as storages by means of special software. Note that this memory is comparatively slow.

tmp: is the printer's temporary read/write memory (SDRAM SIMMs). It will lose its content when the power is turned off or at a power failure. Thus, do not use SDRAM for valuable data that cannot be recreated, but copy it to "/c/". One advantage of using "tmp:" instead of "/c" is that data can be written to SDRAM faster than to the flash memory. To speed up operation, the Intermec Fingerprint firmware (except program modules with dynamic downloading) is copied from "/rom/" to "tmp:" at startup and used from "tmp:".

uart1: is the standard RS-232 port.

uart2: is an additional serial port on an optional interface board.

uart3: is an additional serial port on an optional interface board.

usb1: is the standard USB (Universal Serial Bus) port.

wand: is any input from an Code 128 bar code not containing any FNC3 character via a bar code wand or reader connected to the wand interface.

DEVICES, cont.

Example

```
DEVICES
```

yields for example:

```
c:
card1:
centronics:     (only if an optional parallel interface board is fitted)
console:
dll:
finisher:
lock:
net1:           (only if an EasyLAN interface board is fitted)
par:
rom:
rs485:          (only if an optional serial interface board is fitted)
storage:
tmp:
uart1:
uart2:          (only if an optional serial interface board is fitted)
uart3:          (only if an optional serial interface board is fitted)
usb1:
wand:
```

DIM

Purpose

Statement specifying the dimensions of an array.

Syntax

DIM<<nvar>|<svar>>(<nexp$_1$>[,<nexp$_2$>...])....[,<<nvar>|<svar>>(<nexp$_1$>[,<nexp$_2$>...])]

<nvar>|<svar>	is the name of the array.
<nexp$_1$>	is the max. subscript value for the first dimension.
<nexp$_{2-10}$>	are, optionally, the max. subscript value for the following dimensions (No. 2-10).

Remarks

An array is created by entering a variable followed by a number of subscripts (max 10) separated by commas. All the subscripts are enclosed by parentheses. Each subscript represents a dimension. The number of subscripts in an array variable, the first time (regardless of line number) it is referred to, decides its number of dimensions. The number of elements in each dimension is by default restricted to four (No. 0-3).

If more than 4 elements in any dimension is desired, a DIM statement must be issued. Note that 0 = 1:st element, 1 = 2:nd element, etc.

For example ARRAY$(1,2,3) creates a three-dimensional array, where the dimensions each contain 4 elements (0-3) respectively. This corresponds to the statement DIM ARRAY$(3,3,3).

It is not possible to change the number of dimensions of an array that already has been created during runtime. (Error 57, "Subscript out of range" will occur.)

Considering the printer's limited memory and other practical reasons, be careful not to make the arrays larger than necessary. A DIM statement can be used to limit the amount of memory set aside for the array.

Examples

This example creates an array containing three dimensions with 13 elements each:

```
100   DIM NAME$(12,12,12)
```

Here, two one-dimensional arrays are created on the same program line:

```
10    DIM PRODUCT$(15), PRICE%(12)
20    PRODUCT$(2)="PRINTER"
30    PRICE%(2)=1995
40    PRINT PRODUCT$(2);" $";PRICE%(2)
RUN
```

yields:

```
PRINTER $1995
```

DIR

Purpose

Statement specifying the print direction.

Syntax

DIR<nexp>

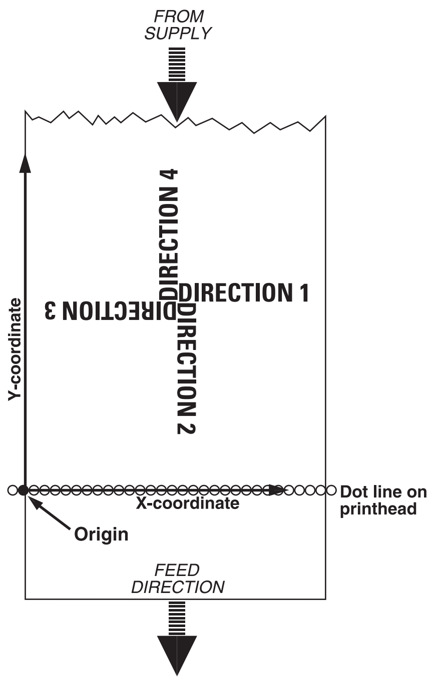
<nexp>	is the print direction (1, 2, 3, or 4).
Default value:	1
Reset to default by:	PRINTFEED execution

Remarks

A change of print direction affects all printing statements, that is PRTXT, PRBAR, PRIMAGE, PRBOX, and PRLINE statements that are executed later in the program until a new DIR statement or a PRINTFEED statement is executed.

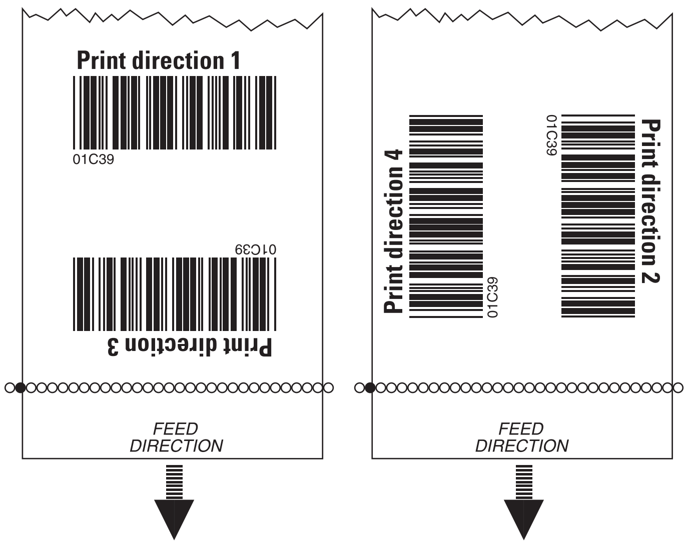
The print direction is specified in relation to the media feed direction as illustrated below. The print direction affects the various types of objects as follows:

Text:



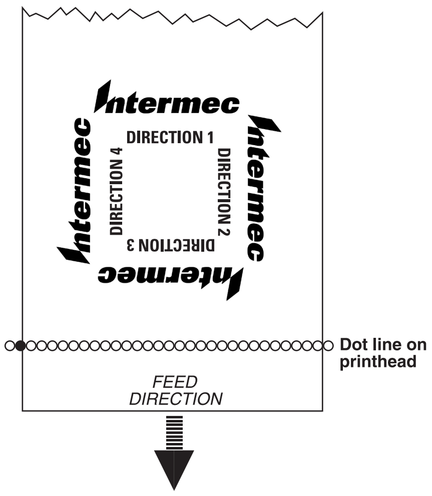
DIR, cont.

Bar Codes:



Horizontal "picket fence" printing vs. vertical "ladder" printing.

Images:



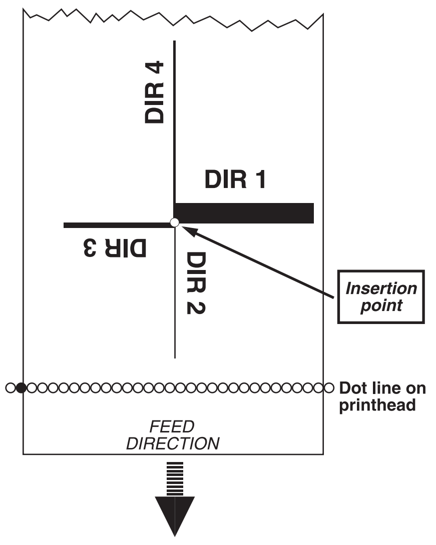
The relation of the image and the print direction depends how the image was drawn. An image can only be "rotated" 180°. Thus, it may be useful to have two copies of the image available with different extensions for either horizontal or vertical printing:

DIR 1 & 3, use extension .1

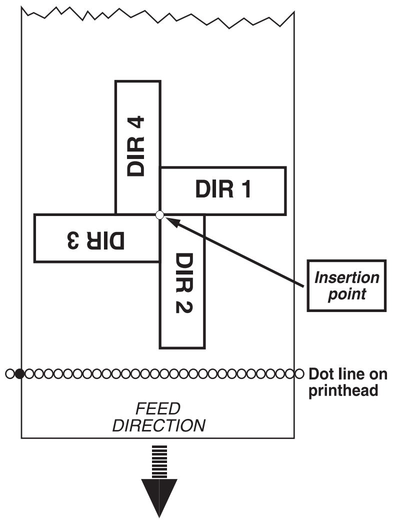
DIR 2 & 4, use extension .2

DIR, cont.

Lines:



Boxes:



Examples

Printing a label with one line of text and drawing a line beneath the text:

```
10    PRPOS 30,300
20    DIR 1
30    ALIGN 4
40    FONT "Swiss 721 BT",18
50    PRTXT "TEXT PRINTING"
60    PRPOS 30,280
70    PRLINE 555,10
80    PRINTFEED
RUN
```

Printing the same information vertically necessitates new positioning to avoid Error 1003, “Field out of label."

```
10    PRPOS 300,30                             (new position)
20    DIR 4                                   (new direction)
30    ALIGN 4
40    FONT "Swiss 721 BT",18
50    PRTXT "TEXT PRINTING"
60    PRPOS 320,30 (new position)
70    PRLINE 555,10
80    PRINTFEED
RUN
```

DIRNAME$

Purpose

Function returning the names of the directories stored in the specified part of the printer's memory.

Syntax

DIRNAME$[(<sexp>)]

<sexp>	is the name of the memory device from which the first directory name will be listed.

Remarks

In <sexp>, parts of directory names and wildcards (*) are allowed. If <sexp> is omitted, the next directory name in the same memory device is listed. Can be repeated. When there are no directories left to list, the output string will be empty. Also see FILENAME$.

Example

```
FILES,A
Files on /c
./                              0   ../              0
.setup.saved                  239   DIR1/
STDIO                           3

4124672 bytes free   242 bytes used

PRINT DIRNAME$("/c/")
.

Ok
PRINT DIRNAME$
..

Ok
PRINT DIRNAME$
DIR1

Ok
```

END

Purpose

Statement ending the execution of the current program or subroutine and closing all OPENed files and devices.

Syntax

END

Remarks

END can be placed anywhere in a program, but is usually placed at the end. It is also useful for separating the “main” program from possible subroutines with higher line numbers. It is possible to issue several END statements in the same program.

Example

A part of a program, which produces fixed line-spacing, may look this way:

```
10   FONT"Swiss 721 BT"
20   X%=300:Y%=350
30   INPUT A$
40   PRPOS X%,Y%
50   PRTXT A$
60   Y%=Y%-50
70   IF Y%>=50 GOTO 30
80   PRINTFEED
90   END
```

The Y-coordinate will be decremented by 50 dots for each new line until it reaches the value 50. The END statement terminates the program.

EOF

Purpose

Function for checking for an end-of-file condition.

Syntax

EOF(<nexp>)

<nexp>	is the number assigned to the file when it was OPENed.

Remarks

The EOF function can be used with files OPENed for sequential input in connection with the statements INPUT#, LINE INPUT#, and INPUT$ to avoid the error condition "Input past end" which has no error message. When the EOF function encounters the end of a file, it returns the value -1 (true). If not, it returns the value 0 (false).

Example

```
10    DIM A%(10)
20    OPEN "DATA" FOR OUTPUT AS #1
30    FOR I%=1 TO 10
40    PRINT #1, I%*1123
50    NEXT I%
60    CLOSE #1
70    OPEN "DATA" FOR INPUT AS #2
80    I%=0
90    WHILE NOT EOF(2)
100   INPUT #2, A%(I%):PRINT A%(I%)
110   I%=I%+1:WEND
120   IF EOF(2) THEN PRINT "End of File"
RUN
```

yields:

```
1123
2246
3369
4492
5615
6738
7861
8984
10107
11230
End of File
```

ERL

Purpose

Function returning the number of the line on which an error condition has occurred.

Syntax

ERL

Remarks

Also useful in connection with an ON ERROR GOTO statement.

Examples

You can check at which line the last error since power up occurred like this:

```
PRINT ERL
```

yields for example

```
40
```

In this example, the line number of the line, where an error has occurred, decides the action to be taken (in this case the font size is too large for the label width):

```
10   ON ERROR GOTO 1000
20   FONT "Swiss 721 BT",100
30   PRTXT "HELLO EVERYBODY"
40   PRINTFEED
50   END
1000 IF ERL=40 THEN PRINT "PRINT ERROR"
1010 RESUME NEXT
RUN
```

yields:

```
PRINT ERROR
```

You can use the ERL function in programs without line numbers too, because such programs have automatically generated hidden line numbers that are revealed when the program is LISTed. This is the same program as above but without line numbers:

```
NEW
IMMEDIATE OFF
ON ERROR GOTO QAAA
FONT "Swiss 721 BT",100
PRTXT "HELLO EVERYBODY"
PRINTFEED
END
QAAA: IF ERL=40 THEN PRINT "PRINT ERROR"
RESUME NEXT
IMMEDIATE ON
RUN
```

yields:

```
PRINT ERROR
```

ERR

Purpose

Function returning the code number of an error that has occurred.

Syntax

ERR

Remarks

The firmware is able to detect a number of error conditions. The errors are represented by code numbers according to Chapter 7, “Error Messages.” The ERR function enables the program to read the coded error number. Thereby you may design your program to take proper action depending on which type of error that may have occurred.

Example

In this example, the code number of the error decides the action to be taken:

```
10    ON ERROR GOTO 1000
. . . . .
. . . . .
100   PRTXT "HELLO"
110   PRINTFEED
120   END
. . . . .
. . . . .
. . . . .
1000  IF ERR=1005 THEN PRINT "OUT OF PAPER"
1010  RESUME NEXT
```

You can also check the number of the last error since power up:

```
PRINT ERR
```

yields for example:

```
1022
```

ERR$

Purpose

Function for returning the explanation of an error code in plain text.

Syntax

ERR$(<nexp>)

<nexp>	is the error code number

Remarks

The explanation of the error is returned in English according to Chapter 7 “Error Messages.”

Example

`PRINT ERR$(1003)` yields:

```
Field out of label
```

ERROR

Purpose

Statement for defining error messages and enabling error handling for specified error conditions (Intermec Direct Protocol only).

Syntax

ERROR <nexp>[,<sexp>]

<nexp>	is the number of the error condition.
<sexp>	is the desired error message.

Remarks

The ERROR statement can only be used in the Intermec Direct Protocol for the purpose of enabling error-handling and creating customized error messages, as described below.

The built-in error-handler of the Intermec Direct Protocol will always handle the following standard errors (display message inside brackets):

• 15	Font not found	[Font not found]
• 18	Disk full	[Disk full]
• 37	Cutter device not found	[Cutter device not found]
• 43	Memory overflow	[Memory overflow]
• 1003	Field out of label	[Field out of label]
• 1005	Out of paper	[Out of paper]
• 1006	No field to print	[No field(s)]
• 1022	Head lifted	[Head lifted]
• 1027	Out of transfer ribbon	[Out of ribbon]
• 1031	Next label not found	[Label not found]
• 1058	Transfer ribbon is installed	[Ribbon installed]
• 1606	Testfeed not done	[Testfeed not done]

Other errors will not be handled unless they have been specified by an ERROR statement. The number of the error should be entered according to the list of error messages at the end of this manual.

The ERROR statement also allows you to edit a suitable message in any language. This message will appear in the printer's display window if the error occurs. The error message will be truncated to 33 characters. Character No. 1-16 will appear on the upper line and character 18-33 will appear on the lower line, whereas character No. 17 always is ignored.

ANSI control characters can be used in the error message string, see "Printer Function Control; Display" in the *Intermec Fingerprint, Tutorial*. An empty string removes any previously entered message for the error in question. Likewise, an existing message can be replaced by a new one.

When a standard error or an error defined by an ERROR statement is detected, the printer sets its standard IN port to BUSY, sets the "Status" LED to red, and displays the error messages. The error message will be cleared, the LED is set to green, and the standard IN port will be set to READY when the printer's <Print> key is pressed. In some cases, the error must also be cleared, for example by loading a fresh stock of labels.

ERROR, cont.

Note: In printers, that are not fitted with the Intermec Readiness Indicator (IRI) console, the green Ready LED will be switched off and the red Error LED will be switched on when an error is detected. The opposite will happen when the error is cleared.

Error messages are not saved in the printer's memory, but new ERROR statements will have to be downloaded after each power up. Therefore, it is recommended to save a set of ERROR statements as a file in the host computer.

The ERROR statements affect both the error messages in the printer's display window and the error messages returned to the host via the standard OUT channel (see SETSTDIO statement).

By default, no error messages are returned to the host in the Intermec Direct Protocol, since the statement INPUT ON sets the verbosity level to off, that is SYSVAR (18)= 0. However, the verbosity level can be changed by means of VERBON/VERBOFF statements or the SYSVAR (18) system variable.

Different types of error messages to be returned on the standard OUT channel can be selected by means of the SYSVAR (19) system variable. If SYSVAR (19) is set to 2 or 3, the error message specified by ERROR is transmitted. If no such error message is available, a standard error message in English will be transmitted (see list of Error Messages in Chapter 7).

Examples

In these examples, a few errors are specified. Note the blank spaces for character position 17 in each message (space characters are indicated by doubleheaded arrows):

```
ERROR 1010,"HARDWARE↔↔↔↔↔↔↔↔↔ERROR" ↵
ERROR 1029,"PRINTHEAD↔VOLT-↔↔AGE↔TOO↔HIGH" ↵
```

EXECUTE

Purpose

Statement for executing a Fingerprint program line or a file with Fingerprint program lines from within another Fingerprint program.

Syntax

EXECUTE<sexp>

<sexp>	is one line of Fingerprint instructions or the name of a file containing at least one line of a Fingerprint program.

Remarks

This statement allows you to create a library of layouts, subroutines, texts, etc, which can be executed as a part of a program without having to merge the programs.

The program called by EXECUTE must not contain any line numbers or line labels.

If the EXECUTE statement is followed by a string of Fingerprint instructions, they should be separated by colons.

When an error occurs in an EXECUTE file, the line number in the error message is that of the EXECUTE file, not of the program where the EXECUTE statement is issued.

EXECUTE is only allowed in the execute mode, not in the immediate mode (yields Error 69).

Recursive call of EXECUTE is not allowed (yields Error 78).

Example

This example shows how a preprogrammed file containing a bar code is executed as a part of a Fingerprint program, where the input data and printfeed are added:

```
IMMEDIATE OFF
DIR 1
ALIGN 7
BARSET "CODE39",2,1,3,120
BARFONT "Swiss 721 BT",10,8,5,1,1
BARFONT ON
IMMEDIATE ON
SAVE "tmp:BARCODE.PRG",L

NEW
10   PRPOS 30,400
20   EXECUTE "tmp:BARCODE.PRG"
30   PRBAR "ABC"
40   PRINTFEED
RUN
```

FIELD

Purpose

Statement for creating a single-record buffer for a random file and dividing the buffer into fields to which string variables are assigned.

Syntax

FIELD[#]<nexp$_1$>,<nexp$_2$>AS<svar$_1$>[,<nexp$_3$>AS<svar$_2$>...]

#	indicates that whatever follows is a number. Optional.
<nexp$_1$>	is the number assigned to the file when it was OPENed.
<nexp$_{2-n}$>	is the number of bytes to be reserved for the string variable that follows. (Null not allowed.)
<svar$_{1-n}$>	is the designation of the string variable, for which space has been reserved.

Remarks

The buffer is divided into fields, each of which is given an individual length in bytes. A string variable is assigned to each field. This statement does not put any data in the buffer, it only creates and formats the buffer, allowing you to place the data using LSET and RSET statements.

Before using this statement, consider the maximum number of characters (incl. space characters) needed for each variable and check that the total does not exceed the record size given when the file was OPENed (by default 128 bytes).

When a file is CLOSEd, all its FIELD definitions will be lost.

Example

This example opens and formats a file buffer for a single record. The buffer is divided into three fields, with the size of 25, 30, and 20 bytes respectively.

```
10   OPEN "ADDRESSES" AS #8 LEN=75
20   FIELD#8,25 AS F1$, 30 AS F2$, 20 AS F3$
```

(Imagine a spreadsheet matrix where the file is the complete spreadsheet, the records are the lines and the fields are the columns. The buffer can only contain one such line at the time.)

FIELDNO

Purpose

Function getting the current field number for partial clearing of the print buffer by a CLL statement.

Syntax

FIELDNO

Remarks

By assigning the FIELDNO function to one or several numeric variables, you can divide the print buffer into portions, which can be cleared using a CLL statement.

Example

```
10    PRPOS 100,300
20    FONT "Swiss 721 BT"
30    PRTXT "HAPPY"
40    A%=FIELDNO
50    PRPOS 100,250
60    PRTXT "NEW YEAR"
70    B%=FIELDNO
80    PRPOS 100, 200
90    PRTXT "EVERYBODY!"
100   PRINTFEED
110   CLL B%
120   PRPOS 100,200
130   PRTXT "TO YOU!"
140   PRINTFEED
150   CLL A%
160   PRPOS 100,250
170   PRTXT "BIRTHDAY"
180   PRPOS 100,200
190   PRTXT "DEAR TOM!"
200   PRINTFEED
RUN
```

yields three labels:

```
#1                  #2                    #3
HAPPY               HAPPY                 HAPPY
NEW YEAR        NEW YEAR             BIRTHDAY
EVERYBODY!          TO YOU!               DEAR TOM!
```

FILE& LOAD

Purpose Statement for receiving and storing binary files in the printer's memory.

Syntax **FILE& LOAD[<nexp$_1$>,]<sexp>,<nexp$_2$>[,<nexp$_3$>]**

<nexp$_1$>	is optionally the number of bytes to skip before starting to read the file data.
<sexp>	is the desired name of the file when stored in the printer's memory.
<nexp$_2$>	is the size of the file in number of bytes.
<nexp$_3$>	optionally specifies a communication channel OPENed for INPUT by the number assigned to the device. (Default: Std IN channel.)

Remarks This statement prepares the printer to receive a binary file on the standard IN channel (see SETSTDIO statement) or on another communication channel OPENed for INPUT.

Another, but more cumbersome, way of obtaining the same result is to use the TRANSFER KERMIT statement.

Image files and font files can also be downloaded using the IMAGE LOAD statement.

As opposed to IMAGE LOAD and TRANSFER KERMIT statements, FILE& LOAD will not immediately install the fonts, but the font files will remain as files in the printer's memory until next power-up.

The optional first parameter makes it possible to use this statement in MS-DOS (CR/LF problem).

The name of the file, when stored in the printer's memory, may consist of max. 30 characters including possible extension.

The size of the original file should be given in bytes according to its size in the host.

Before the FILE& LOAD statement can be used on a serial channel, the setup must be changed to 8 characters, RTS/CTS handshake. When a FILE& LOAD statement is executed, the execution stops and waits for the number of bytes specified in the statement to be received. During the transfer of file data to the printer, there is a 25 sec. timeout between characters. If a new character has not been received within the timeout limit, an error occurs (Error 80, "Download timeout"). When the specified number of characters have been received, the execution is resumed.

Example

```
10   OPEN "uart2:" FOR INPUT AS 5
20   FILE& LOAD "FILE1.PRG",65692,5
30   CLOSE 5
```

FILENAME$

Purpose

Function returning the names of the files stored in the specified part of the printer's memory.

Syntax

FILENAME$[(<sexp>)]

<sexp>	is the name of the memory device from which the first file name (in alphabetical order) will be listed. Parts of file names and wildcards (*) are allowed. Maximum size is 30 characters. If <sexp> is omitted, the next file name in the same memory device is listed. Can be repeated. When there are no files left to list, the output string will be empty.

Remarks

The specified memory device must be mounted. The file name must correspond to the name of the file stored in the memory device in regard of upper- and lowercase characters. Wildcards (* = ASCII 42 dec.) can be used. The list may include all type of files. Even system files, that are preceded by a period character (for example .FONTALIAS), may be listed. No directories will be listed and the order of listing is not specified. Also see DIRNAME$.

Example

This example shows how all files in the printer's permanent memory (`/c`) are listed:

```
FILES,A
Files on /c

./                     0  ../                    0
.setup.saved         239  DIR1/                  0
STDIO                  3

4124672 bytes free   242 bytes used

PRINT FILENAME$("/c/")
.setup.saved

Ok
PRINT FILENAME$
STDIO

PRINT FILENAME$

Ok
```

FILES

Purpose

Statement for listing the files stored in one of the printer’s directories to the standard OUT channel.

Syntax

FILES[<sexp>][,R][,A]

<sexp>	optionally specifies the directory (see DEVICES).
R	lists directories recursively
A	lists all files including system files (that is, files with a name starting with a period (.) character.

Remarks

If no directory is specified, files in the printer’s current directory will be listed. As default, the current directory is the printer’s permanent memory ("/c"), see CHDIR statement.

By including a reference to a memory device ("/c", "tmp:", "/rom", "card1:", "lock:", or "storage:", see DEVICES statement), the files of the specifies directory will be returned without having to change the current directory.

If the “A" flag is omitted, all files, except system files, will be listed The flags A and R can be entered in any order, but R is always processed first.

The number of bytes for each file and the total number of free and used bytes in the specified directory will also be included in the list.

Examples

The presentation may look like this on the host screen:

```
FILES "/c",R
Files on /c

STDIO                2   FILE2                4
DIR1/                0

Files on /c/DIR1/

FILE1                4   DIR2/                0
STDIO                2

No files on /c/DIR1/DIR2

4121600 bytes free   12 bytes used
```

FILES, cont.

```
FILES,R,A
Files on /c

./                      0   ../                 0
DIR1/                   0   FILE2               4
STDIO                   2   .setup.saved      239

Files on /c/DIR1/

./                      0   ../                 0
DIR2/                   0   STDIO               2
FILE1                   4

Files on /c/DIR1/DIR2/
./                      0   ../                 0

4121600 bytes free   251 bytes used


FILES "/c/DIR1"
Files on /c/DIR1

FILE1                   4   DIR2/               0
STDIO                   2

4121600 bytes free   6 bytes used
```

FLOATCALC$

Purpose

Function for calculation with float numbers.

Syntax

FLOATCALC$(<sexp$_1$>,<sexp$_2$>,<sexp$_3$>[,<nexp$_1$>])

<sexp$_1$>	is the first operand.
<sexp$_2$>	is the operator (+, -, *, or /).
<sexp$_3$>	is the second operand.
<nexp$_1$>	is, optionally, the precision in decimals (default 10).

Remarks

Operands are float numbers, that is, a string of digits with a decimal point to separate decimals from integers. Operands can also contain leading plus (+), minus (-), and space characters. Space characters are ignored, whereas the usual mathematical rules apply to plus and minus signs. All other characters (or plus, minus, and space characters in other positions than leading) generate errors.

Note the mathematical rules:

- -	yields	+
- +	yields	-
+ -	yields	-
+ +	yields	+

The following arithmetic operators are allowed:

+	addition	ASCII 043 dec
-	subtraction	ASCII 045 dec
*	multiplication	ASCII 042 dec
/	division	ASCII 047 dec

Any other type of operators or other characters will generate an error.

The precision parameter optionally specifies the number of decimals in the result of the calculation. The result will be truncated accordingly. For example, if the number of decimals is specified as 5, the result 5.76123999 will be presented as 5.76123. The result of a FLOATCALC$ function can be formatted using a FORMAT$ function.

Examples

Addition:

```
A$ = "234.9"
B$ = "1001"
PRINT FLOATCALC$ (A$,"+",B$,5)
```

yields:

```
1235.90000
```

Subtraction:

```
A$ = "234.9"
C% = 2
PRINT FLOATCALC$ (A$,"-",100.013,C%)
```

yields:

```
134.88
```

FONT (FT)

Purpose

Statement for selecting a scaleable TrueType or TrueDoc single-byte font or a single-byte bitmap font for the printing of the subsequent PRTXT statements.

Syntax

FONT|FT<sexp$_1$>[,<nexp$_1$>[,<nexp$_2$>[,<nexp$_3$>]]]

<sexp$_1$>	is the name of the font. Default: "Swiss 721 BT".
<nexp$_1$>	is optionally the height in points of the font. Default: 12 points. Use MAG to enlarge with bitmap fonts.
<nexp$_2$>	is the clockwise slant in degrees (0–90°). Default: 0. Does not work with bitmap fonts.
<nexp$_3$>	is the width enlargement in percent relative the height (1-1000). Default: 100. Does not work with bitmap fonts.
Reset to default by:	PRINTFEED execution.

Remarks

Intermec Fingerprint supports scaleable fonts in TrueType and TrueDoc format that comply with the Unicode standard. A large number of scaleable fonts are available on special request, so it is quite possible that your printer is fitted with a unique selection of fonts. Use a FONTS statement to list the names of all fonts installed in your own printer to the standard OUT channel.

To maintain compatibility with programs created in earlier versions of Intermec Fingerprint, you can also specify bitmap font names, for example "SW030RSN" or "MS060BMN.2". In case of standard bitmap font name, the firmware will select the corresponding scaleable font in the printer's memory and set its parameters so its direction, appearance, and size come as close to the specified bitmap font as possible. A prerequisite is that the printer's memory contains the standard complement of outline fonts. Non-standard bitmap fonts can also be used. They will not produce any outline fonts, but will retain their bitmap format. Any extension to the bitmap font name is of no consequence. See Chapter 6, "Fonts" in this manual.

The height of the font is given in points (same as in your PC), which means that a text will be printed in the same size regardless of the printhead density of the printer. The unit of measure is points (1 point = 1/72 inch   0.352 mm) and specifies the height of the font including ascenders and descenders. Sizes less than 4 points will be unreadable. In case of bitmap fonts, it is recommended to use MAG to enlarge the font instead of specifying a font height (works only in multiples of 12 points).

Any font may be magnified up to 4 times separately in regard of height and width using a MAG statement. Bitmap fonts will get somewhat jagged edges when magnified, whereas outline fonts will remain smooth.

FONT (FT), cont.

Slanting means that you can create the same effect as in ITALIC characters.The higher value, the more askew the upright parts of the characters will come. Slanting increases clockwise. Values greater than 65-70° will be unreadable. Slanting cannot be used with bitmap fonts.

Slanting value: 10 *ABCDEFGH*

Slanting value: 20 *ABCDEFGH*

A scaleable font can enlarged in regard of width relative the height. The value is given as percent (1-1000). This means that if the value is 100, there is no change in the appearance of the characters, whereas if the value is given as, for example, 50 or 200, the width will the half the height or double the height respectively. When using this parameter, all parameters in the syntax must be included in the statement, that is, name, height, slant, and width.

The standard complement of fonts listed in Chapter 6 can be supplemented with more fonts using three methods:

Downloading fonts from a Font Install Card

The card must be inserted before the printer is started. At startup the fonts are automatically downloaded, installed, and permanently stored in the printer's memory. The fonts can be used without the card being present.

Using fonts from a Font Card
The card must be inserted before the printer is started. At startup the fonts are automatically installed, but not copied to the printer's memory. Thus, the card must always be present before such a font can be used.

Downloading font files
Font files can be downloaded and installed by means of either of the two statements IMAGE LOAD and TRANSFER KERMIT. There is no need to restart the printer before using the font in question.

It is possible to create aliases for one or several font to get shorter or more adequate names. Refer to Chapter 6 for further explanation.

Examples

Printing one line of 12p text with default direction and alignment:

```
10   FONT "Swiss 721 BT"
20   PRTXT "HELLO"
30   PRINTFEED
RUN
```

Printing the same text but with 24p size, 20° slant, and 75% width:

```
10   FONT "Swiss 721 BT",24,20,75
20   PRTXT "HELLO"
30   PRINTFEED
RUN
```

FONTD

Purpose

Statement for selecting a scaleable TrueType or TrueDoc double-byte font for the printing of the subsequent PRTXT statements.

Syntax

FONTD<sexp$_1$>[,<nexp$_1$>[,<nexp$_2$>[,<nexp$_3$>]]]

<sexp$_1$>	is the name of the font. Default: none.
<nexp$_1$>	is optionally the height in points of the font. Default: 12 points.
<nexp$_2$>	is the clockwise slant in degrees (0-90°). Default: 0.
<nexp$_3$>	is the width enlargement in percent relative the height (1-1000). Default: 100.
Reset to default by:	PRINTFEED execution or CLL.

Remarks

This statement is identical to the FONT statement, but is used for fonts specified by a double byte (16 bits) instead of a single byte (7 or 8 bits). To use a double-byte font, a double-byte character set must be selected using a NASCD statement. Usually, if the first byte has an ASCII value between 161 dec. (A1 hex) and 254 dec (FE hex), the character will be treated as a double-byte character and the firmware waits for next byte to make the 16 bit address complete. The character will be printed using the font specified by FONTD and according to the character set specified by NASCD and the Unicode standard.

On the other hand, if the first byte has an ASCII value below 161 dec. (A1 hex), the character is treated as a single byte character and next byte received will be regarded as the start of a new character. This implies that the character set specified by NASC and the font specified by FONT will be used. However, the selected Unicode double-byte character set may specify some other ASCII value as the breaking point between single and double byte character sets.

Note that 8 bit communication must be selected.

Only writing from left to right in the selected print direction is supported.

Example

The following text contains both single- and double-byte fonts. The double-byte font and its character set are stored in a Font Install Card:

```
10   NASC 46
20   FONT "Swiss 721 BT", 24, 10
30   FONTD "Chinese"
40   NASCD "rom:BIG5.NCD"
50   PRTXT CHR$(65);CHR$(161);CHR$(162)
60   PRINTFEED
RUN
```

This program yields a printed text line that starts with the Latin character A (ASCII 65 dec.) followed by the Chinese font that corresponds to the address 161+162 dec. in the character set “BIG5.NCD".

FONTNAME$

Purpose

Function returning the names of the fonts stored in the printer's memory.

Syntax

FONTNAME$(<nexp>)

<nexp>	the result of the expression should be either false or true, where... False (0) indicates first font. True (≠0) indicates next font.

Remarks

FONTNAME$(0) produces the first name in the memory.

FONTNAME$(0) produces next name. Can be repeated as long as there are any fontnames left.

Example

Use a program like this to list all fontnames:

```
10   A$ = FONTNAME$ (0)
20   IF A$ = "" THEN END
30   PRINT A$
40   A$ = FONTNAME$ (-1)
50   GOTO 20
RUN
```

yields for example:

```
-UPC11.1
-UPC11.2
-UPC21.1
-UPC21.2
-UPC31.1
-UPC31.2
-UPC51.1
-UPC51.2
Century Schoolbook BT
DingDings SWA
Dutch 801 Bold BT
Dutch 801 Roman BT
Futura Light BT
Letter Gothic 12 Pitch BT
MS030RMN
MS030RMN.1
MS030RMN.2
MS050RMN
MS050RMN.1
MS050RMN.2
MS060BMN
MS060BMN.1
MS060BMN.2
Monospace 821 BT
Monospace 821 Bold BT
OB035RM1
```

etc, etc.

FONTS

Purpose

Statement returning the names of all fonts stored in the printer's memory to the standard OUT channel.

Syntax

FONTS

Example

A list of the fonts stored in the printer may look like this:

```
FONTS
```

yields for example:

```
Century Schoolbook BT       DingDings SWA
Dutch 801 Bold BT           Dutch 801 Roman BT
Futura Light BT             Letter Gothic 12 Pitch BT
MS030RMN                    MS030RMN.1
MS030RMN.2                  MS050RMN
MS050RMN.1                  MS050RMN.2
MS060BMN                    MS060BMN.1
MS060BMN.2                  Monospace 821 BT
Monospace 821 Bold BT       OB035RM1
OB035RM1.1                  OB035RM1.2
OCR-A BT                    OCR-B 10 Pitch BT
Prestige 12 Pitch Bold BT   SW020BSN
SW020BSN.1                  SW020BSN.2
SW030RSN                    SW030RSN.1
SW030RSN.2                  SW050RSN
SW050RSN.1                  SW050RSN.2
SW060BSN                    SW060BSN.1
SW060BSN.2                  SW080BSN
SW080BSN.1                  SW080BSN.2
SW120BSN                    SW120BSN.1
SW120BSN.2                  Swiss 721 BT
Swiss 721 Bold BT           Swiss 721 Bold Condensed BT
Zurich Extra Condensed BT

3569264 bytes free   1717240 bytes used
Ok
```

FOR...TO...NEXT

Purpose

Statement for creating a loop in the program execution, where a counter is incremented or decremented until a specified value is reached.

Syntax

FOR<nvar>=<nexp$_1$>TO<nexp$_2$>[STEP<nexp$_3$>]NEXT[<nvar>]

<nvar>	is the variable to be used as a counter.
<nexp$_1$>	is the initial value of the counter.
<nexp$_2$>	is the final value of the counter.
<nexp$_3$>	is the value of the increment (decrement).

Remarks

This statement is always used in connection with a NEXT statement.

The counter (<nvar>) is given an initial value by the numeric expression (<nexp$_1$>). If no increment value is given (STEP <nexp$_3$>), the value 1 is assumed. A negative increment value will produce a decremental loop. Each time the statement NEXT is encountered, the loop will be executed again until the final value, specified by (<nexp$_2$>), is reached. Then the execution will proceed from the first line after the NEXT statement.

If the optional variable is omitted in the NEXT statement, the program execution will loop back to the most recently encountered FOR statement. If the NEXT statement does include a variable, the execution will loop back to the FOR statement specified by the same variable.

FOR...NEXT loops can be nested, which means that a loop can contain another loop, etc. However, each loop must have a unique counter designation and the inside loop must be concluded by a NEXT statement before the outside loop can be executed.

Examples

The counter A% is incremented from 10 to 50 in steps of 20:

```
10    FOR A%=10 TO 50 STEP 20
20    PRINT A%
30    NEXT
RUN
```

yields:

```
10
30
50
```

The counter B% is decremented from 50 to 10 in steps of 20:

```
10    FOR A%=50 TO 10 STEP -20
20    PRINT A%
30    NEXT
RUN
```

yields:

```
50
30
10
```

FORMAT

Purpose

Statement for formatting the printer's permanent memory, or formatting a CompactFlash memory card.

Syntax

FORMAT<sexp>[,<nexp$_1$>[,<nexp$_2$>]][,A]

<sexp>	specifies the device to be formatted either as "/c" or "card1:"
<nexp$_1$>	Specifies the number of entries in the root directory (only applicable when <sexp> = "card1:" and "A" flag is set). Default: 208 entries.
<nexp$_2$>	Specifies the number of bytes per sector (only applicable when <sexp> = "card1:" and "A" flag is set). Default: 512 bytes per sector.

Remarks

FORMAT "/c"
Formats the printers permanent memory partially or completely.
System files are distinguished by a leading period character, for example .setup.saved. This makes it possible to format the permanent memory without removing the system files.

If no "A" flag is included in the statement, all files excluding those starting with a period character (.) will be removed ("soft" formatting).

If an "A" flag is included in the statement, all files including those starting with a period character (.) will be removed ("hard" formatting).

Be careful. There is no way to undo a FORMAT operation.

FORMAT "card1:"
Formats a CompactFlash card, which is inserted in the printer's optional memory card adapter, to MS-DOS format. Optionally, the number of entries in the root directory (that is number of files on the card) and the number of bytes per sector can be specified, provided an "A" flag is included in the statement ("hard" formatting).

When a FORMAT statement is executed, any existing data or previous formatting in the card will be erased. After formatting, such a memory card can be OPENed for INPUT/OUTPUT/APPEND or RANDOM access and can also be used in a PC for storing MS-DOS files. The DOS-formatted memory card is referred to as device "card1:".

FORMAT, cont.

Examples

Issuing the statement FILES before and after a FORMAT "/c" statement shows how the memory is affected. Note that system files starting with a period character are not removed, since the FORMAT statement does not contain any “A” flag:

```
FILES "/c",A
```

yields for example:

```
Files on /c

./                     0    ../                  0
APPLICATION            1    boot/                0
ADMIN/                 0    .setup.saved       222
STDIO                  4
2222080 bytes free          227 bytes used

Ok
FORMAT "/c"

Ok

FILES "/c",A
```

yields for example:

```
Files on /c

./                     0    ../                  0
boot/                  0    ADMIN/               0
.setup.saved         222
2224128 bytes free          222 bytes used
```

In the following statement, a CompactFlash memory card is formatted to MS-DOS format in the immediate mode. The number of entries is increased from 208 (default) to 500 and the size of the sectors in decreased from 512 bps (default) to 256 in order to make the card better suited for more but smaller files. The “A” flag specifies “hard" formatting.

```
FORMAT "card1:",500,256,A
```

FORMAT DATE$

Purpose

Statement for specifying the format of the string returned by DATE$("F") and DATEADD$(..... ,"F") instructions.

Syntax

FORMAT DATE$<sexp>

<sexp>	is a string representing the order between year, month and date plus possible separating characters. "Y" represents Year (one digit per Y). "M" represents Month (one digit per M). "D" represents Day (one digit per D).
Default:	YYMMDD
Reset to default by:	Empty string ("")

Remarks

DATE$ and DATEADD$ will only return formatted dates if these functionss include the flag "F".

In the FORMAT DATE$ statement, each Y, M or D character generates one digit from the number of the year, month or day respectively, starting from the end. If the number of Y's exceeds 4, or the number of M's or D's exceeds 2, the exceeding characters generate leading space characters.

Examples (the year is 2003):

Y	generates	3	
YY	generates	03	
YYY	generates	003	
YYYY	generates	2003	
YYYYY	generates	$_{\leftrightarrow}$2003	($_{\leftrightarrow}$ represents a space)

Separating characters are returned as entered in the string. Any character except Y, M, or D are regarded as separators.

The date format is saved in the temporary memory and has to be transmitted to the printer after each power-up.

Examples

Changing the date format according to British standard:
```
FORMAT DATE$ "DD/MM/YY"
```

Changing date format back to default (YYMMDD):
```
FORMAT DATE$ ""
```

Changing the date format to Swedish standard:
```
FORMAT DATE$ "YY-MM-DD"
```

FORMAT INPUT

Purpose

Statement for specifying separators for the LAYOUT RUN statement used in the Intermec Direct Protocol.

Syntax

FORMAT INPUT<sexp$_1$>[,<sexp$_2$>[,<sexp$_3$>[,<sexp$_4$>]]]

<sexp$_1$>	is the start -of-text separator, default STX (ASCII 02 dec.).
<sexp$_2$>	is the end-of-text separator, default EOT (ASCII 04 dec.).
<sexp$_3$>	is the field separator, default CR (ASCII 13 dec.).
<sexp$_4$>	is a string of characters to be filtered out.

Remarks

The LAYOUT RUN statement is used in the Intermec Direct Protocol to transmit variable data to a predefined layout. By default, the string of input data to the various layout fields starts with a STX character and ends with a EOT character. The various fields are separated by CR (carriage return) characters.

To provide full compatibility with various protocols and computer systems, these separators can be changed at will by means of the FORMAT INPUT statement. Each separator can have a maximum length of 10 characters.

As an option, it is possible to specify a string of max. 10 characters to be filtered out. By default, the string is empty and will be reset to default if a new FORMAT INPUT with less than four arguments is issued.

There is a timeout if ETX is not found within 60 seconds after STX has been received.

Always execute the FORMAT INPUT statement in the Immediate Mode. If you are using the Intermec Direct Protocol, exit it using an INPUT OFF statement before changing the separators using a FORMAT INPUT statement. Then you can enter the Intermec Direct Protocol again using an INPUT ON statement.

An error will occur if you, for some reason, issue a FORMAT INPUT statement where one, two or three separators are identical to those already in effect without leaving the Intermec Direct Protocol.

If a certain separating character cannot be produced by the keyboard of the host, use a CHR$ function to specify the character by its ASCII value.

The separators are stored in the temporary memory and must to be transmitted to the printer after each power-up.

Example

Changing the start-of-text separator to #, the end-of-text separator to LF (linefeed), and the field separator to @ after having temporarily switched to the Immediate Mode.

```
INPUT OFF ↵
FORMAT INPUT "#",CHR(10),"@" ↵
INPUT ON ↵
```

FORMAT TIME$

Purpose

Statement for specifying the format of the string returned by TIME$("F") and TIMEADD$("F") instructions.

Syntax

FORMAT TIME$<sexp>

<sexp>	is a string representing the order between hours, minutes and seconds plus possible separating characters. "H" represents hours in a 24 hour cycle (one digit per H). "h" represents hours in a 12 hour cycle (one digit per h). "M" represents minutes (one digit per M). "S" represents seconds (one digit per S). "P" represents AM/PM in connection with a 12 hour cycle. "p" represents am/pm in connection with a 12 hour cycle. All other character produce separator characters.
Default:	HHMMSS
Reset to default by:	Empty string

Remarks

Each H, h, M, and S character generates one digit. If the number of each character exceeds 2, leading space characters are inserted. Each uppercase or lowercase P character generates one character of AM/PM or am/pm respectively, when a 12-hour cycle is selected.

Hour, minute and second fields are right-justified, whereas am/pm and AM/PM fields are left-justified.

Example (the hour is 8 o'clock in the morning):

h	generates	8	P	generates	A
hh	generates	08	PP	generates	AM
hhh	generates	$_{\leftrightarrow}$08	p	generates	a
			pp	generates	am

To get 12-hour cycle, all hour format characters must be lowercase "h".

Separating characters are returned as entered in the string. Any character but H, h, M, S, P, or p are regarded as separators.

The time format is saved in the temporary memory and has to be transmitted to the printer after each power-up.

Examples

Changing the time format according to Swedish standard:

```
FORMAT TIME$ "HH.MM.SS"
```

Changing the date format to British standard:

```
FORMAT TIME$ "hh:MM pp"
```

FORMAT$

Purpose

Function for formatting a number represented by a string.

Syntax

FORMAT$(<sexp$_1$>,<sexp$_2$>)

<sexp$_1$>	is the string of numerals, optionally with decimals, which is to be formatted.
<sexp$_2$>	specifies the format of the string.

Remarks

The original string (<sexp$_1$>) is a string of digits, optionally with a decimal point to separate decimals from integers. It can also contain leading plus (+), minus (-), and space characters. Space characters are ignored, whereas the usual mathematical rules apply to plus and minus signs. All other characters (or plus, minus, and space characters in other positions than leading) generate errors.

Note the mathematical rules:

- -	yields	+
- +	yields	-
+ -	yields	-
+ +	yields	+

The format is specified by a string (<sexp$_2$>). Different format will give different result. The string can contain any characters, but some have special meanings. Note the explanation of the following characters.

`0` = **Digit place holder, display a digit or zero.**

If the input number has fewer digits than there are zeros (on either side of the decimal separator) in the format string, leading or trailing zeros are displayed. If the number has more digits to the left side of the decimal separator than there are zeros to the left side of the separator in the format string the digits will be displayed. If the number has more digits to the right of the separator than there are zeros to the right of the decimal separator in the format string, the decimal will be truncated to as many decimal places as there are zeros.

# = **Digit placeholder, display a digit or nothing.**
If there is a digit in the expression being formatted in the position where the # appears in the format string, display the digit or otherwise display nothing in that position. If the number has more digits to the left side of the decimal separator than there are # to the left side of the separator in the format string the digits will be displayed.

`.` = Decimal separator, to separate the integer and the decimal digits.
`,` = Decimal separator, to separate the integer and the decimal digits.
\ = Display the next character in the format string.

The backslash itself is not displayed. To display a \, use two backslashes. The only character, which will be displayed in the formatted string without a backslash is `space`.

FORMAT$, cont.

`space` = **Space**
A space will be displayed as literal character wherever it is in the expression format.

- An empty format string is equivalent to `"0.##########"`.
- `0` and `#` cannot be mixed in every way. Before the decimal separator, use `#` first and then `0`. After the decimal separator, use `0` first and then `#`. For example: `####00.000###` is OK and `#00##0.##0#00` is not OK.
- A point or a comma separates integers and decimals. The decimal separator used in the format is the one that will be the returned separator type. Independent of the separator type in the number the format type will control the return type. Default type is a point.
- A format can consist of separators as space between thousands either a unit as `$`. For example: `"$ ### ### 000.00"`.
- The attached number string will be truncated to the quantity of decimal in the format.
- Characters will not be displayed on the left side of the decimal separator if there is a `#` on the left side of the characters and the string to be formatted do not have a digit in the same position as the `#`. On the right side of the decimal separator, characters will not be displayed if there is a `#` on the right side of the characters and the string to be formatted do not have a digit in the same position as the `#`. For example:

Format string:	"\$#\t\e\x\t0.0\t\e\x\t#\$"			
String to be formatted:	1.1	55	0.33	55.33
Returned strings:	1.1	$5text5.0$	$0.3text3$	$5text5.3text3$

Input number:		"5"	"-5"	"0.5"	"55555"	"0.666666666666"
Input format:		Returned number:				
""	=>	5	-5	0.5	55555	0.6666666666
"0"	=>	5	-5	0	55555	0
"0.00"	=>	5.00	-5.00	0.50	55555.00	0.66
"\$0,0"	=>	$5,0	$-5,0	$0.5	$55555,0	$0,6
"0.0##"	=>	5.0	-5.0	0.5	55555.0	0.666
"###\,000.0"	=>	005.0	-005.0	000.5	55,555.0	000.6
"# 0 0.0"	=>	0 5.0	-0 5.0	0 0.5	555 5 5.0	0 0.6

Examples

The examples on the next page show how FLOATCALC$ and FORMAT$ functions can be combined.

FORMAT$, cont.

Addition

```
B$="234.9"
C$="1001"
D$="# ##0.##"
A$=FLOATCALC$(B$,"+",C$,15)
PRINT A$
```

yields:

```
"1235.900000000000000"

PRINT FORMAT$(A$,D$)
```

yields:

```
"1 235.9"
```

Subtraction

```
A$=FLOATCALC$("234.90","-","100.013",2)
PRINT A$
```

yields:

```
"134.88"

PRINT FORMAT$(A$,"\$ 0,000#")
```

yields:

```
"$ 134,880"
```

Note: If a higher precision is used in FLOATCALC$, A$ will yield "$134,887".

Multiplication

```
B$="3"
A$=FLOATCALC$("100", "*", B$, 1)
PRINT A$
```

yields:

```
"300.0"

C$="0 0 0,00###"
PRINT FORMAT$(A$,C$)
```

yields:

```
"3 0 0,00"
```

Division

```
B$="1.0"
A$=FLOATCALC$(B$,"/","3.0")
PRINT A$
```

yields:

```
"0.3333333333"

PRINT FORMAT$(A$,"\$ 000.00###")
```

yields:

```
"$ 000.33333"
```

FORMFEED (FF)

Purpose

Statement for feeding out or pulling back a certain length of media.

Syntax

FORMFEED|FF[<nexp>]

<nexp>	is, optionally, the feed length expressed as a positive or negative number of dots.

Remarks

If no value is entered after the FORMFEED statement, the printer will feed out one single label, ticket, tag, or a portion of continuous stock according to the printer's setup. See start- and stopadjustments and media type in the User's Guide for the printer model in question.

If a value is entered after the FORMFEED statement, the media will be fed out or pulled back the corresponding number of dots:

- A positive number of dots makes the printer feed out the specified length of media.
- A negative number of dots makes the printer pull back the specified length of media. Be careful not to enter a value larger than the length of the label to avoid causing a media jam.

It is important whether a FORMFEED statement is issued before or after a PRINTFEED statement:
- FORMFEED statement issued before PRINTFEED affects the position of the origin on the first copy to be printed.
- FORMFEED statement issued after PRINTFEED does not affect the position of the origin on the first copy, but next copy will be affected.

Do not use FORMFEED as a replacement for start- and stopadjustments in the Setup Mode or in connection with batch printing.

Examples

Printing a line of text and feeding out an extra 60 dots of media after printing:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,200
30   PRTXT "HELLO"
40   PRINTFEED
50   FORMFEED 60
RUN
```

Pulling back the media 20 dots before printing:

```
10   FORMFEED -20
20   FONT "Swiss 721 BT"
30   PRPOS 30,200
40   PRTXT "HELLO"
50   PRINTFEED
RUN
```

In this case, the positioning of the text line will be performed after the media has been pulled back.

FRE

Purpose

Function returning the number of free bytes in spcified part of the printer's memory.

Syntax

FRE(<<sexp>|<nexp>>)

<sexp>	is the designation of the part of the printer's memory from which the number of free bytes should be returned, for example "/c", "tmp:", "card1:".
<nexp>	is a dummy argument. Returns the number of free bytes in the printer's temporary memory ("tmp:").

Remarks

The firmware looks for a colon (:) sign in the argument for the FRE function. If the argument is valid name of a memory device, the number of free bytes in that device is returned.

If the argument specifies device "card1:", but no card is inserted, Error 1039, "Not mounted" will occur.

If the name of a device, that is not a part of the printer's memory (for example "uart1:" or "console:"), is entered as an argument, the FRE function will return 0.

Refer to DEVICES for more information on memory and non-memory devices.

If the argument contains a colon, but is not a valid name of any device (for example "QWERTY:"), Error 1013, "Device not found" will occur.

Any argument, that does not include a colon sign (for example "7" or "QWERTY"), will return the amount of free bytes in the printer's temporary memory ("tmp:").

Example

```
PRINT FRE("tmp:")
```

yields for example:

```
2382384
```

```
PRINT FRE("uart1:")
```

yields:

```
0
```

```
PRINT FRE(1)
```

yields for example:

```
2382384
```

FUNCTEST

Purpose

Statement for performing various hardware tests.

Syntax

FUNCTEST<sexp>,<svar>

<sexp>	is the type of test to be performed:	
	"CARD"	
	"HEAD"	(only "HEAD" yields a meaningful response)
	"KERNEL"	
	"ROMn"	
<svar>	is the variable in which the result will be placed.	

Remarks

The test has a number of possible responses:

<sexp> = "CARD"

NOT IMPLEMENTED	Not supported.

<sexp> = "HEAD"

HEAD OK, SIZE:n DOTS	The test was successful. n is the number of dots on the printhead.
HEAD LIFTED	Printhead is lifted and must be lowered before test can be performed.
FAULTY PRINTHEAD	One or more dots on the printhead are not working. Note that the voltage for the printhead is not checked. Use the HEAD function for additional printhead tests.

<sexp> = "KERNEL"

NOT IMPLEMENTED	Not supported.

<sexp> = "ROMn"

NOT APPLICABLE	Not supported

Any other input to <sexp> yields an empty string.

Example

This example shows how a test program using the FUNCTEST statement may be composed:

```
10   FUNCTEST "HEAD", A$
20   PRINT "HEADTEST:", A$
RUN
```

yields for example:

```
HEADTEST: HEAD OK,SIZE:832 DOTS

Ok
```

FUNCTEST$

Purpose Function returning the result of various hardware tests.

Syntax **FUNCTEST$(<sexp>)**

<sexp>	is the type of test to be performed:	
	"CARD"	
	"HEAD"	(only "HEAD" yields a meaningful response)
	"KERNEL"	
	"ROMn"	

Remarks The test has a number of possible responses:

<sexp> = "CARD"

NOT IMPLEMENTED	Not supported.

<sexp> = "HEAD"

HEAD OK, SIZE:n DOTS	The test was successful. n is the number of dots on the printhead.
HEAD LIFTED	Printhead is lifted and must be lowered before test can be performed.
FAULTY PRINTHEAD	One or more dots on the printhead are not working. Note that the voltage for the printhead is not checked. Use the HEAD function for additional printhead tests.

<sexp> = "KERNEL"

NOT IMPLEMENTED	Not supported.

<sexp> = "ROMn"

NOT APPLICABLE	Not supported

Any other input to <sexp> yields an empty string.

Example This example shows how a test program using the FUNCTEST$ function may be composed (compare with the example for FUNCTEST statement):

```
PRINT "HEADTEST:", FUNCTEST$ ("HEAD")
```

yields for example:

```
HEADTEST: HEAD OK,SIZE:1280 DOTS

Ok
```

GET

Purpose

Statement for reading a record from a random file to a random buffer.

Syntax

GET[#]<nexp$_1$>,<nexp$_2$>

#	indicates that whatever follows is a number. Optional.
<nexp$_1$>	is the number assigned to the file when it was OPENed.
<nexp$_2$>	is the number of the record. Must be $\neq 0$.

Remarks

The GET statement is used to read a certain record in a certain random file to a buffer, where the record will be assigned to variables according to the FIELD statement given for the buffer. After the GET statement has been executed, you can use references to the variables defined by the FIELD statement to read the characters in the random buffer.

Numeric expressions, which have been converted to string expressions by STR$ functions before being put into the buffer, can be converted back to numeric expressions using VAL functions.

Example

```
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    SNAME$="SMITH"
40    CNAME$="JOHN"
50    PHONE$="12345630"
60    LSET F1$=SNAME$
70    LSET F2$=CNAME$
80    RSET F3$=PHONE$
90    PUT #8,1
100   CLOSE#8
RUN

SAVE "PROGRAM 1.PRG "

NEW
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    GET #8,1
40    PRINT F1$,F2$,F3$
RUN
```

yields:

```
SMITH – – – JOHN – – – – – –  12345630
```

GETASSOC$

Purpose

Function for getting a value from a string association.

Syntax

GETASSOC$ (<sexp$_1$>, <sexp$_2$>)

<sexp$_1$>	is the name of the association (case-sensitive).
<sexp$_2$>	is the name of a tuple in the association.

Remarks

An association is an array of tuples, where each tuple consists of a name and a value.

Example

This example shows how a string, including three stringnames associated with three start values, will be defined and one of them (time) will be changed:

```
10   QUERYSTRING$=
     "time=UNKNOWN&label=321&desc=DEF"
20   MAKEASSOC"QARRAY",QUERYSTRING$,"HTTP"
30   QTIME$=GETASSOC$("QARRAY","time")
40   QLABELS%=VAL(GETASSOC$("QARRAY","label"))
50   QDESC$=GETASSOC$("QARRAY","desc")
60   PRINT"time=";QTIME$,"LABEL=";QLABELS%,
"DESCRIPTION=";QDESC$
70   SETASSOC"QARRAY","time",time$
80   PRINT"time=";GETASSOC$("QARRAY","time")
RUN
```

yields:

```
time=UNKNOWN  LABEL=321 DESCRIP   TION=DEF
time=153355
```

GETASSOCNAME$

Purpose

Function for traversing the tuples of a string association.

Syntax

GETASSOCNAME$(<sexp>,<nexp>)

<sexp>	is the association to be traversed (case-sensitive).
<nexp>	specifies the tuple in the association. <nvar> = 0 specifies first tuple. <nvar> ≠ 0 specifies next tuple.

Remarks

An association is an array of tuples, where each tuple consists of a name and a value. To get the first position in the string association, <nvar> should be zero. Consecutive calls to GETASSOCNAME$ witn <nvar> non zero will traverse all variables in an undefined order. When a blank string ("") is returned, the last variable has been traversed.

Example

This example shows how "QARRAY" is traversed (run example from GETASSOC first):

```
10 LVAL$=GETASSOCNAME$("QARRAY",0)
20 WHILE LVAL$<>""
30 RVAL$=GETASSOC$("QARRAY",LVAL$)
40 PRINT LVAL$;"=";RVAL$
50 LVAL$=GETASSOCNAME$("QARRAY",1)
60 WEND
RUN
```

yields:

```
label=321
desc=DEF
time=153355
```

GETPFSVAR

Purpose

Function for recovering saved variables.

Syntax

GETPFSVAR(<sexp>[,D])

<sexp>	is the name of the variable (uppercase characters only).
D	optionally specifies that the variable is to be deleted after recovery.

Remarks

This function is used to recover variables registered to be saved at power failure by means of a SETPFSVAR statement and returns either -1 on success or 0 at failure.

If a D flag is included, the variable is deleted after it has been recovered. This can be used to make sure that the variable is up to date and that no old obsolete value is recovered.

Related instructions are SETPFSVAR, DELETEPFSVAR, and LIST-PFSVAR.

Example

```
10    IF NOT GETPFSVAR("QS$") THEN QS$ ="<this is
      the default vaule, set a new one>"
20    IF NOT GETPFSVAR("QCPS%") THEN PRINT "No
      copies available":END
30    QSTATUS%=GETPFSVAR("AWE$",D):IF QSTATUS%
      THEN PRINT "Recovered successfully!"
40    SETPFSVAR "QCPS%"
50    'Build label
60    .....
99    PRINTFEED; QCPS%=QCPS%
100   .....
```

GOSUB

Purpose Statement for branching to a subroutine.

Syntax **GOSUB<ncon>|<line label>**

<ncon>|<line label>	is the number or label of the first line in the desired subroutine.

Remarks After branching, the subroutine will be executed line by line until a RETURN statement is encountered.

The same subroutine can be branched to many times from different lines in the main program. GOSUB always remembers where the branching took place, which makes it possible to return to the correct line in the main program after the subroutine has been executed.

Subroutines may be nested, which means that a subroutine may contain a GOSUB statement for branching to a secondary subroutine and so on.

Subroutines are normally placed on program lines with higher numbers than the main program. The main program should be appended by an END statement to avoid unintentional execution of subroutines.

Example This example makes use of line numbers:

```
10   PRINT "This is the main program"
20   GOSUB 1000
30   PRINT "You’re back in the main program"
40   END
1000 PRINT "This is subroutine 1"
1010 GOSUB 2000
1020 PRINT "You’re back from subroutine 2 to 1"
1030 RETURN
2000 PRINT "This is subroutine 2"
2010 GOSUB 3000
2020 PRINT "You’re back from subroutine 3 to 2"
2030 RETURN
3000 PRINT "This is subroutine 3"
3010 PRINT "You’re leaving subroutine 3"
3020 RETURN
RUN
```

yields:

```
This is the main program
This is subroutine 1
This is subroutine 2
This is subroutine 3
You’re leaving subroutine 3
You’re back from subroutine 3 to 2
You’re back from subroutine 2 to 1
You’re back in the main program

Ok
```

GOSUB, cont.

In this examples, line numbers have been omitted and line labels are used to make the program branch to subroutines:

```
IMMEDIATE OFF
PRINT "This is the main program"
GOSUB SUB1
PRINT "You're back in the main program"
END
SUB1: PRINT "This is subroutine 1"
GOSUB SUB2
PRINT "You're back from subroutine 2 to 1"
RETURN
SUB2: PRINT "This is subroutine 2"
GOSUB SUB3
PRINT "You're back from subroutine 3 to 2"
RETURN
SUB3: PRINT "This is subroutine 3"
PRINT "You're leaving subroutine 3"
RETURN
```

GOTO

Purpose Statement for branching unconditionally to a specified line.

Syntax **GOTO<ncon>|<line label>**

<ncon>/<line label>	is the number or label of the line to be branched to.

Remarks If the specified line contains an executable statement, both that statement and all that follows will be executed. If the specified line does not exist, an error condition will occur.

The GOTO statement can also be used in the immediate mode to resume execution of a program, which has been terminated using a STOP statement, at a specified program line.

Example In this example the first bar of the tune "Colonel Boogie" will be played only if the title is entered correctly. Otherwise the message "Try again" will be displayed until you manage to type the right name.

```
10    A$="COLONEL BOOGIE"
20    B$="TRY AGAIN"
30    INPUT "TITLE"; C$
40    IF C$=A$ GOTO 100 ELSE PRINT B$
50    GOTO 30
60    END
100   SOUND 392,15
110   SOUND 330,20
120   SOUND 330,15
130   SOUND 349,15
140   SOUND 392,15
150   SOUND 659,25
160   SOUND 659,20
170   SOUND 523,25
180   GOTO 60
RUN
```

yields:

```
TITLE?
```

Note the way GOTO is used in line 50 to create a loop, which makes the printer await the condition specified in line 40 before the execution is resumed. Instead of line numbers, line labels can be used following the same principles as illustrated in the second example for GOSUB statement.

HEAD

Purpose

Function returning the result of a thermal printhead check.

Syntax

HEAD(<nexp$_1$>)

<nexp$_1$> ≥ 0	specifies the number of a dot for which the resistance in ohms will be returned.	
<nexp$_1$> = -1	printhead check:	Returns -1 (true) if OK Returns 0 (false) if error
<nexp$_1$> = -7	returns mean printhead resistance in ohms.	

<nexp$_2$> = HEAD(<sexp>)

<nexp$_2$>	returns the number (quantity) of faulty dots.
<sexp>	returns the dot number and resistance for each faulty dot.

Remarks

This function allows you to examine the printhead in regard of dot resistance. There is no guarantee that all defect “dots" will detected by the HEAD function, since only the resistance is checked. For example, dirty or cracked dots can only be detected visually.

The detection of a possibly faulty dot or printhead by means of the dot sensing facility does not automatically imply that the printhead is defect and that replacement will be covered by the warranty. Intermec reserves the right of physical examination of the printhead before any replacement free of charge can be discussed.

<nexp1> 0
A positive value specifies a single dot on the printhead and returns its resistance value as a number of ohms. A dot resistance value that deviates considerably from the mean resistance value of the printhead (see below) indicates that the dot may be faulty. The dot numbering starts at 0 (zero), that is, in a 832 dots printhead, the dots are numbered 0-831.

<nexp1> = -1
A check of the complete printhead is performed.

HEAD(-1)=-1	The printhead is within the allowed limits (no dot is more than ±15% from the mean resistance value). This does not guarantee the printout quality.
HEAD(-1)=0	A possible error has been detected.

<nexp1> = -7
The mean resistance value in ohms of all dots of the printhead is returned.

The second version of the HEAD function measures the dot resistance for every dot in the printhead and faulty dots are reported to the system, so you do not need to use a SET FAULTY DOT statement to report bad dots one at the time.

HEAD, cont.

Examples

Read the resistance value of dot No. 5:

```
PRINT HEAD(5)
```

Perform a printhead check:

```
PRINT HEAD(-1)
```

Read the printhead's mean resistance value:

```
PRINT HEAD(-7)
```

Check printhead for faulty dots and their respective resistance values:

```
A%=HEAD(B$)
```

yields for example:

```
Ok
PRINT A%
5

Ok
PRINT B$
25, 2944
42, 2944
106, 2944
107, 2944
140, 2944

Ok
```

IF...THEN...(ELSE)

Purpose

Statement for conditional execution controlled by the result of a numeric expression.

Syntax

IF<nexp>[,]THEN<stmt$_1$>[ELSE<stmt$_2$>]

```
IF<nexp>[,]THEN  ↵
<stmt1>          ↵
[...<stmt1+n>]   ↵
[ELSE            ↵
<stmt2>          ↵
[...<stmt2+n>]]  ↵
ENDIF            ↵
```

<nexp>	is a numeric expression, which is either true or false.
<stmt$_1$>	is the statement or list of statements telling the program what to do, should the IF-condition be true.
<stmt$_2$>	is an optional statement or list of statements specifying what will happen, should the IF-condition be false.

Remarks

THEN and ELSE statements may be nested.

Multiple THEN and ELSE statements can alternatively be entered on separate lines. If so, the instruction should be appended by ENDIF. See second example below.

Examples

These two examples illustrates the different syntaxes:

```
10   A%=100:B%=20
20   C$="A LARGER THAN B"
30   D$="A NOT LARGER THAN B"
40   IF A%>B% THEN PRINT C$ ELSE PRINT D$
RUN
```

yields:

```
A LARGER THAN B
```

```
10   A%=VAL(TIME$)
20   IF A%>120000 THEN
30   PRINT "TIME IS ";TIME$; ". ";
40   PRINT "GO TO LUNCH!"
50   ELSE
60   PRINT "CARRY ON - ";
70   PRINT "THERE'S MORE WORK TO DO!"
80   ENDIF
RUN
```

yields for example:

```
TIME IS 121500. GO TO LUNCH!
```

IF...THEN...(ELSE), cont.

IF ... THEN are often used in connection with GOTO. In this example, line numbering is used. Also see the example for the GOTO statement.

```
10   A%=100
20   B%=50
30   IF A%=B% THEN GOTO 50 ELSE PRINT "NOT
EQUAL"
40   END
50   PRINT "EQUAL":END
RUN
```

yields:

```
NOT EQUAL
```

This example correspond to the preceding example, but line labels are used instead of line numbers.

```
IMMEDIATE OFF
A%=100
B%=50
IF A%=B% THEN GOTO QQQ ELSE PRINT "NOT EQUAL"
END
QQQ: PRINT "EQUAL":END
IMMEDIATE ON
RUN
```

yields:

```
NOT EQUAL
```

IMAGE BUFFER MIRROR

Purpose

Statement for mirror the print image around the Y-axis.

Syntax

IMAGE BUFFER MIRROR

Remarks

This statement mirrors the current defined image buffer around the Y-axis, that is, the feed direction. Fields defined after the IMAGE BUFFER MIRROR statement is executed are rendered normally. The image buffer width is always 8-bit aligned, even when the X-start parameter in the setup is not. Thus, it is recommended to test that the mirrored image is printed sidewise where intended. In some cases, a small correction using the PRPOS statement or the X-start parameter could be necessary.

Example

```
NEW
10 PRPOS 50,300
20 FONT "Swiss 721 BT",40
30 PRTXT "MIRROR"
40 IMAGE BUFFER MIRROR
50 PRPOS 50,100
60 PRTXT "NORMAL"
70 PRINTFEED
```

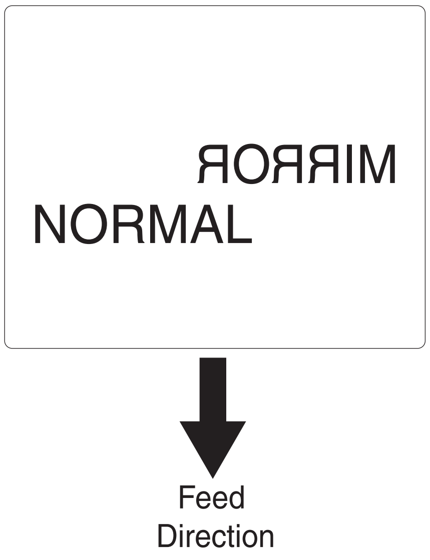


IMAGE BUFFER SAVE

Purpose

Statement for saving the content of the image buffer as a file.

Syntax

IMAGE BUFFER SAVE<sexp>

<sexp>	is the desired name of the file, optionally with a reference to the device where the file should be saved.

Remarks

This statement saves the current content of the print buffer as an image file in RLL format. After saving, the file is automatically installed as an image, that can be printed using a PRIMAGE statement in DIR 1 and DIR 3. Thereby, you can create label templates, to which variable data easily can be added at will.

The size of the print buffer image depends on the size of the print image at the moment the buffer is saved. The width is decided by the Media, Media Size, Width setup value with the first pixel according to the Media, Media Size, Xstart setup value. The height is decided by the actual height in y-dimension of the print image. Note that space characters or invisible "white" parts of an image are included in the height of the print image, even if they are not visible on the printed label.

Example

```
IMAGE BUFFER SAVE "TEMPLATE7"
```

IMAGE LOAD

Purpose

Statement for receiving, converting and installing image and font files.

Syntax

IMAGE LOAD[<nexp$_1$>,]<sexp$_1$>,<nexp$_2$>[,<sexp$_2$>[,<nexp$_3$>]]

<nexp$_1$>	is optionally the number of bytes to skip before starting to read the data.
<sexp$_1$>	is the desired name of the image or font to be created.
<nexp$_2$>	is the size of the original file in number of bytes.
<sexp$_2$>	is an optional flag: "S" specifies that the image or font will be saved in the printer's permanent memory ("/c"). Avoid this option (slow). An empty string ("") specifies that the image or font will be stored in the printer's temporary memory ("tmp:").
<nexp$_3$>	optionally specifies a communication channel OPENed for INPUT by the number assigned to the device. (Default: Std IN channel.)

Remarks

This statement prepares the printer to receive a .PCX image file, an image file in the internal Intermec Fingerprint bitmap format, or a font file on the standard IN channel (see SETSTDIO statement) or on another communication channel OPEN for INPUT. When the file is received, it will automatically be converted to an image in the internal bitmap format of Intermec Fingerprint or to a scaleable font respectively.

The optional first parameter makes it possible to use this statement in MS-DOS (CR/LF problem).

The name of an image may consist of max. 30 characters including possible extension. The image will have the same direction as the original image file and can only be rotated 180° using a DIR statement. We therefore recommend that you include the extension .1 or .2 to indicate for which print directions the image is intended. Font file names are only restricted to 30 characters. The size of the original file should be given in bytes according to its size in the host.

Before IMAGE LOAD can be used on a serial channel, the setup must be changed to 8 characters, CTS/RTS handshake. When an IMAGE LOAD statement is executed, the execution stops and waits for the number of bytes specified in the statement to be received. During the transfer of image file data to the printer, there is a 25 seconds timeout between characters. If a new character has not been received within the timeout limit, Error 80 "Download timeout" occurs. When the specified number of characters have been received, the execution is resumed.

If the downloading was successful, the downloaded image or font will be installed automatically and can be used without any rebooting.

Example

```
IMAGE LOAD "Logotype.1",400,""
```

IMAGENAME$

Purpose

Function returning the names of the images stored in the printer's memory.

Syntax

IMAGENAME$(<nexp>)

<nexp>	is the result of the expression which is either false or true: False (0) indicates first image. True (≠0) indicates next image.

Remarks

This function can be used to produce a list of all images (another method is to use the IMAGES statement).

Image files downloaded by means of a TRANSFER KERMIT statement will not be returned, since the software will regard them as files rather than images.

IMAGENAME$(0) produces the first name in the memory.

IMAGENAME$(0) produces next name. Can be repeated as long there are any image names left.

Example

Use a program like this to list all image names:

```
10   A$=IMAGENAME$(0)
20   IF A$=""THEN END
30   PRINT A$
40   A$=IMAGENAME$(-1)
50   GOTO 20
RUN
```

yields for example:

```
CHESS2X2.1
CHESS4X4.1
DIAMONDS.1
GLOBE.1

Ok
```

IMAGES

Purpose

Statement for returning the names of all images stored in the printer's memory to the standard OUT channel.

Syntax

IMAGES

Remarks

This statement can be used to list all image names (another method is to use an IMAGENAME$ function).

Image files downloaded by means of a TRANSFER KERMIT statement will not be printed, since the firmware will regard them as files rather than images.

Example

A list of images stored in the printer's memory may look like this:

```
IMAGES
```

yields for example:

```
CHESS2X2.1                                 CHESS4X4.1
DIAMONDS.1                                 GLOBE.1

3568692 bytes free   1717812 bytes used
Ok
```

IMMEDIATE

Purpose

Statement for enabling or disabling the immediate mode of Intermec Fingerprint in connection with program editing without line numbers, for reading the current mode, or for reading the current standard IN and OUT channels.

Syntax

IMMEDIATE ON|OFF|MODE|STDIO

ON	Enables the Immediate Mode
OFF	Disables the Immediate Mode
MODE	Prints a line to the STDOUT port with information on the current status of the following modes (ON or OFF): - Execution - Immediate - Input - Layout Input - Debug STDIO (dbstdio)
STDIO	Prints two lines to the STDOUT port with information on current settings for the STDIN and STDOUT channels.

Remarks

IMMEDIATE ON|OFF

Before starting to write a program without line numbers, the immediate mode must be disabled by means of an IMMEDIATE OFF statement. If not, each line will be executed immediately.

After an IMMEDIATE OFF statement, program lines can be entered without any leading line numbers. References to lines are done using "line labels", which are called in GOTO or GOSUB and related statements.

A line label is a name followed by a colon (:). The label must not interfere with any keywords or start with a digit and the line must start with the line label. When a line label is used as a reference to another line, for example within a GOTO statement, the colon should be omitted.

The program should be appended by a IMMEDIATE ON statement. At the execution of this statement, the program lines will be numbered automatically in ten-step incremental order, starting with the first line (10-20-30-40-50...). These line numbers will not appear on the screen until the program is LISTed, LOADed, or MERGEd. Line labels will not be converted to line numbers.

Do not issue a RUN statement before the IMMEDIATE ON statement, or an error will occur.

IMMEDIATE, cont.

IMMEDIATE MODE
Execution On/Off indicates if a Fingerprint program is running or not.

Immediate On/Off indicates whether the Immediate Mode is enabled or disabled as specified by IMMEDIATE ON/OFF.

Input On/Off indicates whether the Direct Protocol is enabled or disabled as specified by INPUT ON/OFF.

Layout Input On/Off indicates whether or not a layout is being recorded in the Direct Protocol as specified by LAYOUT INPUT and LAYOUT END.

Dbstdio On/Off indicates whether the debug standard I/O is active or not.

The following conditions are not reported:
- Running a Fingerprint application.
- Execution of a TRANSFER KERMIT, FILE& LOAD, IMAGE LOAD, LOAD, and STORE INPUT instruction.
- Running external commands (ush), for example RUN"rz......"
- Running the Setup Mode or execution of a SETUP statement.

IMMEDIATE STDIO
Two lines will be transmitted on the STDOUT port with information on the current STDIN and STDOUT channels regarding port, baud rate, character length, parity, and stop bits.

Examples

A program can be written without using any line numbers, as illustrated by this short example. QQQ is used as a line label:

```
IMMEDIATE OFF
```

yields:

```
Ok
PRINT "LINE 1"
GOSUB QQQ
END
QQQ: PRINT "LINE 2"
RETURN
IMMEDIATE ON
Ok
RUN
```

yields:

```
LINE 1
LINE 2
Ok
```

IMMEDIATE, cont.

This example shows how the status of the various modes are checked:

```
IMMEDIATE MODE
```

yields for example:

```
execution=OFF, immediate=ON, input=OFF, layout
input = Off
```

This example shows how the status of the STDIN and STDOUT channels are checked:

```
IMMEDIATE STDIO
```

yields for example:

```
stdin=uart1:, 9600, 8, NONE, 1
stdout=uart1:, 9600, 8, NONE, 1
```

INKEY$

Purpose

Function reading the first character in the receive buffer of the standard IN channel.

Syntax

INKEY$

Remarks

For information on standard I/O channels, see SETSTDIO statement. By default, "uart1:" is the standard I/O channel.

As opposed to the INPUT statement, INKEY$ does not interrupt the program flow to wait for input data, unless a loop is created by means of a GOTO statement, see line 20 in the example below.

INKEY$ is useful when the host computer is unable to end the input data with a “Carriage Return” (CR; ASCII 13 dec.), but must use some other character, for example “End of Text" (ETX; ASCII 3 dec.). Then a routine, which interprets the substitute character as a carriage return, can be created.

Example

In this example, none of the characters received on the standard IN channel will be printed on the host screen until a # character (ASCII 35 decimal) is encountered.

```
10   A$ = INKEY$
20   IF A$ = "" GOTO 10
30   IF A$ = CHR$(35) THEN PRINT B$
40   IF A$ = CHR$(35) THEN END
50   B$ = B$ + A$
60   GOTO 10
RUN
```

Type a number of characters on the keyboard of the host. They will not be printed on the host screen until you type a # character. Then all the characters will appear simultaneously, except for the #-sign.

Note the loop between line 10 and 20, which makes the program wait for you to activate a key.

INPUT (IP)

Purpose

Statement for receiving input data via the standard IN channel during the execution of a program.

Syntax

INPUT|IP[<scon><;|,>]<<nvar>|<svar>>[,<<nvar>|<svar>>...]

<scon><;|,>	is an optional prompt string, followed by a semicolon or comma.
<<nvar>|<svar>>	are variables to which the input data will be assigned.

Remarks

For information on standard I/O channel, see SETSTDIO statement. By default, "uart1:" is the standard I/O channel.

During the execution of a program, an INPUT statement will interrupt the execution. A question mark and/or a prompt will be displayed on the screen of the host to indicate that the program is expecting additional data to be entered. The prompt can be used to tell the operator what type of data he or she is expected to enter.

The prompt will be appended by a question mark if a semicolon (;) is entered after the prompt string. If a comma (,) is used in that position, the printing of the question mark will be suppressed.

If a prompt is not used, the question mark will always be displayed.

Do not enter any comma or semicolon directly after the keyword, only after the prompt, or in order to separate variables.

The input data should be assigned to one or several variables. Each item of data should be separated from next item by a comma. The number of data items entered must correspond to the number of variables in the list, or else an error condition will occur. The variables may be any mix of string variables and numeric variables, but the type of input data must agree with the type of the variable, to which the data is assigned.

Input can also be done directly to the system variables TIME$, DATE$, and SYSVAR.

The maximum number of characters that can be read using an INPUT statement is 32,767 characters.

Note that INPUT filters out any incoming ASCII 00 dec. characters (NUL).

INPUT does not support auto-hunting (see SETSTDIO).

INPUT (IP), cont.

Examples

This example shows input to one numeric variable and one string variable:

```
10   INPUT "ADDRESS";A%,B$
20   PRINT A%;" ";B$
30   IF A% > 0 THEN GOTO 50
40   GOTO 10
50   END
RUN
```

yields:

```
ADDRESS?
```

When the prompt “ADDRESS?” appears on the screen, you can type the input data on the terminal’s keyboard, for example:

```
999, HILL STREET
```

Note the separating comma.

If the input text data contains a comma, which shall be printed, you must enclose the input data with quotation marks ("...."), for example:

```
999, "HILL STREET, HILLSBOROUGH"
```

Numeric input data must not include any decimal points.

This example shows how the date can be set directly from the keyboard of the host:

```
INPUT "Enter date: ",DATE$
```

yields:

```
Enter date:
```

When the prompt “Enter date:" appears on the screen of the host, you can type the date as a six-digit combination of year, month and day (see DATE$ variable). Time can also be set using the same method.

INPUT ON/OFF

Purpose

Statement enabling or disabling the Intermec Direct Protocol.

Syntax

INPUT ON|OFF

Default:	INPUT OFF

Remarks

These statements are used to enter or leave the Intermec Direct Protocol.

INPUT ON	Enables the Intermec Direct Protocol: - Enables reception of input data to a stored layout - Starts the error handler - Sets the verbosity to off (SYSVAR (18) = 0) - Shows “Direct Protocol 8.00” in the display
INPUT OFF	Disables the Intermec Direct Protocol: - Disables reception of input data to a stored layout - Stops the error handler - Resets the verbosity to the level selected before last INPUT ON was executed - Shows “Fingerprint 8.00” in the display

The following instructions will only work with the Intermec Direct Protocol, that is after an INPUT ON statement has been executed:

COUNT&	ERROR	FORMAT INPUT
INPUT OFF	LAYOUT END	LAYOUT INPUT
LAYOUT RUN		

Example

This example illustrates how the Intermec Direct Protocol is enabled, how new separators are specified, how a layout is stored in the printer’s memory, how variable data are combined with the layout, and how a label is printed. Finally, the Intermec Direct Protocol is disabled:

```
INPUT  ON ↵
FORMAT INPUT "#","@","&" ↵
LAYOUT INPUT "tmp:LABEL1" ↵
FT "Swiss 721 BT" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$ ↵
LAYOUT END ↵
LAYOUT RUN "tmp:LABEL1" ↵
#Line number 1&Line number 2&@ ↵
PF ↵
INPUT OFF ↵
```

INPUT#

Purpose

Statement for reading a string of data from an OPEN device or sequential file.

Syntax

INPUT#<nexp>,<<nvar>|<svar>>[,<<nvar>|<svar>>...]

<nexp>	is the number assigned to the file or device when it was OPENed.
<<nvar>|<svar>>	is the variable to which the input data will be assigned.

Remarks

This statement resembles the INPUT statement, but allows the input to come from other devices than the standard IN channel or from various files. Like the INPUT statement, commas can be used to assign different portions of the input to different variables. INPUT# does not allow prompts to be used.

When reading from a sequential file, the records can be read one after the other by the repeated issuing of INPUT# statements with the same file reference.

Once a file record has been read, it cannot be read again until the file is CLOSEd and then OPENed again.

The maximum number of characters that can be read using an INPUT# statement is 32,767 characters.

Note that INPUT# filters out any incoming ASCII 00 dec. characters (NUL).

Example

This example assigns data from the first record in the sequential file "Addresses" to the three string variables A$, B$, and C$ and from the second record in the same file to the string variables D$ and E$:

```
. . . . .
. . . . .
100  OPEN "ADDRESSES" FOR INPUT AS #5
110  INPUT#5, A$, B$, C$
120  INPUT#5, D$, E$
. . . . .
. . . . .
```

INPUT$

Purpose

Function returning a string of data, limited in regard of number of characters, from the standard IN channel, or optionally from an OPENed file or device.

Syntax

INPUT$(<nexp$_1$>[,<nexp$_2$>])

<nexp$_1$>	is the number of characters to be read.
<nexp$_2$>	optionally specifies a file or device using the number assigned to it when it was OPENed.

Remarks

If no file or device is specified, the input will come from the standard I/O channel (default "uart1:", see SETSTDIO statement). Otherwise, it will come from the specified file or device. The execution will be held until the specified number of characters has been received from the keyboard console, file, or communication channel. If a file does not contain the specified number of characters, the execution will be resumed as soon as all available characters in the file have been received.

The maximum number of characters that can be returned using an INPUT$ statement is 65,536 characters.

Examples

This example reads a sequence of 25 characters from the printer’s built-in keyboard and assigns them to a string variable named Z$:

```
. . . . .
. . . . .
1000 OPEN "CONSOLE:" FOR INPUT AS #1
1010 Z$=INPUT$(25,1)
. . . . .
. . . . .
```

In this example, 10 characters are read from the standard IN channel and assigned to a variable.

```
10   A$=INPUT$(10)
```

INSTR

Purpose

Function searching a specified string for a certain character, or sequence of characters, and returning its position in relation to the start of the string.

Syntax

INSTR([<nexp>,]<sexp$_1$>,<sexp$_2$>)

<nexp>	is , optionally, the position where the search will start.
<sexp$_1$>	is the string to be searched.
<sexp$_2$>	is the character(s) for which the string will be searched.

Remarks

INSTR allows you to search a string for some particular character(s) and return the position of the character, or the first character in the sequence, as a number of characters positions measured from the start of the string.

As an option, it is possible to specify a certain position in the string from which the search will start. If no start position is specified, the search will start at the beginning of the string.

The result will be zero if
- the start position value exceeds the length of the string.
- the string is empty.
- the searched combination of characters cannot be found.

Examples

In this example, the string "INTERMEC_PRINTER_AB" is searched for the character combination "AB". No start position is specified.

```
10   A$="INTERMEC PRINTER AB"
20   B$="AB"
30   PRINT INSTR(A$,B$)
RUN
```

yields:

```
18
```

In next example, the string "INTERMEC_PRINTER_AB" is searched for the character "I" and the start position is specified as 4.

```
10   A$="INTERMEC PRINTER AB"
20   B$="I"
30   PRINT INSTR(4,A$,B$)
RUN
```

yields:

```
12
```

INVIMAGE (II)

Purpose

Statement for inversing the printing of text and images from “black-on-white" to “white-on-black."

Syntax

INVIMAGE | II

Default:	NORIMAGE
Reset to default by:	PRINTFEED execution

Remarks

This statement can only be used in connection with the printing of text and images (PRTXT and PRIMAGE). In the matrix of the font or image, all “white” dots will be black and all black dots will be “white.” (Obviously, “white” means that the dots will not be subjected to heat and the media therefore will retain its original color, whereas “black” means the color of the printing.)

This implies that most fonts will be printed on a black background which ascends and descends slightly more than most of the characters. Not all fonts are suited for inverse printing. Thin lines, serifs, and ornaments may be difficult to distinguish. There may also be an imbalance between the ascending and descending black background.

The same principles apply to images. The normally invisible background may be larger than expected or be less favourably balanced. Small “white” details tend to be blurred out by the black background. Therefore, before using an inverse image, make a printout sample.

INVIMAGE will be revoked by a NORIMAGE statement.

Example

```
10   PRPOS 30,300
20   DIR 1
30   ALIGN 4
40   INVIMAGE
50   FONT "Swiss 721 BT"
60   PRTXT "Inverse printing"
70   PRINTFEED
RUN
```

KEY BEEP

Purpose

Statement for resetting the frequency and duration of the sound produced by the beeper, when any of the keys on the printer's keyboard is pressed down.

Syntax

KEY$_{\leftrightarrow}$BEEP<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is the frequency of the sound in Hz.	
<nexp$_2$>	is the duration of the sound in periods of 0.020 seconds each (max. 15,0000 = 5 minutes).	
Default:	Frequency:	1200 Hz
	Duration:	0.030 sec.

Remarks

This statement sets the response for all keys of the printer. To turn off the audible key response, set the frequency to a value higher than 9999.

Note that the beeper is disabled during printing.

The table below illustrates the relation between frequencies and the musical scale (same as in the SOUND statement).

Key	Hz	Key	Hz	Key	Hz	Key	Hz
C	131	C	262	C	523	C	1047
C#	138	C#	277	C#	554	C#	1109
D	147	D	294	D	587	D	1175
D#	155	D#	311	D#	622	D#	1245
E	165	E	330	E	659	E	1319
F	175	F	349	F	699	F	1397
F#	185	F#	370	F#	740	F#	1480
G	196	G	392	G	784	G	1568
G#	208	G#	415	G#	831	G#	1662
A	220	A	440	A	880	A	1760
A#	233	A#	466	A#	933	A#	1865
B	247	B	494	B	988	B	1976
(small octave)		(one-line octave)		(two-line octave)		(three-line octave)	

Example

In this example, the beeper will produce an A in the one-line octave for 1 second each time a key is pressed down.

```
10    KEY BEEP 440,50
. . . . .
. . . . .
```

KEY ON/OFF

Purpose

Statement enabling or disabling a specified key on the printer's front panel to be used in connection with an ON KEY...GOSUB statement.

Syntax

KEY(<nexp>)OFF|ON

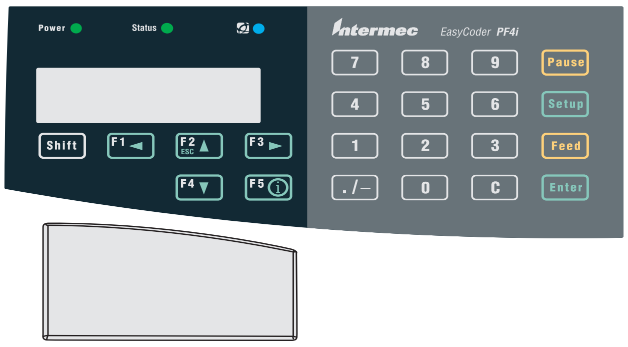
<nexp>	is the id. number of one of the keys on the printer's front panel (see illustration below).
OFF|ON	disables|enables the specified key.

Remarks

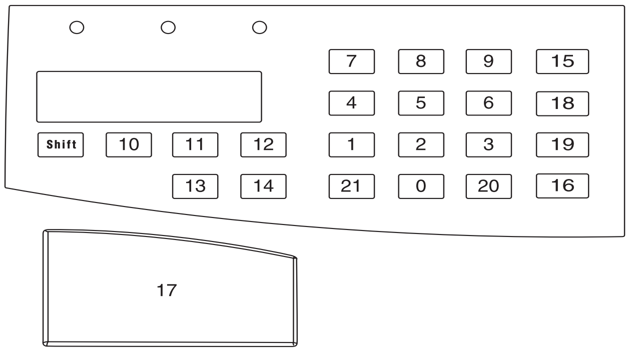
Using an ON KEY... GOSUB statement, any key (except the <Shift> key) can be assigned to make the program branch to a subroutine. The keys are enabled/disabled individually and are specified by means of their respective id. numbers in unshifted and/or shifted position. To specify a shifted key, add 100 to the unshifted id. number the key, as illustrated below.

Please note the difference between the id. numbers of the keys and the ASCII values they are able to produce (see KEYBMAP$).

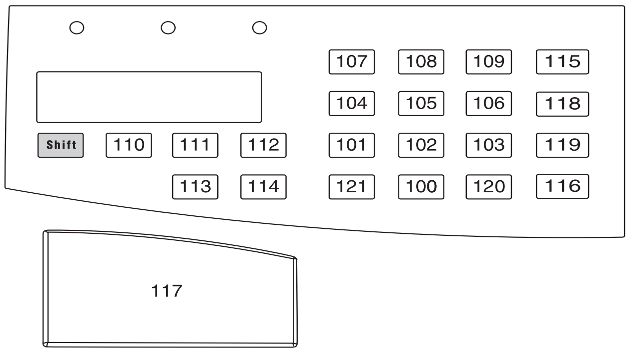
EasyCoder PF-series



Actual keyboard appearance



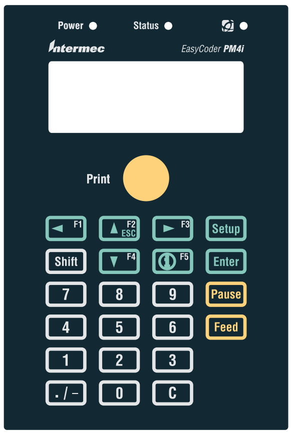
Unshifted keys id. numbers



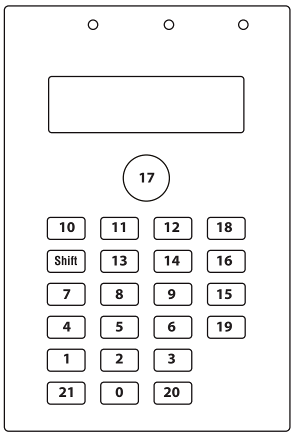
Shifted keys id. numbers

KEY ON/OFF, cont.

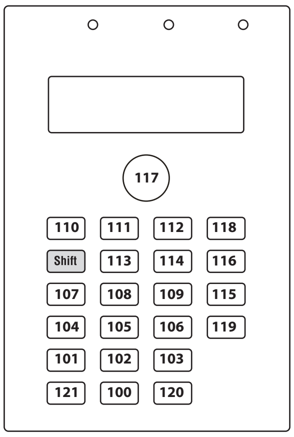
EasyCoder PM4i



Actual keyboard appearance



Unshifted keys; id. numbers



Shifted keys; id. numbers

Example

In this example, the ◄/F1 key (id. No. 10) is first enabled, then used for branching to a subroutine and finally disabled.

```
10    KEY (10) ON
20    ON KEY (10) GOSUB 1000
30    KEY (10) OFF
```

KEYBMAP$

Purpose

Variable returning or setting the keyboard map table.

Syntax

Read the map table:	**<svar> = KEYBMAP$(<nexp>)**
<svar>	returns the keyboard mapping
<nexp>	is the type of string to be returned: 0 = Unshifted 64 characters 1 = Shifted 64 characters

Set the map table:	**KEYBMAP$(<nexp>) = <sexp>**
<nexp>	is the type of string to be remapped: 0 = Unshifted 64 characters 1 = Shifted 64 characters
<sexp>	is the string specifying the ASCII value for each key position in the selected type of string.

Remarks

In the KEYBMAP$ statement, each key on the printer's front panel has two characteristics:

- The physical location of the key (position number). (This is not the same thing as the key's Id. No, see KEY ON/OFF or ON KEY GOSUB.)
- The ASCII decimal value that will be produced when the key is pressed. (Compare with BREAK.)

In principle, each physical key can produce two different ASCII values, one in unshifted position and another in shifted position. One key is appointed <Shift> key. When the <Shift> key is pressed at the same time as another key, the unshifted ASCII value of the latter will be increased by 128.

You can use the KEYBMAP$ instruction in two ways:

Reading the keyboard mapping
You can read how the keyboard is mapped in regard of either unshifted or shifted characters. The printer will return a string of ASCII values in ascending key position number. Because many keys return non-printable ASCII values (ASCII 00-31 dec.), all will not be returned to the screen of the host or printed on a label.

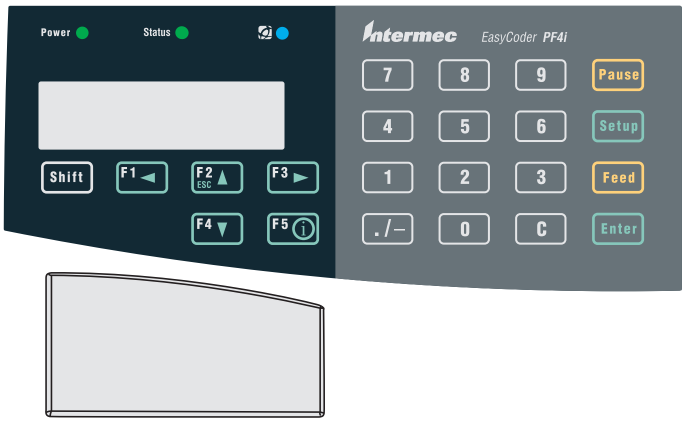
Changing the keyboard mapping
You can change the mapping of the keyboard, so a key will produce another ASCII value than before. To do that, you must create a string which specifies the ASCII value for each of all unshifted or shifted key positions in ascending order. Regardless of what the keyboard looks like, there are always 64 theoretical key positions.

KEYBMAP$, cont.

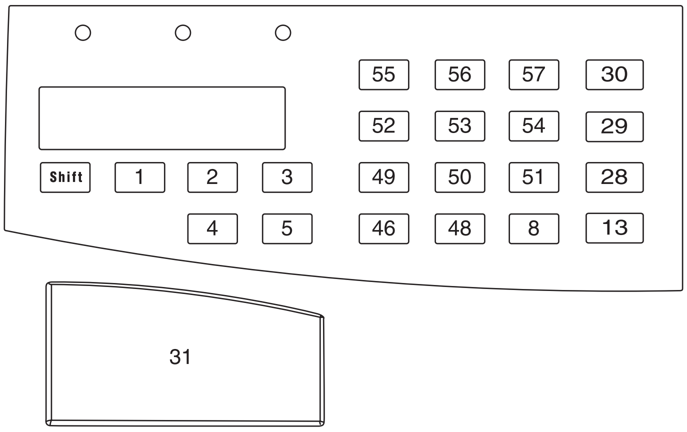
Characters, that cannot be produced by the keyboard of the host, can be substituted by CHR$ functions, where the character is specified by its ASCII decimal value according to the selected character set (see NASC statement.) The same applies to special characters. Key positions which should be disabled or are not included in the physical keyboard can be mapped as NUL, using the function CHR$(0). Note that the position of the <Shift> key cannot be remapped.

The keyboards return the following ASCII values:

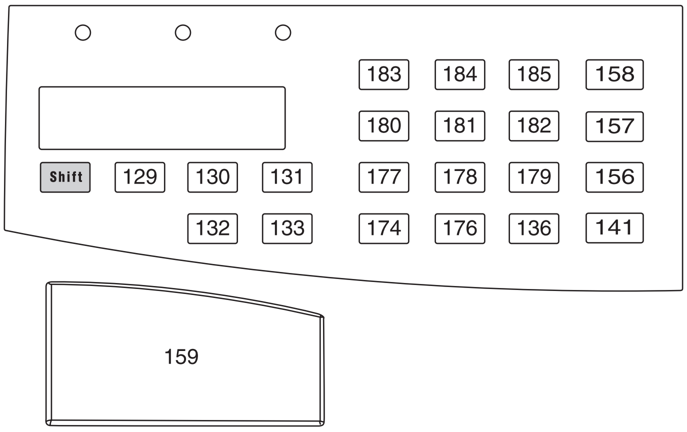
EasyCoder PF-series



Actual keyboard appearance



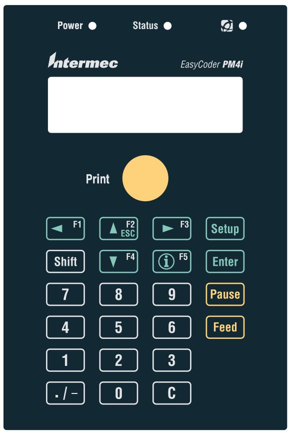
Unshifted keys ASCII values



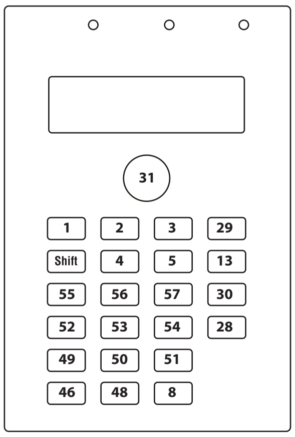
Shifted keys ASCII values

KEYBMAP$, cont.

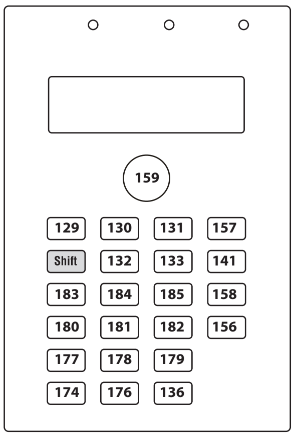
EasyCoder PM4i



Actual keyboard appearance

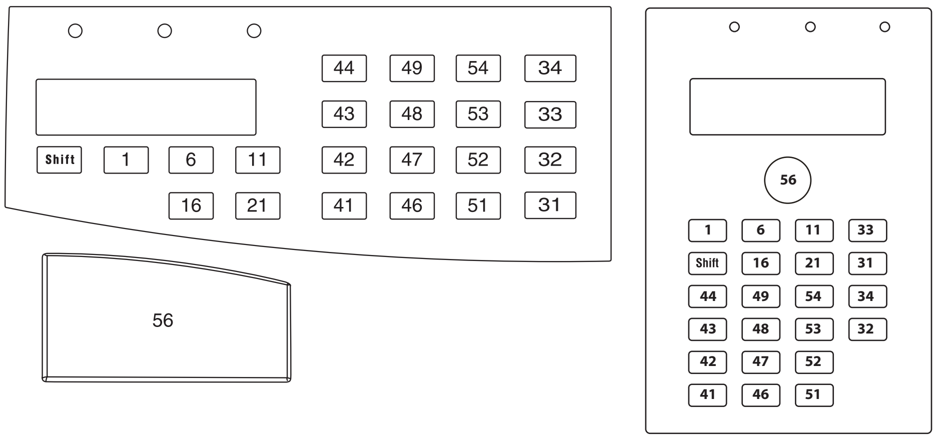


Unshifted keys; ASCII values



Shifted keys; ASCII values

Note: In the Setup Mode, the keys have fixed positions and are not affected by any KEYBMAP$ statement. KEYBMAP$ only affects the keys when used outside the Setup Mode.



Position numbers of the EasyCoder PF2/4i and PM4i keyboards. The keys marked "Shift" cannot be remapped.

Examples

The following example illustrates the mapping of the keyboard for Easy-Coder PF4i (unshifted keys only).

```
10    B$=CHR$(1)+STRING$(4,0)+CHR$(2)+
      STRING$(4,0)+CHR$(3)
20    B$=B$+STRING$(4,0)+CHR$(4)+STRING$(4,0)+
      CHR$(5)+STRING$(9,0)
30    B$=B$+CHR$(13)+CHR$(28)+CHR$(29)+CHR$(30)+
      STRING$(6,0)
40    B$=B$+".147"+CHR$(0)+"0258"+CHR$(0)+CHR$(8)
      +"369"+CHR$(0)+(CHR$(31)+STRING$(8,0)
50    KEYBMAP$(0)=B$
RUN
```

KILL

Purpose

Statement for deleting a file, directory, or complete directory sub-trees from the printer's memory or from a CompactFlash memory card inserted in the memory card adapter.

Syntax

KILL<sexp>[,R[,A]]

<sexp>	is the file or directory to be deleted.
R	recursively removes all non-system files in the specified sub-tree and then removes all empty directories in the same sub-tree.
A	optionally specifies that system files also should be removed.

Remarks

The name of the file to be deleted must match the name given when the file was saved, see SAVE statement. The name must include the extension. If no extension was entered manually by the operator when the file was SAVEd, the extension ".PRG" was added automatically.

To KILL a file residing in another directory than the current one (see CHDIR statement), you must include a reference to the directory in question when you specify the file, for example "card1:<filename>.XYZ".

KILL cannot be used for files residing in "rom:", "storage:", or "lock:".

A directory cannot be removed if it contains files or directories unless the R flag is included in the KILL statement. Otherwise error 1073, "Directory not empty" will occur.

A trailing slash character (/) may be added to directory names, but is not necessary.

The A and R flags are only applicable when removing directories, doing otherwise will result in error 1034, "Not a directory".

The root directory of any device cannot be removed.

Note the symmetry with FILES. FILES<sexp>,R and FILES<sexp>,R,A list files and directories that will be removed using KILL<sexp>,R and KILL<sexp>,R,A respectively.

The current directory may be removed (for example KILL CURDIR$,R). The current directory is not changed after such a command, but is invalid and a successful CHDIR statement is necessary to restore the current directory to one that exists (CHDIR".." may not work).

Also see CHDIR, FILES, and MKDIR

KILL, cont.

Example

```
10   ON ERROR GOTO 1000
20   CHDIR("/c")
30   MKDIR "DIR1"                    Create the directory DIR1
40   FILES
50   COPY "STDIO", "DIR1"            Copy STDIO into DIR1
60   FILES "DIR1"                    List files in DIR1
70   KILL "DIR1"                     Try to remove DIR1
80   KILL "DIR1",R                   Remove the directory recursively
90   FILES
100  END
1000 PRINT "error number "; ERR;"in line ";ERL
1010 RESUME NEXT
RUN
```

yields for example:

```
Files on /c

DIR1/                0  APPLICATION          0
boot/                0  ADMIN/               0
STDIO                4

22210562 bytes free  4 bytes used

STDIO                4

2220032 bytes free  4 bytes used
Error number 1073 in line 70
Files on /c

APPLICATION          0  boot/                0
ADMIN/               0  STDIO                4

2222080 bytes free   4 bytes used
```

LAYOUT

Purpose

Statement for handling of layout files.

Syntax

LAYOUT[F,] <sexp$_1$>,<sexp$_2$>,<svar>|<sexp$_3$>,<nvar>|<sexp$_4$>

F,	optionally allows use of data and error files instead of arrays
<sexp$_1$>	is the layout file.
<sexp$_2$>	is the logotype name file.
<svar>|<sexp$_3$>	is the data array (<svar>) or data file (<sexp$_3$>).
<nvar>|<sexp$_4$>	is the error array (<nvar>) or error file (<sexp$_4$>).

Remarks

<sexp$_1$>: Layout file format sorted in ascending order (Records 1-n, 52 bytes each)
Input: H = hex digit, D = Numeric digit, C = Alpha character

Byte #	Parameter	Layout Type	Input	Notes
0-1	Element number		HH	

2	Layout type
	A = Logotype by name
	B = Bar code
	C = Text
	E = Bar code extended field
	H = Barfont on/off
	J = Baradjust (corresponds to BARADJUST stmt)
	L = Logotype by number
	S = Line
	X = Box
			C	
				
				Note 1

3	Direction Barfont on/off (0=off; 1=on) Security	A,B,C,L,S,or X H E	D D D	
4	Alignment Aspect height ratio	A,B,C,L,S,X E	D D	
5-8	X-position Aspect width ratio Baradjust left	A,B,C,L,S,or X E J	DDDD D DDDD	
9-12	Y-position Rows in bar code Baradjust right	A,B,C,L,S,or X E J	DDDD DD DDDD	
13-22	Font name Logotype name Bar code name Barfont name Line length Box width Columns in bar code Truncate according to code spec's	C A B H S X E E	C_1-C_{10} C_1-C_{10} C_1-C_{10} C_1-C_{10} DDDD DDDD DD D	Note 2 Byte 13-14 Byte 15
23-42	Fixed text or alphanumeric data Fixed numeric data Logotype number Box height Line thickness	B or C B L X S	C_1-C_{20} D_1-D_{20} DD DDDD DDDD	
43-44	No of char. to print (of byte 23-42)	B or C	DD	
45-46	Image type (I = inverse image) Bar code ratio (wide/narrow bars)	A,C, or L B	C DD	
47	Vertical magnification Bar code magnification	A,C, or L B	D or C D or C	Note 3 Note 3
48	Horizontal magnification	A,C, or L	D	
49-51	Bar code height Line thickness	B X	DDD DDD	

LAYOUT, cont.

Note 1: The bar code extended field record (E) corresponds to the six last parameters in the BARSET statement. Must have a lower element number than the corresponding bar code record (B), which specifies the other bar code parameters.

Note 2: The maximum font name length in the LAYOUT statement is 10 characters. Most font names in Intermec Fingerprint are longer. A work-around method is to use font name aliases with a maximum length of 10 characters, See Chapter 6, "Fonts".

Note 3: If a magnification of 0-9 is sufficient, enter a numeric digit. If a higher magnification than 9 is required, enter the character with the ASCII decimal number minus 48 that corresponds to the desired magnification, that is, if magnification 10 is desired, enter the character : (colon, ASCII 58 dec).

Logotype name file format #1: (no embedded spaces in name) Record 1–n, 10 bytes each.	
C_1...C_{10}	Name for logotype No. 1
...	
...	
C_1...C_{10}	Name for logotype No. n

Logotype name file format #2: (Records sorted in ascending logotype number order) Record 1-n, 13 bytes each.	
DD	Logotype number (2 digits)
C	always ":" (colon). Separator. Distinguishes format 2.
C_1...C_{10}	Name of logotype (10 characters)

Note: Logotype name file formats #1 and #2 are alternative.

Data array/file format: (sorted in ascending order) One array position/One file line.	
HH	Element number
C_1...C_n	Data
If a data element cannot be used in the layout, an error will occur and the index of the unused element and error code -1 is placed in the error array/file.	

Error array/file format: (sorted in ascending order)	
Array position/File line No. 0:	Record number for error 1
Array position/File line No.1:	Error number for error 1
...	
...	
Array position/File line No. 2n-2:	Record number for error n
Array position/File line No. 2n-1:	Error number for error n

LAYOUT, cont.

To improve the performance, it is strongly recommended to create the layout and logotype name files in the printer's temporary memory ("tmp:"). Once they have been created in "tmp:", they could be copied to the printer's permanent memory to avoid losing them at power off.

Do not confuse this statement with the statements LAYOUT INPUT, LAYOUT END, and LAYOUT RUN.

Example

Note that the 10 characters available to define a font in the LAYOUT statement in most cases cannot accommodate modern outline font names. Instead, use font aliases as described in Chapter 6. In the example below, the font aliases are indicated by lowercase italic typing (lines 90–120, 150).

```
10    DIM QERR%(10)
20    LAYDATA$(0)="01DAY"
30    LAYDATA$(1)="04123456789012"
40    QERR%(0)=0
50    OPEN "tmp:LOGNAME.DAT" FOR OUTPUT AS 19
60    PRINT# 19,"DIAMONDS.1";
70    CLOSE 19
80    OPEN "tmp:LAYOUT.DAT" FOR OUTPUT AS 6
90    PRINT# 6,"01C11100 10  font alias                    00I 11   ";
100   PRINT# 6,"01C11100 40  font alias                    00  22   ";
110   PRINT# 6,"01C11100 100 font aliasWEDNES              06I 11   ";
120   PRINT# 6,"01C11100 130 font aliasSATURNUS            05I 11   ";
130   PRINT# 6,"02L11300  70          1                        33   ";
140   PRINT# 6,"03S11100 210 300       3                             ";
150   PRINT# 6,"04H1         font alias                             ";
160   PRINT# 6,"04B14100 300 EAN13                         0 312 100";
170   CLOSE 6
180   LAYOUT "tmp:LAYOUT.DAT","tmp:LOGNAME.DAT",LAYDATA$,QERR%
190   IF QERR% (1) = 0 THEN GOTO 260
200   PRINT "-ERROR- LAYOUT 1"
210   I%=0
220   IF QERR%(I%)=0 THEN GOTO 260
230   PRINT "    ERROR  ";QERR%(I%+1);" in record ";QERR%(I%)
240   I%=I%+2
250   GOTO 220
260   PRINTFEED
```

LAYOUT END

Purpose

Statement for stopping the recording of a layout description and saving the layout (Intermec Direct Protocol only).

Syntax

LAYOUT END

Remarks

This statement can only be used in the Intermec Direct Protocol after a layout has been recorded by means of a LAYOUT INPUT statement. After a LAYOUT END statement has been executed, no more data will be added to the layout.

By default, the layout will be saved in the printer's permanent memory ("/c"). To speed up the execution it can, as an alternative, be saved in the temporary memory (see LAYOUT INPUT statement). The layout can be copied and killed as any other program file.

Example

This example illustrates how the Intermec Direct Protocol is enabled, how new separators are specified, how a layout is stored in the printer's temporary memory, how variable data are combined with the layout, and how a label is printed. Finally, the Intermec Direct Protocol is disabled:

```
INPUT  ON  ↵
FORMAT INPUT "#","@","&" ↵
LAYOUT INPUT "tmp:LABEL1" ↵
FT "Swiss 721 BT"↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$ ↵
LAYOUT END ↵
LAYOUT RUN "tmp:LABEL1" ↵
#Line number 1&Line number 2&@ ↵
PF ↵
INPUT OFF ↵
```

LAYOUT INPUT

Purpose

Statement for starting the recording of a layout description (Intermec Direct Protocol only).

Syntax

LAYOUT INPUT <sexp>

<sexp>	is the desired name of the layout (max. 30 characters) including name of the device where the layout is to be stored.

Remarks

This statement can only be used in the Intermec Direct Protocol and starts the recording of a layout. All formatting instructions, such as PRPOS, MAG, FONT, BARFONT, BARSET, PRTXT, PRBAR, PRIMAGE, PRBOX, PRLINE, etc., which are transmitted to the printer on the standard IN channel after a LAYOUT INPUT statement and before a LAYOUT END statement, will be included in the layout.

Layouts cannot be created in "/c" (which by default is the current directory), but must be created in the printer's temporary memory ("tmp:"), or possibly in a CompactFlash card ("card1:"). Once a layout has been created in the temporary memory ("tmp:"), it can be copied to either "/c" or "card1:" so it will not be lost at power-off or reboot.

Variable input data to text, bar code, and image fields can be provided separately using a LAYOUT RUN statement. Such variable data are indicated in the layout by string variables VARn$ where "n" is the number of the field in the LAYOUT RUN string of data. For example, the statement PRTXT "Hello" in the layout results in a fixed text, whereas the statement PRTXT VAR1$ results in a variable text, which is provided by the first field in a LAYOUT RUN string.

The layout must not contain any PRINTFEED statements. The layout will not be saved until a LAYOUT END statement is executed.

Example

In this example, the Intermec Direct Protocol is enabled, new separators are specified, a layout is stored in "tmp:", data are combined with the layout, and a label is printed. Finally, the Direct Protocol is disabled:

```
INPUT  ON  ↵
FORMAT INPUT "#","@","&" ↵
LAYOUT INPUT "tmp:LABEL1" ↵
FT "Swiss 721 BT" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$ ↵
LAYOUT END ↵
LAYOUT RUN "tmp:LABEL1" ↵
#Line number 1&Line number 2&@ ↵
PF ↵
INPUT OFF ↵
```

LAYOUT RUN

Purpose

Statement for providing variable input data to a predefined layout (Intermec Fingerprint Direct Protocol only).

Syntax

LAYOUT RUN <sexp>

<sexp>	is the name of the layout as specified in the LAYOUT INPUT statement.

Remarks

This instruction can only be used in the Intermec Direct Protocol and is used to select a predefined layout in a specified part of the printer's memory (see LAYOUT INPUT statement) and provide input to string variables in the layout. Such variables are indicated by VARn$, where "n" indicates a field in the string of data that should follow the LAYOUT RUN statement.

The string of input data should be composed according to the following syntax, where <STX> is the start-of-text separator, <CR> is the field separator and <EOT> is the end-of-text separator (see FORMAT INPUT statement):
<STX><input to VAR1$><CR><input to VAR2$><CR>....<input to VARn$><CR><EOT>

Before reverting to "normal" Fingerprint printing after having used variable data (LAYOUT INPUT, LAYOUT END, and LAYOUT RUN), the data must be cleared using LAYOUT RUN with an empty string (**LAYOUT RUN ""**).

Example

This example illustrates how the Intermec Direct Protocol is enabled, how new separators are specified, how a layout is stored in the printer's temporary memory, how variable data are combined with the layout, and how a label is printed. Finally, the Intermec Direct Protocol is disabled:

```
INPUT  ON  ↵
FORMAT INPUT "#","@","&" ↵
LAYOUT INPUT "tmp:LABEL1" ↵
FT "Swiss 721 BT" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$ ↵
LAYOUT END ↵
LAYOUT RUN "tmp:LABEL1" ↵
#Line number 1&Line number 2&@ ↵
PF ↵
INPUT OFF ↵
```

LBLCOND

Purpose

Statement for overriding the media feed setup.

Syntax

LBLCOND<nexp$_1$>,<nexp$_2$>|<nexp$_3$>

<nexp$_1$>	specifies the type of action: 0 = Overriding the stop adjust. 1 = Overriding the start adjust. 2 =Turning off the Label Stop Sensor/Black Mark Sensor. 3 = Selecting the mode specified by <nexp$_3$>
<nexp$_2$>	specifies <nexp$_1$> = 0, 1, or 2 as a number of dots.
<nexp$_3$>	Specifies one of the following modes: 0 = Legacy Mode 1 = IPL Mode 2 = Gap Truncate Mode Default: LBLCOND 3,2

Remarks

This instruction allows you to override the printer's feed-adjust setup or to temporarily disable the label stop sensor or black mark sensor:

<nexp$_1$> = 0	temporarily sets the stop adjust to the value specified by <nexp$_2$>.
<nexp$_1$> = 1	temporarily sets the start adjust to the value specified by <nexp$_2$>.
<nexp$_1$> = 2	makes the label stop sensor (LSS) or black mark sensor temporarily ignore any gaps or marks detected within the length of media feed specified by <nexp$_2$>. However, the label length must be greater than than the distance between the LSS and the tear bar (if not, use LBLCOND 3,xx). This allows the use of labels of such shapes that would make the LSS react prematurely, or tickets with preprint at the back of the media that would interfere with the detection of the black mark.
<nexp$_1$> = 3	is useful as an alternative to LBLCOND 2,xx when the length of the label or ticket is shorter than the distance between the LSS and the tear bar. It makes it possible to select one of the modes specified by <nexp$_3$>.

Legacy Mode (<nexp$_3$> = 0)
If the print image is longer than the physical length of the label or ticket, the print image will extend into the next label until the media feed stops according to the stop adjust setup (for example when the gap becomes aligned with the tear bar). This means that the print image may be truncated, the next label may have to be discarded, and some of the print image may coincide with a gap or slot. This mode was called "Default Mode" in earler versions of Intermec Fingerprint.

LBLCOND, cont.

IPL Mode (<nexp$_3$> = 1)
If the print image is longer than the physical length of the label or ticket, the print image will extend into the following label(s) until the entire print image has been printed. Then the media is fed out to the next gap or mark according to the stop adjust setup. This means that the print image will not be truncated but may extend into one or more consecutive labels, and some of the print image may coincide with gaps or slots.

Gap Truncate Mode (<nexp$_3$> = 2)
If the print image is longer than the physical length of the label or ticket, only the part of the print image that fits on the label or ticket will be printed and the remainder will be ignored. This means that some of the print image may not be printed at all, but the following labels will not be affected.

Verifying a start adjust or stop adjust value in the Setup Mode by pressing key No. 16 (normally labeled “Enter”), or setting the value using a setup file or setup string, will revoke any LBLCOND statement for the parameter in question.

The label stop sensor will be returned to normal operation by the statement: LBLCOND 2,0

All current LBLCOND statements will be revoked at startup or the execution of a REBOOT statement. This means that the start and stop adjust will be decided by the setup and the label stop sensor will work normally.

Example

In this example, the start adjust value in the setup mode is overridden and the label stop sensor is set to ignore any gaps in the web within 20 mm (160 dots at 8 dots/mm; 240 dots at 12 dots/mm) of media feed:

```
10   LBLCOND 1,5: LBLCOND 2,160
20   FONT "Swiss 721 BT"
30   PRTXT "Hello"
40   PRINTFEED
RUN
```

LED ON/OFF

Purpose

Statement for controlling the dual-color “Status” indicator.

Syntax

LED<nexp>ON|OFF|BLINK [,DATAIN]

<nexp>	specifies the color of the Status indicator: 0 controls the green LED (default ON). 1 controls the red LED (default OFF)

Remarks

All present Intermec Fingerprint printers are equipped with three LED (Light Emitting Diode) control lamps on the front panel. The center one of the LEDs (normally marked “Status” on the keyboard overlay) can be used to indicate, for example, when an error occurs, when the printer is ready, or when data is received.

Under the Direct Protocol, the “Status” LED starts blinking green, when data are received on the standard input channel, and switches to solid green, when the channel has been silent for 0.8 seconds. Under the Direct Protocol, the Status LED may also be affected by the error handler, see ERROR statement.

BLINK mode means that the “Status” LED is lit for 0.4 seconds at an interval of 0.4 seconds.

If the DATA IN flag is set with the BLINK mode, reception of data on the standard input channel controls whether the “Status” LED shall blink or be switched off, exactly as the behavior under the Direct Protocol.

If you set LED 0 ON and LED 1 ON, the “Status” LED will get solid red.

Note: If the printer has a console that does not includes the blue Intermec Readiness Indicator, please refer to *Intermec Fingerprint v8.10, Programmer's Reference Manual* for a description of the LED ON/OFF statement.

Example

In this example, the “Status” LED will be solid red if you, for example, attempt to run the program with a raised printhead. Lower the printhead and a label will be fed out. The “Status” LED switches to solid green.

```
10    LED 0 ON
20    LED 1 OFF
30    ON ERROR GOTO 1000
40    PRPOS 30,300
50    FONT "Swiss 721 BT"
60    PRTXT "OK!"
70    PRINTFEED
80    LED 0 ON
90    LED 1 OFF
100   END
.....
.....
1000  LED 0 OFF
1010  LED 1 ON
1020  RESUME
```

LEFT$

Purpose

Function returning a specified number of characters from a given string starting from the extreme left side of the string, that is from the start.

Syntax

LEFT$(<sexp>,<nexp>)

<sexp>	is the string from which the characters will be returned.
<nexp>	is the number of characters to be returned.

Remarks

This function is the complementary function for RIGHT$, which returns the characters starting from the extreme right side, that is from the end.

If the number of characters to be returned is greater than the number of characters in the string, then the entire string will be returned. If the number of characters is set to zero, a null string will be returned.

Examples

```
10   PRINT LEFT$("THERMAL PRINTER",7)
RUN
```

yields:

```
THERMAL
```

```
10   A$="THERMAL PRINTER":B$="LABEL"
20   PRINT LEFT$(A$,8);LEFT$(B$,10);"S"
RUN
```

yields:

```
THERMAL LABELS
```

LEN

Purpose

Function returning the number of character positions in a string.

Syntax

LEN(<sexp>)

<sexp>	is the string from which the number of characters will be returned.

Remarks

The number of characters to be returned includes unprintable characters, but the quotation marks enclosing the string expression are not included.

Examples

In this example, lines 40 and 50 illustrate two ways of using the LEN function, when the number of characters from several string expressions are to be added up.

```
10    A$="INTERMEC"                             (8 char.)
20    B$="THERMAL"                              (7 char.)
30    C$="PRINTERS"                             (8 char.)
40    PRINT LEN(A$+B$+C$)
50    PRINT LEN(A$)+LEN(B$)+LEN(C$)
RUN
```

yields:

```
23
23
```

This example illustrates that unprintable characters, for example space characters, are included in the value returned by the LEN function:

```
PRINT LEN("INTERMEC THERMAL PRINTERS")
```

yields:

```
25
```

LET

Purpose

Statement for assigning the value of an expression to a variable.

Syntax

[LET]<<nvar>=<nexp>>|<<svar>=<sexp>>

<nvar>	is the numeric variable to which a value will be assigned.
<nexp>	is the numeric expression from which the value will be assigned to the numeric variable.
or...	
<svar>	is the string variable to which the content of the string expression will be assigned.
<sexp>	is the string expression from which the content will be assigned to the string variable.

Remarks

The keyword LET is not necessary, but retained for compatibility with old versions of Intermec Fingerprint. The equal sign (=) is sufficient to make the assignment. Both the expression and the variable most be either string or numeric.

Example

```
10   LET A%=100                              (numeric variable)
20   B%=150                                  (numeric variable)
30   LET C$="INTERMEC"                       (string variable)
40   D$="THERMAL PRINTERS"                   (string variable)
50   PRINT A%+B%,C$+" "+D$
RUN
                                                        yields:
250  INTERMEC THERMAL PRINTERS
```

LINE INPUT

Purpose

Statement for assigning an entire line, including punctuation marks, from the standard IN channel to a single string variable.

Syntax

LINE↔INPUT[<scon>;]<svar>

<scon>;	is an optional prompt plus a semicolon
<svar>	is the string variable to which the input line is assigned.

Remarks

For information on standard I/O channel, see SETSTDIO statement. By default, "uart1:" is the standard I/O channel.

LINE INPUT differs from INPUT in that an entire line of max. 32,767 characters will be read. Possible commas will appear as punctuation marks in the string instead of dividing the line into portions.

During the execution of a program, a LINE INPUT statement will interrupt the execution. You can make a prompt being displayed on the host screen to notify the operator that the program is expecting additional data to be entered. The input is terminated and the program execution is resumed when a carriage return character (ASCII 13 decimal) is encountered. The carriage return character will not be included in the input line.

Note that LINE INPUT filters out any incoming ASCII 00 dec. characters (NUL).

Example

Print your own visiting card like this:

```
10    LINE INPUT "ENTER NAME: ";A$
20    LINE INPUT "ENTER STREET: ";B$
30    LINE INPUT "ENTER CITY: ";C$
40    LINE INPUT "ENTER STATE + ZIPCODE: ";D$
50    LINE INPUT "ENTER PHONE NO: ";E$
60    FONT "Swiss 721 BT", 8
70    ALIGN 5
80    PRPOS 160,300:PRTXT A$
90    PRPOS 160,250:PRTXT B$
100   PRPOS 160,200:PRTXT C$
110   PRPOS 160,150:PRTXT D$
120   PRPOS 160,100:PRTXT "Phone: "+E$
130   PRINTFEED
RUN
```

LINE INPUT#

Purpose

Statement for assigning an entire line, including punctuation marks, from a sequential file or a device to a single string variable.

Syntax

LINE$_{\leftrightarrow}$INPUT#<nexp>,<svar>

<nexp>	is the number assigned to the file when it was OPENed.
<svar>	is the string variable to which the input line is assigned.

Remarks

This statement differs from the INPUT# statement in that an entire line of max. 32,767 characters will be read, and possible commas in the line will be included in the string as punctuation marks instead of dividing it into portions.

When reading from a sequential file, the lines can be read one after the other by the repeated issuing of LINE INPUT# statements, using the same file reference.

Once a line has been read, it cannot be read again until the file is CLOSEd and then OPENed again.

The LINE INPUT# statement is useful when the lines in a file has been broken into fields.

Note that LINE INPUT# filters out any incoming ASCII 00 dec. characters (NUL).

Example

This example assigns data from the three first lines of the file "Addresses" to the string variables A$, B$, and C$ respectively:

```
. . . . .
. . . . .
100  OPEN "ADDRESSES" FOR INPUT AS #5
110  LINE INPUT# 5, A$
120  LINE INPUT# 5, B$
130  LINE INPUT# 5, C$
. . . . .
. . . . .
```

LIST

Purpose

Statement for listing the current program completely or partially, or listing all variables, to the standard OUT channel.

Syntax

LIST[[<ncon$_1$>[–<ncont$_2$>]]|,V|,B]

<ncon$_1$>	is a single line, or the first line number in a range of lines.
<ncon$_2$>	is optionally the last line number in a range of lines.
,V	lists all variables.
,B	lists all breakpoints.

Remarks

This instruction is useful after LOADing a program, or if you during programming have changed any program lines, renumbered the lines, or added new lines and want to bring some order in the presentation on the screen of the host. LIST also removes unnecessary characters and adds assumed keywords. The instruction is usually given in the immediate mode, that is on a line without any preceding line number.

The LIST statement can be used in seven different ways:

- If no line number is entered after LIST, the entire current program will be listed. In case the program has been written without line numbers (see IMMEDIATE ON/OFF statements), the lines will be automatically numbered with 10-step incrementation starting with line number 10 (10-20-30-40-50....).
- If a single line number is entered after LIST, only the specified line will be listed.
- If a line number followed by a hyphen (-) is entered after LIST, all lines from the specified line to the end of the program will be listed.
- If a hyphen (-) followed by line number is entered after LIST, all lines from the start of the program through the specified line will be listed.
- If two line numbers are entered after LIST, they will specify the first and last line in a range of lines to be listed.
- If LIST,V is entered, all integer variables, integer array variables, string variables, and string array variables in the printer's memory will be listed.
- If LIST,B is entered, all breakpoints of the Fingerprint Debugger (see DBBREAK) will be printed in line number order. Line labels that have not been updated, which occurs at program execution, may be mis-placed.

LIST, cont.

Examples

`LIST`	Lists all lines in the program.
`LIST 100`	Lists line No. 100 only.
`LIST 100-`	Lists all lines from line No 100 to the end of the program.
`LIST -500`	Lists all lines from the start of the program through line No. 500.
`LIST 100-500`	Lists all lines from line 100 through line 500.
`LIST,V`	Lists all variables.
`LIST,B`	Lists all breakpoints.

LISTPFSVAR

Purpose Statement for listing variables saved at power failure.

Syntax **LISTPFSVAR**

Remarks Related instructions are SETPFSVAR, GETPFSVAR, and DELETE-PFSVAR.

Example

```
LISTPFSVAR
```

yields for example:

```
QS$
QCPS%
A%
```

LOAD

Purpose

Statement for loading a copy of a program, residing in the current directory or in another specified directory, into the printer's working memory.

Syntax

LOAD<scon>

<scon>	is the program to be loaded into the working memory.

Remarks

If the program has the extension .PRG, the name of the program can be given with or without any extension. Otherwise, the extension must be included in the name. If the program resides in another directory than the current one (see CHDIR statement), the name must also contain a reference to the directory in question.

LOAD closes any open files and deletes all program lines and variables residing in the working memory before loading the specified program. If the previous program in the working memory has not been saved, see SAVE statement, it will be lost and cannot be retrieved.

While the program is loaded, a syntax check is performed. If a syntax error is detected, the loading will be interupted and an error message will be transmitted on the standard OUT channel.

Examples

Load the program "LABEL127.PRG" from the current directory:

```
LOAD "LABEL127"
```

or

```
LOAD "LABEL127.PRG"
```

When “Ok” appears on the screen, the loading is completed. Use a LIST statement to display the program on the screen of your terminal.

You may also load a program stored in another directory than the current one, for example the read-only memory ("/rom") or a CompactFlash memory card ("card1:"). Start the file name by specifying the directory, for example:

```
LOAD "/rom/MKAUTO.PRG"
```

or

```
LOAD "card1:PROGRAM1.PRG"
```

This will create a copy, which you can list or change and then save under a new name.

LOC

Purpose

Function returning the current position in an OPEN file or the status of the buffers in an OPEN communication channel.

Syntax

LOC(<nexp>)

<nexp>	is the number assigned to the file or communication channel when it was OPENed.

Remarks

In a random file, LOC will return the number of the last record read or written by the use of GET or PUT statements respectively.

In a sequential file, the number of 128-byte blocks, that have been read or written since the file was OPENed, will be returned.

LOC can also be used to check the receive or transmit buffer of the specified communication channel:
- If the channel is OPENed for INPUT, the remaining number of characters (bytes) to be read from the receive buffer is returned.
- If the channel is OPENed for OUTPUT, the remaining free space (bytes) in the transmit buffer is returned.

The number of bytes includes characters that will be MAPped as NUL.

Examples

This example closes the file "addresses" when record No. 100 has been read from the file:

```
10   OPEN "ADDRESSES" FOR INPUT AS #1
.....
.....
.....
200  IF LOC(1)=100 THEN CLOSE #1
.....
.....
```

This example reads the number of bytes which remains to be received from the receive buffer of "uart2:":

```
100  OPEN "uart2:" FOR INPUT AS #2
110  A%=LOC(2)
120  PRINT A%
```

LOF

Purpose

Function returning the length in bytes of an OPEN sequential or random file, or returning the status of the buffers in an OPEN communication channel.

Syntax

LOF(<nexp>)

(<nexp>)	is the number assigned to the file or communication channel when it was OPENed.

Remarks

LOF can also be used to check the receive or transmit buffer of the specified communication channel:

- If a channel is OPENed for INPUT, the remaining free space (bytes) in the receive buffer is returned.
- If a channel is OPENed for OUTPUT, the remaining number of characters to be transmitted from the transmit buffer is returned.

Examples

The first example illustrates how the length of the file "Pricelist" is returned:

```
10   OPEN "PRICELIST" AS #5
20   A%=LOF(5)
30   PRINT A%
. . . .
. . . .
```

The second example shows how the number of free bytes in the receive buffer of communication channel "uart2:" is calculated:

```
100  OPEN "uart2:" FOR INPUT AS #2
110  A%=LOF(2)
120  PRINT A%
```

LSET

Purpose

Statement for placing data left-justified into a field in a random file buffer.

Syntax

LSET<svar>=<sexp>

<svar>	is the string variable assigned to the field by a FIELD statement.
<sexp>	holds the input data.

Remarks

After having OPENed a file and formatted it using a FIELD statement, you can enter data into the random file buffer using the LSET and RSET statements (RSET right-justifies the data).

The input data can only be stored in the buffer as string expressions. Therefore, a numeric expression must be converted to string format by the use of an STR$ function before an LSET or RSET statement is executed.

If the length of the input data is less than the length of the field, the data will be left justified and the remaining number of bytes will be printed as space characters.

If the length of the input data exceeds the length of the field, the input data will be truncated on the right side.

Example

```
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    SNAME$="SMITH"
40    CNAME$="JOHN"
50    PHONE$="12345630"
60    LSET F1$=SNAME$
70    LSET F2$=CNAME$
80    RSET F3$=PHONE$
90    PUT #8,1
100   CLOSE#8
RUN
SAVE "PROGRAM 1.PRG "
NEW
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    GET #8,1
40    PRINT F1$,F2$,F3$
RUN
```

yields:

```
SMITH — — — JOHN — — — — — —  12345630
```

LTS& ON/OFF

Purpose

Statement for enabling or disabling the label taken sensor.

Syntax

LTS& ON|OFF

Default: LTS& OFF

Remarks

The label taken sensor (LTS) is a photoelectric device that can be fitted in the vicinity of the printer's label outfeed slot and detects if a printed label or ticket has been removed or not. (Usually, a self-adhesive label is not fed out completely, but will remain partly stuck to the liner so it will not fall off.)

Using the LTS ON statement, you can order the printer to stop the execution at next PRINTFEED statement until the LTS no longer detects any label. Then the PRINTFEED is executed. This is most useful when printing batches of labels or tickets. As soon as a label is taken, the next one is printed and awaits being taken care of.

The same result can also be obtained in a more cumbersome way by a program based on the PRSTAT(2) function.

LTS& OFF revokes LTS& ON.

Example

```
10   LTS& ON
20   FOR A%=1 TO 5
30   B$=STR$(A%)
40   FONT "Swiss 721 BT"
50   PRPOS 200,200
60   PRTXT B$
70   PRINTFEED
80   NEXT
RUN
```

MAG

Purpose

Statement for magnifying a font, barfont, or image up to four times separately in regard of height and width.

Syntax

MAG<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is the magnification in regard of height (1, 2, 3, or 4).
<nexp$_2$>	is the magnification in regard of width (1, 2, 3, or 4).
Default value:	1,1
Reset to default by:	PRINTFEED execution

Remarks

Magnification makes the object grow in directions away from the selected anchor point, see ALIGN statement.

The MAG statement has become more or less obsolete for fonts and bar fonts with the implementation of scaleable fonts. Even if MAG works for such fonts, the printout quality will be much better by using a larger font size rather than magnifying a smaller one. However, the MAG statement is retained to allow compatibility with programs originally written for older Intermec Fingerprint versions.

The MAG statement also works with images. However, since the MAG statement simply enlarges the bitmap pattern of an image, it gives a better printout quality to download and use a larger version of an image rather than magnifying a smaller one.

Note that the MAG statement cannot be used for bar code patterns (use BARHEIGHT and BARMAG statement for that purpose).

Example

This example illustrates how the image "GLOBE.1" is printed both with its original size and magnified 4 times. Note the jagged edges of the curves in the enlarged image.

```
10   ALIGN 2
20   PRPOS 300,50
30   FONT "Swiss 721 BT"
40   PRTXT "Normal Size"
50   PRPOS 300,125
60   PRIMAGE "GLOBE.1"
70   PRPOS 300,300
80   PRTXT "Enlarged 4X"
90   PRPOS 300,375
100  MAG 4,4
110  PRIMAGE "GLOBE.1"
120  PRINTFEED
RUN
```

MAKEASSOC

Purpose

Statement for creating an association.

Syntax

MAKEASSOC <sexp$_1$>, <sexp$_2$>, <sexp$_3$>

<sexp$_1$>	specifies the name of the association to be created (case-sensitive).
<sexp$_2$>	contains an argument list of parameter tuples according to the convention in <sexp$_3$>.
<sexp $_3$>	should always be "HTTP" (case sensitive).

Remarks

HTTP implies that the argument list in <sexp$_2$> is encoded in “x-www-url-encoding.”

Example

This example shows how a string, including three stringnames associated with three start values, will be defined and one of them (time) will be changed:

```
10   QUERYSTRING$ =
     "time=UNKNOWN&label=321&desc=DEF"
20   MAKEASSOC "QARRAY", QUERYSTRING$, "HTTP"
30   QTIME$ = GETASSOC$("QARRAY", "time")
40   QLABELS% = VAL(GETASSOC$("QARRAY","label"))

50   QDESC$ = GETASSOC$("QARRAY", "desc")
60   PRINT "time=";QTIME$, "LABEL=";QLABELS%,
     "DESCRIPTION=";QDESC$
70   SETASSOC "QARRAY", "time", time$
80   PRINT "time="; GETASSOC$("QARRAY", "time")
RUN
```

yields:

```
time=UNKNOWN  LABEL=321 DESCRIP   TION=DEF
time=153355
```

MAP

Purpose

Statement for changing the ASCII value of a character when received on the standard IN channel, or optionally on another specified communication channel.

Syntax

MAP[<nexp$_1$>,]<nexp$_2$>,<nexp$_3$>

<nexp$_1$>	optionally specifies a communication channel: 0 = "console:" 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:" Default: Standard I/O channel.
<nexp$_2$>	is the original ASCII decimal value.
<nexp$_3$>	is the new ASCII decimal value after mapping.

Remarks

This statement is used to modify a character set (see NASC and NASCD statements) or to filter out undesired character. If you for example want a “Q” (ASCII 81 dec.) to be printed as the letter “Z” (ASCII 90 dec.), the MAP statement should be entered as: `MAP 81,90`

The mapping interprets any ASCII 81 dec. value received on the standard IN channel as ASCII 90 dec., that is when you press “Q” on the keyboard of the host, the character “Z” will be printed (see note). However, pressing “Z” will still produce a “Z”, because that character has not been remapped.

To reset the mapping performed above, map the character back to its original ASCII value like this: `MAP 81,81`

When a character is received by the printer, it is processed in regard of possible MAP statements before it “enters” the Intermec Fingerprint firmware. That allows you to filter out undesired control characters, which may confuse the Intermec Fingerprint firmware, for example by mapping them as NUL (ASCII 0 decimal).

After processing, the selected character set (see NASC and NASCD statements) controls how characters will be printed or displayed. If none of the character sets meets your demands completely, use MAP statements to modify the set that comes closest. Note that MAP statements will be processed before any COMSET or ON KEY..GOSUB strings are checked. NASC and NASCD statements will be processed last.

Do not map any characters to ASCII values occupied by characters used in Intermec Fingerprint instructions, for example keywords, operators, %, $, #, and certain punctuation marks. Mapping will be reset to normal at power-up or reboot.

MAP, cont.

Examples

You can check what characters the host produces using a simple program. Pressing different keys on the host should produce the corresponding characters both on the label and on the screen of the host. If not, try another character set (see NASC). In this example we presume that the keyboard produces ASCII 81 dec. and ASCII 90 dec. when you press the Q and Z keys respectively. Should any unexpected characters be printed on the labels or the screen, check the manuals of the host for information on what ASCII values will be produced by the various keys and how the screen will present various ASCII values received from the printer.

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,100
30   INPUT "Enter character";A$
40   PRTXT A$
50   PRINTFEED
```

By adding a MAP statement in line 5, you can test what happens. In this case we remap the character Q to be printed as Z, as in the explanation on the previous page. After printing, we map the character Q back to its original position.

```
5    MAP 81,90
10   FONT "Swiss 721 BT"
20   PRPOS 30,100
30   INPUT "Enter character";A$
40   PRTXT A$
50   PRINTFEED
60   MAP 81,81
```

Assume that a device connected to "uart2:" produces strings always starting with the control character STX (ASCII 2 decimal). STX can be filtered out by mapping it as NUL (ASCII 0 decimal):

```
10   MAP 2,2,0
```

Should "uart2:" be appointed standard IN channel (see SETSTDIO), the first parameter can be omitted from the example above:

```
10   MAP 2,0
```

MERGE

Purpose

Statement for merging a program in the printer's current directory, or optionally in another specified directory, with the program currently residing in the printer's working memory.

Syntax

MERGE<scon>

<scon>	is the name (optionally including a reference to another directory than the current one) of the program, which is to be merged with the program currently residing in the printer's working memory.

Remarks

MERGE creates a copy of a program stored in the current directory (see CHDIR statement), or optionally in a specified other directory, and blends its lines into the program currently residing in the printer's working memory.

Caution

If there are lines with the same numbers in both programs, the lines in the program currently residing in the working memory will be replaced by the corresponding lines in the MERGEd program. This also applies to programs written without line numbers, since they will automatically be assigned hidden line numbers (10-20-30... etc.) at the execution of the IMMEDIATE ON statement. In order to avoid overwriting any lines, you may SAVE a program without line numbers using a SAVE <scon>, L statement. When MERGEd, it will be appended to the current program and assigned line numbers that start with the number of the last line of the current program plus 10. For safety reasons, a backup copy of the current program is recommended before issuing a MERGE statement.

MERGE makes it possible to store blocks of program instructions, which are frequently used, and include them into new programs. The printer's ROM memory contains a number of useful programs, which also can be MERGEd into programs of your own creation.

Caution

Be careful not to include any MERGE statement as a part of a program, or else the execution will stop after the MERGE statement has been executed.

The EXECUTE statement offers an alternative method for combining Fingerprint programs.

Examples

The program "XYZ.PRG" will be merged with the current program.
If there are identical line numbers in both programs, the lines from "XYZ.PRG"will replace those in the current program.

```
MERGE "XYZ.PRG"                    (from current directory)
MERGE "/c/XYZ.PRG"                 (from permanent memory)
MERGE "tmp:XYZ.PRG"                (from temporary memory)
MERGE "card1:XYZ.PRG"              (from memory card)
```

MID$

Purpose

Function returning a specified part of a string.

Syntax

MID$(<sexp>,<nexp$_1$>[,<nexp$_2$>])

<sexp>	is the original string.
<nexp$_1$>	is the start position in the original string.
[,<nexp$_2$>]	is the number of characters to be returned (optional).

Remarks

<sexp> is the original string from which a specified part is to be returned.

<nexp$_1$> specifies which character position in the original string is to be the first character in the part to be returned.

<nexp$_2$> restricts the number of characters to be returned. This information is optional. If omitted, all characters from the start position specified by <nexp$_1$> to the end of the string will be returned.

If the value of <nexp$_1$> is less than or equal to zero, then Error 44, "Parameter out of range" will occur.

If the value of <nexp$_2$> is less than zero, then Error 44, "Parameter out of range" will occur.

If the value of <nexp$_1$> exceeds the length of the original string, an empty string will be returned, but no error condition will occur.

If the value of <nexp$_1$> does not exceed the length of the original string, but the sum of <nexp$_1$> and <nexp$_2$> exceeds the length of the original string, the remainder of the original string will be returned.

Examples

```
10   A$=MID$("INTERMEC PRINTERS",6,3)
20   PRINT A$
RUN
```

yields:

```
MEC
```

```
10   A$="INTERMEC PRINTERS"
20   B%=10
30   C%=7
40   D$=MID$(A$,B%,C%)
50   PRINT D$
RUN
```

yields:

```
PRINTER
```

MKDIR

Purpose

Statement for creating a directory.

Syntax

MKDIR<sexp>

<sexp>	specifies the directory to be created.

Remarks

<sexp> can end with a slash (/) character, but it is not necessary. Only the device `/c` (or `"c:"`) supports creating directories.

Example

```
NEW
MKDIR "DIR1"
SAVE "DIR1/PROGRAM.PRG
FILES "/c/DIR1"
```

yields:

```
FILES on /c/DIR1
PROGRAM.PRG                  2
2220032 bytes free   2 bytes used
```

NAME DATE$

Purpose

Statement for formatting the month parameter in return strings of DATE$("F") and DATEADD$(...,"F").

Syntax

NAME DATE$ <nexp>, <sexp>

<nexp>	is the month number (1-12).
<sexp>	is the desired name of the month.

Remarks

This statement allows you to assign names to the different months in any form and language you like. The names will be returned instead of the corresponding numbers in connection with DATE$("F") and DATEADD$("F") instructions, provided that a FORMAT DATE$ statement has been executed.

The number of characters assigned to represent months in the FORMAT DATE$ statement decides how much of the names, as specified in the NAME DATE$ statement, will be returned. The names will be truncated at the left side. For example:

```
FORMAT DATE$ "YY.MMM:DD"
NAME DATE$ 1,"JANUARY"
PRINT DATE$("F")
```

yields for example:

```
03.ARY.06
```

Usually, it is best to restrict the month parameter in the FORMAT DATE$ statement to 2 or 3 characters (MM or MMM) and enter the names of the months in the NAME DATE$ statement accordingly.

Example

This example shows how to make the printer return dates in accordance with British standard:

```
10   DATE$="030115"
20   NAME DATE$ 1, "JAN"
30   NAME DATE$ 2, "FEB"
40   NAME DATE$ 3, "MAR"
50   NAME DATE$ 4, "APR"
60   NAME DATE$ 5, "MAY"
70   NAME DATE$ 6, "JUN"
80   NAME DATE$ 7, "JUL"
. . . . .
140  FORMAT DATE$ "MMM DD, YYYY"
150  PRINT DATE$("F")
RUN
```

yields:

```
JAN 15, 2003
```

NAME WEEKDAY$

Purpose

Statement for formatting the day parameter in return strings of WEEKDAY$.

Syntax

NAME WEEKDAY$ <nexp>, <sexp>

<nexp>	is the number of the weekday according to the WEEKDAY$ function syntax (Monday = 1... Sunday = 7).
<sexp>	is the desired name of the weekday. Default: Full English name in lowercase characters, that is Monday, Tuesday, etc.

Remarks

This statement allows you to assign names to the different weekdays in any form and language you like. The names will be returned instead of the corresponding numbers in connection with WEEKDAY$ function.

Example

This example shows how to make the printer return the name of the weekday as an English 3-letter abbreviation:

```
10   FORMAT DATE$ ", MM/DD/YY"
20   DATE$="031201"
30   NAME WEEKDAY$ 1, "Mon"
40   NAME WEEKDAY$ 2, "Tue"
50   NAME WEEKDAY$ 3, "Wed"
60   NAME WEEKDAY$ 4, "Thu"
70   NAME WEEKDAY$ 5, "Fri"
80   NAME WEEKDAY$ 6, "Sat"
90   NAME WEEKDAY$ 7, "Sun"
100  PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN
```

yields:

```
Fri, 12/01/03
```

NASC

Purpose

Statement for selecting a single-byte character set.

Syntax

NASC<nexp>

<nexp> is the reference number of a character set:

1	=	Roman 8 (default)
33	=	French
34	=	Spanish
39	=	Italian
44	=	English (UK)
46	=	Swedish
47	=	Norwegian
49	=	German
81	=	Japanese Latin (romají)
351	=	Portuguese
-1	=	PCMAP
- 2	=	ANSI (same as 1252)
850	=	MS-DOS Latin 1
851	=	MS-DOS Greek 1
852	=	MS-DOS Latin 2
855	=	MS-DOS Cyrillic
857	=	MS-DOS Turkish
1250	=	Windows Latin 2 (Central Europe)
1251	=	Windows Cyrillic (Slavic)
1252	=	Windows Latin 1 (ANSI, same as -2)
1253	=	Windows Greek
1254	=	Windows Latin 5 (Turkish)
1257	=	Windows Baltic Rim

Remarks

Please refer to Chapter 4 for complete character set tables.

By default, after processing of possible MAP statements, the Intermec Fingerprint firmware will print and, when applicable, display all characters according to the Roman 8 character set. However, the Intermec Fingerprint firmware contains a number of other character sets, which allows you to print and display such characters that are characteristic for a number of countries or language areas, or to adapt the printer for the operating system of the host.

That implies that a certain ASCII code received by the printer may result in a different character is printed or displayed depending on which character set has been selected.

If none of the character sets available contains the desired character(s), use a MAP statement to reMAP the character set that comes closest to your needs. Note that MAP statements are processed before NASC statements.

NASC, cont.

A NASC statement will have the following consequences:

Text printing
Text on labels etc. will be printed according to the selected character set. However, parts of the label, that already has been processed and stored in the print buffer before the NASC statement is executed, will not be affected. This implies that labels may be multi-lingual.

LCD Display
New messages in the display will be affected by a NASC statement. However, a message that is already displayed will not be updated automatically. The display is, for all practical reasons, able to show all printable characters. In the Setup Mode, all characters are mapped according to US-ASCII standard.

Communication
Data transmitted via any of the communication channels will not be affected as the data is defined as ASCII values, not as alphanumeric characters. The active character set of the receiving unit will decide the graphic presentation of the input data, for example the screen of the host.

Bar Code Printing
The pattern of the bars reflects the ASCII values of the input data and is not affected by a NASC statement. The bar code interpretation (the human readable characters below the bar pattern) is affected by a NASC statement. However, the interpretation of bar codes, that have been processed and are stored in the print buffer, will not be affected.

Example

This example selects the Italian character set:

```
10    NASC 39
```

NASCD

Purpose

Statement for selecting a double-byte character set according to the Unicode standard.

Syntax

NASCD <sexp>

<sexp>	is the name of the character set.
Default:	"" (disables double-byte interpretation).

Remarks

When a double-byte character set has been selected, the firmware will usually treat all characters from ASCII 161 dec. to ASCII 254 dec (ASCII A1-FE hex) as the first part of a two-byte character. Next character byte received will specify the second part. However, the selected Unicode double-byte character set may specify some other ASCII value as the breaking point between single and double byte character sets.

There are various ways to produce double-byte characters from the keyboard of the computer. By selecting the proper character set using a NASCD statement, the typed-in ASCII values will be translated to the corresponding Unicode values, so the desired glyph will be printed.

Double-byte fonts and character set tables are available from Intermec on special request.

Example

The following text contains both single- and double-byte fonts. The double-byte font and its character set are stored in a Font Install Card:

```
10   NASC 46
20   FONT "Swiss 721 BT", 24, 10
30   FONTD "Chinese"
40   NASCD "rom:BIG5.NCD"
50   PRTXT CHR$(65);CHR$(161);CHR$(162)
60   PRINTFEED
RUN
```

This program yields a printed text line that starts with the Latin character A (ASCII 65 dec.) followed by the Chinese font that corresponds to the address 161+162 dec. in the character set “BIG5.NCD.”

NEW

Purpose

Statement for clearing the printer's working memory in order to allow a new program to be created.

Syntax

NEW

Remarks

The NEW statement will delete the program currently residing in the printer's working memory, close all files, and clear all variables and break-points. If the current program has not been saved (see SAVE statement), it will be lost and cannot be restored.

In the Intermec Direct Protocol, all counters will be removed when a NEW statement is executed.

Note that clearing the printer's working memory does not imply that the host screen will be cleared too. The lines of the previous program will remain on the screen until gradually being replaced by new lines.

Example

```
NEW
```

NORIMAGE (NI)

Purpose

Statement for returning to normal printing after an INVIMAGE statement has been issued.

Syntax

NORIMAGE|NI

Remarks

Normal image is the default type of printing and means that text and images will be printed in black-on-white.

Using an INVIMAGE statement, the printing of text and images can be inversed. Such inverse printing will be discontinued for all PRTXT and PRIMAGE statements that follows the encounter of a NORIMAGE statement.

Example

In this example, the first line is printed in inversed fashion and the second line in the normal fashion:

```
10   PRPOS 30,300
20   ALIGN 4
30   INVIMAGE
40   FONT "Swiss 721 BT"
50   PRTXT "INVERSE PRINTING"
60   PRPOS 30, 200
70   NORIMAGE
80   PRTXT "NORMAL PRINTING"
90   PRINTFEED
RUN
```

ON BREAK GOSUB

Purpose

Statement for branching to a subroutine, when break interrupt instruction is received.

Syntax

ON↔BREAK<nexp>GOSUB<ncon>|<line label>

<nexp>	is one of the following communication channels: 0 = "console:" 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:"
<ncon>|<line label>	is the number or label of the program line to be branched to.

Remarks

This statement is closely related BREAK and BREAK ON/OFF. When break interrupt is enabled (see BREAK ON) and the operator issues a break interrupt instruction (see BREAK), the execution of the currently running program will be interrupted and branched to a specified line in a subroutine.

Examples

In this example, the printer emits a special signal when a break interrupt is issued from the printer’s keyboard:

```
10   ON BREAK 0 GOSUB 1000
20   GOTO 20
. . . . .
. . . . .
1000 FOR A%=1 TO 3
1010 SOUND 440,50
1020 SOUND 349,50
1030 NEXT A%
1040 END
```

The same example without line numbers will look like this:

```
IMMEDIATE OFF
ON BREAK 0 GOSUB QQQ
WWW: GOTO WWW
. . . . .
. . . . .
QQQ: FOR A%=1 TO 3
SOUND 440,50
SOUND 349,50
NEXT A%
END
IMMEDIATE ON
```

ON COMSET GOSUB

Purpose

Statement for branching to a subroutine, when the background reception of data on the specified communication channel is interrupted.

Syntax

ON↔COMSET<nexp$_1$>GOSUB<nexp$_2$>|<line label>

<nexp$_1$>	is one of the following communication channels: 0 = "console:" 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 6 = "usb1:"
<nexp$_2$>/<line label>	is number or label of the program line to be branched to.

Remarks

This statement is closely related to COMSET, COMSTAT, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. It is used to branch to a subroutine when one of the following conditions occur:
- End character is received.
- Attention string received.
- Max. number of characters received.

These three parameters are set for the specified communication channel by a COMSET statement.

Examples

In this example, the program branches to a subroutine for reading the buffer of the communication channel:

```
1     REM Exit program with #STOP&
10    COMSET1,"#","&","ZYX","=",50
20    ON COMSET 1 GOSUB 2000
30    COMSET 1 ON
40    IF A$ <> "STOP" THEN GOTO 40
50    COMSET 1 OFF
.....
.....
1000  END
2000  A$= COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```

ON COMSET GOSUB, cont.

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
REM Exit program with #STOP&
COMSET1,"#","&","ZYX","=",50
ON COMSET 1 GOSUB QQQ
COMSET 1 ON
WWW: IF A$ <> "STOP" THEN GOTO WWW
COMSET 1 OFF
.....
.....
END
QQQ: A$=COMBUF$(1)
PRINT A$
COMSET 1 ON
RETURN
IMMEDIATE ON
```

ON ERROR GOTO

Purpose

Statement for branching to an error-handling subroutine when an error occurs.

Syntax

ON↔ERROR↔GOTO<ncon>|<line label>

<ncon>	is the number or label of the line to which the program should branch when an error condition occurs.

Remarks

If any kind of error condition occurs after this statement has been encountered, the standard error-trapping routine will be ignored and the program will branch to the specified line, which should be the first line in an error-handling subroutine.

If the line number is 0, the standard error-trapping routine will be enabled and no error-branching within the current program will be executed.

Examples

If you try to run this example with the printhead raised (or if any other error occurs), a warning signal will sound and the error LED will be lighted.

```
10   LED 0 ON:LED 1 OFF
20   ON ERROR GOTO 1000
30   FONT "Swiss 721 BT"
40   PRTXT "HELLO"
50   PRINTFEED
60   END
. . . . .
1000 LED 0 OFF:LED 1 ON
1010 FOR A%=1 TO 3
1020 SOUND 440,50
1030 SOUND 359,50
1040 NEXT A%
1050 RESUME NEXT
```

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
LED 0 ON:LED 1 OFF
ON ERROR GOTO QQQ
FONT "Swiss 721 BT"
PRTXT "HELLO"
PRINTFEED
END
. . . . .
QQQ: LED 0 OFF:LED 1 ON
FOR A%=1 TO 3
SOUND 440,50
SOUND 359,50
NEXT A%
RESUME NEXT
IMMEDIATE ON
```

ON GOSUB

Purpose

Statement for conditional branching to one of several subroutines.

Syntax

ON<nexp>GOSUB<ncon>|<line label>[,<ncon>|<line label>...]

<nexp>	is a numeric expression that determines which line the program should branch to.
<ncon>/<line label>	is the number or label of the line, or list of lines, to which the program should branch.

Remarks

This statement is closely related to the ON GOTO statement. The numeric expression may result in any positive value. The expression is truncated to an integer value before the statement is executed. If the resulting value is negative, 0, or larger than the number of subroutines, the statement will be ignored.

The value of the numeric expression determines which of the subroutines the program should branch to. For example, if the the value of the numeric expression is 2, the program will branch to the second subroutine in the list.

Examples

In this example, different texts will be printed on the screen depending on which of the keys 1-3 you press on the keyboard of the host.

```
10    INPUT "PRESS KEY 1-3 ", A%
20    ON A% GOSUB 1000,2000,3000
30    END
1000  PRINT "You have pressed key 1"
1010  RETURN
2000  PRINT "You have pressed key 2"
2010  RETURN
3000  PRINT "You have pressed key 3"
3010  RETURN
```

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
INPUT "PRESS KEY 1-3 ", A%
ON A% GOSUB QQQ,WWW,ZZZ
END
QQQ: PRINT "You have pressed key 1"
RETURN
WWW: PRINT "You have pressed key 2"
RETURN
ZZZ: PRINT "You have pressed key 3"
RETURN
IMMEDIATE ON
```

ON GOTO

Purpose

Statement for conditional branching to one of several lines.

Syntax

ON<nexp>GOTO<ncon>|<line label>[,<ncon>|<line label>...]

<nexp>	is a numeric expression that determines which line the program should branch to.
<ncon>/<line label>	is the number or label of the line, or list of lines, to which the program should branch.

Remarks

This statement is closely related to the ON GOSUB statement. The nu1meric expression may result in any positive value. The expression is truncated to an integer value before the statement is executed. If the resulting value is negative, 0, or larger than the number of lines, the statement will be ignored.

The value of the numeric expression determines which of the lines the program should branch to. For example, if the the value of the numeric expression is 2, the program will branch to the second line in the list.

Examples

In this example, different texts will be printed on the screen depending on which of the keys 1-3 you press on the keyboard of the host.

```
10   INPUT "PRESS KEY 1-3 ", A%
20   ON A% GOTO 1000,2000,3000
30   END
1000 PRINT "You have pressed key 1"
1010 GOTO 30
2000 PRINT "You have pressed key 2"
2010 GOTO 30
3000 PRINT "You have pressed key 3"
3010 GOTO 30
```

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
INPUT "PRESS KEY 1-3 ", A%
ON A% GOSUB QQQ,WWW,ZZZ
YYY: END
QQQ: PRINT "You have pressed key 1"
GOTO YYY
WWW: PRINT "You have pressed key 2"
GOTO YYY
ZZZ: PRINT "You have pressed key 3"
GOTO YYY
IMMEDIATE ON
```

ON HTTP GOTO

Purpose

Statement for branching to a subroutine when a request for an application CGI is received.

Syntax

ON↔HTTP↔GOTO<ncon>|<line label>

<ncon>/<line label>	is the number or label of the line to which the program will branch when the CGI request is received.

Remarks

This statement is used in connection with EasyLAN 100i and defines a Fingerprint subroutine that handles the CGI-request. Setting the handler's line number or line label to 0 disables the handler.

When a request for an application CGI is received, the current execution point will be pushed on to the stack and then the execution will commence in the handler with stdin and stdout redirected from/to the Web browser.

Related instruction: RESUME HTTP.

ON KEY GOSUB

Purpose

Statement for branching to a subroutine when a specified key on the printer's front panel is activated.

Syntax

ON↔KEY(<nexp>)GOSUB<ncon>|<line label>

<nexp>	is the id. number of one of the keys on the printer's front panel (see illustration below).
<ncon>/<line label>	is the number or label of the line to which the program will branch when the specified key is pressed down.

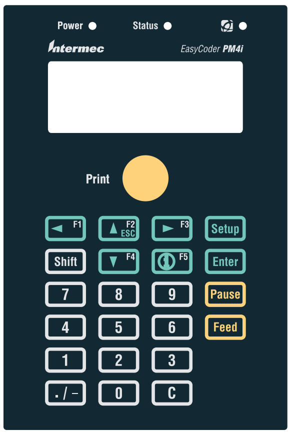
Remarks

All Intermec Fingerprint-compatible printer models are fitted with a membrane-switch keyboard. Each key can be enabled individually using its id. number in a KEY ON statement. Then the key can be assigned, alone or in combination with the <Shift> key, to make the program branch to a subroutine using an ON KEY... GOSUB statement. The <Shift> key adds 100 to the unshifted id. number of each key, as illustrated below.

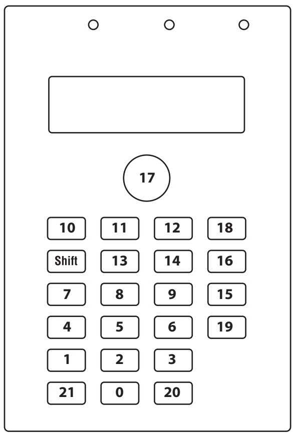
Note the difference between the id. numbers of the keys and the ASCII values they are able to produce (see for example BREAK).

Note that BREAK takes precedence over any ON KEY statement, provided that break interrupt is not disabled for the "console:" by a BREAK 0 OFF statement.

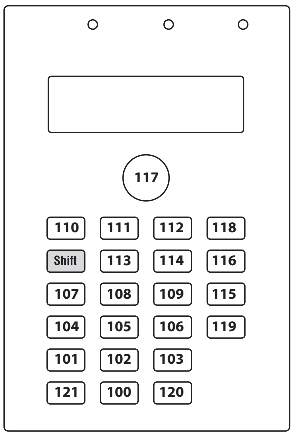
EasyCoder PM4i



Actual keyboard appearance



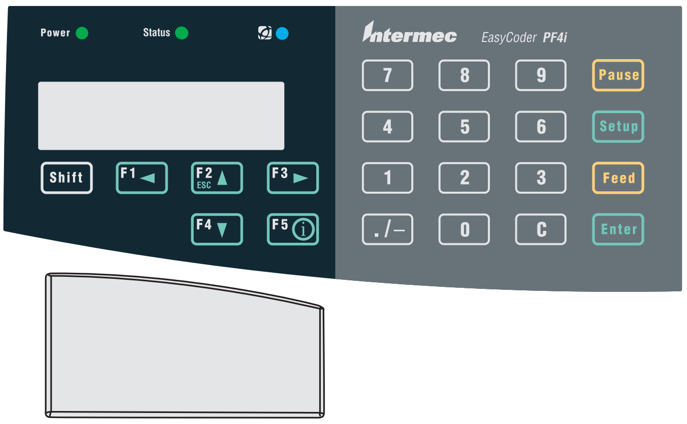
Unshifted keys; id. numbers



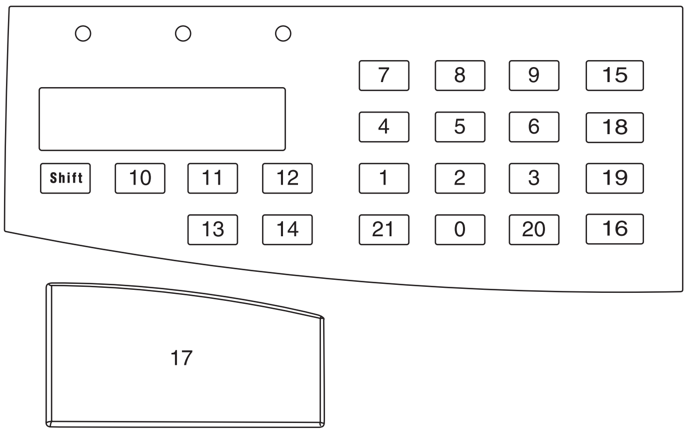
Shifted keys; id. numbers

ON KEY GOSUB, cont.

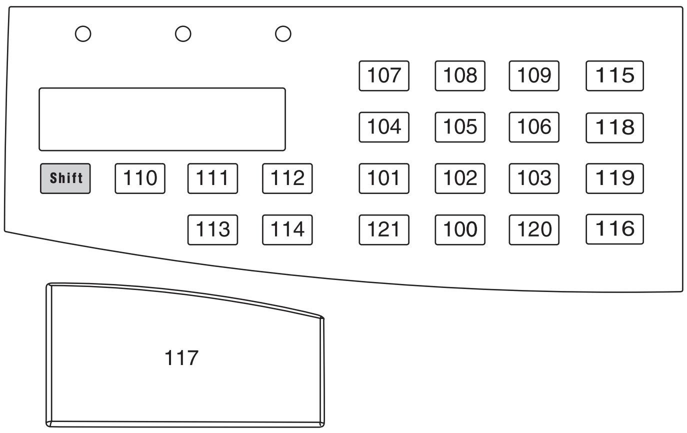
EasyCoder PF-series



Actual keyboard appearance



Unshifted keys id. numbers



Shifted keys id. numbers

ON KEY GOSUB, cont.

Examples

This example illustrates how activating the ◄/F1 key (id. No. 10) will make the program branch to a subroutine, which contains the PRINTFEED statement. Note line 30 where the execution will wait for the key to be pressed.

```
10   ON KEY (10) GOSUB 1000
20   KEY (10) ON
30   GOTO 30
.....
.....
.....
1000 FONT "Swiss 721 BT"
1010 PRPOS 30,100
1020 PRTXT "HELLO"
1030 PRINTFEED
1040 END
RUN
```

The same example can be written without line numbers this way:

```
IMMEDIATE OFF
ON KEY (10) GOSUB QQQ
KEY (10) ON
WWW: GOTO WWW
.....
.....
.....
QQQ: FONT "Swiss 721 BT"
PRPOS 30,100
PRTXT "HELLO"
PRINTFEED
END
IMMEDIATE ON
RUN
```

ON/OFF LINE

Purpose

Statement controlling the SELECT signal on the "centronics:" communication channel.

Syntax

ON|OFF↔LINE<nexp>

<nexp>	specifies the communication channel: 4 = "centronics:" 6 = "usb1:"

Remarks

Pin 13 in the Centronics/IEEE 1284 interface connector contains the SELECT signal:

- ON LINE 4 sets the SELECT signal high.
- OFF LINE 4 sets the SELECT signal low.

If no ON/OFF LINE statement is issued, the SELECT signal will be high, that is the Centronics channel will be ON LINE.

ON LINE/OFF LINE for the serial channel "usb1:" is implemented according to USB Device Class for Printing Devices v1.09, January 2000.

Example

In this example, the "centronics:" communication channel is disabled, while a new setup is performed on the printer by means of a setup file, and then enabled:

```
10   OFF LINE 4
20   SETUP "New Setup.SYS"
30   ON LINE 4
. . . . .
. . . . .
. . . . .
```

OPEN

Purpose

Statement for opening a file or device—or creating a new file—for input, output, or append, allocating a buffer, and specifying the mode of access.

Syntax

OPEN<sexp>[FOR$_{\leftrightarrow}$<INPUT|OUTPUT|APPEND>$_{\leftrightarrow}$]AS [#]<nexp$_1$> [LEN=<nexp$_2$>]

<sexp>	is the file or device to be opened, of the file to becreated. File names must not contain any colon character (:).
#	indicates that whatever follows is a number. Optional.
<nexp$_1$>	is a designation number for the OPENed file or device.
<nexp$_2$>	is, optionally, the length of the record in bytes (default 128 bytes).

Remarks

An OPEN statement must be executed before a file or device can be used for input, output, and/or append. A maximum of 25 files and/or devices can be open at the same time.

Sequential Access Mode
The access mode can optionally be specified as sequential INPUT, OUTPUT, or APPEND:

INPUT	Sequential input from the file/device, replacing existing data. Existing files/devices only.
OUTPUT	Sequential output to the file/device, replacing existing data.
APPEND	Sequential output to the file/device, where new data will be appended without replacing existing data.

Random Access Mode
If no access mode is specified in the statement, the file/device is opened for both input and output (RANDOM access mode). FIELD, LSET, RSET, PUT, and GET can only be used on records in files OPENed in the RANDOM access mode.

Please refer to the DEVICES statement for information on which devices can be opened for the different modes of access.

Lists of the files stored in the various parts of your printer's memory can be obtained by the use of the FILES statements.

Electronic Keys
Each key circuit may contain a number of "key items." There are two types of key items:

• Lock	(device "lock:")
• Storage	(device "storage:")

Each key item has a file name consisting of max. 4 characters, usually appended by a password. The password consists of a delimiter character (?) indicating the password followed by the actual password (max. 4 characters). Failure to include the correct password (if such is required) in the file name will result in an error.

OPEN, cont.

Examples

Allow sequential output to the printer's display using the OPEN statement this way:

```
10    OPEN "console:" FOR OUTPUT AS #1
20    PRINT#1:PRINT#1
30    PRINT#1, "GONE TO LUNCH"
40    PRINT#1, "BACK SOON";
RUN
```

The text will appear on the printer's display as:

```
GONE TO LUNCH
BACK SOON
```

Open the file "PRICELIST" for random access with the reference number #8 and a record length of 254 bytes:

```
10    OPEN "PRICELIST" AS #8 LEN=254
```

Open the file "ADDRESSES" for sequential input with the reference number #4 and a record length of 128 bytes.

```
10    OPEN "ADDRESSES" FOR INPUT AS #4
```

This example shows how a few lines can be added to a program to make it possible to unlock it using an electronic key:

```
10    OPEN "lock:LCK1?PAS1" FOR INPUT AS #1
20    INPUT#1, A$
30    IF A$ AND 1 <>1 THEN GOTO 90000
.....
.....
.....
80000 CLOSE #1
90000 PRINT "Access to program denied!"
90010 END
```

OPTIMIZE BATCH ON/OFF

Purpose

Statement for enabling/disabling optimizing for batch printing.

Syntax

OPTIMIZE "BATCH"↔ON|OFF

ON|OFF	enables/disables optimizing respectively.
Default:	Disabled (OFF)

Remarks

This facility is intended to speed up batch printing, which means the uninterrupted printing of large numbers of identical or very similar labels. OPTIMIZE BATCH is not recommended for the printing of labels with frequently varying content.

The program execution will not wait for the printing of the label to be completed, but proceeds executing next label image into the other of the two image buffers as soon as possible.

By default, OPTIMIZE BATCH is disabled (OFF). However, if the following conditions are all fulfilled, the OPTIMIZE BATCH is automatically enabled (ON):
1. A value larger than 1 has been entered for the PRINTFEED statement.
2. LTS& OFF (default)
3. CUT OFF (default)

Examples

Run these two examples and watch the differences in the printer's performance:

```
10      OPTIMIZE "BATCH" ON
20      FOR I%=1 TO 10
30      PRTXT I%
40      PRINT "Before printfeed"
50      PRINTFEED
60      PRINT "After printfeed"
70      NEXT
RUN
```

```
10      OPTIMIZE "BATCH" OFF
20      FOR I%=1 TO 10
30      PRTXT I%
40      PRINT "Before printfeed"
50      PRINTFEED
60      PRINT "After printfeed"
70      NEXT
RUN
```

PORTIN

Purpose

Function reading the status of a port on a Serial/Industrial Interface Board.

Syntax

PORTIN(<nexp>)

<nexp>	is the number of the port to be read:	
	IN ports (optical):	101-108 (301-308)
	OUT ports (relay):	201-204 (401-404)
	OUT ports (optical):	221-228 (421-428)

Remarks

This function works with the Serial/Industrial Interface Board and is able to read the status of 8 IN ports with optocouplers, 8 OUT ports with optocouplers, and 4 OUT ports with relays. For information on how to set the OUT ports, please refer to the PORTOUT statement.

A current can be lead through an optocoupler in each IN port:
- If the current is on, the PORTIN function returns the value -1 (true).
- If the current is off, the PORTIN function returns the value 0 (false).

This feature is intended to allow the execution of the Intermec Fingerprint to be controlled by various types of external sensors or non-digital switches.

The status of the OUT ports, as set by PORTOUT statements, can also be read by PORTIN functions.

Some printers, like EasyCoder PM4i can carry two Serial/Industrial Interface boards. In this case, the ports on the inner board (that is, the board closest to the CPU board) are specified by the low numbers (101-108, 201-204, and 221-228) while the ports on the outer board are specified by the high numbers (301-308, 401-404, and 421-428).

Please refer to the documentation of the Serial/Industrial Interface Board for more information.

Example

The status of IN port 101 on a Serial/Industrial Interface Board decides when a label is to be printed. The printing will be held until the current is switched off:

```
10   FONT "Swiss 721 BT"
20   PRTXT "POWER IS OFF"
30   IF PORTIN (101) THEN GOTO 30
40   PRINTFEED
50   END
```

PORTOUT ON/OFF

Purpose

Statement for setting one of four relay port or one of eight optical ports on a Serial/Industrial Interface Board to either on or off.

Syntax

PORTOUT (<nexp>) ON|OFF

<nexp>	is the number of the port to be set:	
	OUT ports (relay):	201-204 (401-404)
	OUT ports (optical):	221-228 (421-428)

Remarks

This statement works with the Serial/Industrial Interface Board and is able to control 8 IN ports with optocouplers, 8 OUT ports with optocouplers, and 4 OUT ports with relays. For information on how to read the status of the various ports, please refer to the PORTIN function.

This feature is intended to allow the execution of the Intermec Fingerprint program to control various external units like gates, lamps, or conveyor belts.

Some printers, like EasyCoder PM4i can carry two Serial/Industrial Interface boards. In this case, the ports on the inner board (that is, the board closest to the CPU board) are specified by the low numbers (201-204 and 221-228) while the ports on the outer board are specified by the high numbers (401-404 and 421-428).

Please refer to the documentation of the Serial/Industrial Interface Board for more information.

Example

The relay of OUT port 201 on a Serial/Industrial Interface Board is Opened and then Closed like this:

```
. . . . .
. . . . .
1000 PORTOUT (201) ON
. . . . .
. . . . .
2000 PORTOUT (201) OFF
. . . . .
. . . . .
```

PRBAR (PB)

Purpose

Statement for providing input data to a bar code.

Syntax

PRBAR|PB<<sexp>|<nexp>>

<<sexp>|<nexp>>	is the input data to the bar code generator.

Remarks

The bar code must be defined by BARSET, BARTYPE, BARRATIO, BARHEIGHT, BARMAG, BARFONT, and/or BARFONT ON/OFF statements, or by the corresponding default values.

Make sure that the type of input data (numeric or string) and the number of characters agree with the specification for the selected bar code type. Information on some of the most commonly used bar codes are provided at the end of this manual.

Examples

Two different bar codes, one with numeric input data and one with string input data, can be generated this way. The input data could also have been entered in the form of variables:

```
10   BARFONT "Swiss 721 BT", 8 ON
20   PRPOS 50,400
30   ALIGN 7
40   BARSET "INT2OF5",2,1,3,120
50   PRBAR 45673
60   PRPOS 50,200
70   BARSET "CODE39",3,1,2,100
80   PRBAR "ABC"
90   PRINTFEED
RUN
```

PRBAR (PB), cont.

This example shows how the following information is used to create a Maxicode symbology:

Zip Code:	84170
Zip Code Extension:	1280
Country Code:	840
Class of Service:	001
Message Header:	[)><RS>01
Year:	96
Tracking Number:	1Z12345675
SCAC:	UPSN
UPS Shipper Number:	12345E
Julian Day of Pickup:	089
Shipment ID:	1324567
Package:	1/1
Weight:	10.1
Address Validation:	Y
Ship to Street:	1 Main ST
Ship to City:	PITTSBURGH
Ship to State:	PA

```
10     PRPOS 100,100
20     DIR 1
30     ALIGN 1
40     a$= "84170"+CHR$(10)+"1280"+CHR$(10)+"840"+CHR$(10)+"001"
       +CHR$(10)+"[)>"+CHR$(30)+"01"+CHR$(29)+"96"+"1Z12345675"
       +CHR$(29)+"UPSN"+CHR$(29)+"12345E"+CHR$(29)+"089"+CHR$(29)
       +"1234567"+CHR$(29)+"1/1"+CHR$(29)+"10.1"+CHR$(29)+"Y"
       +CHR$(29)+"1 MAIN ST"
50     b$= CHR$(29)+"PITTSBURGH"+CHR$(29)+"PA"+CHR$(29)+CHR$(30)+
       CHR$(4)+CHR$(10)+"2"+CHR$(10)+"1"+CHR$(10)+"1"
60     BARTYPE "MAXICODE"
70     PRBAR a$;b$
80     PRINTFEED
RUN
```

PRBOX (PX)

Purpose

Statement for creating a box, optionally containing a single text line or a frame of mulitiple hyphenated text lines.

Syntax

PRBOX|PX<nexp$_1$>,<nexp$_2$>,<nexp$_3$>[,<sexp$_1$>[,<nexp$_4$>[,<nexp$_5$>[,<sexp$_2$>[,<sexp$_3$>]]]]]

<nexp$_1$>	is the height of the box in dots (1-6000).
<nexp$_2$>	is the width of the box in dots (1-6000).
<nexp$_3$>	is the line weight in dots (0/1-6000).
<sexp$_1$>	is the framed text to be written inside the box (max. 300 char./line, max. 20 lines). Single-byte fonts only.
<nexp$_4$>	is the horizontal distance between inner edge of the box line and the text frame (-100 to 100 dots). Default: 0.
<nexp$_5$>	is the vertical distance between the inner edge of the box line and text frame and also between each line of text in the frame (-100 to 100 dots). Default: Same value as <nexp$_4$>.
<sexp$_2$>	is a line delimiter (max. 9 characters), which replaces the default delimiter string CHR$(10) or CHR$(13). Each time this delimiter is encountered in the text string (<sexp$_1$>, the rest of the text is wrapped to the next line.
<sexp$_3$>	is a control string for hyphen delimiter and replacement, see Remarks.

Remarks

This statement has two purposes: to create a rectangular white box surrounded by a line with a certain thickness, or to specify a text frame that can contain up to 20 lines of hyphenated text. These two purposes can be combined so a text frame is surrounded by a black box.

Creating a simple box:

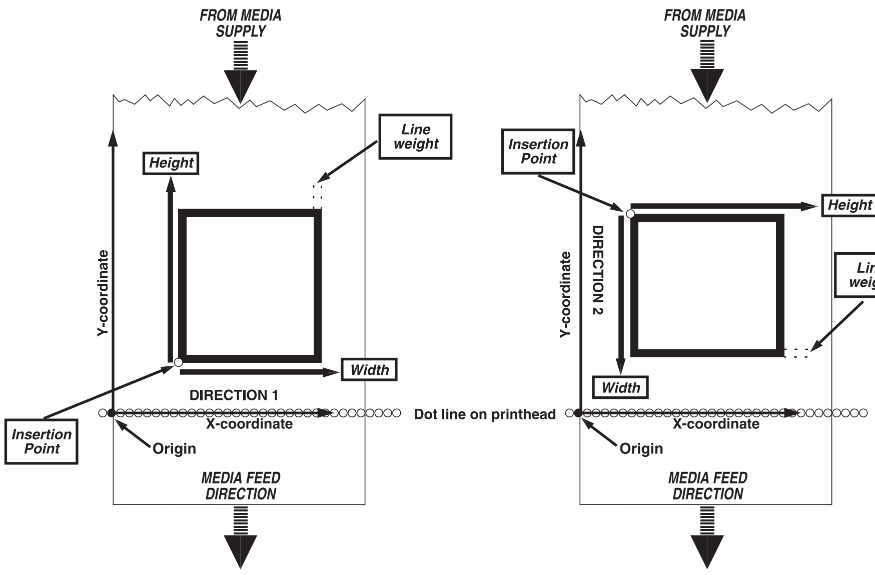
In this case you only need to specify the first three parameters, that is height, width, and line weight (thickness). The box will be drawn with its anchor point (see ALIGN) at the insertion point, as specified by the nearest preceding PRPOS statement. A box can be aligned left, right, or center along its baseline.

The print direction specifies how the box is rotated in relation to its anchor point.

The line weight (thickness) grows inward from the anchor point. The heavier the line, the less white area inside the box. Thus, it is possible to create a black area using a box with very heavy lines. For a simple box without any text field, the line weight must be >0. The white area inside a box can be used for printing. Boxes, lines, and text may cross (also see XORMODE ON/OFF).

PRBOX, cont.

The illustration below shows how the height and width of the box are defined for different print directions.



Creating a multiline text field

The PRBOX statement can also be used to create an area in which a field of wrapped and hyphenated text can be printed. As opposed to the PRTXT statement, there is no need to specify each line of text separately. The text field can be framed by the box (line weight > 0), or the box can be invisible (line weight = 0). The maximum number of characters on each line is 300 and the maximum number of lines is 20.

The position of the text frame inside the box is affected by the direction (see DIR statement), the alignment (see ALIGN statement), and by two parameters in the PRBOX statement (<nexp$_4$> and <nexp$_5$>.

The direction rotates the box with its text field around the anchor point as specified by the alignment. The alignment specifies the anchor point of the box itself as left-, right-, or center-aligned (see ALIGN), and at the the same time also decides how the field will be aligned inside the box (9 possible positions) and if the text lines will be left, right, or center justified.

In the following description, horizontal and vertical should be understood in relation to how the text is printed. (That means that in directions 2&4, horizontal and vertical have opposite meanings than in directions 1&3).

The horizontal distance between the inner edge of the box line and the borders of the text field is specified by <nexp$_4$>:

- In case of ALIGN 1, 4, or 7, it decides the distance between the inner edge of the left side box line and the left-hand edge of the text field.
- In case of ALIGN 3, 6, or 9, it decides the distance between the inner edge of the right side box line and the right-hand edge of the text field.
- In case of ALIGN 2, 5, or 8, this parameter has no consequence.

PRBOX, cont.

The vertical distance between the inner edge of the box line and the borders of the text field as well as the empty vertical space between the character cells of two adjacent lines (line spacing) is specified by <nexp$_5$>:

- In case of ALIGN 1, 2, or 3, it decides the distance between the inner edge of the bottom box line and the bottom edge of the text field as well as line spacing.
- In case of ALIGN 7, 8, or 9, it decides the distance between the inner edge of the right side box line and the right-hand edge of the text field as well as line spacing.
- In case of ALIGN 4, 5, or 6, this parameter only decides line spacing.

See the illustration on next page for examples of how the alignment affects the location of multi-line text.

If the text in <sexp$_1$> is entered as a continuous string of characters without any spaces, linefeeds, or carriage returns, the text will wrap to the next line when there is no room left for any more characters on a line.

If any combination of a carriage return (CR = ASCII 13 dec,) and a linefeed (LF = ASCII 10 dec.) is encountered, the remaining text will be wrapped once to the next line.

Space characters (ASCII 32 dec.) will also initiate a line wrap. If there are more than one space character, the wrapping will be at the last one that fits into the line in question.

You can replace the default line delimiters (CR, LF, and CR/LF) with another line delimiter specified in a string of max. 9 characters (<sexp$_2$>). This delimiter will not be printed, even if it is a printable character. Each time the delimiter is encountered, the text will wrap to a new line.

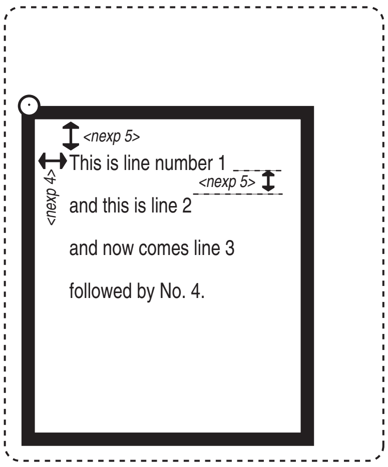
Hyphenation Support
In <sexp$_3$> you can modify the way hyphenation will be performed using a special syntax described later on.

You can put “invisible” hyphen delimiters in the text string at suitable wrap-around positions. The delimiter is by default a hyphen sign (ASCII 45 dec.). However, you can use a string of any characters up to nine characters long instead, but be careful so it will not be confused with the text. If a wrap-around is performed, the corresponding hyphen delimiter will by default be printed as a hyphen sign (ASCII 45 dec.), whereas hyphen delimiters not used for wrap-around will not be printed.

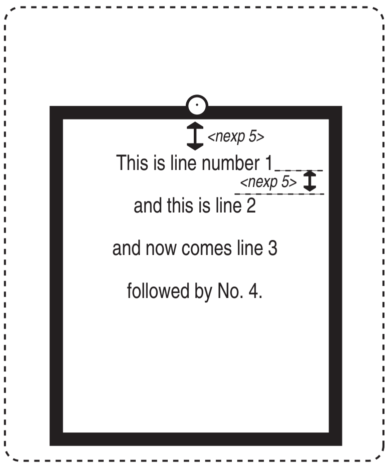
If you for some reason would like to print some other character(s) than hyphens, you can specify a string of hyphen replacement characters. It is possible to use a string up to nine characters long, but the shorter the string the lesser risk that a line will wrap outside the box.

If you have a text string with long words and have not inserted all necessary line delimiters, a line-wrap may occur unexpectedly. You can optionally specify a hyphen delimeters for this case as well. Default: None.

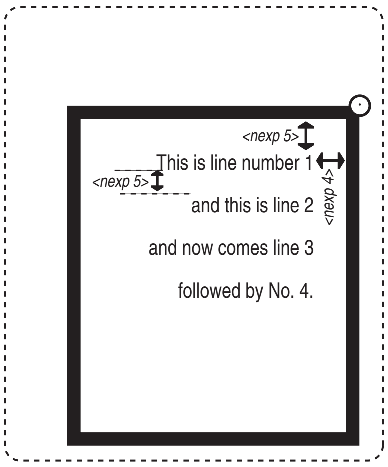
PRBOX, cont.



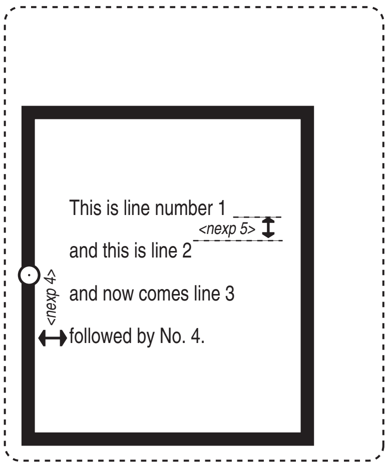
ALIGN 7



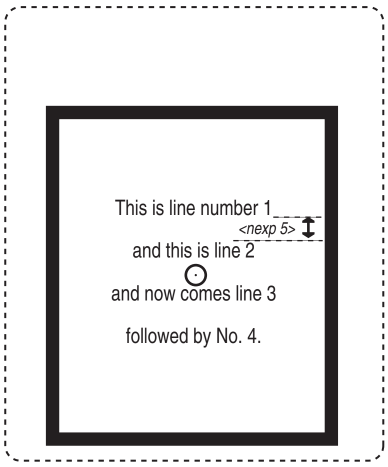
ALIGN 8



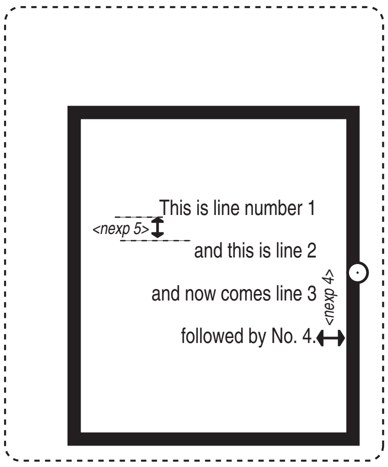
ALIGN 9



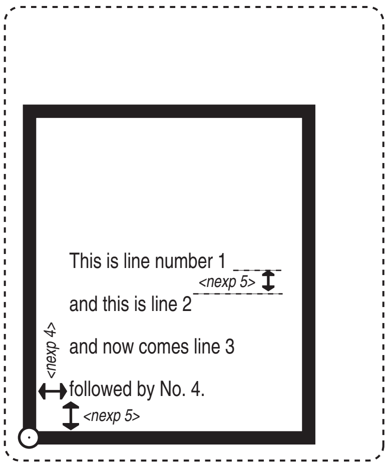
ALIGN 4



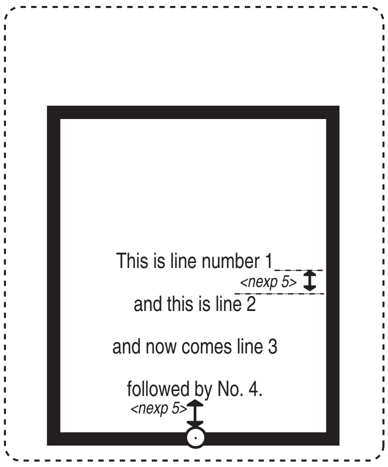
ALIGN 5



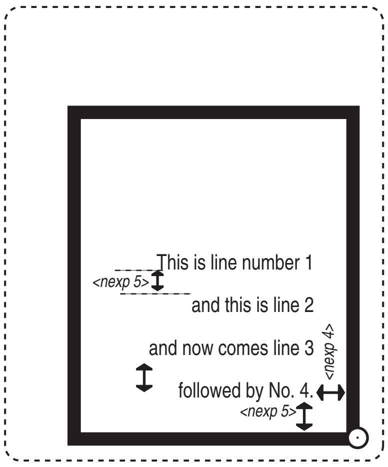
ALIGN 6



ALIGN 1



ALIGN 2



ALIGN 3

PRBOX, cont.

Specify the parameter <$sexp_3$> in PRBOX using the following syntax:

<$sexp_3$>=<$sexp_{3a}$>[space<$sexp_{3b}$>[space<$sexp_{3c}$>]]

<$sexp_{3a}$>	is a soft hyphen delimiter. If the text does not have enough room on one line, the rest of the text will be wrapped from the last space or from the position marked by the soft hyphen delimiter. Exception: Two adjacent soft hyphen delimiters revoke each other. Default: Normal hyphen (-). Max length: 9 characters.
space	is a string delimiter with the value CHR$(32).
<$sexp_{3b}$>	is one or more characters, that will be printed at the end of a line which has been hyphenated according to a hyphen delimiter (see <$sexp_{3a}$>). Default: Normal hyphen (-). Max length: 9 characters (less is preferred).
<$sexp_{3c}$>	is a string of hyphen extension characters, used on single words which are too long to be printed on one line and have no hyphen delimiter specified. The hyphen extension character(s) will be printed at the right end of line and the remainder of the word will be printed on the next line. Default: No character. Max length: 9 characters.

If no <$sexp_3$> is specified, the rule for hyphen delimiter and replacement will be the same as for printing hyphens in text. Two adjacent hyphens will be printed as one.

Examples

This examples draws a rectangle without any text:

```
10   PRPOS 50,50
20   PRBOX 200,400,5
30   PRINTFEED
RUN
```

This program illustrates a multi-line text field with line wrap, where “&S” is the soft hyphen delimiter:

```
10   DIR 1
20   ALIGN 8
30   R$="Hyphen&Sated words will be divid&Sed
     into sylla&Sbles."
40   NL$="NEWLINE"
50   S$="&S&Special Cases and EXTRAORDINARY long
     words."
60   T$=R$+NL$+S$
70   PRPOS 300,300
80   PRBOX 700,500,20,T$,25,1,NL$,"&S - +"
90   PRINTFEED
RUN
```

PRBUF

Purpose

Statement for receiving and printing bitmap image data using the PRBUF protocol.

Syntax

PRBUF<$nexp_1$>[,<$nexp_2$]<new line><image data>

<$nexp_1$>	is the number of bytes of the image in PRBUF protocol.
<$nexp_2$>	is, optionally, a timeout between characters in TICKS (0.01 sec). Default ≈ 12.7 sec./character.
<new line>	is any combination of CR, CR/LF, or LF.
<image data>	is the image according to the PRBUF protocol.

Remarks

This statement is useful for receiving and printing bitmap images from, for example, a Windows printer driver. It is more effective and requires less memory than using a STORE IMAGE...PRIMAGE sequence. The bitmap image is printed directly and is not saved anywhere in the printer's memory after the image buffer has been cleared.

At the PRBUF statement, the printer waits for image data to be received on the standard IN channel. PRBUF only works with binary transfers, that is XON/XOFF must be disabled. You can optionally set a timeout between characters (default 12.7 sec.) When the specified number of bytes according to the PRBUF protocol have been received, the image data are processed directly into the printer's image buffer and printed without requiring any more Fingerprint instructions.

PRBUF does not work if <$nexp_1$> bytes cannot be allocated. If memory is low, it is possible to download the bitmap image in two or more blocks.

The field settings (alignment, clipping, direction, xor mode, inverse image, magnification, x-position, and y-position) are handled by the current protocol, but the basic rule is that x- and y-positions, field clipping, and xor mode are handled and the other attributes are ignored.

If PRPOS x,y, then the real print position will be PRPOS x,y+1.

The PRBUF protocol is decribed in Chapter 3, "Image Transfer."

The <newline> is not part of the statement, but any combination of carriage return (ASCII 13 dec,) and/or linefeed (ASCII 10 dec.) is allowed without interfering with the PRBUF protocol.

Example

This example shows how the printer is instructed to receive and print 1,424 bytes of image data according to the PRBUF protocol:

```
PRBUF 1424 ↵
<binary image data>
```

PRIMAGE (PM)

Purpose

Statement for selecting an image stored in the printer's memory.

Syntax

PRIMAGE|PM<sexp>

<sexp>	is the full name of the desired image including extension.

Remarks

An image is positioned according to the preceding PRPOS, DIR, and ALIGN statements. It can be magnified by means of a MAG statement.

For the best printout quality, create and download a larger version of the image rather than magnifying a smaller one.

All images provided by Intermec have an extension which indicates for which directions the image is intended:

- Extension .1 indicates print directions 1 & 3.
- Extension .2 indicates print directions 2 & 4.

Even if the Intermec Fingerprint firmware does not require such an extension, we strongly recommend you to follow the same convention when creating your own images as to make it easier to select the correct image.

Example

This example illustrates the printing of a label containing an image printed "upside down":

```
10   PRPOS 200,200
20   DIR 3
30   ALIGN 5
40   PRIMAGE "GLOBE.1"
50   PRINTFEED
RUN
```

PRINT (?)

Purpose

Statement for printing data to the standard OUT channel.

Syntax

PRINT|?[<<nexp>|<sexp>>[<,|;><<nexp>|<sexp>>...][;]]

<<nexp>|<sexp>>	are string or numeric expressions, which will be printed to the standard OUT channel.

Remarks

If no expressions are specified after the PRINT statement, it will yield a blank line. If one or more expressions are listed, the expression(s) will be processed and the resulting values will be presented on standard OUT channel (see SETSTDIO statement), for example usually on the screen of the host. The shorthand form of PRINT is a question mark.

Do not confuse the PRINT statement with the PRINTFEED statement.

Each line is divided into zones of 10 character positions each. These zones can be used for positioning the values:

- A comma sign (,) between the expressions causes next value to be printed at the beginning of next zone.
- A semicolon sign (;) between the expressions causes next value to be printed immediately after the last value.
- A plus sign (+) between two string expressions also causes next value to be printed immediately after the last value. (Plus signs cannot be used between numeric expressions.)
- If the list of expressions is terminated by a semicolon, the next PRINT statement will be added on the same line. Otherwise, a carriage return is performed at the end of the line. If the printed line is wider than the screen, the software will automatically wrap to a new line and go on printing.

Printed numbers are always followed by a space character.

Printed negative numbers are preceded by a minus sign.

Example

```
10    LET X%=10
20    LET A$="A"
30    PRINT X%;X%+1,X%+5;X%-25
40    PRINT A$+A$;A$,A$
50    PRINT X%;
60    ? "PIECES"
RUN
```

yields:

```
10 11     15 -15
AAA       A
10 PIECES
```

PRINT KEY ON/OFF

Purpose

Statement for enabling or disabling printing of a label by pressing the Print key.

Syntax

PRINT KEY ON|OFF

Default:	PRINT KEY OFF

Remarks

In the Immediate Mode and in the Intermec Direct Protocol, the <Print> key can be enabled to issue printing commands, corresponding to PRINTFEED statements. This implies that each time the <Print> key is pressed, one single label, ticket, tag, or portion of continuous stock will be printed and fed out.

Note that this statement cannot be entered in the Programming Mode (use KEY ON and ON KEY GOSUB statements instead).

Example

This example shows how the Print key is enabled in the Intermec Direct Protocol and a label is printed (abbreviated instructions are used whenever available):

```
INPUT ON ↵
PRINT KEY ON ↵
PP 100,100 ↵
FT "Swiss 721 BT" ↵
PT "TEST LABEL" ↵
```

[Press the <Print> key]

```
INPUT OFF ↵
```

PRINT#

Purpose

Statement for printing data to a specified OPENed device or sequential file.

Syntax

PRINT#<nexp$_1$>[,<<nexp$_2$>|<sexp$_1$>>[<,|;><<nexp$_3$>|<sexp$_2$>>...][;]]

<nexp$_1$>	is the number assigned to the file or device when it was OPENed.
<<nexp$_{2-n}$>|<sexp1$_{-n}$>>	are the string or numeric expressions, which will be printed to the specified file or device.

Remarks

Expressions can be separated by commas or semicolons according to the same rules as for the PRINT statement. It is important that the expressions are separated properly, so they can be read back when needed, or be presented correctly on the printer's LCD display.

PRINT# can only be used to print to sequential files, not to random files.

When sending data to the printer's display ("console:"), PRINT# will work same way as PRINT does on the standard OUT channel. The display can for example be cleared by sending PRINT#<nexp> twice (see line 20 in the example below).

Example

The display on the printer's keyboard console is able to show two lines with 16 characters each. Before sending any text, the device must be OPENed (line 10) and both lines on the display must be cleared (line 20). Note the trailing semicolon on line 40!

```
10   OPEN "console:" FOR OUTPUT AS #1
20   PRINT# 1:PRINT# 1
30   PRINT# 1,"OUT OF LABELS"
40   PRINT# 1,"PLEASE RELOAD!";
50   CLOSE# 1
RUN
```

Since the last line was appended by a semicolon, there will be no carriage return and the text will appear on both line on the printer's display as:

```
OUT OF LABELS
PLEASE RELOAD!
```

An alternative method is to send all the data to the display in a single PRINT# statement. Character No. 1-16 will be displayed on the upper line and character No. 17-33 will be displayed on the lower line, whereas character No. 17 will be ignored. Note the trailing semicolon on line 30! (The double-headed arrows in line 30 represent space characters.)

```
10   OPEN "console:" FOR OUTPUT AS #1
20   PRINT# 1: PRINT# 1
30   PRINT# 1,"OUT↔OF↔LABELS↔↔↔↔PLEASE↔
     RELOAD!";
40   CLOSE# 1
RUN
```

PRINTFEED (PF)

Purpose

Statement for printing and feeding out one or a specified number of labels, tickets, tags, or portions of strip, according to the printer's setup.

Syntax

PRINTFEED|PF [<nexp$_1$>] | [-1,<nexp$_2$>]

<nexp$_1$>	specifies number of copies to be printed.
-1,<nexp$_2$>	specifies that <nexp$_2$> number of identical copies of the last printed label should be reprinted (cannot be used with Intermec Direct Protocol).

Remarks

Each time a PRINTFEED statement without any appending value is executed, one new label, ticket, tag, or portion of continuous stock will be printed.

The PRINTFEED statement can optionally be appended by a numeric expression, which specifies the number of copies to be printed. In the Intermec Direct Protocol, possible counter, time, and date values will be updated between copies printed using a predefined layout. Note that you must never include any PRINTFEED statements in layouts in the Intermec Direct Protocol.

If the number of copies is >1 and LTS& and CUT are disabled (= LTS& OFF and CUT OFF), the BATCH optimizing strategy is automatically enabled, which corresponds to an OPTIMIZE BATCH ON statement. When theses conditions are no longer fulfilled, BATCH optimizing strategy is automatically disabled, which corresponds to an OPTIMIZE BATCH OFF statement.

It is also possible to reprint a specified number of copies of the last printed label, for example after an out-of-media condition (also see PRSTAT).

The execution of a PRINTFEED statement clears the following statements to their default values:

ALIGN	BARRATIO	INVIMAGE
BARFONT	BARTYPE	MAG
BARFONT ON/OFF	BARSET	PRPOS
BARHEIGHT	DIR	XORMODE ON
BARMAG	FONT	

Fields defined by statements, that have been executed before the PRINT-FEED statement, are not affected. Note that, when using a PRINTFEED in a loop, all formatting parameters are reset to default each time the PRINTFEED statement is executed and must therefore be included inside the loop.

PRINTFEED (PF), cont.

The length of media to be fed out at execution of a PRINTFEED statement is decided by the choice of media type in the printer's setup (label w gaps, ticket w gaps, fix length strip, or var length strip) and globally by the start and stop adjustment setup (positive or negative). Refer to the User's Guide for more information. The length of media to be fed out can be further modified by an additional positive or negative FORMFEED statement, either before or after the PRINTFEED statement.

Examples

Printing a single label with one line of text:

```
10   FONT "Swiss 721 BT"
20   PRTXT "Hello!"
30   PRINTFEED
RUN
```

Printing five identical labels with one line of text:

```
10   FONT "Swiss 721 BT"
20   PRTXT "Hello!"
30   PRINTFEED 5
RUN
```

Printing five labels using a FOR...NEXT loop. Note that formatting parameters are placed inside the loop:

```
10   FOR A%=1 TO 5
20   FONT "Swiss 721 BT"
30   PRPOS 200, 100
40   DIR 3
50   ALIGN 5
60   PRTXT "Hello!"
70   PRINTFEED
80   NEXT A%
RUN
```

Printing of five labels in the Intermec Direct Protocol, illustrating how the TICKS value is updated between labels, provided a predefined layout is used (1 TICK = 0.01 sec):

```
INPUT  ON  ↵
FORMAT INPUT "#","@","&" ↵
LAYOUT INPUT "tmp:LABEL1" ↵
FT "Swiss 721 BT" ↵
PP 100,100 ↵
PT TICKS ↵
PP 100,200 ↵
PT VAR1$ ↵
LAYOUT END ↵
LAYOUT RUN "tmp:LABEL1" ↵
#See how time flies&@ ↵
PF 5 ↵
INPUT OFF ↵
```

PRINTONE

Purpose

Statement for printing characters, specified by their ASCII values, to the standard OUT channel.

Syntax

PRINTONE<nexp>[<,|;><nexp>...][;]

<nexp>	is the ASCII decimal value of a character, which will be printed to the standard OUT channel.

Remarks

When, for some reason, certain characters cannot be produced by the host computer, they can be substituted by the corresponding ASCII decimal values using the PRINTONE statement. The characters will be printed, according to the currently selected character set (see NASC statement), to the standard OUT channel, that is usually to the screen of the host. PRINTONE is very similar to the PRINT statement and the use of commas and semicolons follows the same rules.

Example

```
PRINTONE 80;82;73;67;69;58,36;52;57;46;57;53
```

yields:

```
PRICE:    $49.95
```

PRINTONE#

Purpose

Statement for printing characters specified by their ASCII values to a device or sequential file.

Syntax

PRINTONE#<nexp$_1$>[,<nexp$_2$>[<,|;><nexp$_3$>...][;]]

<nexp$_1$>	is the number assigned to the file or device when it was OPENed.
<nexp$_{2-n}$>	is the ASCII decimal value of the character, which is to be printed to the specified file or device.

Remarks

This statement is useful, when the host for some reason cannot produce certain characters. The ASCII values entered will produce characters according to the currently selected character set, see NASC. The ASCII values can be separated by commas or semicolons according to the same rules as for the PRINT# statement.

PRINTONE# can only be used to print to sequential files, not to random files.

When sending data to the printer's display, PRINTONE# will work in a way similar to PRINT#. The display can be cleared by sending PRINT#<nexp> twice (see line 20 in the example below).

Example

The display on the printer's keyboard console is able to show two lines with 16 characters each. Before sending any text, the device must be OPENed and the display be cleared. Note the trailing semicolon sign on line 40.

```
10   OPEN "console:" FOR OUTPUT AS #1
20   PRINT# 1:PRINT# 1
30   PRINTONE# 1,80;82;69;83;83
40   PRINTONE# 1,69;78;84;69;82;
50   CLOSE #1
RUN
```

Since the last line was appended by a semicolon, there will be no carriage return and the text will appear on both line on the printer's display as:

```
PRESS
ENTER
```

PRLINE (PL)

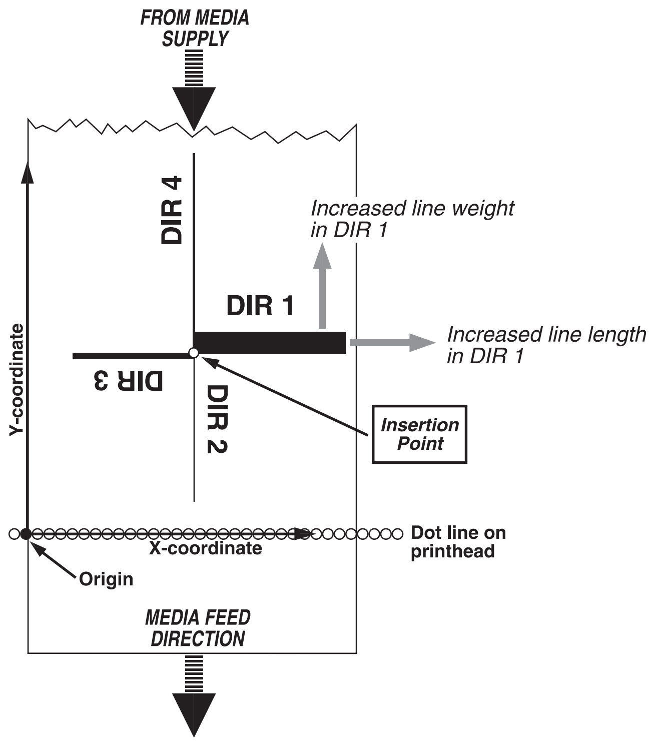
Purpose Statement for creating a line.

Syntax **PRLINE|PL<nexp$_1$>,<nexp$_2$>**

<nexp$_1$>	is the length of the line in dots (max. 6000).
<nexp$_2$>	is the line weight in dots (max. 6000).

Remarks The line will be drawn from the insertion point and away according to the nearest preceding DIR and ALIGN statements (that is the line runs in parallel with any text printed in the selected direction).

A line can be ALIGNed left, right or center. The anchor points are situated at the bottom of the line, which means that with an increasing line weight (thickness), the line will the grow upward in relation to the selected direction. In the illustration below, all lines are aligned left. Lines may cross (see XORMODE ON/OFF statement).



Example This example draws a 2.5 cm (1 inch) long and 10 dots thick line across the media in an 8 dots/mm printer:

```
10   PRPOS 50,100
20   PRLINE 200,10
30   PRINTFEED
RUN
```

PRPOS (PP)

Purpose

Statement for specifying the insertion point for a line of text, a bar code, an image, a box, or a line.

Syntax

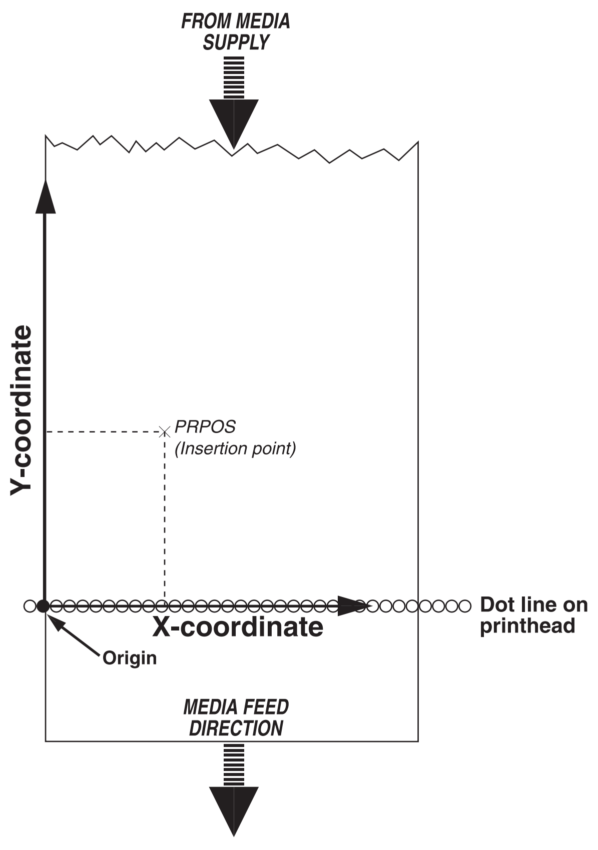
PRPOS|PP<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is the X-coordinate (number of dots from the origin).
<nexp$_2$>	is the Y-coordinate (number of dots from the origin).
Default value:	0,0
Reset to default by:	PRINTFEED execution.

Remarks

When the printer is set up, a “print window” is created. This involves specifying the location of the origin along the X-axis, setting the max. print width along the X-axis from origin, and setting the max. print length along the Y-axis from origin.

The X-coordinate goes across the media path and the Y-coordinate along the media feed direction, as illustrated below. They are set in relation to the origin on the printhead, not in relation to the media. Thus, the position where an object actually will be printed depends on the relation between printhead and media at the moment when the printing starts.



PRPOS (PP), cont.

The insertion point must be selected so the field in question will fit inside the print window. This implies that the print direction, the size of the field including “invisible” parts of for example an image, the alignment, and other formatting instructions must be considered. A field that does not fit entirely inside the print window will cause Error 1003, “Field out of label”, except when a CLIP ON statement is issued.

To find out the present insertion point, use the PRSTAT function.

Examples

Programming and printing a line of text:

```
10    FONT "Swiss 721 BT"
20    PRPOS 30,200
30    PRTXT "HELLO"
40    PRINTFEED
RUN
```

Each text line is normally positioned separately by is own PRPOS statement. If no position is given for a printable statement, it will be printed immediately after the preceding printable statement.

```
10    FONT "Swiss 721 BT"
20    PRPOS 30,200
30    PRTXT "SUMMER"
40    PRTXT "TIME"
50    PRINTFEED
RUN
```

yields a label with the text:

```
SUMMERTIME
```

A program for fixed line-spacing of text may be composed this way (another way is to use the extended PRBOX statement):

```
10    FONT"Swiss 721 BT"
20    X%=30:Y%=500
30    INPUT A$
40    PRPOS X%,Y%
50    PRTXT A$
60    Y%=Y%-50
70    IF Y%>=50 GOTO 30
80    PRINTFEED
90    END
RUN
```

Enter the text for each line after the question mark shown on the screen of the host. The Y-coordinate will be decremented by 50 dots for each new line until it reaches the value 50, which means that ten lines will be printed.

PRSTAT

Purpose

Function returning the printer's current status or, optionally, the current position of the insertion point.

Syntax

PRSTAT[(<nexp>)]

<nexp> = 1	returns the X-position for the insertion point at DIR 1&3.
<nexp> = 2	returns the Y-position for the insertion point at DIR 2&4.
<nexp> = 3	returns the X-position of the corner with the lowest coordinates of the last object.
<nexp> = 4	returns the Y-position of the corner with the lowest coordinates of the last object.
<nexp> = 5	returns the width along the X-axis of the last object.
<nexp> = 6	returns the height along the Y-axis of the last object.
<nexp> = 7	returns the print job identifier
<nexp> = 8	returns the print job state (see table below).
<nexp> = 9	returns the print job error code.
<nexp> = 10	returns the remaining number of copies to be printed in a batch print job.

Remarks

PRSTAT
Returns a numeric expression, which is the sum of the values given by the following conditions, at the moment when the PRSTAT function is executed:

- OK	0
- Printhead lifted	1
- Label not removed (see note)	2
- Printer out of media	4
- Printer out of transfer ribbon (TTR) or ribbon installed (DT)	8
- Printhead voltage too high	16
- Printer is feeding	32

Note: Always returns 0 in printers not fitted with a label taken sensor.

If two error conditions occur at the same time, for example the printhead is lifted and the printer is out of media, the sum will be (1+4) = 5. Every combination of errors will result in a unique sum. You can use it to branch to a subroutine which notifies the operator, interrupts the program or whatever you like. When checking for out-of-media conditions, the use of error codes 1031 "Next label not found" and 1005 "Out of paper" gives more reliable result (multiple checks).

PRSTAT(1) & PRSTAT(2)
The current position of the insertion point in regard of either the X or the Y position can be returned, depending on the selected print direction. This is useful for for example measuring the length of a text or a bar code.

PRSTAT, cont.

PRSTAT(3)-PRSTAT(6)
These functions are used to return the position and size of the last object regardless of RENDER ON/OFF. Their values are not updated by the execution of a PRBUF statement.

PRSTAT(7)-PRSTAT(10)
These functions are used to detect if a print job has been interrupted, so steps can be taken to reprint missing copies (see PRINTFEED).

PRSTAT (7) returns a print job identifier that is automatically assigned to the print job by the firmware.

PRSTAT (8) returns the state of the print job as a numeric expression, which is the sum of the values given by the following conditions:

- Print cycle not set up for printing, perhaps due to out-of-ribbon	0
- The previous print cycle never ended (timeout)	1
- Print cycle has started	2
- All lines successfully printed	4
- Printing truncated (media shorter than print image)	8
- Printhead strobing error or label length exceeded	16
- Ribbon low	32

PRSTAT (8) = 6 or 22 indicates a successfully printed label (in the latter case error “next label not found” may have been detected).

PRSTAT (9) returns the error code (see Chapter 7, “Error Messages”) detected by the print engine during printfeed. It is used together with PRSTAT(8) to determine the error cause when using OPTIMIZE "BATCH" ON.

PRSTAT (10) returns the number of copies that remains to be printed in an interrupted batch print job.

Examples

This examples shows how two error conditions are checked:

```
10   A% = PRSTAT
20   IF A% AND 1 THEN GOSUB 1000
30   IF A% AND 4 THEN GOSUB 1010
40   END
.....
1000 PRINT "Printhead is lifted":RETURN
1010 PRINT "Printer out of media":RETURN
RUN
```

This example illustrates how you can check the length of a text:

```
10   PRPOS 100,100: FONT "Swiss 721 BT"
20   PRTXT "ABCDEFGHIJKLM"
30   PRINT PRSTAT(1)
RUN
```

yields:

```
519
```

PRTXT (PT)

Purpose

Statement for providing the input data for a text field.

Syntax

PRTXT|PT<<nexp>|<sexp>>[;<<nexp>|<sexp>>...][;]

<<nexp>|<sexp>>	specifies one line of text (max. 300 characters)

Remarks

A text field consists of one line of text. The text field must be defined in regard of FONT or FONTD and may be further defined and positioned by DIR, ALIGN, MAG, PRPOS, INVIMAGE, or NORIMAGE statements or their respective default values.

Two or more expressions can be combined to form a text line. They must be separated by semicolons (;) and will be printed adjacently. Plus signs can also be used for the same purpose, but only between string expressions.

String constants must be enclosed by quotation marks, whereas numeric constants or any kind of variables must not.

Examples

Programming and printing a line of text:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   PRTXT "How do you do?"
40   PRINTFEED
RUN
```

Several string constants and string variables can be combined into one line of text by the use of plus signs or semicolons:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   PRTXT "SUN";"SHINE"
40   A$="MOON"
50   B$="LIGHT"
60   PRPOS 30,200
70   PRTXT A$+B$
80   PRINTFEED
RUN
```

yields a label with the text:

```
SUNSHINE
MOONLIGHT
```

PRTXT (PT), cont.

Numeric constants and numeric variables can be combined by the use of semicolons, but plus signs cannot be used in connection with numeric expressions:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   PRTXT 123;456
40   A%=222
50   B%=555
60   PRPOS 30,200
70   PRTXT A%;B%
80   PRINTFEED
RUN
```

yields a label with the text:

```
123456
222555
```

Numeric and string expressions can be mixed on the same line, for example:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   A$="December"
40   B%=27
50   PRTXT A$;" ";B%;" ";"2003"
80   PRINTFEED
RUN
```

yields a label with the text:

```
December 27 2003
```

Two program lines of text will be printed on the same line if the first program line is appended by a semicolon:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   PRTXT "HAPPY"+" ";
40   PRTXT "BIRTHDAY"
50   PRINTFEED
RUN
```

yields a label with the text:

```
HAPPY BIRTHDAY
```

PUT

Purpose

Statement for writing a given record from the random buffer to a given random file.

Syntax

PUT[#]<nexp$_1$>,<nexp$_2$>

#	indicates that whatever follows is a number. Optional.
<nexp$_1$>	is the number assigned to the file when it was OPENed.
<nexp$_2$>	is the number of the record. Must be ≥1.

Remarks

Use LSET or RSET statements to place data in the random buffer before issuing the PUT statement.

Example

```
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    SNAME$="SMITH"
40    CNAME$="JOHN"
50    PHONE$="12345630"
60    LSET F1$=SNAME$
70    LSET F2$=CNAME$
80    RSET F3$=PHONE$
90    PUT #8,1
100   CLOSE#8
RUN

SAVE "PROGRAM 1.PRG "

NEW
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    GET #8,1
40    PRINT F1$,F2$,F3$
RUN
```

yields:

SMITH↔↔↔JOHN↔↔↔↔↔↔12345630

RANDOM

Purpose

Function generating a random integer within a specified interval.

Syntax

RANDOM(<nexp$_1$>,<nexp$_2$>)

<nexp$_1$>	is the first integer in the interval.
<nexp$_2$>	is the last integer in the interval.

Remarks

<nexp$_1$> ≤ <random integer> ≤ <nexp$_2$>, that is, the random integer will be:

Equal to or greater than <nexp$_1$>

Equal to or less than <nexp$_2$>

Example

The following example will produce ten random integers between 1 and 100:

```
10   FOR I%=1 TO 10
20   A% = RANDOM (1,100)
30   PRINT A%
40   NEXT I%
RUN
```

yields for example:

```
31
45
82
1
13
16
41
77
20
70
```

RANDOMIZE

Purpose

Statement for reseeding the random number generator, optionally with a specified value.

Syntax

RANDOMIZE[<nexp>]

<nexp>	is the integer (0 -99999999) with which the random number generator will be reseeded

Remarks

If no value is specified, a message will appear asking you to enter a value between 0 and 99,999,999.

Examples

In the following example, no reseeding integer is specified in the program. Thus a prompt will appear, asking you to do so:

```
10    RANDOMIZE
20    A%=RANDOM1,100)
30    PRINT A%
RUN
Random Number Seed (0 to 99999999) ?
Enter 555
```

yields for example:

```
36
```

When the reseeding integer is specified, no prompt will appear:

```
10    RANDOMIZE 556
20    A%=RANDOM(1,100)
30    PRINT A%
RUN
```

yields for example:

```
68
```

A higher degree of randomization will be obtained in the random integer generator is reseeded with a more or less random integer, for example provided by a TICKS function:

```
10    A%=TICKS
20    RANDOMIZE A%
30    B%=RANDOM(1,100)
40    PRINT B%
RUN
```

yields for example:

```
42
```

READY

Purpose

Statement for ordering a ready signal, for example XON, CTS/RTS or PE, to be transmitted from the printer on the specified communication channel.

Syntax

READY[<nexp>]

<nexp>	optionally specifies a communication channel: 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 6 = "usb1:"

Remarks

The selected communication protocol usually contains some “ready” signal, which tells the host computer that the printer is ready to receive more data. The READY statement allows you to order a ready signal to be transmitted on the specified communication channel. If no channel is specified, the signal will be transmitted on the standard OUT channel (see SETSTDIO statement).

The READY signal is used to revoke a previously transmitted BUSYsignal. However, the printer may still be unable to receive more data, for example because of a full receive buffer.

For the "centronics:" communication channel, BUSY/READY controls the PE (paper end) signal on pin 12 according to an error-trapping routine (READY = PE low).

Example

You may, for example, want to allow the printer to receive more data on "uart2:" after the process of printing a label is completed. (Running this example may require an optional interface board to be fitted):

```
10   FONT "Swiss 721 BT"
20   PRTEXT "HELLO!"
30   BUSY2
40   PRINTFEED
50   READY2
RUN
```

REBOOT

Purpose Statement for restarting the printer.

Syntax **REBOOT**

Remarks This statement has exactly the same effect as switching off and on the power to the printer.

REDIRECT OUT

Purpose

Statement fo redirecting the output data to a created file.

Syntax

REDIRECT$_{\leftrightarrow}$OUT[<sexp>]

<sexp>	is, optionally, the name of the file to be created and where the output will be stored.

Remarks

Normally the output data will be transmitted on the standard output channel (see SETSTDIO statement). In most cases, this means the screen of the host. However, by means of a REDIRECT OUT <sexp> statement, a file can be created to which the output will be redirected. That implies that no data will be echoed back to the host. Normal operation, with the output being transmitted on the standard output channel again, will be resumed when a REDIRECT OUT statement without any appending file name is executed.

Example

In this example, a file ("LIST.DAT") is created to which the names of the files in the printer's permanent memory is redirected. The redirection is then terminated (line No. 30) and the file is OPENed for input.

```
10   REDIRECT OUT "LIST.DAT"
20   FILES "/c"
30   REDIRECT OUT
40   OPEN "LIST.DAT" FOR INPUT AS #1
. . . . .
. . . . .
. . . . .
```

REM (')

Purpose

Statement for adding headlines and explanations to the program without including them in the execution.

Syntax

REM|'<remark>

<remark>	is a text inserted in the program for explanatory purpose. Max. 32,767 characters per line.

Remarks

A REM statement may either be entered on a program line of its own or be inserted at the end of a line containing another instruction. In the latter case, REM must be preceded by a colon (":REM").

A shorthand form for REM is an apostrophe (ASCII 39 dec.).

It is possible to branch to a line of REM statement. Execution will then continue at the first executable line after the REM line.

REM statements slow down execution and transfer of data and also take up valuable memory space. Therefore, use REM statements with judgement.

Example

A program containing REM statements:

```
10    'Label format No. 1
20    FONT "Swiss 721 BT"
30    PRPOS 30,100
40    DIR 1 :REM Print across web
50    ALIGN 4 :REM Aligned left/baseline
60    MAG 2,2 :'Double height and width
70    PRTXT "HELLO"
80    PRINTFEED
RUN
```

REMOVE IMAGE

Purpose

Statement for removing a specified image from the printer's memory.

Syntax

REMOVE↔IMAGE <sexp>

<sexp>	is the full name including extension of the image to be removed.

Remarks

Useful for removing obsolete or faulty images from the printer's memory in order to save valuable memory space.

Note that there is a distinction between on one hand images and on the other hand image files (compare with IMAGES and FILES statements). This implies that REMOVE IMAGE statements can only be used for images downloaded by means of a STORE statement (see STORE and STORE IMAGE). Image files downloaded using for example a TRANSFER KERMIT statement should be removed the same way as other files using a KILL statement.

Be careful, REMOVE IMAGE is irreversible!

Example

```
10   REMOVE IMAGE "LOGOTYPE.1"
RUN
```

RENDER ON/OFF

Purpose

Statement for enabling/disabling rendering of text, bar code, image, box, and line fields.

Syntax

RENDER ON|OFF

ON	enables rendering (default).
OFF	disables rendering.

Remarks

These statements are intended to get information regarding size and position of a field without actually rendering it, that is the field will not be printed when the program is executed. The information on the field is retrieved using PRSTAT functions.

RENDER OFF disables the rendering, which means that PRTXT, PRBAR, PRIMAGE, PRLINE, and PRBOX statements will not give any result when a PRINTFEED statement is executed. Any other statements than PRPOS will not update the insertion point. Field numbers (see FIELDNO) will not be updated. Statement such as CLIP ON/OFF, XORMODE ON/OFF, or BARSET will retain their usual meanings. PRBUF will render a field regardless of RENDER ON/OFF.

RENDER ON enables field rendering after a RENDER OFF statement.

Duplicate statement have no effect, that is if a RENDER OFF statement has been executed, another RENDER OFF statement will be ignored. The same applies to RENDER ON.

Example

This examples retrieves information on the size of a text field which was not rendered. (The actual result may vary depending on font, font size, and printer type.)

```
10   RENDER OFF
20   PRTXT "Render off"
30   PRINT "Width:",PRSTAT(5),"Height:",
     PRSTAT(6)
40   RENDER ON
50   PRINTFEED
RUN
```

yields for example:

```
Width:  153     Height:   46

No field to print in line 50

Ok
```

RENUM

Purpose

Statement for renumbering the lines of the program currently residing in the printer's working memory.

Syntax

RENUM[<ncon$_1$>][,[<ncon$_2$>][,<ncon$_3$>]]

<ncon$_1$>	is the first line number of the new sequence.
<ncon$_2$>	is the line in the current program at which renumbering is to start.
<ncon$_3$>	is the desired increment between line numbers in the new sequence.
Default values:	10, 1, 10

Remarks

This statement is useful for providing space for more program lines when expanding an existing program, and for renumbering programs written without line numbers, for example after being LISTed, LOADed, or MERGEd. Line references following GOTO statements will be renumbered accordingly. Use a LIST statement to print the new numbers on the screen.

Example

A program may be renumbered like this:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,100
30   PRTXT "HELLO"
40   A%=A%+1
50   PRINTFEED
60   IF A%<3 GOTO 40
70   END
RENUM 100,20,50
LIST
```

yields:

```
10   FONT "Swiss 721 BT"
100  PRPOS 30,100
150  PRTXT "HELLO"
200  A%=A%+1
250  PRINTFEED
300  IF A%<3 GOTO 200
350  END
```

Note that the line number in the GOTO statement on line 300 has changed. Line 10 is not renumbered, since line 20 was specified as starting point. The new increment is 50.

REPRINT ON/OFF

Purpose

Statement for enabling/disabling reprinting of a label in the Direct Protocol.

Syntax

REPRINT <ON|OFF>

ON	Enables reprinting (default)
OFF	Disables reprinting

Remarks

The REPRINT ON/OFF statement is used to enable or disable reprinting of a label. REPRINT OFF also affects and overrides the behaviour of the PRINT KEY ON statement. If REPRINT OFF is entered before PRINT KEY ON, no error message is shown but PRINT KEY is set to ON and an empty label is printed when the <Print> key is pressed.

If REPRINT is set to OFF, there is no way to reprint an old print job. If a PRINTFEED statement is sent to the printer after a print job has been completed, a blank label is fed out and the error 1006 "No field to print" occurs. However, the REPRINT OFF statement does not clear the print buffer, which only occurs after a PRINTFEED statement has been executed (see example 2 below).

Leaving and re-entering the Direct Protocol does not reset the REPRINT status. A reboot resets the REPRINT status to its default value (ON).

A REPRINT OFF statement prevents automatic reprinting after the error has been cleared for the following errors:
1005 "Out of paper"
1022 "Head lifted"
1031 "Next label not found"
1058 "Transfer ribbon is installed"

Error 1027 "Out of transfer ribbon" does not generate the display prompt "Continue-Reprint" if REPRINT is set to OFF.

Examples

This example disables reprinting:

```
INPUT ON                          Enter the Direct Protocol
REPRINT OFF                       Disable reprinting
```

This example shows a special case, when the first and second PRINTFEED statements generates a printed label. After the second PRINTFEED, the print buffer is cleared and a "No field to print" error occurs.

```
INPUT ON                          Enter the Direct Protocol
PRTXT "PRINT1"                    Print the text "Print1"
PRINTFEED                         Yields a print label
REPRINT OFF                       Disable reprinting
PRINTFEED                         Yields a printed label, clears print buffer
PRINTFEED                         Yields a blank label, generates an error
```

RESUME

Purpose

Statement for resuming program execution after an error-handling subroutine has been executed.

Syntax

RESUME[<<ncon>|<line label>|<NEXT>|<0>>|<HTTP>]

<ncon>	is the number or label of the line to which the program should return.

Remarks

RESUME must only be used in connection with error-handling subroutines (see ON ERROR GOTO).

There are five ways of using RESUME:

RESUME	Execution is resumed at the statement where the error occurred.
RESUME 0	Same as RESUME.
RESUME NEXT	Execution is resumed at the statement immediately following the one that caused the error.
RESUME <ncon>	Execution is resumed at the specified line.
RESUME <line label>	Execution is resumed at the specified line label.
RESUME <HTTP>	Execution is resumed at the point where it was branched by an ON HTTP GOTO statement. Stdin and stdout are restored to their original values.

Examples

This short program is the basis for two examples of alternative subroutines:

```
10   ON ERROR GOTO 1000
20   FONT "Swiss 721 BT"
30   PRPOS 100,100
40   PRTXT "HELLO"
50   PRPOS 100, 300
60   PRIMAGE "GLOBE.1"
70   PRINTFEED
80   END
```

1. A font is selected automatically and execution is resumed from the line after where the error occurred. If another error than the specified error condition occurs, the execution is terminated.

```
1000 IF ERR=15 THEN FONT "Swiss 721 BT":RESUME
     NEXT
1010 RESUME 80
```

2. An error message is displayed and the execution goes on from the line following the one where the error occurred.

```
1000 IF ERR=15 THEN PRINT "Font not found"
1010 RESUME NEXT
```

RETURN

Purpose

Statement for returning to the main program after having branched to a subroutine because of a GOSUB statement.

Syntax

RETURN[<ncon>|<line label>]

<ncon>	is optionally the number or label of a line in the main program to return to.

Remarks

When the statement RETURN is encountered during the execution of a subroutine, the execution will return to the main program. Execution will continue from the statement immediately following the most recently executed GOSUB or from an optionally specified line.

If a RETURN statement is encountered without a GOSUB statement having been previously executed, Error 28, “Return without Gosub” will occur.

Example

```
10   PRINT "This is the main program"
20   GOSUB 1000
30   PRINT "You’re back in the main program"
40   END
1000 PRINT "This is subroutine 1"
1010 GOSUB 2000
1020 PRINT "You’re back in subroutine 1"
1030 RETURN
2000 PRINT "This is subroutine 2"
2010 GOSUB 3000
2020 PRINT "You’re back in subroutine 2"
2030 RETURN
3000 PRINT "This is subroutine 3"
3010 PRINT "You’re leaving subroutine 3"
3020 RETURN
RUN
```

yields:

```
This is the main program
This is subroutine 1
This is subroutine 2
This is subroutine 3
You’re leaving subroutine 3
You’re back in subroutine 2
You’re back in subroutine 1
You’re back in the main program
```

RIGHT$

Purpose

Function returning a specified number of characters from a given string starting from the extreme right side (end) of the string.

Syntax

RIGHT$(<sexp>,<nexp>)

<sexp>	is the string from which the characters will be returned.
<nexp>	specifies the number of characters to be returned.

Remarks

This function is the complementary function for LEFT$, which returns the characters starting from the extreme left side, that is from the start.

If the number of characters to be returned is greater than the number of characters in the string, then the entire string will be returned. If the number of characters is set to zero, a null string will be returned.

Examples

```
PRINT RIGHT$("THERMAL_PRINTER",7)
```

yields:

```
PRINTER
```

```
10   A$="THERMAL_PRINTER":B$ = "LABEL"
20   PRINT RIGHT$(B$,5);RIGHT$(A$,8);"S"
RUN
```

yields:

```
LABEL_PRINTERS
```

RSET

Purpose

Statement for placing data right-justified into a field in a random file buffer.

Syntax

RSET<svar>=<sexp>

<svar>	is the string variable assigned to the field by a FIELD statement.
<sexp>	holds the input data.

Remarks

After having OPENed a file and formatted it using a FIELD statement, you can enter data into the random file buffer using the RSET and LSET statements (LSET left-justifies the data).

The input data can only be stored in the buffer as string expressions. Therefore, a numeric expression must be converted to string by the use of a STR$ function before an LSET or RSET statement is executed.

If the length of the input data is less than the field, the data will be right justified and the remaining number of bytes will be printed as space characters.

If the length of the input data exceeds the length of the field, the input data will be truncated on the left side.

Example

```
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    SNAME$="SMITH"
40    CNAME$="JOHN"
50    PHONE$="12345630"
60    LSET F1$=SNAME$
70    LSET F2$=CNAME$
80    RSET F3$=PHONE$
90    PUT #8,1
100   CLOSE#8
RUN

SAVE "PROGRAM 1.PRG "

NEW
10    OPEN "PHONELIST" AS #8 LEN=26
20    FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30    GET #8,1
40    PRINT F1$,F2$,F3$
RUN
```

yields:

```
SMITH↔↔↔JOHN↔↔↔↔↔↔12345630
```

RUN

Purpose

Statement for starting the execution of a program.

Syntax

RUN[<<scon>|<ncon>>]

<scon>	optionally specifies an existing program to be run.
<ncon>	optionally specifies the number of a line in the current program where the execution will start.

Remarks

The RUN statement starts the execution of the program currently residing in the printer's working memory, or optionally of a specified program residing elsewhere. The execution will begin at the line with the lowest number, or optionally from a specified line in the current program.

If a program stored in another directory than the current one (see CHDIR statement), and has not been LOADed, its designation must be preceded by a reference to that device ("/c", "tmp:", "/rom", or "card1:", see the last example).

Never use RUN on a numbered line or in a line without number in the Programming Mode, or Error 40, "Run statement in program" will occur.

A RUN statement executed in the Intermec Direct Protocol will make the printer switch to the Immediate Mode, that is it has the same effect as an INPUT OFF statement.

Examples

Order the execution of a program this way:

```
RUN
```
Executes the current program from its first line.

```
RUN 40
```
Executes the current program, starting from line 40.

```
RUN "TEST"
```
Executes the program "TEST.PRG" from its first line.

```
RUN "TEST.PRG"
```
Executes the program "TEST.PRG" from its first line.

```
RUN "/rom/FILELIST.PRG"
```
Executes the program "FILELIST.PRG", which is stored in the read-only memory, from its first line.

SAVE

Purpose

Statement for saving a file in the printer's memory or optionally in a memory card.

Syntax

SAVE<scon>[,P|L]

<scon>	is the name of the file, optionally starting with a reference to a directory (see DEVICES). Allowed input: Max. 30 characters incl. extension. Max. 26 characters excl. extension
P	optionally protects the file.
L	optionally saves the file without line numbers.

Remarks

When a file is SAVEd, it must be given a designation consisting of max. 30 characters including extension. By default, the program will automatically append the name with the extension .PRG and convert all lowercase characters to uppercase. The name must not contain any quotation marks ("). By starting the file name with a period character (.), you can avoid having it removed at a soft formatting operation, see FORMAT statement. Such a file will also be listed differently, see FILES statement.

When saving a file in a directory other than the current one (see CHDIR statement), a reference to that directory must be included in the file name. Files can only be SAVEd in the printer's permanent memory ("/c"), the printer's temporary memory ("tmp:"), or in an optional CompactFlash memory card ("card1:"). If a file with the selected name already exists in the selected directory, that file will be deleted and replaced by the new file without any warning. You can continue to work with a file after saving it, until a NEW, LOAD, KILL, or REBOOT instruction is issued.

A **protected file** (SAVE <filename>,P) is encrypted at saving and cannot be LISTed after being LOADed. Program lines cannot be removed, changed, or added. Once a file has been protected, it cannot be deprotected again. Therefore, it is advisable to save an unprotected copy, should a programming error be detected later on. If you are going to use an electronic key to prevent unautorized access to a file, you should protect it.

A SAVEd program can be MERGEd with the program currently residing in the printer's working memory. If the program is SAVEd normally, there is a risk that the line numbers automatically assigned to the program may interfere with the line numbers in the current program. Therefore, you can choose to SAVE the program **without line numbers** (SAVE <filename>,L). That entails that the MERGEd program will be appended to the current program and its lines will be assigned line numbers in ten-step incremental order, starting with the number of the last line in the current program plus 10. In this case, the MERGEd program should either make use of line labels for referring to other lines, or not contain any such instructions at all.

SAVE, cont.

Examples

```
SAVE "Label14"
```
saves the file as “LABEL 14.PRG" in current directory.

```
SAVE "Label14",P
```
saves and protects the file "LABEL14.PRG".

```
SAVE "Label14",L
```
saves the file "LABEL14.PRG" without line numbers.

```
SAVE "/c/MY LABELS/Label14"
```
saves the file "LABEL14.PRG" in the directory “MY LABELS” in the printer’s permanent memory.

```
SAVE "card1:Label14.PRG"
```
saves the file in an optional CompactFlash memory card.

SET FAULTY DOT

Purpose

Statement for marking one or several dots on the printhead as faulty, or marking all faulty dots as correct.

Syntax

SET↔FAULTY↔DOT<nexp$_1$>[,<nexp$_n$>...]

<nexp$_1$>	is the number of the dot to be marked as faulty. Successive executions add more faulty dots.
<nexp$_1$> = -1	marks all dots as correct (default).

Remarks

This statement is closely related to the HEAD function and the BARADJUST statement. You can check the printhead for possible faulty dots by means of the HEAD function and mark them as faulty, using the SET FAULTY DOT statement. Using the BARADJUST statement, you can allow the firmware to automatically reposition horizontal bar codes sideways so as to place the faulty dots between the bars, where no harm to the readability will be done.

Once a number a dot has been marked faulty by a SET FAULTY DOT statement, it will remain so until all dots are marked as correct by a SET FAULTY DOT -1 statement.

Note that the HEAD function makes it possible to mark all faulty dots using a single instruction instead of specifying each faulty dot in a SET FAULTY DOT.

Example

This example illustrates how a bar code is repositioned by means of BARADJUST when a number of dots are marked as faulty by a SET FAULTY DOTS statement. Type RUN and send various numbers of faulty dots from the host a few times and see how the bar code moves sideways across the label.

```
10   INPUT "No. of faulty dots"; A%
20   FOR B% = 1 TO A%
30   C% = C% + 1
40   SET FAULTY DOT C%
50   NEXT
60   D% = A%+2
70   BARADJUST D%, D%
80   PRPOS 0, 30
90   BARTYPE "CODE39"
100  PRBAR "ABC"
110  SET FAULTY DOT -1
120  PRINTFEED
RUN
```

SETASSOC

Purpose

Statement for setting a value for a tuple in a string association.

Syntax

SETASSOC <sexp$_1$>, <sexp$_2$>, <sexp$_3$>

<sexp$_1$>	is the name of the association (case-sensitive).
<sexp$_2$>	is the name of the tuple
<sexp$_3$>	is the value of the tuple.

Remarks

An association is an array of tuples, where each tuple consists of a name and a value.

Example

This example shows how a string, including three string names associated with three start values, will be defined and one of them (time) will be changed:

```
10   QUERYSTRING$=
     "time=UNKNOWN&label=321&desc=DEF"
20   MAKEASSOC "QARRAY",QUERYSTRING$,"HTTP"
30   QTIME$=GETASSOC$("QARRAY","time")
40   QLABELS%=VAL(GETASSOC$("QARRAY","label"))
50   QDESC$=GETASSOC$("QARRAY","desc")
60   PRINT"time=";QTIME$,"LABEL=";QLABELS%,
     "DESCRIPTION=";QDESC$
70   SETASSOC "QARRAY","time",time$
80   PRINT "time=";GETASSOC$("QARRAY","time")
RUN
```

yields:

```
time=UNKNOWN  LABEL=321 DESCRIP   TION=DEF
time=153355
```

SETPFSVAR

Purpose

Statement for registering variable to be saved at power failure.

Syntax

SETPFSVAR<sexp>[,<nexp>]

<sexp>	is the name of a numeric or string variable (uppercase characters only).
<nexp>	is, optionally, the size in bytes of a string variable (max. 230).

Remarks

When a program is loaded, it is copied to and executed in the printer's temporary memory ("tmp:"). Should an unexpected power failure occur, the printer tries to save as much data as possible in the short time available before all power is lost. To minimize the risk of lose important variable values at a power failure, you can register numeric and string variables to be saved. There is 2176 bytes (incl. overhead) available for this purpose.

However, should the power failure occur while the printer is printing, there will not be any power left to save the current variables.

When you register a string variable, you must also specify its size in bytes.

Related instructions are GETPFSVAR, DELETEPFSVAR, and LIST-PFSVAR.

Examples

Example with string variable:

```
100  IF QA$="" THEN QA$="Hello":QA%=LEN(QA$)
110  SETPFSVAR "QA$",QA%
```

Example with numeric variable:

```
200  SETPFSVAR"QCPS%"
```

SETSTDIO

Purpose

Statement for selecting standard IN and OUT communication channel.

Syntax

SETSTDIO<nexp$_1$>[,<nexp$_2$>]

<nexp$_1$>	is the desired input/output channel: 100 = autohunting enabled (default) 0 = "console:" 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 4 = "centronics:" 5 = "net1:" 6 = "usb1:"
<nexp$_2$>	optionally specifies an output channel other than the input channel: 0 = "console:" 1 = "uart1:" 2 = "uart2:" 3 = "uart3:" 5 = "net1:" 6 = "usb1:"

Remarks

The printer is controlled from its host via a communication channel. By default, autohunting is selected. Autohunting means that all available channels are continuously scanned for input. When data is received on a channel, it is regarded as standard input/output channel. If no data is received on the present standard input channel within a 2 second timeout period, the firmware scans all other existing channels (except "console:") looking for input data. The channel where input data is first found will now be appointed the new stdin/stdout channel. The same procedure is repeated infinitely as long as autohunting is enabled.

There are some restrictions that apply to autohunting:
- If "centronics:" is used as input channel and autohunting is enabled, "uart1:" is selected stdout channel.
- Autohunting does not work with "console:".
- Autohunting does not work with COMSET or INPUT.

It is also possible to specify a certain channel as permanent stdin and/or stdout channel. If only one channel is specified, it will serve as both standard input (stdin) and standard output (stdout) channel. Alternatively, different channels can be selected for stdin and stdout.

For programming, it is recommended to use "uart1:" both as stdin and stdout channel. If another channel is selected, use the same serial channel for both input and output. The "centronics:" channel can only be used for input to the printer and is thus not suited for programming.

SETSTDIO, cont.

Example

This example selects the "uart2:" communication channel as the standard input and output channel:

```
10   SETSTDIO 2
. . . .
. . . .
```

This example enables autohunting for input and "uart1:" for output:

```
10   SETSTDIO 100,1
. . . .
. . . .
```

SETUP

Purpose

Statement for entering the printer's Setup Mode or changing the setup.

Syntax

SETUP

If no parameter is specified, the printer enters the Setup Mode.

SETUP <sexp>

<sexp>	is the name of an existing setup file that will be used to change the printer's entire current setup, or a string used to change a single parameter in the printer's current setup.

SETUP <sexp$_1$>,<sexp$_2$>

<sexp$_1$>	is the name of a setup section (EasyLAN User's Guide).
<sexp$_2$>	is the name of a file that will be used to change the specified setup section.

SETUP <sexp$_1$>,<sexp$_2$>,<sexp$_3$>

<sexp$_1$>	is the name of a setup section (see EasyLAN User's Guide). Not implemented for "prt".
<sexp$_2$ >	is the name of the setup object (see EasyLAN User's Guide).
<sexp$_3$>	specifies the new value (see EasyLAN User's Guide).

Remarks

The SETUP statement can be used for several purposes as illustrated above. Related instructions are SETUP GET and SETUP WRITE.

By default, the setup parameters are saved as a file in the printer's permanent memory. However, using SYSVAR (35) it is possible to decide that any new change will not be saved (volatile). See SYSVAR.

The methods of manual setup via the printer's built-in keyboard are described in the User's Guides manuals for the various printer models. You can also use setup files and setup strings to change the setup as a part of the program execution, or to change the setup remotely from the host.

A setup file may contain new values for one or several setup parameters, whereas a setup string only can change a single parameter. Another difference is that, while the creation of setup files requires several operations, setup strings can be created in a single operation which makes them suitable for use with the Intermec Direct protocol.

When a SETUP<sexp> statement is encountered, the setup will be changed accordingly, then the program execution will be resumed. Note that some printing instructions (ALIGN, DIR, FONT, and PRPOS) may be changed when test labels are printed.

SETUP, cont.

The content of setup files can be listed using the program FILELIST.PRG stored in the printer's permanent memory (`"/rom/"`), or by COPYing the file to the communication channel of the host, usually "uart1:".

Setup files or setup strings have a special syntax for each parameter that must be followed exactly. Variable numeric input data are indicated by "n" – "nnnnn", alternative data are indicated by bold characters separated by vertical bars (|). Compulsory space characters are indicated by double-headed arrows (↔). Note that some parameters listed below may only apply to a certain printer model or an optional device.

Do not include any double-headed arrows or vertical bars when typing a setup string or file!

"SER-COM,**UART1|UART2|UART3**,BAUDRATE,**300|600|1200|2400|4800|9600|19200|38400|57600|115200**"

"SER-COM,**UART1|UART2|UART3**,CHAR↔LENGTH,**7|8**"

"SER-COM,**UART1|UART2|UART3**,PARITY,**NONE|EVEN|ODD|MARK|SPACE**"

"SER-COM,**UART1|UART2|UART3**,STOPBITS,**1|2**"

"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,RTS/CTS,**ENABLE|DISABLE**"

"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,ENQ/ACK,**ENABLE|DISABLE**"

"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,XON/XOFF,DATA↔FROM↔HOST,**ENABLE|DISABLE**"

"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,XON/XOFF,DATA↔TO↔HOST,**ENABLE|DISABLE**"

"SER-COM,UART2,PROT↔ADDR,**ENABLE|DISABLE**"

"SER-COM,**UART1|UART2|UART3**,NEW↔LINE,**CR/LF|LF|CR**"

"SER-COM,**UART1|UART2|UART3**,REC↔BUF,nnnnn"

"SER-COM,**UART1|UART2|UART3**1,TRANS↔BUF,nnnnn"

"SER-COM,UART2,PROTOCOL↔ADDR.,nn"

"NET-COM,NET1,NEW↔LINE,**CR/LF|LF|CR**"

"NETWORK,IP↔SELECTION,**DHCP+BOOTP|MANUAL|DHCP|BOOTP**"

"NETWORK,IP↔ADDRESS,nnn.nnn.nnn.nnn"

"NETWORK,NETMASK,nnn.nnn.n.n"

"NETWORK,DEFAULT↔ROUTER,nnn.nnn.nnn.nnn"

"NETWORK,NAME↔SERVER,nnn.nnn.n.n"

"FEEDADJ,STARTADJ,nnnn" *(negative value allowed)*

"FEEDADJ,STOPADJ,nnnn" *(negative value allowed)*

"MEDIA,MEDIA↔SIZE,XSTART,nnnn"

"MEDIA,MEDIA↔SIZE,WIDTH,nnnn"

"MEDIA,MEDIA↔SIZE,LENGTH,nnnnn"

"MEDIA,MEDIA↔TYPE,**LABEL↔(w↔GAPS)|TICKET↔(w↔MARK)|TICKET↔(w↔GAPS)|FIX↔LENGTH↔STRIP|VAR↔LENGTH STRIP**"

"MEDIA,PAPER↔TYPE,**TRANSFER|DIRECT↔THERMAL**"

"MEDIA,PAPER↔TYPE,DIRECT↔THERMAL,LABEL↔CONSTANT,nnn"

"MEDIA,PAPER↔TYPE,DIRECT↔THERMAL,LABEL↔FACTOR,nnn"

"MEDIA,PAPER↔TYPE,TRANSFER,RIBBON↔CONSTANT,nnn"

"MEDIA,PAPER↔TYPE,TRANSFER,RIBBON↔FACTOR,nnn"

"MEDIA,PAPER↔TYPE,TRANSFER,LABEL↔OFFSET,nnn"

"MEDIA,PAPER↔TYPE,TRANSFER,LOW↔DIAMETER,nnn"

"MEDIA,CONTRAST,**-10%|-8%|-6%|-4%|-2%|+0%|+2%|+4%|+6%|+8%|+10%**"

"MEDIA,PAPER,LOW↔DIAMETER,nnn"

"PRINT↔DEFS,PRINT↔SPEED,nnn"

"PRINT↔DEFS,LTS↔VALUE,nn"

SETUP, cont.

Examples

This example enables a key for branching to the Setup Mode:

```
10    ON KEY(18) GOSUB 1000
20    KEY(18)ON
.....
1000  SETUP
1010  RETURN
```

This example shows how a new file is OPENed for output and each parameter in the setup is changed by means of PRINT# statements. Then the file is CLOSEd. Any lines, except the first and the last line in the example, may be omitted. Finally, the printer's setup is changed using this file.

```
10    OPEN "/tmp/SETUP.SYS" FOR OUTPUT AS #1
20    PRINT#1,"SER-COM,UART1,BAUDRATE,19200"
30    PRINT#1,"SER-COM,UART1,CHAR LENGTH,7"
40    PRINT#1,"SER-COM,UART1,PARITY,EVEN"
50    PRINT#1,"SER-COM,UART1,STOPBITS,2"
60    PRINT#1,"SER-COM,UART1,FLOWCONTROL,RTS/CTS,ENABLE"
70    PRINT#1,"SER-COM,UART1,FLOWCONTROL,ENQ/ACK,ENABLE"
80    PRINT#1,"SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA FROM
      HOST,ENABLE"
90    PRINT#1,"SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA TO
      HOST,ENABLE"
100   PRINT#1,"SER-COM,UART1,NEW LINE,CR"
110   PRINT#1,"SER-COM,UART1,REC BUF,800"
120   PRINT#1,"SER-COM,UART1,TRANS BUF,800"
130   PRINT#1,"FEEDADJ,STARTADJ,-135"
140   PRINT#1,"FEEDADJ,STOPADJ,-36"
150   PRINT#1,"MEDIA,MEDIA SIZE,XSTART,50"
160   PRINT#1,"MEDIA,MEDIA SIZE,WIDTH,1000"
170   PRINT#1,"MEDIA,MEDIA SIZE,LENGTH,2000"
180   PRINT#1,"MEDIA,MEDIA TYPE,LABEL (w GAPS)"
190   PRINT#1,"MEDIA,PAPER TYPE,TRANSFER"
200   PRINT#1,"MEDIA,PAPER TYPE,TRANSFER,RIBBON CONSTANT,110"
210   PRINT#1,"MEDIA,PAPER TYPE,TRANSFER,RIBBON FACTOR,25"
220   PRINT#1,"MEDIA,PAPER TYPE,TRANSFER,LABEL OFFSET,00"
230   PRINT#1,"TRANSFER,LOW DIAMETER,30"
230   PRINT#1,"MEDIA,CONTRAST,-4%"
240   PRINT#1,"PRINT DEFS,PRINT SPEED,200"
250   CLOSE
260   SETUP "/tmp/SETUP.SYS"
```

This example shows how a setup parameter is changed in the Immediate Mode or the Intermec Direct Protocol, using a setup string.

```
SETUP"MEDIA,MEDIA TYPE,VAR LENGTH STRIP" ↵
```

This method can also be used in the Programming Mode, for example:

```
10 SETUP"MEDIA,MEDIA TYPE,VAR LENGTH STRIP"
```

SETUP GET

Purpose

Statement for getting the current setting for a single setup object.

Syntax

SETUP GET<sexp$_1$>,<sexp$_2$>,<sexp$_3$>

<sexp$_1$>	specifies the setup section.
<sexp$_2$>	specifies the setup object.
<sexp$_3$>	stores the result.

Remarks

Refer to *Intermec EasyLAN User's Guide* for a list of setup sections and objects.

Examples

```
SETUP GET "lan1","RTEL_PR1",A$
SETUP GET "prt","MEDIA,MEDIA TYPE", B$
SETUP GET "alerts","lts",C$
```

SETUP WRITE

Purpose

Statement for creating a file containing the printer's current setup or for returning it on a specified communication channel.

Syntax

SETUP WRITE[<sexp$_1$>] ,<sexp$_2$>

<sexp$_1$>	is an optional parameter specifying the setup section.
<sexp$_2$>	is the name of a file or device to which the printer's current setup is to be written.

Remarks

The SETUP WRITE statement is useful when you want to return to the printer's current setup at a later moment. You can make a copy of the current setup using SETUP WRITE<filename>, change the setup using a SETUP <filename> statement, and, when so is required, return to the original setup by issuing a new SETUP<filename> statement containing the name of the file created by the SETUP WRITE<filename> statement.

It is strongly recommended to create the file in the printer's temporary memory ("tmp:"), for example SETUP WRITE "tmp:OLDSETUP". Once it has been created in "tmp:", it can be copied to the printer's permanent memory "/c" so it will not be lost at power off.

Another application of SETUP WRITE is transmitting the printer's current setup on a serial communication channel, for example SETUP WRITE "uart1:".

Setup sections are used in connection with EasyLAN. Refer to the *EasyLAN User's Guide* for a list of setup sections.

SETUP WRITE returns the printer's setup in the following order (the example shows a standard EasyCoder PF4i printer):

```
SETUP WRITE "uart1:"
```

yields:

```
SER-COM,UART1,BAUDRATE,9600
SER-COM,UART1,CHAR LENGTH,8
SER-COM,UART1,PARITY,NONE
SER-COM,UART1,STOPBITS,1
SER-COM,UART1,FLOWCONTROL,RTS/CTS,DISABLE
SER-COM,UART1,FLOWCONTROL,ENQ/ACK,DISABLE
SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA FROM HOST,DISABLE
SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA TO HOST,DISABLE
SER-COM,UART1,NEW LINE,CR/LF
SER-COM,UART1,REC BUF,1024
SER-COM,UART1,TRANS BUF,1024
FEEDADJ,STARTADJ,0
FEEDADJ,STOPADJ,0
MEDIA,MEDIA SIZE,XSTART,24
MEDIA,MEDIA SIZE,WIDTH,832
MEDIA,MEDIA SIZE,LENGTH,1200
MEDIA,MEDIA TYPE,LABEL (w GAPS)
```

SETUP WRITE, cont.

```
MEDIA,PAPER TYPE,TRANSFER
MEDIA,PAPER TYPE,DIRECT THERMAL,LABEL CONSTANT,85
MEDIA,PAPER TYPE,DIRECT THERMAL,LABEL FACTOR,40
MEDIA,PAPER TYPE,TRANSFER,RIBBON CONSTANT,95
MEDIA,PAPER TYPE,TRANSFER,RIBBON FACTOR,25
MEDIA,PAPER TYPE,TRANSFER,LABEL OFFSET,0
MEDIA,PAPER TYPE,TRANSFER,RIBBON SENSOR,14
MEDIA,PAPER TYPE,TRANSFER,LOW DIAMETER,36
MEDIA,CONTRAST,0%
#MEDIA,TESTFEED,26 28 0 10
PRINT DEFS,HEAD RESIST, 702
PRINT DEFS,PRINT SPEED,100
```

Note that when a SETUP WRITE file is used to change the setup, the printer's present TESTFEED adjustment is not affected.

Examples

In this example, the current setup is saved in the printer's temporary memory under the name "SETUP1.SYS". Then the start adjustment is changed to "200" by the creation of a new setup file named "SETUP2. SYS." The setup file is finally used to change the printer's setup.

```
10    SETUP WRITE "tmp:SETUP1.SYS"
20    OPEN "tmp:SETUP2.SYS" FOR OUTPUT AS #1
30    PRINT#1,"FEEDADJ,STARTADJ,200"
40    CLOSE
50    SETUP "tmp:SETUP2.SYS"
```

In this example, the setup section "prt" is returned on the serial channel "uart1:":

```
SETUP WRITE "prt","uart1:"
```

SGN

Purpose

Function returning the sign (positive, zero, or negative) of a specified numeric expression.

Syntax

SGN(<nexp>)

<nexp>	is the numeric expression from which the sign will be returned.

Remarks

The sign will be returned in this form:

SGN(<nexp>) = -1	(negative)
SGN(<nexp>) = 0	(zero)
SGN(<nexp>) = 1	(positive)

Examples

Positive numeric expression:

```
10    A%=(5+5)
20    PRINT SGN(A%)
RUN
```

yields:

```
1
```

Negative numeric expression:

```
10    A%=(5-10)
20    PRINT SGN(A%)
RUN
```

yields:

```
-1
```

Zero numeric expression:

```
10    A%=(5-5)
20    PRINT SGN(A%)
RUN
```

yields:

```
0
```

SORT

Purpose

Statement for sorting a one-dimensional array.

Syntax

SORT<<nvar>|<svar>>,<nexp$_1$>,<nexp$_2$>,<nexp$_3$>

<<nvar>|<svar>>	is the array to be sorted.
<nexp$_1$>	is the number of the first element.
<nexp$_2$>	is the number of the last element.
<nexp$_3$>	> 0: Ascending sorting < 0: Descending sorting = 0: Illegal value In a string array, the value specifies the position according to which the array will be sorted.

Remarks

A numeric or string array can be sorted, in its entity or within a specified range of elements in ASCII value order.

The 4:th parameter (<nexp$_3$>) is used differently for numeric and string arrays. The sign always specifies ascending or descending order. For numeric arrays, the value is of no consequence, but for string arrays, the value specifies for which character position the elements will be sorted. <nexp$_3$> = 0 results in Error 41, "Parameter out of range."

Example

One numeric and one string array are sorted in descending order. The string array is sorted in ascending according to the third character position in each string:

```
10    ARRAY% (0) = 1001
20    ARRAY% (1) = 1002
30    ARRAY% (2) = 1003
40    ARRAY% (3) = 1004
50    ARRAY$ (0) = "ALPHA"
60    ARRAY$ (1) = "BETA"
70    ARRAY$ (2) = "GAMMA"
80    ARRAY$ (3) = "DELTA"
90    SORT ARRAY%,0,3,-1
100   SORT ARRAY$,0,3,3
110   FOR I% = 0 TO 3
120   PRINT ARRAY% (I%), ARRAY$ (I%)
130   NEXT
RUN
```

yields:

```
1004 DELTA
1003 GAMMA
1002 ALPHA
1001 BETA
```

SOUND

Purpose

Statement for making the printer's beeper produce a sound specified in regard of frequency and duration.

Syntax

SOUND<nexp$_1$>,<nexp$_2$>

<nexp$_1$>	is the frequency of the sound in Hz.
<nexp$_2$>	is the duration of the sound in periods of 0.020 sec. each (max. 15,0000 = 5 minutes).

Remarks

This statement allows you include significant sound signals in your programs, for example to notify the operator that various errors have occurred. A sound with approximately the specified frequency will be produced for the specified duration. If the program encounters a new SOUND statement, it will not be executed until the previous sound has been on for the specified duration.

The SOUND statement even allows you to make melodies, although the musical quality may be somewhat limited. The following table illustrates the frequencies corresponding to the notes in the musical scale. To create a period of silence, set the frequency to value higher than 9,999 Hz.

Key	Hz	Key	Hz	Key	Hz	Key	Hz
C	131	C	262	C	523	C	1047
C#	138	C#	277	C#	554	C#	1109
D	147	D	294	D	587	D	1175
D#	155	D#	311	D#	622	D#	1245
E	165	E	330	E	659	E	1319
F	175	F	349	F	699	F	1397
F#	185	F#	370	F#	740	F#	1480
G	196	G	392	G	784	G	1568
G#	208	G#	415	G#	831	G#	1662
A	220	A	440	A	880	A	1760
A#	233	A#	466	A#	933	A#	1865
B	247	B	494	B	988	B	1976
(small octave)		(one-line octave)		(two-line octave)		(three-line octave)	

Example

The tune “Colonel Boogie" starts like this:

```
10   SOUND 392,10
20   SOUND 330,15
30   SOUND 330,10
40   SOUND 349,10
50   SOUND 392,10
60   SOUND 659,18
70   SOUND 659,18
80   SOUND 523,25
```

SPACE$

Purpose Function returning a specified number of space characters.

Syntax **SPACE$(<nexp>)**

<nexp>	is the number of space characters to be returned.

Remarks This function is useful for more complicated spacing, for example in tables.

Examples Printing of two left-justified columns on the screen:

```
10    FOR Q%=1 TO 6
20    VERBOFF:INPUT "",A$
30    VERBON:PRINT A$;
40    VERBOFF:INPUT "",B$
50    VERBON
60    C$=SPACE$(25-LEN(A$))
70    PRINT C$+B$
80    NEXT Q%
90    END
RUN

Enter:
January ↵
February ↵
March ↵
April ↵
May ↵
June ↵
July ↵
August ↵
September ↵
October ↵
November ↵
December ↵
```

yields:

```
January                  February
March                    April
May                      June
July                     August
September                October
November                 December
```

SPLIT

Purpose

Function splitting a string into an array according to the position of a specified separator character and returning the number of elements in the array.

Syntax

SPLIT(<sexp$_1$>,<sexp$_2$>,<nexp>)

<sexp$_1$>	is the string to be split.
<sexp$_2$>	is the string array in which the parts of the split string should be put.
<nexp>	specifies the ASCII value for the separator.

Remarks

The string is divided by a specified separating character which may found an infinite number of times in the string. Each part of the string will become an element in the string array, but the separator character itself will not be included in the array.

Should the string be split into more than four elements, Error 57, "Subscript out of range" will occur. To avoid this error, issue a DIM statement to create a larger array before the string is split.

Example

In this example a string is divided into five parts by the separator character # (ASCII 35 decimal). The result will be an array of five elements numbered 0-4 as specified by a DIM statement. Finally, the number of elements is also printed on the screen.

```
10    A$="ONE#TWO#THREE#FOUR#FIVE"
20    B$="ARRAY$"
30    DIM ARRAY$(5)
40    C%=SPLIT(A$,B$,35)
50    PRINT ARRAY$(0)
60    PRINT ARRAY$(1)
70    PRINT ARRAY$(2)
80    PRINT ARRAY$(3)
90    PRINT ARRAY$(4)
100   PRINT C%
RUN
```

yields:

```
ONE
TWO
THREE
FOUR
FIVE
5
```

STOP

Purpose

Statement for terminating execution of a program and to return to immediate mode.

Syntax

STOP

Remarks

When a STOP statement is encountered, the following message is returned to the host:

```
Break in line <line number>
```

You can resume execution where it was stopped by means of a CONT statement or at a specified program line using a GOTO statement in the immediate mode.

STOP is usually used in conjunction with CONT for debugging. When execution is stopped, you can examine or change the values of variables using direct mode statements. You may then use CONT to resume execution. CONT is invalid if the program has been editied during the break.

Example

```
10    A%=100
20    B%=50
30    IF A%=B% THEN GOTO QQQ ELSE STOP
40    GOTO 30
50    QQQ:PRINT "Equal"

Ok
RUN
Break in line 30

Ok
PRINT A%
100

Ok
PRINT B%
50

Ok
B%=100

OK
CONT
Equal

Ok
```

STORE IMAGE

Purpose

Statement for setting up parameters for storing an image in the printer's memory.

Syntax

STORE$_{\leftrightarrow}$IMAGE [RLL] [KILL]<sexp$_1$>,<nexp$_1$>,<nexp$_2$>,[<nexp$_3$>],<sexp$_2$>

[RLL]	optionally indicates RLL compression.	
[KILL]	optionally specifies that the image will be erased from the temporary memory at startup (recommended).	
<sexp$_1$>	is the name of the image (max 30 char. incl. extension).	
<nexp$_1$>	is the width of the image in bits (=dots).	
<nexp$_2$>	is the height of the image in bits (=dots).	
[<nexp$_3$>]	is the size of the images in bytes (RLL only).	
<sexp$_2$>	is the name of the protocol:	"INTELHEX"
		"UBI00"
		"UBI01"
		"UBI02"
		"UBI03"
		"UBI10"

Remarks

The name of the protocol must be entered in one sequence (for example "INTELHEX"). Upper- or lowercase letter can be used at will. Refer to the Chapter 3, "Image Transfer" for information on protocols.

STORE IMAGE RLL is used when the image to be received is compressed into RLL format. In this case the size of the image must be included in the list of parameters (<nexp$_3$>).

STORE IMAGE KILL implies that the image will be stored in the printer's temporary memory, which is erased at power off or REBOOT. It is strongly recommended to use this option to improve the performance. If you need to store the image permanently, copy it from the temporary memory ("tmp:") to the permanent memory ("/c") after the download is completed.

A STORE IMAGE statement must precede any STORE INPUT statement.

Example

This example shows how an Intelhex file is received via the standard input channel and stored in the printer's temporary memory:

```
10   STORE OFF
20   INPUT "Name:", N$
30   INPUT "Width:", W%
40   INPUT "Height:", H%
50   INPUT "Protocol:", P$
60   STORE IMAGE N$, W%, H%, P$
70   INPUT "", F$
80   STORE F$
90   IF MID$(F$,8,2,)<>"01" THEN GOTO 70
100  STORE OFF
```

STORE INPUT

Purpose

Statement for receiving and storing protocol frames of image data in the printer's memory.

Syntax

STORE↔INPUT<nexp$_1$>[,<nexp$_2$>]

<nexp$_1$>	is the timeout in ticks (0.01 sec.) before next character is received.
<nexp$_2$>	is, optionally, the number assigned to a device when it was OPENed for INPUT. Default: Standard IN channel.

Remarks

The STORE INPUT statement receives and stores a protocol frame of image data as specified by preceding INPUT and STORE IMAGE statements. It also performs an end frame check. (STORE INPUT substitutes the old STORE statement (not documented in this manual.)

STORE INPUT works differently for various types of protocol:

INTELHEX	Receives and stores frames until timeout or end frame is received.
UBI00-03	Receives and stores frames until timeout or required number of bytes are received.
UBI10	Receives and stores frames until timeout or end frame is received.

Examples

This example shows how an Intelhex file is stored using the STORE IMAGE statement. The number of input parameters may vary depending on type of protocol, see STORE INPUT statement.

```
10    STORE OFF
20    INPUT "Name:", N$
30    INPUT "Width:", W%
40    INPUT "Height:", H%
50    INPUT "Protocol:", P$
60    STORE IMAGE N$, W%, H%, P$
70    STORE INPUT 100
80    STORE OFF
```

To receive the input from another channel than std IN channel, the device must be OPENed for INPUT and a reference be included in the STORE INPUT statement.

```
10    STORE OFF
20    OPEN "uart2:" FOR INPUT AS #9
30    INPUT "Name:", N$
40    INPUT "Width:", W%
50    INPUT "Height:", H%
60    INPUT "Protocol:", P$
70    STORE IMAGE N$, W%, H%, P$
80    STORE INPUT 100,9
90    CLOSE #9
100   STORE OFF
```

STORE OFF

Purpose

Statement for terminating the storing of an image and resetting the storing parameters.

Syntax

STORE↔OFF

Remarks

After having stored all protocol frames of an image, the storing must be terminated by a STORE OFF statement. Even if you want to store another image, you must still issue a STORE OFF statement before the parameters for the new image can be set up using a new STORE IMAGE statement.

It is recommended always to start an image storing procedure by issuing a STORE OFF statement to clear the parameters of any existing STORE IMAGE statement.

Example

This example shows how an Intelhex file is received via the standard IN channel and stored in the printer's memory:

```
10   STORE OFF
20   INPUT "Name:", N$
30   INPUT "Width:", W%
40   INPUT "Height:", H%
50   INPUT "Protocol:", P$
60   STORE IMAGE N$, W%, H%, P$
70   STORE INPUT 100
80   STORE OFF
```

STR$

Purpose Function returning the string representation of a numeric expression.

Syntax **STR$(<nexp>)**

<nexp>	is the numeric expression from which the string representation will be returned.

Remarks This is the complementary function for the VAL function.

Example In this example, the value of the numeric variable A% is converted to string representation and assigned to the string variable A$:

```
10   A%=123
20   A$=STR$(A%)
30   PRINT A%+A%
40   PRINT A$+A$
RUN
```

yields:

```
246
123123
```

STRING$

Purpose

Function repeatedly returning the character of a specified ASCII value, or the first character in a specified string.

Syntax

STRING$(<nexp$_1$>,<<nexp$_2$>|<sexp>>)

<nexp$_1$>	is the number of times the specified character should be repeated.
<nexp$_2$>	is the ASCII decimal code of the character to be repeated.
<sexp>	is a string expression, from which the first character will be repeated.

Remarks

The character to be repeated is specified either by its ASCII decimal code according to the selected character set (see NASC), or as the first character in a specified string expression.

Example

In this example, both ways of using STRING$ are illustrated. The character "*" is ASCII 42 decimal:

```
10        A$="*INTERMEC*"
20        LEADING$ = STRING$(10,42)
30        TRAILING$ = STRING$(10,A$)
40        PRINT LEADING$; A$; TRAILING$
RUN
```

yields:

```
***********INTERMEC***********
```

SYSHEALTH

Purpose

Variable for setting or getting the Fingerprint application's view of the system health and control the Intermec Readiness Indicator (IRI) on the printer's front panel.

Syntax

Setting the system health: SYSHEALTH=<nexp>

<nexp>	sets the Fingerprint application's view of the system health and controls the Intermec Readiness Indicator (IRI): 1 IRI off 2 IRI blink 3 IRI on (default)

Getting the system health: <nvar>=SYSHEALTH

<nvar>	returns the current system health status as indicated by the IRI: 1 translates to IRI off 2 translates to IRI blink 3 translates to IRI on

Remarks

The readiness of the printer, individually or as a part of a solution, is indicated by the blue Intermec Readiness Indicator (IRI).

If the IRI blinks or is switched off, the printer is not ready. Further information can be obtained in the display window by pressing the <F5/i> key. In case of several errors or similar conditions occuring simultaneously, only the most significant error is displayed. Once this error has been cleared, next remaining error is displayed.

Provided the printer is connected to a network, all conditions that prevents printing are reported to the Easy ADC Console. The Easy ADC Console is a PC-based software which allows a supervisor to monitor all connected devices that have an Intermec Readiness Indicator, including handheld computers, access points, and printers.

The SYSHEALTH variable offers the programmer the opportunity to add more functions to the IRI in an application than offered by the printer's system (see the User's Guide of the printer in question). However, it does not override the standard IRI handling; that is, the worst case is always reported regardless if it is a system error or an application error.

SYSHEALTH, cont.

Example

This example shows how the IRI can be made to indicate a "Connection refused" condition:

```
10   ON ERROR GOTO 1000
50   TRANSFER NET
     "ftp://wrong.server.com/file","c/myfile"
60   PRINT "XXX"
100  END
1000 IF ERR=1833 THEN SYSHEALTH=2 ELSE
     SYSHEALTH=3
1010 RETURN
```

You can find out the health of the system this way:

```
A%=SYSHEALTH
PRINT A%
```

SYSHEALTH$

Purpose

Function for returning the error causing the current system health status.

Syntax

<svar>=SYSHEALTH$

<svar>	returns the error causing the current status, for example "Head lifted."

Remarks

If SYSHEALTH = 3, SYSHEALTH$ returns "Operational."

Example

```
A$=SYSHEALTH$
PRINT A$
```

yields for example

```
Out of paper
```

SYSVAR

Purpose

System array for reading or setting various system variables.

Syntax

SYSVAR(<nexp>)

<nexp> is the reference number of the system variable:

0	Not intended for public use
1	Not intended for public use
2	Not implemented
3	Not intended for public use
4-8	Not implemented
9-11	Reserved for special applications
12	Not implemented
13	Obsolete
14	Read errors since power on
15	Read errors since last SYSVAR(15)
16	Read number of bytes received at execution of STORE INPUT
17	Read number of frames received at execution of STORE INPUT
18	Read or Set verbosity level
19	Read or Set type of error message
20	Read direct or transfer mode
21	Read printhead density (dots/mm)
22	Read number of printhead dots
23	Read status of transfer ribbon sensor
24	Read if startup has occurred since last SYSVAR(24)
25	Read or Set type of Centronics communication
26	Read ribbon low condition
27	Set condition for label reprinting at out-of-ribbon error
28	Set or read media feed data erase at headlift
29	Read DSR condition on "uart2:"
30	Read DSR condition on "uart3:"
31	Read last sent ACK, NAK, or CAN character in the MUSE protocol.
32	Read odometer value
33	Read DSR condition on "uart1:"
34	Read or Set positioning mode for TrueType characters
35	Setup saving (non-volatile/volatile)
36	Print changes of program modes
37	Set minimum gap length
38	Obsolete
39	Enable/disable slack compensation
40	Not implemented
41	“Next label not found” at predefined feed length
42	Stop media feed in the middle of label gaps
43	Enable/disable file name conversion
44	Enable/disable filtering of NUL characters in background communication.
45	Read printhead resolution
46	Read status of paper low sensor
47	Not implemented
48	Enable/disable bidirectional direct protocol

SYSVAR, cont.

Remarks

1-13.
Not for public use, obsolete, or not implemented.

14. Errors since power up (Read only)
Reads number of errors detected since last power up.

15. Errors since last SYSVAR(15) (Read only)
Reads number of errors detected since last executed SYSVAR(15).

16. Number of bytes received (Read only)
Reads the number of bytes received after the execution of a STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

17. Number of frames received (Read only)
Reads the number of frames received after the execution of a STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

18. Verbosity level (Set or Read)
The verbosity level can be set or read.

In the Immediate and Programming Modes, all levels are enabled by default.

In the Intermec Direct Protocol, all levels are disabled by default.

Different verbosity levels can be selected:

SYSVAR (18) = -1	All levels enabled	(= VERBON)
SYSVAR (18) = 0	No verbosity	(= VERBOFF)
SYSVAR (18) = 1	Echo received characters	
SYSVAR (18) = 2	“Ok” after correct command lines	
SYSVAR (18) = 4	Echo input characters from comm. port	
SYSVAR (18) = 8	Error after failed lines	

The levels can be combined, so for example SYSVAR(18)=3 means both “Echo received characters” and “Ok after correct command line.”

The presently selected verbosity level can also be read and is returned as a numeric value, for example by PRINT SYSVAR(18).

SYSVAR, cont.

19. Type of error message (Set or Read)
Four types of error messages can be selected:

SYSVAR(19) = 1	<string> in line <line> (default)
	for example "Invalid font in line 10"
SYSVAR(19) = 2	Error <number> in line <line>: <string>
	for example "Error 19 in line 10: Invalid font"
SYSVAR(19) = 3	E<number>
	for example "E19"
SYSVAR(19) = 4	Error <number> in line <line>
	for example "Error 19 in line 10"

The presently selected type of error message can also be read and is returned as a numeric value (1-4), for example by PRINT SYSVAR(19).

20. Direct or transfer mode (Read only)
SYSVAR(20) allows you to read if the printer is set up for direct thermal printing or thermal transfer printing, which is decided by your choice of paper type in the printer's setup.

The printer returns:
0 = Direct thermal printing
1 = Thermal transfer printing

21. Printhead density (Read only)
SYSVAR(21) allows you to read the density of the printer's printhead, expressed as number of dots per millimeter.

22. Number of dots (Read only)
SYSVAR(22) allows you to read the number of dots in the printer's printhead.

23. Transfer ribbon sensor (Read only)
SYSVAR(23) allows you to read the status of the transfer ribbon sensor in thermal transfer printers.

The printer returns:
0 = No ribbon detected
1 = Ribbon detected

SYSVAR, cont.

24. Power up since last SYSVAR(24) (Read only)

This system variable is important when using the Intermec Direct Protocol. At power up, all data not saved as programs, files, fonts or images will be deleted, and most instructions will be reset to their respective default values. SYSVAR(24) allows the host to poll the printer to see if a power up has occurred, for example because of a power failure and, if so, download new data and new instructions.

The printer returns:

0	=	No power up since last SYSVAR(24)
1	=	Power up has occurred since last SYSVAR(24)

25. Type of Centronics communication (Set or Read)

Three types of Centronics communication in the compatible mode can be selected or read. (Nibble, byte, ECP and EPP are presently not supported.)

SYSVAR(25) = 0	Standard type Predefined timing for the ACK and BUSY signals when responding to host data is: 500 ns ACK, BUSY inactivated after ACK finishes.
SYSVAR(25) = 1	IBM/Epson type Predefined timing for the ACK and BUSY signals when responding to host data is: 2500 ns ACK, BUSY inactivated as soon as ACK pulse starts.
SYSVAR(25) = 2	Classic type Predefined timing for the ACK and BUSY signals when responding to host data is: BUSY deactivated, wait 2500 ns, then give 2500 ns pulse on ACK.
Default:	0 = Standard type

26: Ribbon low condition (Read only)

This parameter allows you to read the status of the ribbon low sensor, assuming that the printer is fitted with a thermal transfer mechanism. In the Setup Mode (Media/Paper Type/Transfer/Low Diameter), you can specify a diameter in mm of the ribbon supply roll, when SYSVAR(26) will switch from 0 to 1.

The printer returns:
0 = Ribbon not low
1 = Ribbon low

By default, the Low Diameter is set to 0, which disables the ribbon low function. However, if the Low Diameter is set to a higher value than 0 and SYSVAR(26) returns 1, the error condition 1083 “Ribbon Low” occurs at every tenth PRINTFEED operation. Further actions must be taken care of by the running Fingerprint program.

SYSVAR, cont.

27: Condition for label reprinting at out-of-ribbon (Set or Read)
When printing a batch of labels using thermal transfer printing (OPTIMIZE "BATCH" ON or PRINTFEED<n>), a label is deemed erroneous and thus eligible for reprinting if the ribbon has been empty for a distance longer than specified in dots by SYSVAR(27). Default is 0. Non-negative integers only.

28. Erase media feed data at headlift (Set or Read)
The firmware keeps track of all labels (or similar) between the label stop sensor and the dot line of the printhead. If the printhead is lifted, there is a large risk that the media is moved, so the media feed will not work correctly before those labels have been fed out. This parameter allows you to decide or read whether these data should be cleared or not when the printhead is lifted.

SYSVAR(28) = 0	Media feed data are not cleared at headlift
SYSVAR(28) = 1	Media feed data are cleared at headlift (default)
SYSVAR(28) = 2	Media feed data are cleared at headlift and the firmware looks for the first gap or mark and adjusts the media feed using the same data as before the head was lifted.

29: DSR condition on "uart2:" (Read only)
This parameter allows you to read the DSR (Data Send Ready) condition on the serial channel "uart2:"

The printer returns:
0 = No
1 = Yes

.

30: DSR condition on "uart3:" (Read only)
This parameter allows you to read the DSR (Data Send Ready) condition on the serial channel "uart3:"

The printer returns:
0 = No
1 = Yes

31: Last control character sent (Read only)
This parameter allows you to read the last control character sent from the MUSE protocol (special applications).

The printer returns one of the following alternatives:
NUL
ACK
NAK
CAN

SYSVAR, cont.

32: Odometer value (Read only)
Returns the length of media feed past the printhead. Resolution: 10 meters.

33: DSR condition on "uart1:" (Read only)
This parameter allows you to read the DSR (Data Send Ready) condition on the serial channel "uart1:".

The printer returns:
0 = No
1 = Yes

34: TrueType character positioning mode (Set or Read)
This parameter allows you to select one of three modes for the positioning of TrueType characters and also to read for which mode the printer is set.

The modes are:
0 = Standard mode (default)
 This mode was introduced with Intermec Fingerprint 7.2.
1 = Compatible mode
 This mode is compatible with Intermec Fingerprint 7.xx earlier than version 7.2.
2 = Adjusted mode
 This mode was introduced with Intermec Fingerprint 7.2.

35: Setup Saving (Set or Read)
This parameter allows you to decide whether a change in the printer's setup is to be saved as a file (that is be effective after a reboot or power down) or not be saved (volatile). You can also read for which alternative the printer is set. Note that the SYSVAR (35) setting at the moment when the new setup is entered decides whether it will be saved or not.

The alternatives are:
0 = Setup saved to file (non-volatile)Default
1 = Setup not saved to file (volatile)

36: Print changes of program modes (Set or Read)
This parameter is used with the Fingerprint debugger and controls whether changes of program modes should be printed to the Debug Standard Out port (see DBSTDIO).

The options are:
0 = Disable printout (default)
1 = Enable printout

SYSVAR, cont.

37: Set Minimum Gap Length (Set or Read)
The media may have perforations or marks that not are intended to be interpreted as gaps or black marks by the LSS. Using this SYSVAR parameters, it is possible to make the LSS ignore gaps or marks that are shorter that a specified value. (In this context, long and short are related to the media feed direction.) The minimum gap length is specified in dots within a range of 1-32. Default value is 1 mm (0.039 inches). Note that SYSVAR(37) affects PRINTFEED and FORMFEED. For TESTFEED, see SYSVAR(38).

38: Set Equal Safe for TESTFEED
This parameter is obsolete and has no effect, even if it does not cause an error if used.

39: Enable/Disable Slack Compensation (Set or Read)
Label slack compensation is a method of eliminating slack in the belts after having fed the media back. At a negative FORMFEED, the printer will pullback the media slightly more than specified by the FORMFEED statement and the feed the media forward the same distance. For example, if FORMFEED -100 is specified, the printer will pull back the media -112 dots and then feed the media forward +12 dots to take out the slack.
In some applications, this method could be inconvenient, so it is possible to enable/disable it.
The options are:
0 = Disable slack compensation
1 = Enable slack compensation (default)

40: Not Implemented

41: "Next label not found" at Predefined Feed Length (Set or Read)
The automatic detection of the error condition "Next label not found" (error 1031) by the label stop sensor can be overridden by specifying a fixed length in dots. The length should preferably correspond to at least the distance between the tops of two consecutive labels. During printing, error 1031 occurs if the media does not come loose from the core (media glued to core) or if a label is missing on the liner. Especially useful for short labels (10–40 mm/0.4–1.5 inches long). Default value is 0.

SYSVAR, cont.

42: Stop Media Feed in the Middle of Label Gaps (Set or Read)

0	The media feed stops so the middle of a 3 mm (0.12 in) gap becomes aligned with the tear bar when using labels (w gaps). This is the default setting.
1	The media feed stops so the middle of the gap becomes aligned with the tear bar, regardless of gap size.

43: Enable/Disable File Name Conversion (Set or Read)

File name conversion means that lowercase characters will be converted to uppercase and the extension .PRG will be added if an extension is missing.

0	File name conversion is enabled (default)
1	File name conversion is disabled.

44: Enable/Disable filtering of NUL characters (Set or Read)

SYSVAR(44) controls the filtering of NUL characters in background communication (see COMBUF$).

0	Enables filtering (default)
1	Disables filtering

45: Read Printhead Resolution (Read only)

SYSVAR(45) returns the resolution of the printhead expressed in dots per inch (dpi).

46: Read status of Paper Low Sensor (Read only)

0	Indicates that the diameter of the media supply is larger than specified in the Setup Mode.
1	Indicates that the detected diameter of the media supply roll is equal or less than the diameter specified in the Setup Mode (Media/Paper/Low Diameter). The error condition 1084 "Paper low" will occur. This error does not stop the printing, but interrupts any program that does not handle it.

48: Enable/Disable Bidirectional Direct Protocol (Set or Read)

0	Disables use of direct commands (default)
1	Scans stdIN channel for direct commands. Can only be set in the Direct Protocol.

Examples

Reading the value of a system variable, in this case the transfer ribbon sensor:

```
PRINT SYSVAR(23)
```

Setting the value of a system variable. In this case the standard type of Centronics is selected:

```
SYSVAR(25)= 0
```

TESTFEED

Purpose

Statement for adjusting the label stop, ribbon end/low, and paper low sensors while running the media and ribbon feed mechanisms.

Syntax

TESTFEED[<nexp>]

<nexp>	is an optional feed length in dots.

Remarks

The TESTFEED statement feeds <nexp> dots while calibrating the label stop/black mark sensor (LSS) for the characteristics of the media presently loaded in the printer. The adjustment is needed to detect media, gaps, black marks, and out-of-paper conditions, therefore this must be done for all media types.

If <nexp> is omitted, <nexp> is automatically set to 1.5 times the media length specified in the setup. For the TESTFEED operation to be successful, at least one gap or black mark must pass the LSS. In case if "Ticket w Mark", the best result is obtained if entering a <nexp> value of 1200 or any other reasonable number.

When a TESTFEED is executed, the ribbon end/low and paper low sensor are also calibrated (if installed). However, this does not apply when the testfeed is ordered using the testfeed option in the Setup Mode.

In the Immediate Mode, a TESTFEED is performed when the <Shift> and <Feed> keys are pressed simultaneously.

Since the TESTFEED is essential for a proper media load, some facility for issuing a TESTFEED statement should be included in all custom-made label-printing programs (see the example below).

Example

This program performs a TESTFEED statement when the <Shift> and <Feed> key are pressed simultaneously on the printer's built-in keyboard:

```
10   ON KEY (119) GOSUB QTESTFEED
20   KEY (119) ON
30   QLOOP:
40   GOTO QLOOP
. . . . .
. . . . .
. . . . .
1000 QTESTFEED:
1010 TESTFEED
1020 RETURN
```

TICKS

Purpose

Function returning the time, that has passed since the last power up in the printer, expressed in number of “TICKS" (1 TICK = 0.01 seconds.)

Syntax

TICKS

Remarks

TICKS allows you to measure time more exactly than the TIME$ variable, which cannot handle time units smaller than 1 second.

The TICKS counter is reset to zero at power up.

Example

```
10   A%=TICKS
20   PRINT A%
RUN
```

yields for example:

```
1081287
```

The time which has passed since the printer was started is 10812.87 seconds, that is 3 hours 12.87 seconds.

TIME$

Purpose

Variable for setting or returning the current time.

Syntax

Setting the time: TIME$=<sexp>

<sexp>	sets the current time by a 6-digit number specifying Hour, Minute and Second.

Reading the time: <svar>=TIME$[(<sexp>)]

<svar>	returns the current time according to the printer's clock.
<sexp>	is an optional flag "F", indicating that the time will be returned according to the format specified by FORMAT TIME$.

Remarks

This variable works best if a real-time clock circuit (RTC) is fitted on the printer's CPU board. The RTC is battery backed-up and will keep record of the time even if the power is turned off or lost.

If no RTC is installed, the internal clock will be used. After startup, an error will occur when trying to read the date or time before the internal clock has been manually set using either a DATE$ or a TIME$ variable. If only the date is set, the internal clock starts at 00:00:00 and if only the time is set, the internal clock starts at Jan 01 1980. After setting the internal clock, you can use the DATE$ and TIME$ variables the same way as when an RTC is fitted, until a power off or REBOOT causes the date and time values to be lost.

Time is always entered and, by default, returned as HHMMSS, where:

HH	=	Hour	Two digits (00-23)
MM	=	Minute	Two digits (00-59)
SS	=	Second	Two digit (00-59)

Time is entered as a 24-hour cycle, for example 8 o'clock pm is entered as "200000".

The clock will be reset at the exact moment, when the appending carriage return character is received, for example when you press the Return key (Immediate Mode and Intermec Direct Protocol), or when the instruction is executed (Programming Mode).

The format for how the printer will return time from a TIME$("F") variable can be changed using a FORMAT TIME$ statement.

Example

Setting and reading the time, then printing it on the screen of the host:

```
10   TIME$ = "154300"
20   FORMAT TIME$ "HH.MM"
30   PRINT "Time is "+TIME$("F")
RUN
```

yields:

```
Time is 15.43
```

TIMEADD$

Purpose

Function returning a new time after a number of seconds have been added to, or subtracted from, the current time or optionally a specified time.

Syntax

TIMEADD$([<sexp$_1$>,]<nexp>[,<sexp$_2$>])

<sexp$_1$>	is any time given according to the TIME$ format, which a certain number of seconds should be added to or subtracted from.
<nexp>	is the number of seconds to be added to (or subtracted from) the current time, or optionally the time specified by <sexp$_1$>.
<sexp$_2$>	is an optional flag "F", indicating that the time will be returned according to the format specified by FORMAT TIME$.

Remarks

The original time (<sexp$_1$>) should always be entered according to the TIME$ format (HHMMSS), where:

HH	=	Hour	Two digits (00-23)
MM	=	Minute	Two digits (00-59)
SS	=	Second	Two digits (00-59)

Time is entered as a 24-hour cycle, for example 8 o'clock pm is entered as "200000".

The number of seconds to be added or subtracted from the original time should be specified as a positive or negative numeric expression respectively.

If no "F" flag is included in the TIMEADD$ function, the result will be returned according to the TIME$ format, see above.

If the TIMEADD$ function includes an "F" flag, the result will be returned in the format specified by FORMAT TIME$.

Examples

```
10   A%=30
20   B$=TIMEADD$ ("133050",A%)
30   PRINT B$
RUN
```

yields:

```
133120
```

```
10   TIME$="133050"
20   FORMAT TIME$ "hh.mm.ss pp"
30   A% = -40
40   PRINT TIMEADD$(A%,"F")
RUN
```

yields:

```
01.30.10 pm
```

TIMEDIFF

Purpose

Function returning the difference between two specified moments of time in number of seconds.

Syntax

TIMEDIFF(<sexp$_1$>,<sexp$_2$>)

<sexp$_1$>	is one of two moments of time (time 1).
<sexp$_2$>	is the other of the two moments (time 2).

Remarks

To get the result as a positive value, the two moments of time, for which the difference is to be calculated, should be entered with the earlier moment (time 1) first and the later moment (time 2) last, see the first example below.

If the later moment (time 2) is entered first, the resulting value will be negative, see the second example below.

The time should be entered according to the format for the TIME$ variable, that is in the order HHMMSS, where:

HH	=	Hour	Two digits (00-23)
MM	=	Minute	Two digits (00-59)
SS	=	Second	Two digits (00-59)

Time is entered as a 24-hour cycle, for example 8 o'clock pm is entered as "200000".

The resulting difference in seconds will be returned.

Examples

```
PRINT TIMEDIFF ("133050","133120")
```

yields:

```
30
```

```
PRINT TIMEDIFF ("133120","133050")
```

yields:

```
-30
```

TRANSFER KERMIT

Purpose

Statement for transferring data files using KERMIT communication protocol.

Syntax

TRANSFER$_{\leftrightarrow}$K[ERMIT]<sexp$_1$>[,<sexp$_2$>[,<sexp$_3$>[,sexp$_4$>]]]

<sexp$_1$>	specifies the direction of the transmission by the expression "S " (= send) or "R" (= receive).
<sexp$_2$>	is, optionally, the name of the file transmitted from the printer (default "KERMIT.FILE").
<sexp$_3$>	specifies, optionally, the input device as "uart1:", "uart2:", or "uart3:" (default: std IN channel).
<sexp$_4$>	specifies, optionally, the output device as "uart1:", "uart2:", or "uart3:" (default: std OUT channel).

Remarks

Kermit is a protocol for serial binary transfer of a complete file between for example a PC and a printer. Kermit is included in Windows HyperTerminal and in many other communication programs.

Warning, tests have shown that Microsoft Windows Terminal, versions 3.0 and 3.1, is unable to receive a file from the printer, even if capable of sending a file to the printer.

Consult the application program manual or the reference volume "Kermit -A File Transfer Protocol" by Frank da Cruz (Digital Press 1987, ISBN 0-932376-88-6).

TRANSFER KERMIT can only handle one single file at a time.

When transmitting files from the printer to the host, carefully observe possible restrictions concerning the number of characters, etc. in the file name, that is imposed by the operating system of the host.

When receiving a file, you must start the transmission within 30 seconds from completing the TRANSFER KERMIT "R" statement. The printer will store the file in the current directory "/c", "tmp:", or "card1:", see CHDIR statement. (Obviously, files cannot be received into "/rom".) If there already exists a file in the current directory with the same name as the one to be transferred, the existing file will be replaced by the new file. Thus, it is up to you to keep record of the files already stored in the current directory (see FILES statement). Before transfer, give the new file a name that is not already occupied by an existing file, unless you want to replace the existing file. If you use TRANSFER KERMIT to download a font or image file, the font or image will automatically be installed after the downloading is completed without any need for a reboot.

TRANSFER KERMIT, cont.

Examples

Setting up the printer for file reception on the standard IN channel:

```
TRANSFER KERMIT "R"
```

Transmission from printer to host of the file "FILE1.TXT" on a channel other than the standard OUT channel:

```
TRANSFER K "S","FILE1.TXT","uart2:","uart2:"
```

TRANSFER NET

Purpose

Statement for transferring files to and from the printer using FTP.

Syntax

TRANSFER N[ET] <sexp$_1$>,<sexp$_2$>[,<sexp$_3$>]

<sexp$_1$>	is the source. If the source is a local file, this file will be sent from the printer to the destination specified by <sexp$_2$>. If the source is a URI, this file will be fetched from the server and sent to the printer and stored at the location specified by <sexp$_2$>.
<sexp$_2$>	is the destination of the file transfer.
<sexp$_3$>	is an optional account secret.

Remarks

Limitations:
TRANSFER NET is not a complete ftp client. It only supports file transfer to and from the printer in binary format. Only one file can be transferred per command. File transfer between two local or two remote files is not supported.

Local files:
A local file is a path to an existing file (when sending from the printer) or to the file that will be created (when fetching to the printer). If a local file already exists when fetching a file to the printer, the existing file will be replaced, if it is not write-protected. A read-protected file will not be sent. If the destination is a local directory, the fetched file will get the same name as the source file.

URIs:
An URI shall be entered in the format

```
ftp://[<user>:<password>@]<server>[:port]/<path>
```

Entries inside square brackets [...] are optional. The following default values are used:

- user:	anonymous
- password:	nopass@<ip address>
- port:	21

If the destination is a URI specifying a directory, the sent file will get the same name as the source file.

Account Secret:
If the user does not want to reveal his/her username and password in plain text in a Fingerprint program, the account secret option can be used. The account secret holds the secret information and cannot be read by any user. To create an account secret, use the external command RUN "secret". Listing and deleting accounts secrets are reserved for admin.

Fonts and Images:

Downloaded fonts and images will be auto-installed.

TRANSFER NET, cont.

Example

This example shows how the file README.uploads is fetched from sunet's ftp server and stored as UPLOAD.TXT in the current directory. Default user, password and port number are used.

```
TRANSFER NET "ftp://ftp.sunet.se/README.uploads",
"UPLOAD.TXT"
```

TRANSFER STATUS

Purpose

Statement for checking last TRANSFER KERMIT or TRANSFER ZMODEM operation.

Syntax

TRANSFER↔S[TATUS]<nvar>,<svar>

<nvar>	is a five-element one-dimensional numeric array where the elements will return:	
	0: Number of packets.	(Kermit only)
	1: Number of NAK's.	(Kermit only)
	2: ASCII value of last status character.	(Kermit only)
	3: Last error.	(Kermit and ZMODEM)
	4: Block check type used.	(Kermit only)
<svar>	is a two-element one-dimensional string array where the elements will return:	
	0: Type of protocol.	("KERMIT" or "ZMODEM")
	1: Last file name received.	

Remarks

After a file transfer using the Kermit or ZMODEM protocol has been performed (see TRANSFER KERMIT and TRANSFER ZMODEM statements), you can check how the transfer was performed. Note that the numeric array requires the use of a DIM statement, since the array will contain more than four elements.

Example

```
10   TRANSFER KERMIT "R"
20   DIM A%(4)
30   TRANSFER STATUS A%, B$
40   PRINT A%(0), A%(1), A%(2), A%(3), A%(4)
50   PRINT B$(0), B$(1)
.....
.....
.....
```

TRANSFER ZMODEM

Purpose

Statement for transferring data files using ZMODEM communication protocol.

Syntax

TRANSFER↔Z[MODEM]<sexp$_1$>[,<sexp$_2$>[,<sexp$_3$>[,sexp$_4$>]]]

<sexp$_1$>	specifies the direction of the transmission by the expression "S " (= send) or "R" (= receive).
<sexp$_2$>	is, optionally, the name of the file transmitted from the printer (default "ZMODEM.FILE").
<sexp$_3$>	specifies, optionally, the input device as "uart1:", "uart2:", or "uart3:" (default: std IN channel).
<sexp$_4$>	specifies, optionally, the output device as "uart1:", "uart2:", or "uart3:" (default: std OUT channel).

Remarks

ZMODEM is a protocol for serial transfer of a complete file between for example a PC and a printer. For more information on the ZMODEM protocol, please refer to http://www.omen.com. Related instructions are RZ (receive data using the ZMODEM protocol) and SZ (send data using the ZMODEM protocol).

TRANSFER ZMODEM can only handle one single file at a time.

When transmitting files from the printer to the host, carefully observe possible restrictions concerning the number of characters etc. in the file name, that is imposed by the operating system of the host.

When receiving a file, you must start the transmission within 30 seconds from completing the TRANSFER ZMODEM "R" statement. The printer will store the file in the current directory "/c", "tmp:", or "card1:", see CHDIR statement. (Obviously, files cannot be received into "/rom".) If there already exists a file in the current directory with the same name as the one to be transferred, the existing file will be replaced by the new file. Thus, it is up to you to keep record of the files already stored in the current directory (see FILES statement). Before transfer, give the new file a name that is not already occupied by an existing file, unless you want to replace the existing file. If you use TRANSFER ZMODEM to download a font or image file, the font or image will automatically be installed after the downloading is completed without any need for a reboot.

Examples

Setting up the printer for file reception on the standard IN channel:

```
TRANSFER ZMODEM "R"
```

Transmission from printer to host of the file "FILE1.TXT" on a channel other than the standard OUT channel:

```
TRANSFER Z "S","FILE1.TXT","uart2:","uart2:"
```

TRANSFER$

Purpose

Function executing a transfer from source to destination as specified by a TRANSFERSET statement.

Syntax

TRANSFER$(<nexp>)

<nexp>	is the character time-out in ticks (10 milliseconds).

Remarks

The TRANSFER$ function executes the transfer from source to destination as specified by the TRANSFERSET statement. It also checks the transfer and breaks it if no character has been transmitted before the specified time-out has expired or if any break character, as specified by the break character string in the TRANSFERSET statement, is encountered.

If the transmission was interrupted because a character in the break set was encountered, that character will be returned.

If the transmission was interrupted because of a time-out error, an empty string will be returned.

If the transmission was interrupted because of the reception of a character on any other communication channel than the source (as specified by TRANSFERSET statement), an empty string will be returned.

Example

The transfer will be executed by the TRANSFER$ function in line 60 and possible interruptions will be indicated by a break character or empty string ("") in the string variable C$.

```
10   OPEN "LABEL1.PRG" FOR INPUT AS #1
20   OPEN "UART1:" FOR OUTPUT AS #2
30   A$=CHR$(13)
40   B$=CHR$(10)
50   TRANSFERSET #1, #2, A$+B$
60   C$=TRANSFER$(100)
.....
.....
.....
```

TRANSFERSET

Purpose Statement for entering setup for the TRANSFER$ function.

Syntax **TRANSFERSET[#]<nexp$_1$>,[#]<nexp$_2$>,<sexp>[,<nexp$_3$>]**

#	optional number sign.
<nexp$_1$>	is the number of the source (the file or device OPENed for input).
<nexp$_2$>	is the number of the destination file (the file or device OPENed for output or append).
<sexp>	is a set of break characters.
<nexp$_3$>	optionally enables or disables break on any other channel than the source: <nexp> = 0 Break disabled <nexp> ≠ 0 Break enabled
Default:	Standard I/O with no break characters. Break on any other channel enabled.

Remarks This statement sets up the transfer of data from a file or device OPENed for input to another file or device OPENed for output or append. The transfer will be interrupted if any character in a string of break characters, specified in this statement, is encountered (optionally on another specified channel). The actual transfer is executed by means of a TRANSFER$ function, that also returns the break character that has caused any possible interruption.

Example In this example, the data transfer from a file in the current directory to an external device connected to the communication port "uart1:" will be interrupted as soon as a carriage return or a line feed character is encountered in the file.

```
10   OPEN "LABEL1.PRG" FOR INPUT AS #1
20   OPEN "UART1:" FOR OUTPUT AS #2
30   A$=CHR$(13)
40   B$=CHR$(10)
50   TRANSFERSET #1, #2, A$+B$
60   C$=TRANSFER$(100)
.....
.....
.....
```

TRON/TROFF

Purpose Statements enabling/disabling tracing of the program execution.

Syntax **TRON|TROFF**

TRON	enables tracing.
TROFF	disables tracing (default)

Remarks Useful for debugging purposes. When tracing is enabled, each line number of the program is displayed on the screen within parentheses as the execution goes on.

Tracing will be disabled when a TROFF statement is executed.

Example

```
10   PRINT "HELLO"
20   INPUT"Enter Text"; A$
30   PRINT A$
TRON
RUN
```

yields:

```
(10)  HELLO
(20)  Enter text?                (Operator enters "WORLD")
(30)  WORLD
```

VAL

Purpose

Function returning the numeric representation of a string expression.

Syntax

VAL(<sexp>)

<sexp>	is the string expression from which the numeric representation will be returned.

Remarks

VAL is the complementary function for STR$.

VAL ignores space characters from the argument string to determine the result.

If the first character in the string expression is anything else but a digit, a plus sign, or a minus sign, the VAL function returns the value 0.

Example

In this example, the values of the string variables A$ and B$ are read and assigned to the numeric variables A% and B%:

```
10   A$="123, MAIN STREET"
20   A%=VAL(A$)
30   B$="PHONE 123456"
40   B%=VAL(B$)
50   PRINT A$
60   PRINT A%
70   PRINT B$
80   PRINT B%
RUN
```

yields:

```
123, MAIN STREET
123
PHONE 123456
0
```

VERBON/VERBOFF

Purpose

Statements for specifying the verbosity level of the communication from the printer on the standard OUT channel (serial communication only).

Syntax

VERBON|VERBOFF

VERBON	enables all verbosity levels (default).
VERBOFF	disables all verbosity levels.

Remarks

VERBON
By default, when a character is received on the standard IN channel (see SETSTDIO statement), the corresponding character will be echoed back on the standard OUT channel. As the serial channel "uart1:" is by default selected both standard IN and OUT channel, this implies that when you enter a character on the keyboard of the host, the same character will appear on the screen after having been transmitted to the printer and back.

When an instruction has been successfully executed, “Ok” will be displayed on the screen, else an error message will be returned. Obviously, this requires two-way communication, so verbosity has no meaning in case of the parallel "centronics:" communication protocol.

VERBON corresponds to SYSVAR(18) = -1.

VERBOFF corresponds to SYSVAR(18) = 0.

Other verbosity levels can be selected using SYSVAR(18), and the type of error message can be selected using SYSVAR (19).

VERBOFF
All responses will be suppressed, which means that no characters or error messages will be echoed back. VERBOFF statements do not affect question marks and prompts displayed as a result of an INPUT statement. Instructions like DEVICES, FILES, FONTS, IMAGES, LIST, and PRINT will also work normally.

Example

This example shows how VERBOFF is used to suppress the printing of INPUT data in lines 20 and 40 during the actual typing on the host, and VERBON to allow the printing of the resulting string variables on the screen:

```
10   FOR Q%=1 TO 6
20   VERBOFF:INPUT "", A$
30   VERBON:PRINT A$;
40   VERBOFF:INPUT "", B$
50   VERBON
60   C$=SPACE$(25-LEN(A$))
70   PRINT C$+B$
80   NEXT Q%
90   END
```

VERSION$

Purpose

Function returning the version of the firmware, printer family, or type of CPU board.

Syntax

VERSION$[(<nexp>)]

<nexp>	is, optionally, the type of information to be returned: 0 = Version of firmware (default) 1 = Printer family 2 = Type of CPU board

Remarks

The name of the firmware depends on if the printer is running in the Immediate or Programming Modes, or in the Intermec Direct Protocol.

The printer family is returned as one of the following alternatives:
PF2i
PF4i
PM4i

The type of CPU-board is returned as a string of text, for example:

hardware version 4.0	(EasyCoder PF2i, PF4i, PM4i)

Examples

```
PRINT VERSION$(0)
```

yields for example:

```
Fingerprint 8.00
```

```
PRINT VERSION$(1)
```

yields for example:

```
PF4i
```

```
PRINT VERSION$(2)
```

yields for example:

```
hardware version 4.0
```

WEEKDAY

Purpose

Function returning the weekday of a specified date.

Syntax

WEEKDAY(<sexp>)

<sexp>	is the date in DATE$ format from which the weekday will be returned.

Remarks

This function returns the weekday as a numeric constant:
1 = Monday
2 = Tuesday
3 = Wednesday
4 = Thursday
5 = Friday
6 = Saturday
7 = Sunday

The date should be entered according to the syntax for the DATE$ variable, that is in the following order:

YY	=	Year	Last two digits	(for example 2003 = 03)
MM	=	Month	Two digits	(01-12)
DD	=	Day	Two digits	(01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

Example

In this example the weekday for the current date is printed on the screen of the host (another way is to use NAME WEEKDAY$ statement and WEEKDAY$ function):

```
10    B$=DATE$
20    A% = WEEKDAY (B$)
30    IF A% = 1 THEN PRINT "MONDAY"
40    IF A% = 2 THEN PRINT "TUESDAY"
50    IF A% = 3 THEN PRINT "WEDNESDAY"
60    IF A% = 4 THEN PRINT "THURSDAY"
70    IF A% = 5 THEN PRINT "FRIDAY"
80    IF A% = 6 THEN PRINT "SATURDAY"
90    IF A% = 7 THEN PRINT "SUNDAY"
RUN
```

yields for example:

```
THURSDAY
```

WEEKDAY$

Purpose

Returning the name of the weekday from a specified date.

Syntax

WEEKDAY$(<sexp>)

<sexp>	is the date for which the name of the weekday, according to a list of weekday names created by means of NAME WEEKDAY$ statement , will be returned.

Remarks

This function returns the name of the weekday according to a list of weekday names specified by means of NAME WEEKDAY$ statement or—if the name is missing—the full English name in lowercase characters, for example "friday".

The date should be entered according to the syntax for the DATE$ variable, that is in the following order:

YY	=	Year	Last two digits	(for example 2003 = 03)
MM	=	Month	Two digits	(01-12)
DD	=	Day	Two digits	(01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

Example

This example shows how to make the printer return the name of the weekday as a three-letter English abbreviation in connection with a formatted date:

```
10   FORMAT DATE$ ", MM/DD/YY"
20   DATE$="031201"
30   NAME WEEKDAY$ 1, "Mon"
40   NAME WEEKDAY$ 2, "Tue"
50   NAME WEEKDAY$ 3, "Wed"
60   NAME WEEKDAY$ 4, "Thu"
70   NAME WEEKDAY$ 5, "Fri"
80   NAME WEEKDAY$ 6, "Sat"
90   NAME WEEKDAY$ 7, "Sun"
100  PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN
```

yields:

```
MON, 12/01/03
```

WEEKNUMBER

Purpose

Function returning the number of the week for a specified date.

Syntax

WEEKNUMBER(<sexp>[,<nexp])

<sexp>	is the date for which the week number will be returned (1-53).
<nexp>	specifies the calculating function (0-14) as listed below. Default is 0.

Remarks

WEEKNUMBER calculating function:

<nexp>	Week #1 starts...
0	according to ISO 8601 (European standard): • week #1 will start on the last Monday at or before the New Year, if January 1 occurs on a Monday, Tuesday, Wednesday, or Thursday. • week #1 will start on the first Monday after the New Year, if January 1 occurs on a Friday, Saturday, or Sunday.
1	at Sunday in the first week with 7 days in the actual year
2	at January 1:st, with each following week starting on a Sunday
3	at Monday in the first week with 7 days in the actual year
4	at January 1:st, with each following week starting on a Monday
5	at Tuesday in the first week with 7 days in the actual year
6	at January 1:st, with each following week starting on a Tuesday
7	at Wednesday in the first week with 7 days in the actual year
8	at January 1:st, with each following week starting on a Wednesday
9	at Thurday in the first week with 7 days in the actual year
10	at January 1:st, with each following week starting on a Thursday
11	at Friday in the first week with 7 days in the actual year
12	at January 1:st, with each following week starting on a Friday
13	at Saturday in the first week with 7 days in the actual year
14	at January 1:st, with each following week starting on a Saturday

The date should be entered according to the syntax for the DATE$ variable, that is in the following order:

YY	=	Year	Last two digits	(for example 2003 = 03)
MM	=	Month	Two digits	(01-12)
DD	=	Day	Two digits	(01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

Examples

This example returns the week number of December 29, 2002 using calculating function 2:

```
PRINT WEEKNUMBER ("031229",2)
```

yields for example:

```
53
```

WHILE...WEND

Purpose

Statement for executing a series of statements in a loop providing a given condition is true.

Syntax

WHILE <nexp>
<stmt>
[...<stmt>]
WEND

<nexp>	is a numeric expression that is either TRUE (-1) of FALSE (0).
<stmt>	is a statement, or a list of statements on separate lines, that will be executed provided <nexp> is TRUE.

Remarks

If <nexp> is TRUE, all following statements will be executed successively until a WEND statement is encountered. The program execution then goes back to the WHILE statement and repeats the process, provided <nexp> still is TRUE.

If <nexp> is FALSE, the execution resumes at the statement following the WEND statement.

WHILE...WEND statements can be nested. Each WEND matches the most recent WHILE statement.

Example

In this example, the WHILE...WEND loop will only be executed if the character “Y” (ASCII 89 dec.) is entered on the keyboard of the host.

```
10    B%=0
20    WHILE B%<>89
30    INPUT "Want to exit? Press Y=Yes or N=No ",A$
40    B%=ASC(A$)
50    WEND
60    PRINT "The answer is Yes"
70    PRINT "You will exit the program"
80    END
RUN
```

yields:

```
Want to exit? Press Y=Yes or N=No    N
Want to exit? Press Y=Yes or N=No    Y
The answer is Yes
You will exit the program
```

XORMODE ON/OFF

Purpose

Statement for enabling or disabling the xor/flip mode of Intermec Fingerprint in connection with graphical operations.

Syntax

XORMODE ON|OFF

Remarks

When XORMODE is set ON, dots are reversed, as opposed to set, by all graphical operations except bar codes. This means that if, for example two black lines cross, the intersection will be white. If XORMODE is set to OFF, the intersection will be black.

Default is XORMODE OFF. XORMODE is automatically set to default when a PRINTFEED statement is executed or a Fingerprint program has been successfully run.

Example

The following program illustrates the difference between XORMODE ON and XORMODE OFF. The two lines to the left are drawn with XORMODE disabled and the lines to the right with XORMODE enabled.

```
10   XORMODE OFF
20   PRPOS 0,50
30   PRLINE 300,30
40   DIR 4
50   PRPOS 100,0
60   PRLINE 200,30
70   XORMODE ON
80   DIR 1
90   PRPOS 400,50
100  PRLINE 300,30
110  DIR 4
120  PRPOS 500,0
130  PRLINE 200,30
140  PRINTFEED
RUN
```

External Command; Account Secret

Purpose

Creating an Account Secret for use with the TRANSFER NET statement.

Syntax

secret [-t]<application> <name> <string> **(create a secret)**

For user /admin/ two more functions are available:

secret –rm <application> <name> **(delete a secret)**

secret –l **(list all secrets)**

-t	Temporary (will be removed at next reboot). Optional.
-rm	Remove
-l	List
<application>	ftp
<name>	is the name of secret.
<string>	is the secret string.

Remarks

The user may not want to have his/her username and password in plain text in a Fingerprint program. Instead of writing the account info in the URI, the <account secret> parameter of TRANSFER NET can be used. The account secret holds the secret information and cannot be read by any user.

For the application ftp, <string> should have the following structure:
`<user>[:passwd]@<server>` where <server> is:
`<server name>|<server name>:<port number>|*`
If `<server>` is set to "*", the server name supplied in the URI will be used. This means that this account secret can be used with any server.

Account info, that is, user, password, server name and port number, can be stated in both the account secret and the URI. Any parameter supplied in the account secret will have precedence over parameters supplied in the URI. This means that if, for example, the URI states that port 25 should be used and the account secret says port 21, then port 21 will be used.

Examples

Create a temporary account secret, my_account, and use it to send the file beta1.bin in directory /tmp to the server TheServer. The sent file will get the name beta1.bin and will be put in the home directory of myusername on TheServer.

First of all, set SYSVAR(43) to 1 to avoid file name conversion:
```
SYSVAR(43)=1
```

```
RUN "secret -t ftp my_account myusername:mypass-
word@*"
```

```
TRANSFER NET "/tmp/beta1.bin","ftp://TheServer/",
"my_account"
```

External Command; Account Secret, cont.

Create a permanent account secret, frodo, and use it to send the file MY.TXT in the current directory to the server Frodo. The file YOUR.TXT will be put in the directory /absolute/path/.

```
RUN "secret ftp frodo frodo_username:frodo_
password@Frodo"

TRANSFER NET "MY.TXT","ftp://Frodo//absolute/
path/YOUR.TXT","frodo"
```

The server name in the URI will not be used since there is a server name in the account secret. Hence the two following command lines will have the same effect:

```
TRANSFER NET "MY.TXT",ftp://What_ever//absolute/
path/YOUR.TXT,"frodo"
```

and

```
TRANSFER NET "MY.TXT","ftp:////absolute/path/
YOUR.TXT","frodo"
```

External Command; Dynamic Modules

Purpose

External commands for inserting, listing, and removing dynamic modules in the running kernel.

Syntax

RUN "insmod <device><file>" **(insert module)**

<device>	is the device where the dynamic module is stored.
<file>	is the name of the dynamic module.

RUN "rl m" **(list modules)**

<device>	is the device where the dynamic module is stored.
<name>	is the name of the dynamic module.

RUN "rmmod <name>" **(remove module)**

<name>	is the name of the dynamic module.

Remarks

At startup, the kernel is copied from the printer's flash memory ("rom:") to the printer's temporary SDRAM memory ("tmp:"), where it is executed. In order to save space in the SDRAM memory, dynamic program modules could be downloaded and linked to the running kernel when they are needed and be removed when they are not needed any longer.

Dynamic modules also allow new or custom-made program modules to be added to an existing version of Intermec Fingerprint. Previously, it was necessary to create a new Fingerprint version for each new module or combination of modules.

In case of bar codes, the **run "insmod"** command is executed automatically if a bar code referred to in a BARSET or BARTYPE statement is not found in the running kernel. If the bar code still is not found, an error occurs. For all other types of dynamic modules, the run "insmod" command must be perfomed manually. The downloading is made "on-the-fly", so there is no need to reboot the printer or even to stop program execution.

The method of creating dynamic modules is outside the scope of this manual and is not publically released.

Note that insmod, rl m, and rmmod must be entered as lowercase characters.

External Command; ZMODEM

Purpose

External commands for receiving and sending data using the ZMODEM protocol.

Syntax

RUN "rz [<switches>] [<filename>]" (receive data)

<switches>:	
-c	Forces no crash recovery, even if sender requests ZCRESUM (resume interrupted file transfer).
-e	Print last error to std OUT channel.
-l[<logfile>]	Send verbose output to logfile. Default logfile name is "tmp:.zmodemlog".
-r	If ZMCLOB is not set and the file already exists, replace file if the transfer is successful.
-v[<level>]	Set verbosity level. Level is a decimal number. Default level is 1.
-u	Translate file name to uppercase. If a filename is given as parameter, no translation is done.
<filename>	is optionally the name under which the file will be saved.

RUN "sz [<switches>] [<filename>]" (send data)

<switches>:	
-l[<logfile>]	Send verbose output to logfile. Default logfile name is "tmp:.zmodemlog".
-v[<level>]	Set verbosity level. Level is a decimal number. Default level is 1.
<filename>	is the name of the file.

Remarks

Note that rz and sz must be entered in lowercase characters.

If a file name is given in the rz statement, this name overrides the name given by the transmitting unit.

For more information on the ZMODEM protocol, please refer to http://www.omen.com. Related instruction is TRANSFER ZMODEM.

3 Image Transfer

This chapter describes the various image transfer file protocols used in Intermec Fingerprint v8.20.

Protocols

The following five image transfer file protocols are used in connection with the STORE IMAGE statement and use a common format for the image data, as descibed on next page.

Intelhex

Intel hex [Intel Hexadecimal Intellec 8/MDS (I_hex) file format] is a well-known standard format for transfer of bitmap images. Please refer to the standard literature on the subject.

Note that:
• Hex digits in Intelhex frames must be uppercase.
• Null frames may be omitted.
• Frames can be received in any order.
• Maximum file size is 64 kbytes.

UBI00

Each frame contains:

<data bytes>

<data bytes>	Binary images. Modulo 2 bytes.

UBI01

Each frame of data contains:

<data bytes> <checksum>

<data bytes>	Binary images. Modulo 2 bytes.
<checksum>	Modulo 65536 byte-wise sum of what is defined in protocol of “data bytes.” 2 byte binary. MSB, LSB.

UBI02

Each frame of data contains:

<number of data bytes> <data bytes> <checksum>

<number of data bytes>	2 bytes binary. MSB, LSB.
<data bytes>	Binary images. Modulo 2 bytes.
<checksum>	Modulo 65536 byte-wise sum of what is defined in protocol of “number of data bytes” and “data bytes.” 2 byte binary. MSB, LSB.

UBI03

Each frame of data contains:

<start of frame id.> <number of data bytes> <data bytes> <checksum>

<start of frame id.>	1 byte (ASCII 42 dec = "*").
<number of data bytes>	2 bytes binary. MSB, LSB.
<data bytes>	Binary images. Modulo 2 bytes.
<checksum>	Modulo 65536 byte-wise sum of what is defined in protocol of “start of frame id” and “number of data bytes” and ”data bytes.” 2 byte binary. MSB, LSB.

Image Format

The following image format is valid for Intelhex, UBI00, UBI01, UBI02, and UBI03 image transfer protocols, but not for the UBI10 protocol, which is a combined image transfer protocol and format.

A bitmap picture can be encoded in one of two ways, as a plain bit representation or encoded with a Run Lenght Limited (RLL) algoritm.

Pictures can be magnified, by the printer, up to four times independently in both x and y directions.

The pictures can be rotated 180 degres by the printer (that is printed upside-down.) To print a bitmap in all four directions you have to define two bitmaps, one straight and one rotated 90 degrees. To comply with the Intermec Fingerprint convention, use the extension .1 for the straight bitmap and extension .2 for the rotated one.

Bitmap pictures, in both encoding schemes, are printed with the lowest address first, that is the first row of defined data is the first thing out. (This may be somewhat confusing. The only result, if you misinterpret this, is that your picture will come out upside-down.)

Bitmap pattern, bit representation

The bitmap picture is encoded word oriented (16 bits), low byte first. The bits in each byte is read from lsb first (bit 0.)

Bitmap pattern, Run Lenght Limited (RLL)

RLL encoding is a very efficient way of compressing big bitmaps with relatively big black and/or white areas.

The RLL encoded picture is encoded byte oriented (8 bits.) Each byte represents the number of consecutive black or white dots. The sum of bytes for each row must equal the width of the pattern. The first byte represent white dots, the second black and so on. The last byte must alter the color back to white. If the first dot is black just enter a zero first. Valid values for dot fields is 0 to 127 (0 to 7f hex.) To get a row longer then 127, concatenate two rows with zero, for example to get a row of 240 dots, enter 128,0,112.

The next step in our RLL encoding algoritm is to compress identicals rows, two identical rows are compressed by adding a byte in both ends of the dot row, the valid range for these bytes are -1 to -128 (ff to 80 hex.)

Example 1: Bitmap encoding

To clarify this, lets try a simple example. X’s represent black dots in the final printout. The pattern shown is 22 bits wide and 28 rows high.

Note:

- The bit order in each byte. Note also word fill to nearest word (16 bit).
- To the right is a hex representation of the pattern, as it would appear in a memory dump.
- To get the pattern to appear as printed on this page with direction one, the last row (row 27) should have the lowest address.

```
          |byte 3  |byte 2 |byte 1 |byte 0 |
           76543210765432107654321076543210

row      0 ..........XXXXXXXXXXXXXXXXXXXXXX          ff,ff,3f,00
         1 ..........X....................X          01,00,20,00
         2 ..........X..............XX....X          61,00,20,00
         3 ..........X.............X.X....X          a1,00,20,00
         4 ..........X............X..X....X          21,01,20,00
         5 ..........X...........X...X....X          21,02,20,00
         6 ..........X..........X....X....X          21,04,20,00
         7 ..........X.........X.....X....X          21,08,20,00
         8 ..........X........X......X....X          21,10,20,00
         9 ..........X...... X.......X....X          21,20,20,00
        10 ..........X......X........X....X          21,40,20,00
        11 ..........X.....X.........X....X          21,80,20,00
        12 ..........X....X..........X....X          21,00,21,00
        13 ..........X...X...........X....X          21,00,22,00
        14 ..........X..X............X....X          21,00,24,00
        15 ..........X.X.............X....X          21,00,28,00
        16 ..........X.XXXXXXXXXXXXXXXXXX.X          fd,ff,2f,00
        17 ..........X...............X....X          21,00,20,00
        18 ..........X...............X....X          21,00,20,00
        19 ..........X...............X....X          21,00,20,00
        20 ..........X...............X....X          21,00,20,00
        21 ..........X...............X....X          21,00,20,00
        22 ..........X...............X....X          21,00,20,00
        23 ..........X...............X....X          21,00,20,00
        24 ..........X...............X....X          21,00,20,00
        25 ..........X.............XXXXX..X          f9,03,20,00
        26 ..........X....................X          01,00,20,00
        27...........XXXXXXXXXXXXXXXXXXXXXX          ff,ff,3f,00
```

Example 2: RLL Encoding

To clarify this, lets try a simple example. X’s represent black dots in the final print out. The pattern shown is 22 bits wide and 32 rows high.

Note:

- Notice the reverse byte order. Count dots from right.
- To the right is a decimal representation of the pattern.
- To get the pattern to appear as printed on this page with direction one, the last row (row 27) should have the lowest address. Row 18 until 24 is repeted by the data in row 17.

```
row     0 XXXXXXXXXXXXXXXXXXXXXX          0,22,0
        1 X....................X          0,1,20,1,0
        2 X..............XX....X          0,1,4,2,14,1,0
        3 X.............X.X....X          0,1,4,1,1,1,13,1,0
        4 X............X..X....X          0,1,4,1,2,1,12,1,0
        5 X...........X...X....X          0,1,4,1,3,1,11,1,0
        6 X..........X....X....X          0,1,4,1,4,1,10,1,0
        7 X.........X.....X....X          0,1,4,1,5,1,9,1,0
        8 X........X......X....X          0,1,4,1,6,1,8,1,0
        9 X...... X.......X....X          0,1,4,1,7,1,7,1,0
       10 X......X........X....X          0,1,4,1,8,1,6,1,0
       11 X.....X.........X....X          0,1,4,1,9,1,5,1,0
       12 X....X..........X....X          0,1,4,1,10,1,4,1,0
       13 X...X...........X....X          0,1,4,1,11,1,3,1,0
       14 X..X............X....X          0,1,4,1,12,1,2,1,0
       15 X.X.............X....X          0,1,4,1,13,1,1,1,0
       16 X.XXXXXXXXXXXXXXXXXX.X          0,1,1,18,1,1,0
       17 X...............X....X          -8,0,1,4,1,15,1,0,-8
       18 X...............X....X
       19 X...............X....X
       20 X...............X....X
       21 X...............X....X
       22 X...............X....X
       23 X...............X....X
       24 X...............X....X
       25 X.............XXXXX..X          0,1,2,5,13,1,0
       26 X....................X          0,1,20,1,0
       27 XXXXXXXXXXXXXXXXXXXXXX          0,22,0
       28 ..........X...........          -4,11,1,10,-4
       29 ..........X...........
       30 ..........X...........
       31 ..........X...........
       32 ......XXXXXXXXX.......          7,9,6
```

UBI10

UBI10 is a combined protocol/file format for image transfer, as opposed to Intelhex and UBI00-UBI03 protocols described earlier in this chapter.

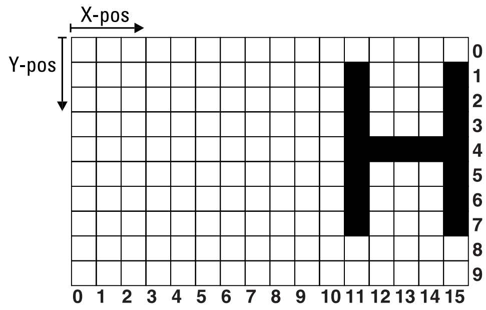
Protocol Description

```
!BG ↵
!X<pos>A ↵
!Y<pos>A ↵
!X<pos>A | !Y<pos>A !SB<bytes>W<data>
!X<pos>A | !Y<pos>A !SB<bytes>W<data>
!X<pos>A | !Y<pos>A !SB<bytes>W<data>
.....
!X<pos>A | !Y<pos>A !SB<bytes>W<data>!EG
!PRINT ↵
```

Frame Definitions

The width of the image in the STORE IMAGE statement should be given as a multiple of 16 bits.

!BG	Begin graphics. Always appended by a carriage return character.
!X<pos>A	Set absolute x position <pos>. The value must be divisible by 8. Default value is 0. Once set, it will affect all consecutive y-positions in the image, until a new x-position is set. Appended by a carriage return character, unless followed by a **!SB<bytes>W<data>** string on the same line.
!Y<pos>A	Set absolute y position <pos>. Default value is 0. Appended by a carriage return character, unless followed by a **!SB<bytes>W<data>** string on the same line.
!SB<bytes>W<data>	Send one line of bitmap with <bytes> number of bytes. <data> is bitmap bytes. Can be preceded by a new x- and/or y-position. If appended by a carriage return character, next **!SB** set of data will be positioned at the current y-position incremented by 1. If no appending carriage return character is used, a new y-position must be specified for next **!SB** set of data.
!EG	End graphics. Always appended by a carriage return character.
!PRINT	End page (end frame). Always appended by a carriage return character.



The image illustrated above contains 2 bytes (= 16 bits) in each horizontal line. By setting the absolute start position to x = 8, you can start counting from the start of the second byte, that is x = 8 in the matrix above. The first 3 bits (x-positions) are white, then comes one black bit followed by three white bits, and finally one black bit. Expressed in 0:s and 1:s, where 0 represents a white bit and 1 a black bit, the pattern will be 00010001. This binary number can be expressed as 11 hex. The same pattern is repeated for each y-position from y = 1 thru y = 7 with the exception of position y = 4, where all bits are black except for the leading three, i.e. the pattern is 00011111, which can be expressed as 1F hex. Use this hexadecimal values as input data as shown in the example below.

Example

The simplified image above is transmitted to the printer. Do not use XON/XOFF (11 hex/13 hex) protocol, since these characters may coincide with input data. Use RTS/CTS instead. Do not strip LF.

```
10    STORE OFF
20    OPEN "uart1:" FOR INPUT AS #1
30    QNAME$="H.1"
40    QWIDTH%=16
50    QHEIGHT%=10
60    QPRO$="UBI10"
70    STORE IMAGE QNAME$,QWIDTH%,QHEIGHT%,QPROT$
80    STORE INPUT 900,4: 'Timeout 9 sec.
90    CLOSE#1
100   STORE OFF
RUN
```

The input string in line 80 should contain the following data. Carriage returns (↵) after each !SB set of data increments the y-position by 1 in consecutive order. It may also be sent as a continuous string.

`!BG↵`	(Begin graphic)
`!X8A↵`	(Set x-position)
`!Y1A!SB1W<11 hex>↵`	(Set y-position + data for y = 1)
`!SB1W<11 hex>↵`	(Data for y = 2)
`!SB1W<11 hex>↵`	(Data for y = 3)
`!SB1W<1F hex>↵`	(Data for y = 4)
`!SB1W<11 hex>↵`	(Data for y = 5)
`!SB1W<11 hex>↵`	(Data for y = 6)
`!SB1W<11 hex>!EG↵`	(Data for y = 7 + end graphics)
`!PRINT↵`	(End frame)

PRBUF Protocol

The PRBUF Protocol is designed for downloading bitmap print image data directly from an application program, such as a Windows printer driver, directly to the printer's image buffer in connection with the PRBUF statement.

The protocol consist of a two-byte header and a number of data bytes:

Header

Byte No 1. is always the @-sign (Commercial at; Unicode 0x0040) and indicates start of the protocol header.

Byte No 2 is:

0	Reserved (bitmap format)
1	Reserved (RLL image format)
2	RLL buffer format
3-255	Reserved

RLL Buffer format

The RLL buffer format is optimized for use by Windows drivers. In most cases the performance of the host outruns the performance of the printer, so it is preferred to to do most of the processing in the host before sending the job down to the printer.

- Data byte 1 & 2 specifies the pixelwidth (unsigned) of data in BIG Endian format for one line.
- Data byte 3 & 4 specifies the pixelheight (unsigned) of the buffer when it is expanded BIG Endian.
- Data byte 5-nn specifies the bitmap in RLL format.

Example of RLL buffer protocol header, 515x212 pixels hexdump:

```
40 02 02 03 00 d4
```

RLL format

The RLL format is good for black and white pixel runs. It compresses data in both dimensions. It works well with one-dimensional bar codes, but grayscales grow in size instead of shrinking. The format is symmetric so that all pixel runs start and end with a white pixel and with line repetitions whenever applicable. This makes the format possible to turn upside down.

The RLL format is specified on the next page.

Specification of the RLL format

<begin><toggling pixelruns><end>

<-   total width of RLL pattern   ->

<begin>	:=	<linereps>|<small white pixelrun>
<end>	:=	<begin>|<empty>
<toggling pixelruns>	:=	<whiteAndBlack pixelruns>| <blackAndWhite pixelruns>| <white pixelrun>|<empty>
<whiteAndBlack pixelruns>	:=	<white pixelrun><black pixelrun>| <small white pixelrun><black pixelrun>
<blackAndWhite pixelruns>	:=	<black pixelrun><white pixelrun>
<linereps>	:=	((-1)-(-128))*-1 number of equal lines
<small white pixelrun>	:=	0-127, number of white pixels
<black pixelrun>	:=	0-255, number of black pixels
<white pixelrun>	:=	0-255, number of white pixels
<empty>	:=	empty, extreme if the entire line fits in one pixelrun.

If there is no line repetion, there does not have to be any line repeat. If the pixelrun is out of range, it must be split into several runs.

Example of RLL format for an eight bit pattern:

-*-*-*-*	1,1,1,1,1,1,1,1,0	Note the last 0 to end with a white pixelrun
--*-*-	0,1,1,1,1,1,1,1,1	begins with a white pixelrun of 0 pixels
--**--**	2,2,2,2,0	repetion, stopped with a white pixelrun of 0 pixels
---- **--**--	-2,0,2,2,2,2,-2	line and pixel repetions

Example of coding a black square of 800 dots to valid RLL format:

```
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-32,0,255,0,255,0,255,0,35,0,-32
```

4 Character Sets

This chapter contains the various single-byte character sets, that can be selected using the NASC statement. Printouts are made using the font "Swiss 721 BT". Other fonts may not include all characters listed in the character sets. Double-byte character sets are not included, but are available separately on special request. For more information on character sets and fonts, refer to the *Intermec Fingerprint, Font Reference Manual*.

Introduction

The following information applies to all single-byte character sets:

- Characters between ASCII 00 decimal and ASCII 31 decimal are unprintable control characters as listed below.
- Characters between ASCII 32 decimal and ASCII 127 decimal can always be printed, regardless of 7-bit or 8-bit communication protocol, provided that the selected font contains the characters in question.
- Characters between ASCII 125 decimal and ASCII 255 decimal can only be printed if the selected font contains the characters in question and an 8-bit communication protocol is used. If you use 7-bit communication, select another national character set (see NASC statement) or use a MAP statement to remap a character set.
- If a character, which does not exist in the selected font, is used, an error condition will occur.

Non-printable control characters (ASCII 00-31 dec)

ASCII	Character	Meaning
00	NUL	Null
01	SOH	Start of heading
02	STX	Start of text
03	ETX	End of text
04	EOT	End of transmission
05	ENQ	Enquiry
06	ACK	Acknowledge
07	BEL	Bell
08	BS	Backspace
09	HT	Horizontal tabulation
10	LF	Line feed
11	VT	Vertical tabulation
12	FF	Form feed
13	CR	Carriage Return
14	SO	Shift out
15	SI	Shift in
16	DLE	Data link escape
17	DC1	Device control one
18	DC2	Device control two
19	DC3	Device control three
20	DC4	Device control four
21	NAK	Negative acknowledge
22	SYN	Syncronous idle
23	ETB	End of transmission block
24	CAN	Cancel
25	EM	End of medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File separator
29	GS	Group separator
30	RS	Record separator
31	US	Unit separator

Roman 8 Character Set

NASC 1

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	ˆ	¨	˜	Ù	Û	₤	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

French Character Set NASC 33

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	à	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	°	ç	§	^	_	µ	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	é	ù	è	¨		€	▯
130	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
140	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	▯				

Spanish Character Set NASC 34

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	¡	Ñ	¿	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	°	ñ	ç	~		€	▯
130	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
140	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	▯				

Italian Character Set NASC 39

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	°	ç	é	^	_	ù	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	à	ò	è	ì		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

English (UK) Character Set NASC 44

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	‾		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	₤	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Swedish Character Set NASC 46

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	¤	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	É	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Ä	Ö	Å	Ü	_	é	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	ä	ö	å	ü		€	▯
130	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
140	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	▯				

Norwegian Character Set

NASC 47

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Æ	Ø	Å	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	æ	ø	å	‾		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	₤	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

German Character Set NASC 49

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Ä	Ö	Ü	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	ä	ö	ü	ß		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Japanese Latin Character Set NASC 81

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	¥	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	₤	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Portuguese Character Set NASC 351

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Ã	Ç	Õ	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	ã	ç	õ	°		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

PCMAP Character Set

NASC -1

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	å	ç	ê	ë	è	ï
140	î	ì	Ä	Å	É	æ	Æ	ô	ö	ò
150	û	ù	ÿ	Ö	Ü	¢	£	¥	§	ƒ
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	`
170	^	½	¼	¡	«	»	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	▯				

ANSI Character Set NASC -2

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ	□	Ž	□	□	‘	’	“	”	•
150	–	—	˜	™	š	›	œ	□	ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	^{2}	^{3}
180	´	µ	¶	·	¸	^{1}	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

MS-DOS Latin 1 Character Set NASC 850

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	å	ç	ê	ë	è	ï
140	î	ì	Ä	Å	É	æ	Æ	ô	ö	ò
150	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	®
170	¬	½	¼	¡	«	»	░	▒	▓	│
180	┤	Á	Â	À	©	╣	║	╗	╝	¢
190	¥	┐	└	┴	┬	├	─	┼	ã	Ã
200	╚	╔	╩	╦	╠	═	╬	¤	ð	Ð
210	Ê	Ë	È	ı	Í	Î	Ï	┘	┌	█
220	▄	¦	Ì	▀	Ó	ß	Ô	Ò	õ	Õ
230	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´
240	-	±	‗	¾	¶	§	÷	¸	°	¨
250	·	¹	³	²	■					

MS-DOS Greek 1 Character Set NASC 851

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	Ά	ç	ê	ë	è	ï
140	î	Έ	Ä	Ή	Ί	□	Ό	ô	ö	Ύ
150	û	ù	Ώ	Ö	Ü	ά	£	έ	ή	ί
160	ϊ	ΐ	ό	ύ	Α	Β	Γ	Δ	Ε	Ζ
170	Η	½	Θ	Ι	«	»	░	▒	▓	│
180	┤	Κ	Λ	Μ	Ν	╣	║	╗	╝	Ξ
190	Ο	┐	└	┴	┬	├	─	┼	Π	Ρ
200	╚	╔	╩	╦	╠	═	╬	Σ	Τ	Υ
210	Φ	Χ	Ψ	Ω	α	β	γ	┘	┌	█
220	▄	δ	ε	▀	ζ	η	θ	ι	κ	λ
230	μ	ν	ξ	ο	π	ρ	σ	ς	τ	΄
240	-	±	υ	φ	χ	§	ψ	¸	°	¨
250	ω	ϋ	ΰ	ώ	■					

MS-DOS Latin 2 Character Set NASC 852

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	ů	ć	ç	ł	ë	Ő	ő
140	î	Ź	Ä	Ć	É	Ĺ	ĺ	ô	ö	Ľ
150	ľ	Ś	ś	Ö	Ü	Ť	ť	Ł	×	č
160	á	í	ó	ú	Ą	ą	Ž	ž	Ę	ę
170	¬	ź	Č	ş	«	»	░	▒	▓	│
180	┤	Á	Â	Ě	Ş	╣	║	╗	╝	Ż
190	ż	┐	└	┴	┬	├	─	┼	Ă	ă
200	╚	╔	╩	╦	╠	═	╬	¤	đ	Đ
210	Ď	Ë	ď	Ň	Í	Î	ě	┘	┌	█
220	▄	Ţ	Ů	▀	Ó	ß	Ô	Ń	ń	ň
230	Š	š	Ŕ	Ú	ŕ	Ű	ý	Ý	ţ	´
240	-	˝	˛	ˇ	˘	§	÷	¸	°	¨
250	˙	ű	Ř	ř	■					

MS-DOS Cyrillic Character Set

NASC 855

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		ђ	Ђ
130	ѓ	Ѓ	ё	Ё	є	Є	ѕ	Ѕ	і	І
140	ї	Ї	ј	Ј	љ	Љ	њ	Њ	ћ	Ћ
150	ќ	Ќ	ў	Ў	џ	Џ	ю	Ю	ъ	Ъ
160	а	А	б	Б	ц	Ц	д	Д	е	Е
170	ф	Ф	г	Г	«	»	░	▒	▓	│
180	┤	х	Х	и	И	╣	║	╗	╝	й
190	Й	┐	└	┴	┬	├	─	┼	к	К
200	╚	╔	╩	╦	╠	═	╬	¤	л	Л
210	м	М	н	Н	о	О	п	┘	┌	█
220	▄	П	я	▀	Я	р	Р	с	С	т
230	Т	у	У	ж	Ж	в	В	ь	Ь	№
240	-	ы	Ы	з	З	ш	Ш	э	Э	щ
250	Щ	ч	Ч	§	■					

MS-DOS Turkish Character Set NASC 857

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	å	ç	ê	ë	è	ï
140	î	ı	Ä	Å	É	æ	Æ	ô	ö	ò
150	û	ù	İ	Ö	Ü	ø	£	Ø	Ş	ş
160	á	í	ó	ú	ñ	Ñ	Ğ	ğ	¿	®
170	¬	½	¼	¡	«	»	░	▒	▓	│
180	┤	Á	Â	À	©	╣	║	╗	╝	¢
190	¥	┐	└	┴	┬	├	─	┼	ã	Ã
200	╚	╔	╩	╦	╠	═	╬	¤	º	ª
210	Ê	Ë	È	▯	Í	Î	Ï	┘	┌	█
220	▄	¦	Ì	▀	Ó	ß	Ô	Ò	õ	Õ
230	µ	▯	×	Ú	Û	Ù	ì	ÿ	¯	´
240	-	±	▯	¾	¶	§	÷	¸	°	¨
250	·	¹	³	²	■					

Windows Latin 2 Character Set NASC 1250

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	□	„	…	†	‡	□	‰	Š	‹
140	Ś	Ť	Ž	Ź	□	‘	’	“	”	•
150	–	—	□	™	š	›	ś	ť	ž	ź
160		ˇ	˘	Ł	¤	Ą	¦	§	¨	©
170	Ş	«	¬	-	®	Ż	°	±	˛	ł
180	´	µ	¶	·	¸	ą	ş	»	Ľ	˝
190	ľ	ż	Ŕ	Á	Â	Ă	Ä	Ĺ	Ć	Ç
200	Č	É	Ę	Ë	Ě	Í	Î	Ď	Đ	Ń
210	Ň	Ó	Ô	Ő	Ö	×	Ř	Ů	Ú	Ű
220	Ü	Ý	Ţ	ß	ŕ	á	â	ă	ä	ĺ
230	ć	ç	č	é	ę	ë	ě	í	î	ď
240	đ	ń	ň	ó	ô	ő	ö	÷	ř	ů
250	ú	ű	ü	ý	ţ	˙				

Windows Cyrillic Character Set NASC 1251

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ђ	Ѓ
130	‚	ѓ	„	…	†	‡	€	‰	Љ	‹
140	Њ	Ќ	Ћ	Џ	ђ	‘	’	“	”	•
150	–	—	□	™	љ	›	њ	ќ	ћ	џ
160		Ў	ў	Ј	¤	Ґ	¦	§	Ё	©
170	Є	«	¬	-	®	Ї	°	±	І	і
180	ґ	µ	¶	·	ё	№	є	»	ј	Ѕ
190	ѕ	ї	А	Б	В	Г	Д	Е	Ж	З
200	И	Й	К	Л	М	Н	О	П	Р	С
210	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы
220	Ь	Э	Ю	Я	а	б	в	г	д	е
230	ж	з	и	й	к	л	м	н	о	п
240	р	с	т	у	ф	х	ц	ч	ш	щ
250	ъ	ы	ь	э	ю	я				

Windows Latin 1 Character Set NASC 1252

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ	□	Ž	□	□	‘	’	“	”	•
150	–	—	˜	™	š	›	œ	□	ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Windows Greek Character Set NASC 1253

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	ƒ	„	…	†	‡	□	‰	□	‹
140	□	□	□	□	□	‘	’	“	”	•
150	–	—	□	™	□	›	□	□	□	□
160		΅	Ά	£	¤	¥	¦	§	¨	©
170	□	«	¬	-	®	―	°	±	^{2}	^{3}
180	΄	µ	¶	·	Έ	Ή	Ί	»	Ό	½
190	Ύ	Ώ	ΐ	Α	Β	Γ	Δ	Ε	Ζ	Η
200	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π	Ρ
210	□	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	Ϊ	Ϋ
220	ά	έ	ή	ί	ΰ	α	β	γ	δ	ε
230	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
240	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω
250	ϊ	ϋ	ό	ύ	ώ	□				

Windows Latin 5 Character Set NASC 1254

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ	□	□	□	□	‘	’	“	”	•
150	–	—	˜	™	š	›	œ	□	□	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ğ	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	İ	Ş	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ğ	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ı	ş	ÿ				

Windows Baltic Rim Character Set NASC 1257

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	□	„	…	†	‡	□	‰	□	‹
140	□	¨	ˇ	¸	□	‘	’	“	”	•
150	–	—	□	™	□	›	□	¯	˛	□
160		□	¢	£	¤	□	¦	§	Ø	©
170	Ŗ	«	¬	-	®	Æ	°	±	²	³
180	´	µ	¶	·	ø	¹	ŗ	»	¼	½
190	¾	æ	Ą	Į	Ā	Ć	Ä	Å	Ę	Ē
200	Č	É	Ź	Ė	Ģ	Ķ	Ī	Ļ	Š	Ń
210	Ņ	Ó	Ō	Õ	Ö	×	Ų	Ł	Ś	Ū
220	Ü	Ż	Ž	ß	ą	į	ā	ć	ä	å
230	ę	ē	č	é	ź	ė	ģ	ķ	ī	ļ
240	š	ń	ņ	ó	ō	õ	ö	÷	ų	ł
250	ś	ū	ü	ż	ž	˙				

5 Bar Codes

This chapter list the bar codes included in the Intermec Fingerprint v8.20 firmware and gives examples of some commonly used bar codes.

Introduction

The printer contains a number of bar code generators, which can produce highly readable bar codes in four different directions.

However, a general rule which applies to all thermal printers is that it is more difficult to print a bar code with the bars across the media path (ladder style) than along the media path (picket fence style.) Therefore, to ensure a highly readable printout, we recommend that you do not use narrow bars less than 3 dots, when printing bar codes with the bars across the media path (ladder style).

No such restrictions apply for bar codes with the bars along the media path (picket fence style).

Another factor, that affects the printout quality of the bar codes, is the print speed. Generally, a lower print speed gives a better quality, especially for ladder style bar codes and at low ambient temperatures. Do not use a higher print speed than necessary and consider the overall print cycle time. In some instances, a lower print speed may actually give better overall performance. We recommend you do your own tests with your unique applications to find the best compromise between printout quality, performance, and media.

Specifications for bar code symbologies can be obtained from organizations like:

EAN International
http://www.ean-int.org

UCC - The Uniform Code Council, Inc. (UCC)
http://www.uc-council.org

AIM International, Inc.
http://www.aimi.org

American National Standard Institute (ANSI)
http://www.ansi.org

Standard Bar Codes

Bar Code	Designation
Codabar	`"CODABAR"`
Code 11	`"CODE11"`
Code 16K	`"CODE16K"`
Code 39	`"CODE39"`
Code 39 full ASCII	`"CODE39A"`
Code 39 w. checksum	`"CODE39C"`
Code 49	`"CODE49"`
Code 93	`"CODE93"`
Code 128	`"CODE128"`
Code 128 subset A	`"CODE128A"`
Code 128 subset B	`"CODE128B"`
Code 128 subset C	`"CODE128C"`
Data Matrix	`"DATAMATRIX"`
DUN-14/16	`"DUN"`
EAN-8	`"EAN8"`
EAN-13	`"EAN13"`
EAN 128	`"EAN128"`
EAN 128 subset A	`"EAN128A"`
EAN 128 subset B	`"EAN128B"`
EAN 128 subset C	`"EAN128C"`
Five-Character Supplemental Code	`"ADDON5"`
Industrial 2 of 5	`"C2OF5IND"`
Industrial 2 of 5 w. checksum	`"C2OF5INDC"`
Interleaved 2 of 5	`"INT2OF5"`
Interleaved 2 of 5 w. checksum	`"INT2OF5C"`
Interleaved 2 of 5 A	`"I2OF5A"`
Matrix 2 of 5	`"C2OF5MAT"`
MaxiCode	`"MAXICODE"`
MicroPDF417	`"MICROPDF417"`
MSI (modified Plessey)	`"MSI"`
PDF 417	`"PDF417"`
Plessey	`"PLESSEY"`
Postnet	`"POSTNET"`
QR Code	`"QRCODE"`
RSS-14	`"RSS14"`
RSS-14 Truncated	`"RSS14T"`
RSS-14 Stacked	`"RSS14S"`
RSS-14 Stacked Omnidirectional	`"RSS14SO"`
RSS-14 Limited	`"RSS14L"`
RSS-14 Expanded	`"RSS14E"`
RSS-14 Expanded Stacked	`"RSS14ES"`
Straight 2 of 5	`"C2OF5"`
Two-Character Supplemental Code	`"ADDON2"`
UCC-128 Serial Shipping Container Code	`"UCC128"`
UPC-5 digits Add-On Code	`"SCCADDON"`

UPC-A	`"UPCA"`
UPC-D1	`"UPCD1"`
UPC-D2	`"UPCD2"`
UPC-D3	`"UPCD3"`
UPC-D4	`"UPCD4"`
UPC-D5	`"UPCD5"`
UPC-E	`"UPCE"`
UPC Shipping Container Code	`"UPCSCC"`

On the following pages, a quick survey of the characteristics of some of the most common bar codes will be given. This information is only intended to help you avoid entering unacceptable parameters or input data. For further information, please refer to the standard literature on the subject of bar codes.

EAN-8

BARTYPE:	"EAN8"
BARRATIO:	Fixed ratio.
	BARRATIO statement ignored.
BARMAG:	Max. 8
BARHEIGHT:	No restriction.
BARFONT:	Barfont generated automatically.
	BARFONT statement ignored.
	BARFONT ON/OFF statements work.
INPUT DATA:	
No. of characters:	7
Check digit:	1 added automatically.
Digits:	0-9
Uppercase letters:	No
Lowercase letters:	No
Punctuation marks:	No
Start characters:	No
Stop characters:	No

EAN-13

BARTYPE:	"EAN13"
BARRATIO:	Fixed ratio.
	BARRATIO statement ignored.
BARMAG:	Max. 8
BARHEIGHT:	No restriction.
BARFONT:	Barfont generated automatically.
	BARFONT statement ignored.
	BARFONT ON/OFF statements work.
INPUT DATA:	
No. of characters:	12
Check digit:	1 added automatically.
Digits:	0-9
Uppercase letters:	No
Lowercase letters:	No
Punctuation marks:	No
Start characters:	No
Stop characters:	No

UPC-E

BARTYPE:	"UPCE"
BARRATIO:	Fixed ratio. BARRATIO statement ignored.
BARMAG:	Max. 8
BARHEIGHT:	No restriction.
BARFONT:	Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF statements work.
INPUT DATA:	
No. of characters:	6
Check digit:	1 added automatically.
Digits:	0-9
Uppercase letters:	No
Lowercase letters:	No
Punctuation marks:	No
Start characters:	No
Stop characters:	No

UPC-A

BARTYPE:	"UPCA"
BARRATIO:	Fixed ratio. BARRATIO statement ignored.
BARMAG:	Max. 8
BARHEIGHT:	No restriction.
BARFONT:	Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF statements work.
INPUT DATA:	
No. of characters:	11
Check digit:	1 added automatically.
Digits:	0-9
Uppercase letters:	No
Lowercase letters:	No
Punctuation marks:	No
Start characters:	No
Stop characters:	No

Interleaved 2 of 5

BARTYPE:	"INT2OF5"
BARRATIO:	2:1-3:1
BARMAG:	No restriction.
BARHEIGHT:	No restriction.
BARFONT:	No restriction.
INPUT DATA:	
No. of characters:	Unlimited
Check digit:	No
Digits:	0-9
Uppercase letters:	No
Lowercase letters:	No
Punctuation marks:	No
Start characters:	Added automatically.
Stop characters:	Added automatically.

Note: A numeric code where input digits are encoded in pairs. If an odd number of digits is entered, a leading zero will be added automatically.

Code 39

BARTYPE:	"CODE39"
BARRATIO:	2:1-3:1
BARMAG:	No restriction, but if the narrow element is less than 4 dots wide, then the ratio must be larger than 2.25:1 (9:4).
BARHEIGHT:	No restriction.
BARFONT:	No restriction.
INPUT DATA:	
No. of characters:	Unlimited.
Check digit:	No
Digits:	0-9
Uppercase letters:	A-Z (no national characters).
Lowercase letters:	No
Punctuation marks:	- . space $ / + %
Start characters:	* (is added automatically).
Stop characters:	* (is added automatically).

Note: An alphanumeric self-checking discrete code.

Code 128

BARTYPE:	"CODE128"
	"CODE128A"
	"CODE128B"
	"CODE128C"
BARRATIO:	Fixed. BARRATIO statement ignored.
BARMAG:	≥ 2.
BARHEIGHT:	No restriction.
BARFONT:	No restriction.

INPUT DATA:	
No. of characters:	Unlimited
Check digit:	1 check digit added automatically.
Input characters:	ASCII 0-127 decimal according to Roman 8 character set.
Function characters:	FNC1: ASCII 128 decimal (see note 1)
	FNC2: ASCII 129 decimal (see note 1)
	FNC3: ASCII 130 decimal (see note 1)
	FNC4: ASCII 131 decimal (see note 1)
Start characters:	See note 2.
Code characters:	See note 1 & 2.
Shift characters:	See note 1 & 2.
Stop character:	Always added automatically.

Note 1:
Function characters FNC1-4, code characters, and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 dec., or the use of an CHR$ function.

FNC2-4 are not allowed in Subset C.

Note 2:
Code 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data), whereas Code 128A, Code 128B, and Code 128C selects subset A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character.
  Shift character: «« (« = ASCII 171 dec.)
- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset:
  Code character: « + A|B|C (« = ASCII 171 dec.)

EAN 128

BARTYPE:	"EAN128" "EAN128A" "EAN128B" "EAN128C"
BARRATIO:	Fixed. BARRATIO statement ignored.
BARMAG:	≥ 2.
BARHEIGHT:	No restriction.
BARFONT:	No restriction.

INPUT DATA:	
No. of characters:	Unlimited.
Check digit:	Trailing symbol check character added automatically.
Input characters:	ASCII 0-127 decimal according to Roman 8 character set.
Start characters:	See note 2.
Code characters:	See note 1 & 2.
Shift characters:	See note 1 & 2.
Stop character:	Always added automatically.

This bar code is identical to Code 128 with the exception that the initial FNC1 function character is generated automatically.

Note 1:
Code characters and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 dec., or the use of an CHR$ function.

Note 2:
EAN 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data), whereas EAN 128A, EAN 128B, and EAN 128C selects subset A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character.
  Shift character: ««   (« = ASCII 171 dec.)
- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset:
  Code character: « + A|B|C   (« = ASCII 171 dec.)

Data Matrix

BARTYPE:	"DATAMATRIX"
BARRATIO:	Fixed. Values will be interpreted as BARMAG.
BARMAG:	<128
BARHEIGHT:	Not applicable.
BARFONT:	Not applicable.

BARSET parameters:

<nexp$_1$>	Large bar ratio	Not applicable.
<nexp$_2$>	Narrow bar ratio	Not applicable.
<nexp$_3$>	Barmag/Enlargement	< 128.
<nexp$_4$>	Barheight	Not applicable.
<nexp$_5$>	Security level	Not applicable.
<nexp$_6$>	Aspect height ratio	Not applicable.
<nexp$_7$>	Aspect width ratio	Not applicable.
<nexp$_8$>	No. of rows	Not applicable.
<nexp$_9$>	No. of columns	Not applicable.
<nexp$_{10}$>	Truncate flag	Not applicable.

INPUT DATA:

No. of characters:	ASCII characters: 2335 when only uppercase A-Z, ampersand (&), period (.), comma (,), minus or hyphen (-), and solidus (/) are represented.
Check digit:	Added automatically.
Input characters:	ASCII 0-255 decimal

MaxiCode

BARTYPE:	"MAXICODE"
BARRATIO:	Not applicable. Input ignored.
BARMAG:	Not applicable. Input ignored.
BARHEIGHT:	Not applicable. Input ignored.
BARFONT:	Not applicable. Input ignored.
BARSET:	Not applicable. Input ignored.

MaxiCode requires 8 fields of data separated by a LF character, which is entered as CHR$(10). Regardless of which mode is chosen, all eight fields must contain valid data and must be present in final in data string, see table below

F_n = Field No, (mode n) = decoded by reader

F_1 (mode 2&3):	5 Characters, numeric (mode 2) or 6 alphanumeric characters (mode3).
F_2 (mode 2):	4 digits [0-9999]
F_3 (mode 2&3):	Country code 3 digits [0-999]
F_4 (mode 2&3):	Service class, 3 digits [0-999]
F_5 (LPM mode 2,3,4,):	User defined message.
F_6:	Mode selector, one digit [2|3|4].
F_7:	Position in structured append, one digit [1-8]
F_8:	Total number of symbols in structure, one digit [1-8]

F_8 in structured append is the trigger for structured append mode.

If F_8 >1, the two first code word in secondary message will be a pad followed by position code word. F_8 has higher precedence than F_7.

If F_7 >1when F_8 = 1, the two first codeword will not signal structured append.

If F_8 >1, F_7 may be >F_8 without error and structured append codeword will signal given values.

No of characters:	Up to 84 on mode 2 & 3 or up to 138 in mode 4.
Check character:	Automatically, Reed-Solomon algorithm.
Data type:	ASCII 0-255

Summary				**Mode 2** Structured data message for US Destinations	**Mode 3** Structured data message for International Destinations	**Mode 4** Standard Symbol
Data element	**Length**	**Type**	**Sample**			
Primary Message Zip code + 4 digit extension	Mode 2: 9 Mode3: 6	Numeric (mode2) Alphanumeric (mode 3)	Mode 2: 152392802 Mode 3: B1050$_{\leftrightarrow}$	Mandatory	Mandatory	N/A
Country Code	3	Numeric	840	Mandatory	Mandatory	N/A
Service Class	3	Numeric	001	Mandatory	Mandatory	N/A
Secondary Message	84/138	Alphanumeric	This is the secondary message			
Header (optional)[1]	9	Numeric	[])^R_S01^G_S97	Optional	Optional	N/A
Sample data streams as decoded by scanner				[])^R_S01^G_S979820 39280^G_S840^G_S001 ^G_SSECONDARY MESSAGE	[])^R_S01^G_S97B105 0_^G_S056^G_S999^G_S SECONDARY MESSAGE	

[1]/. Header is encoded into secondary message.

Refer to Chapter 2; PRBAR for Fingerprint programming example.

PDF417

BARTYPE:	"PDF417"
BARRATIO:	Fixed. Values will be interpreted as BARMAG.
BARMAG:	2-128.
BARHEIGHT:	No restriction.
BARFONT:	Not applicable.

For two-dimensional bar code symbologies, we recommend using the BARSET statement rather than separate statements like BARMAG, BAR-RATIO etc, which do not allow all parameters to be set.

BARSET parameters:

<nexp$_1$>	Large bar ratio	No restriction
<nexp$_2$>	Narrow bar ratio	No restriction
<nexp$_3$>	Barmag/Enlargement	< 128
<nexp$_4$>	Barheight	< 500.
<nexp$_5$>	Security level	1-5
<nexp$_6$>	Aspect height ratio	No restriction
<nexp$_7$>	Aspect width ratio	No restriction
<nexp$_8$>	No. of rows	No restriction.
<nexp$_9$>	No. of columns	No restriction.
<nexp$_{10}$>	Truncate flag	0 or 0. 0 will print a normal symbol.

INPUT DATA:

No. of characters:	1,800 ASCII characters or 2,700 digits depending on level of compactness.
Check digit:	Added automatically.
Input characters:	ASCII 0-255 decimal

Refer to Chapter 2; BARSET for Fingerprint programming example,

MicroPDF417

MicroPDF417 is a multi-row symbology based on PDF417. A limited set of symbol sizes is available where each size has a fixed level of error correction. Most symbol characteristics such as data character encodation, error correction, and symbol character sets are identical to those of PDF417. Up to 250 alphanumeric characters or 366 numeric digits can be encoded in a symbol.

BARTYPE: "MICROPDF417"

BARSET parameters:

<nexp$_1$>	Not applicable		
<nexp$_2$>	Not applicable		
<nexp$_3$>	Element width in dots	1-21	
<nexp$_4$>	Element height in dots	1-127	
<nexp$_5$>	Not applicable		
<nexp$_6$>	Not applicable		
<nexp$_7$>	Not applicable		
<nexp$_8$>	Number of rows	0,4-44	(0=automatic)
<nexp$_9$>	Number of columns	0-4	(0=automatic)
<nexp$_{10}$>	Not applicable		

Setting The Number of Rows And Columns
The symbol size is defined by specifying the number of rows and columns. Not all combinations of rows and columns are allowed. The table below illustrates the valid combinations.

No. of columns	Valid number of rows for each no. of columns									
1	11	14	17	20	24	28	-	-	-	-	-
2	8	11	14	17	20	23	26	-	-	-	-
3	6	8	10	12	15	20	26	32	38	44	-
4	4	6	8	10	12	15	20	26	32	38	44

If the number of rows is set to a value that does not match the valid values for the given number of columns, the printer will automatically choose a larger number from the list of valid values.

Automatic Selection
The number of columns and rows can be set automatically by the printer. If the number of columns is set to 0, the printer will set the number of columns as well as the number of rows automatically, regardless of the number of rows specified. The printer will try to fit the given data into a symbol with as few columns as possible. If the number of columns is non-zero and the number of rows is set to 0, the printer will automatically set the number of rows to the lowest number required to encode the given data.

Limitations
Enhanced applications such as Extended Channel Interpretation (ECI), structured append, reader initialisation, Code 128 emulation, and macro characters are not supported.

Examples

This example shows how a MicroPDF417 bar code is specified using the BARTYPE and BARSET statements.
Bar width: 2 dots
Bar height: 8 dots
Number of rows: 26
Number of columns: 3

```
BARTYPE "MICROPDF417"
BARSET #4,2,8,1,1,1,26,3
```

Note: The bar width and bar height can also be set using BARMAG and BARHEIGHT respectively.

The number of columns and rows are set using the Fingerprint statement BARSET. Parameters number 9 and 10 are the number of rows and columns respectively. Examples A and B below set the number of rows to 12 and the number of columns to 3. The type of bar code is set to MicroPDF417. Not all parameters of the BARSET command are applicable to the MicroPDF417 implementation. The parameters ignored by the implementation are set to '1' in example B (large bar ratio, small bar ratio, security level, aspect height, aspect width).

Example A (Direct Protocol)
```
BARTYPE "MICROPDF417"
BARSET #9, 12
BARSET #10, 3
```

Example B (Direct Protocol)
```
BARSET "MICROPDF417",1,1,2,8,1,1,1,12,3
```

The example code below prints a small MicroPDF417 bar code containing the string "MicroPDF417." The number of rows and columns is set by the printer based on the input string since the number of columns is set to 0.

```
10 BARSET "MICROPDF417",1,1,4,8,1,1,1,0,0
20 PRPOS 50, 50
30 PRBAR "MICROPDF417"
40 PRINTFEED
```

QR Code

BARTYPE:	"QRCODE"	
BARSET parameters:		
<nexp$_1$>	Large bar ratio	Not applicable
<nexp$_2$>	Narrow bar ratio	Not applicable
<nexp$_3$>	Element size in dots	1-127 (default 2)
<nexp$_4$>	QR Code Model	1 (default) or 2 (rec.)
<nexp$_5$>	Security level	1-4 1=L 2=M (default) 3=Q 4=H
<nexp$_6$>	(Mask pattern selection)	Reserved for future use
<nexp$_7$>		Not applicable
<nexp$_8$>		Not applicable
<nexp$_9$>		Not applicable
<nexp$_{10}$>		Not applicable

Input data Capacity (Model 1)

Security level	Numeric	Alphanumeric	8-bit byte data	Kanji
L	1176	707	486	299
M	877	531	365	225
Q	738	447	307	189
H	498	302	207	127

Input data Capacity (Model 2)

Security level	Numeric	Alphanumeric	8-bit byte data	Kanji
L	7089	4296	2953	1817
M	5596	3391	2331	1435
Q	3993	2420	1663	1024
H	3057	1852	1273	784

The unit is number of characters. Mixed mode is supported for all combinations except for combinations containing bot 8-bit byte and Kanji data. The type of data is set automatically by the implementation based on the input characters.

Error Correction Levels

L	7%
M	15%
Q	25%
H	30%

Refer to Chapter 2; BARSET for Fingerprint programming example.

RSS-14

BARTYPE:	"RSS14" "RSS14T" "RSS14S" "RSS14SO" "RSS14L" "RSS14E" "RSS14ES"
BARRATIO:	Not applicable
BARMAG:	Most narrow element width in dots (default 2)
BARHEIGHT:	No restriction. Defines height in dots of a bar code or of each row in a stacked bar code[1]. Default 100.
BARFONT:	Not applicable

BARSET parameters:

<nexp$_1$>	Value insignificant, but a digit must be entered as a place holder
<nexp$_2$>	Value insignificant, but a digit must be entered as a place holder
<nexp$_3$>	Most narrow element width in dots (default 2)
<nexp$_4$>	Height in dots (default 100)[1]
<nexp$_5$>	Segments per row (2-22). RSS14ES only.
<nexp$_6$>	Value insignificant, but a digit must be entered as a place holder
<nexp$_7$>	Separator pattern row height (RSS14S, RSS14SO, RSS14ES only)
<nexp$_8$>	Not applicable
<nexp$_9$>	Not applicable
<nexp$_{10}$>	Not applicable

INPUT DATA:
All except RSS14E and RSS14ES:
13 digits. If less than 13 digits are entered, leading zeros will be added automatically so the string will be 13 digits long.

RSS14E and RSS14ES only:
Max. 71 numeric or 41 alphanumeric characters. Allowed characters:
0-9 A-Z a-z ! " % & ' ( ) * + , - . / : ; < = > ? _ space FNC1 [CHR$(128)]

[1]/. There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp$_4$> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

Type	Width	Min. Height
RSS-14	96X	33X
RSS-14T	96X	13X
RSS-14S	50X	13X (upper 5X + lower 7X + separator1X min.)
RSS-14SO	50X	69X (upper 33X +lower 33X + separator 3*1X)
RSS-14L	71X	10X
RSS-14E	Depends on input	33X
RSS-14ES	Depends on input	34X per row + 3*1X per separator

X = width of the most narrow element as specified by BARMAG or BARSET <nexp$_3$>.

Example of an RSS14S bar code with the following characteristics and with recommended minimum height selected:

Place holder ($nexp_1$):	1
Place holder ($nexp_2$):	1
Most narrow element width in dots:	3
Height in dots:	21
Place holder ($nexp_5$):	1
Place holder ($nexp_6$):	1
Separator pattern row height:	4
Data:	1234567890123

```
BARSET "RSS14S",1,1,3,21,1,1,4
PRBAR "1234567890123"
```

The RSS14E and the RSS14ES can be used for intelligent encoding of the input data. They can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSSE and RSS14ES, see Chapter 7 “Symbol Requirements for RSS Expanded” in the AIM specification: *International Symbology Specification Reduced Space Symbology (RSS), AIM Inc. ITS/99-0012, Version 1.0 1999-10-29*.

Example of RSS14 Expanded for a variable weight item (0,001 kilogram increments) and with recommended minimum height selected:

Start parameter:	#4
Most narrow element width in dots:	2
Height in dots:	68
Segments per row:	4
Place holder ($nexp_6$):	1
Separator pattern row height:	2
Data:	01900123456789003103001750

Explanation of data:	***01 9001234567890 0 3103 001750***
Application Identifier (AI):	*01*
AI 01 item ID:	*9001234567890* *(In this method, the first digit must be 9.)*
Digit:	*0* *(The value is insignificant but a digit must be entered as a place holder.)*
Application Identifier (AI):	*3103*
AI 3103 variable weight element string:	*001750*

```
BARTYPE "RSS14ES"
BARSET #4,2,68,4,1,2
PRBAR "01900123456789003103001750"
```

Setup Bar Codes

Intermec Fingerprint v8.xx-compatible EasyCoder printers can optionally be fitted with an EasySet bar code wand or a scanner (see the printer's User's Guide). By reading a special bar code containing encoded data for one or several setup parameters, the printer's setup can easily be changed, even by a person without any knowledge of Intermec Fingerprint, the Direct Protocol, or their supporting software.

You could print such bar codes in your printer and paste them on a board in the vicintity of the printer. When, for example, the operator needs to switch to another type of media, he or she will only have to pick up the EasySet wand or scanner and read the appropriate bar code.

The only bar code that can be used for this purpose is a Code 128 containing the function character FNC3 (ASCII 130 dec). If the FNC character is missing, the printer will regard the bar code as containing ordinary ASCII input to the "wand:" device.

Please refer to the *EasySet Bar Code Wand Setup* manual for syntax and parameter descriptions.

6 Fonts

This chapter lists the scaleable single-byte fonts included in the Intermec Fingerprint v8.20 firmware and contains printout samples. It also describes the method of creating font aliases and contains character sets for the OCR-A, OCR-B, and DingDings fonts.

Bitmap Fonts

It is possible to use fonts in the "old" Intermec .ATF bitmap font format. This feature improves compatibility with custom-made programs originally created in Fingerprint v6.xx or earlier versions.

Downloading an .ATF font (for example XX030RSN.ATF) to the printer produces three fonts in the memory; one without any extension (for example XX030RSN), one with the extension .1 (for example XX030RSN.1), and one with the extension .2 (for example XX030RSN.2). When using bitmap fonts in Fingerprint v8.xx, the relation between print direction and extension is of no consequence.

It is recommended to exclude the font height parameter in the FONT and BARFONT statements and use MAG to enlarge the font. Slant does not work at all with bitmap fonts.

Font Aliases

The standard font names in Intermec Fingerprint are much longer than in earlier versions of Fingerprint and may be cumbersome to use. They are also incompatible with the LAYOUT statement, which restricts the font name to 10 characters.

However, it is possible to create a file containing a list of font aliases. The file should be named exactly as shown here (note the leading period character that specifies it as a system file):

```
"c:.FONTALIAS"
```

The format of the file should be:

"<Alias name #1>","<Name of font>"[,size[,<slant>[,<width>]]]

"<Alias name #2>","<Name of font>"[,size[,<slant>[,<width>]]]

"<Alias name #3>","<Name of font>"[,size[,<slant>[,<width>]]]

etc., etc.

The file can contain as many fontname aliases as required. The default size is 12 points, the default slant is 0°, and the default width is 100 (%).

A font alias can be used as any other font, but its size, slant, and width cannot be changed.

Examples:

```
"BODYTEXT","Century Schoolbook BT",10,0,80
"HEADLINE","Swiss 721 Bold BT",18,0,110
"WARNING","Swiss 721 BT",12
```

For more information on fonts and character sets, refer to the *Intermec Fingerprint, Font Reference Manual*.

Printout Samples

The printout samples below are in 10 point size, no slant, and 100% width. The quality of these samples does not exactly correspond to the printout quality from your printer, which is affected by printhead density, printing method, type of media and ribbon, and a number of other factors.

Century Schoolbook BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 1234567890
DingDings SWA	✴★✥ ✱✵✩✣✫ ✢✲✯✷✮ ✦✯✸ ✪✵✭✰✳ ✯✶✥✲ ✴★✥ ✬✡✺✹ ✤✯✧ ▼❈❅ ❑◆❉❃❋ ❂❒❏◗■ ❆❏❘ ❊◆❍❐▲ ❏❖❅❒ ▼❈❅ ●❁❚❙ ❄❏❇ ✑✒
Dutch 801 Roman BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 1234567890
Dutch 801 Bold BT	**THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG** **the quick brown fox jumps over the lazy dog 1234567890**
Futura Light BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 1234567890
Letter Gothic 12 Pitch BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 1234567890
Monospace 821 BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 12345
Monospace 821 Bold BT	**THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG** **the quick brown fox jumps over the lazy dog 12345**
OCR-A BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 1234
OCR-B 10 Pitch BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 123456
Prestige 12 Pitch Bold BT	**THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG** **the quick brown fox jumps over the lazy dog 123456**
Swiss 721 BT	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG the quick brown fox jumps over the lazy dog 1234567890
Swiss 721 Bold BT	**THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG** **the quick brown fox jumps over the lazy dog 1234567890**
Swiss 721 Bold Condensed BT	**THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG** **the quick brown fox jumps over the lazy dog 1234567890**
Zurich Extra Condensed Bold	**THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG** **the quick brown fox jumps over the lazy dog**

OCR-A BT Character Set

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	□	□	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	□	□	□	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160	□	⑁	□	£	□	¥	□	□	□	'
170	□	<	□	□	□	□	□	□	,	.
180	□	□	□	□	□	□	□	>	□	□
190	□	□	?	□	▬	□	Ä	Å	Æ	□
200	□	□		□	□	□	□	□	-	Ñ
210	□	■	□	□	Ö	□	Ø	□	□	□
220	Ü	□	⑀	□	⑂	□	□	□	□	□
230	□	□	□	□	□	□	□	□	□	□
240	□	□	□	□	□	□	□	□	□	□
250	□	□	□	□	□	□				

OCR-B 10 Pitch BT Character Set

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	▯	▯	▯
130	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
140	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
160		▯	▯	£	¤	¥	▯	§	¨	'
170	"	▯	▯	▯	▯	▯	▯	▯	†	▯
180	´	m	▯	·	,	▯	"	▯	▯	▯
190	▯	▯	▯	'	—	ˆ	Ä	Å	Æ	▯
200	▯	▯		▯	▯	▯	IJ	ij	▯	Ñ
210	|	█	_	▯	ö	▯	Ø	▯	▯	▯
220	ü	▯	▯	ß	▯	▯	▯	▯	▯	å
230	æ	▯	▯	▯	▯	▯	▯	▯	▯	▯
240	▯	▯	▯	▯	▯	▯	▯	▯	ø	▯
250	▯	▯	▯	▯	▯	▯				

DingDings SWA Character Set

	0	1	2	3	4	5	6	7	8	9
30				✁	✂	✃	✄	☎	✆	✇
40	✈	✉	☛	☞	✌	✍	✎	✏	✐	✑
50	✒	✓	✔	✕	✖	✗	✘	✙	✚	✛
60	✜	✝	✞	✟	✠	✡	✢	✣	✤	✥
70	✦	✧	★	✩	✪	✫	✬	✭	✮	✯
80	✰	✱	✲	✳	✴	✵	✶	✷	✸	✹
90	✺	✻	✼	✽	✾	✿	❀	❁	❂	❃
100	❄	❅	❆	❇	❈	❉	❊	❋	●	❍
110	■	❏	❐	❑	❒	▲	▼	◆	❖	◗
120	❘	❙	❚	❛	❜	❝	❞		❨	❩
130	❪	❫	❬	❭	❮	❯	❰	❱	❲	❳
140	❴	❵	▯	▯	▯	▯	▯	▯	▯	▯
150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
160		❡	❢	❣	❤	❥	❦	❧	♣	♦
170	♥	♠	①	②	③	④	⑤	⑥	⑦	⑧
180	⑨	⑩	❶	❷	❸	❹	❺	❻	❼	❽
190	❾	❿	➀	➁	➂	➃	➄	➅	➆	➇
200	➈	➉	➊	➋	➌	➍	➎	➏	➐	➑
210	➒	➓	➔	→	↔	↕	➘	➙	➚	➛
220	➜	➝	➞	➟	➠	➡	➢	➣	➤	➥
230	➦	➧	➨	➩	➪	➫	➬	➭	➮	➯
240		➱	➲	➳	➴	➵	➶	➷	➸	➹
250	➺	➻	➼	➽	➾	▯				

7 Error Messages

This chapter list the possible error messages that can be returned to the host when an error occurrs.

Code	Message/Explanation
0	No error
1	Syntax error.
2	Unbalanced parenthesis.
3	Feature not implemented.
4	Evaluation syntax error.
5	Unrecognized token.
6	Tokenized line too long.
7	Evaluation stack overflow.
8	Error in exectab.
9	Undefined token.
10	Non-executing token.
11	Evaluation stack underflow.
12	Type mismatch.
13	Line not found.
14	Division with zero.
15	Font not found.
16	Bar code device not found.
17	Bar code type not implemented.
18	Disk full.
19	Error in file name.
20	Input line too long.
21	Error stack overflow.
22	RESUME without error.
23	Image not found.
24	Overflow in temporary string buffer.
25	Wrong number of parameters.
26	Parameter too large.
27	Parameter too small.
28	RETURN without GOSUB
29	Error in startup file.
30	Assign to a read-only variable.
31	Illegal file number.
32	File is already open.
33	Too many files open.
34	File is not open.
37	Cutter device not found.
38	User break.
39	Illegal line number.
40	Run statement in program.
41	Parameter out of range.
42	Illegal bar code ratio.
43	Memory overflow.
44	File is write protected.
45	Unknown store option.

Code	Message/Explanation
46	Store already in progress.
47	Unknown store protocol.
48	No store defined.
49	NEXT without FOR.
50	Bad store record header.
51	Bad store address.
52	Bad store record.
53	Bad store checksum.
54	Bad store record end.
55	Remove in ROM.
56	Illegal communication channel.
57	Subscript out of range.
58	Field overflow.
59	Bad record number.
60	Too many strings.
61	Error in setup file.
62	File is list protected.
63	ENTER function.
64	FOR without NEXT
65	Evaluation overflow.
66	Bad optimizing type.
67	Error from communication channel.
68	Unknown execution entity.
69	Not allowed in immediate mode.
70	Line label not found.
71	Line label already defined.
72	IF without ENDIF.
73	ENDIF without IF.
74	ELSE without ENDIF.
75	ELSE without IF.
76	WHILE without WEND.
77	WEND without WHILE
78	Not allowed in execution mode.
79	Not allowed in a layout.
80	Download timeout
81	Exit to system
82	Invalid cont environment
83	ETX Timeout
1001	Not implemented.
1002	Memory too small.
1003	Field out of label.
1004	Wrong font to chosen direction.
1005	Out of paper.
1006	No field to print.

Code	Message/Explanation
1007	Lss too high.
1008	Lss too low.
1009	Invalid parameter.
1010	Hardware error.
1011	I/O error.
1012	Too many files opened.
1013	Device not found.
1014	File not found.
1015	File is read-only.
1016	Illegal argument.
1017	Result too large.
1018	Bad file descriptor.
1019	Invalid font.
1020	Invalid image.
1021	Too large argument for MAG.
1022	Head lifted.
1023	Incomplete label.
1024	File too large.
1025	File does not exist.
1026	Label pending.
1027	Out of transfer ribbon.
1028	Paper type is not selected.
1029	Printhead voltage too high.
1030	Character is missing in chosen font.
1031	Next label not found.
1032	File name too long.
1033	Too many files are open.
1034	Not a directory.
1035	File pointer is not inside the file.
1036	Subscript out of range.
1037	No acknowledge received within specified time-
1038	Communication checksum error.
1039	Not mounted.
1040	Unknown file operating system.
1041	Error in fos structure.
1042	Internal error in mcs.
1043	Timer table full.
1044	Low battery in memory card.
1045	Media was removed.
1046	Memory checksum error.
1047	Interrupted system call.
1051	Dot resistance measure out of limits.
1052	Error in printhead.
1053	Unable to complete a dot measurement.
1054	Error when trying to write to device.
1055	Error when trying to read from device.
1056	O_BIT open error.
1057	File exists.
1058	Transfer ribbon is installed.
1059	Cutter does not respond.
1061	Wrong type of media.
1062	Not Allowed.
1067	Is a directory
1073	Directory not empty
1076	Permission denied
1077	Broken pipe
1081	Timer expired
1082	Unsupported protocol
1083	Ribbon low
1084	Paper low
1085	Connection timed out
1086	Secret not found
1087	Paper Jam
1101	Illegal character in bar code.
1102	Illegal bar code font.
1103	Too many characters in bar code.
1104	Bar code too large.
1105	Bar code parameter error.
1106	Wrong number of characters.
1107	Illegal bar code size.
1108	Number or rows out of range.
1109	Number of columns out of range.
1201	Insufficient font data loaded.
1202	Transformation matrix out of range.
1203	Font format error.
1204	Specifications not compatible with output
1205	Intelligent transform not supported.
1206	Unsupported output mode requested.
1207	Extended font not supported.
1208	Font specifications not set.
1209	Track kerning data not available.
1210	Pair kerning data not available.
1211	Other Speedo error.
1212	No bitmap or outline device.
1213	Speedo error six.
1214	Squeeze or clip not supported.

Code	Message/Explanation
1215	Character data not available.
1216	Unknown font.
1217	Font format is not supported.
1218	Correct mapping table is not found.
1219	Font is in the wrong direction.
1220	Error in external map table.
1221	Map table was not found.
1222	Double byte map table is missing.
1223	Single byte map table is missing.
1224	Character map function is missing.
1225	Double byte font is not selected.
1301	Index outside collection bounds.
1302	Collection could not be expanded.
1303	Parameter is not a collection.
1304	Item not a member of the collection.
1305	No compare function, or compare returns faulty
1306	Tried to insert a duplicate item.
1601	Reference Font Not Found.
1602	Error in Wand-Device.
1603	Error in Slave Processor.
1604	Print Shift Error.
1605	No Hardware Lock.

Code	Message/Explanation
1606	Testfeed not done.
1607	General Print Error.
1608	Access Denied.
1609	Specified Feed Length Exceeded.
1610	Illegal Character Map File.
1701	Cutter Error1
1702	Cutter Error2
1703	Cutter Error3
1704	Cutter open
1710	Power supply Generic Error
1711	Power supply Pending
1712	Power supply Status OK
1713	Power supply Power Fail
1714	Power supply Over Volt V24
1715	Power supply Under Volt V24
1716	Power supply Over Volt VSTM
1717	Power supply Under Volt VSTM
1718	Power supply Over Temperature
1719	Power supply Error
1820	No route to host
1821	Disc quota exceeded
1833	Connection refused

Intermec Technologies Corporation
Corporate Headquarters
6001 36th Avenue West
Everett, WA 98203
U.S.A.

tel +425.348.2600

fax +425.355.9551

www.intermec.com

Intermec Fingerprint v8.20—Programmer's Reference Manual

1-960582-02