Reference Manual

P/N 1-960367-02
Edition 3
September 1998

Intermec Fingerprint 6.13

Intermec

A UNOVA Company

CONTENTS

Continued!

Intermec Fingerprint 6.13
Reference Manual
Edition 3, September 1998
Part No. 1-960367-02

CONTENTS, cont'd.

Program Instructions, cont'd.

Continued!

CONTENTS, cont'd.

Program Instructions, cont'd.

Continued!

CONTENTS, cont'd.

PREFACE

Intermec Fingerprint is a *Basic*-inspired, printer-resident programming language that has been developed for use with computer-controlled direct thermal and thermal transfer printers manufactured by *Intermec Technologies Corp*.

Intermec Fingerprint 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 *Intermec Fingerprint 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*. A digested version of the *Reference Manual* is available as a *Windows*-compatible Help file for use on a PC.

The Reference Manual is supplemented by the tutorial manuals *"Intermec Fingerprint Programmer's Guide"* and *"Intermec Direct Protocol Programmer's Guide,* which describe how to start up *Intermec Fingerprint* and how to use the various instructions in their proper context.

All information needed by the operator, like how to run the printer, how to load the paper supply and how to maintain the printer, can be found in the *Operator's Guide* and the *User's Manual* for the printer in question.

In the *Technical Manual* for each type of printer you will find information on installation, setup, print resolution, paper specifications, relations between printhead and paper, and other technical information, which is specific for the printer model in question. It also includes information on optional equipment like interface boards, label-taken sensors, cutters, rewinders, and memory cards.

NEWS IN INTERMEC FINGERPRINT 6.13

Compared to the last version of Intermec Fingerprint Reference Manual, i.e. Intermec Fingerprint 6.0, this new version contains the following improvements and enhancements:

- **General *Intermec Fingerprint* enhancements:**
  New setup option for high resistance transfer ribbon (Intermec HR 31) for *EasyCoder 501* with 11.81 dots/mm printhead density.
  A paper cutter can now be fitted on all models of *EasyCoder 501*.

- **Corrections and improvements of RS 422/485 interface:**
  Previously, to decide between RS 422 and RS 485, the XON/XOFF option "Data to host" was used. This has been changed to look at "Data from host" instead, so as to allow the host to send binary data on RS 422 to the printer with XON/XOFF flow control.

  RS 422 (4-wire):
  • PROT_ADDR=DISABLE; XON/XOFF,DATA FROM HOST=ENABLE

  RS 485 (2-wire) point-to-point:
  • PROT_ADDR=DISABLE; XON/XOFF,DATA FROM HOST=DISABLE

  RS 485 (2-wire) multidrop loop:
  • PROT_ADDR=ENABLE; XON/XOFF,DATA FROM HOST=DISABLE

  Not used:
  • PROT_ADDR=ENABLE; XON/XOFF,DATA FROM HOST=ENABLE

  Previously, when
  PROT_ADDR=DISABLE; XON/XOFF,DATA FROM HOST=DISABLE
  was selected, the printer was erroneously put into send mode. The only way around this was to send a character to the port. Now, the interface is set to reception mode and the dummy write is no longer necessary.

- **Improvements of RS 485 interface:**
  After the port has been set for transmission, a delay for at least 10 ms is inserted before writing the data. This is done to take care of a hardware deficiency, which states that a stabilization time is needed after the loop has been turned.

  A possible break character is taken care of if PROT_ADDR=ENABLE and break handling for the RS 485 channel is enabled.

  Previously, it was not possible to use addresses over 9, when the printer was appointed "master". Now, it is possible to use addresses 0 – 31.

Continued!

NEWS IN INTERMEC FINGERPRINT 6.13, cont'd.

❑ **Extended Instructions:**

• **ERROR**
This statement now can set a specified error, in addition to enabling error-handling and creating error messages in the *Intermec Direct Protocol*.

• **FILE& LOAD**
An optional leading parameter has been added that specifies the number of characters to ignore before the real data. This makes it possible to use the instruction as an MS/DOS command (CR/LF problem). The instruction is compatible with *Intermec Fingerprint* 6.0.

• **IMAGE LOAD**
An optional leading parameter has been added that specifies the number of characters to ignore before the real data. This makes it possible to use the instruction as an MS/DOS command (CR/LF problem). The instruction is compatible with *Intermec Fingerprint* 6.0.

• **LAYOUT**
Two layout types have been added:
E = Bar code extended field, sets up complex bar code in regard of:
- Security
- Aspect height
- Aspect width
- Rows in bar code
- Column in bar code
- Truncation

This corresponds to the 6 last parameters in the BARSET statement.
J = Baradjust (adjust left or adjust right)
This corresponds to the BARADJUST statement.

• **SYSVAR**
New parameter. SYSVAR (25). Not intended for public use.

• **VERSION$**
Supports *EasyCoder 401 Linerless* and CPU board 1-040700-30.

❑ **New Instruction:**

• **FONT LOAD**
This instruction downloads and converts .ATF fonts to the printer's internal font format.

NEWS IN INTERMEC FINGERPRINT 6.13, cont'd.

- **Corrections:**
  - **pcx2bmp/IMAGE LOAD**
    When converting a .PCX file to the internal bitmap format (bmp), the picture was cutted if the width was divisible by 8. Image name name was in some cases corrupt after conversion.
  - **IMAGE LOAD**
    If COM ERROR ON was used at the same time, the printer was echoing characters and the transmission was interrupted.
  - **STORE IMAGE**
    UBI01–UBI03 protocols: Checksum calculation was wrong.
    UBI03 protocol was waiting for "*" after checksum.
  - **FORMFEED**
    FORMFEED was not accepted in some application programs. In those applications, FORMFEED 0 worked ok.
  - **FUNCTEST**
    FUNCTEST was consuming memory, about 300 bytes each time.
  - **SPLIT**
    First time after reboot, the splitting did not work if the split array was initialized with empty strings.
  - **BARFONT ON**
    If for example FILE& LOAD was used on a previous line, BARFONT ON syntax could be corrupted.
  - **FORMAT**
    Use of an invalid device name, e.g. FORMAT "qwerty", formatted the current directory instead of producing the error message *"Device not found"*.
  - **REBOOT**
    The DTR signal is now dropped on "uart2:"/"uart3:" at reboot.

- **Other changes:**
  UBI-related text and images removed.
  The image UBI.1 replaced by GLOBE.1.

NEWS IN INTERMEC FINGERPRINT 6.13, cont'd.

- **Remaining bugs:**
  - **BARFONT [ON]/STORE IMAGE KILL**
    If an image is stored in the temporary memory (by STORE IMAGE KILL) and an EAN 13 code with human readables is printed in front of the image, the image is corrupted the first time after reboot.
  - **OPTIMIZE BATCH ON/FORMFEED**
    If OPTIMIZE BATCH ON is selected and FORMFEED 2 is entered, the printer feeds paper and hangs.
  - **Error 1048**
    When error 1048 *"Transfer ribbon is installed"* occurs, the printer enters an infinitite loop, repeating the message.

- **Limitations:**
  - **RIBBON SAVE ON**
    Prints incomplete labels in some cases.
    Workaround solution:
    Increase image buffer size in setup.
    *Estimated size = (Label width in dots/8) * (Label length in dots)*
  - **ACTLEN**
    ACTLEN measures only last printed section instead of the whole label, e.g. on printers fitted with a ribbon save device.
  - **COMSET/LOC on Std IN channel**
    It is not possible to handle NULL in strings.
  - **PRINTFEED in Direct Protocol**
    In the layout mode, PRINTFEED (PF) is slow when printing identical labels.

PROGRAM INSTRUCTIONS

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 keywords.

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

The following abbreviations will be 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 *FUNCTION*

Field of Application **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 *FUNCTION*

Field of Application

Returning the length of the most recently executed PRINTFEED, FORMFEED, or TESTFEED statement.

Syntax

ACTLEN

Remarks

The length of the most recently executed paper feed operation, resulting from 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.

Also see page 9 for remaining bugs and limitations.

Example

In this example, the printer is loaded with 90,5 mm (724 dots) long labels separated by a 3 mm (24 dots) gap. Start- and stop adjust setup values are both 0:

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

yields:

```
755
```

The deviation from the expected result (748) is normal and should have no practical consequences (less than 1 mm).

ALIGN (AN) *STATEMENT*

Field Application

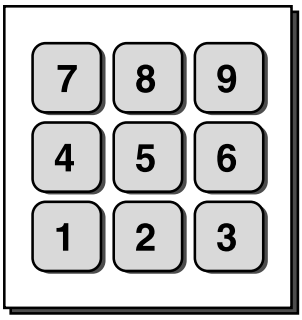
Specifying which part (anchor point) of a text field, bar code field, image field, line or box will be positioned at the insertion point.

Syntax

ALIGN|AN<nexp>

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

Remarks

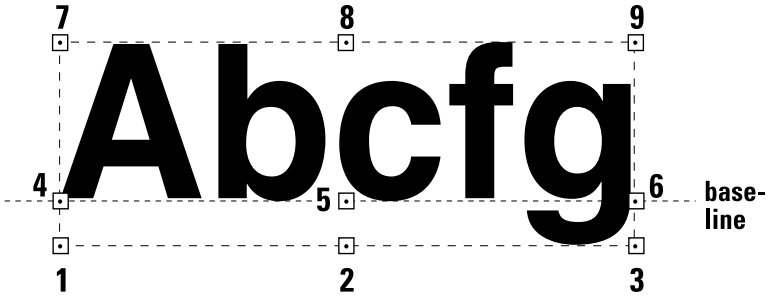


Each text, bar code or image field has nine possible anchor points, whereas lines and boxes have three. One of these points must be selected, or the default value (1) will be used. The selected anchor point decides the position of the object in relation to the insertion point, which is decided by the 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 e.g. the keys on the numeric part of a computer keyboard, as illustrated to the left.

Lines and boxes have three anchor points only – left, centre and right.
The anchor points for the various types of field are illustrated below.

Text field:



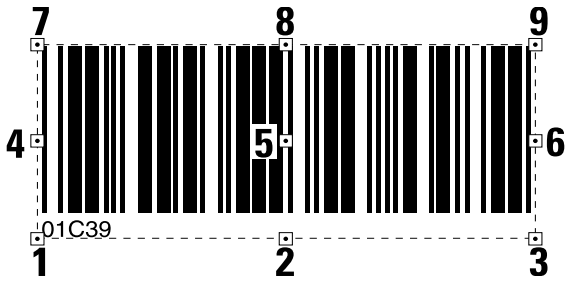
A text field makes up an imaginary box limited in regard of width by the length of the text, and in regard of height by the matrix size of the selected font. In text fields, the anchor points “4”, “5” and “6” are situated on the baseline, as opposed to bar code fields and image fields.

Continued!

ALIGN (AN), cont'd. *STATEMENT*

Remarks, cont'd.

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:

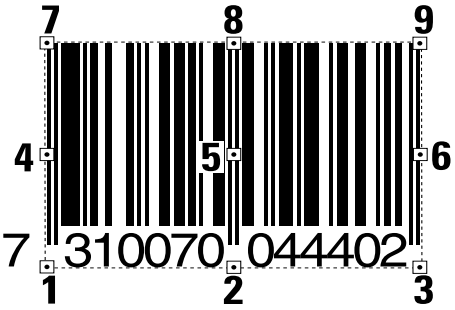
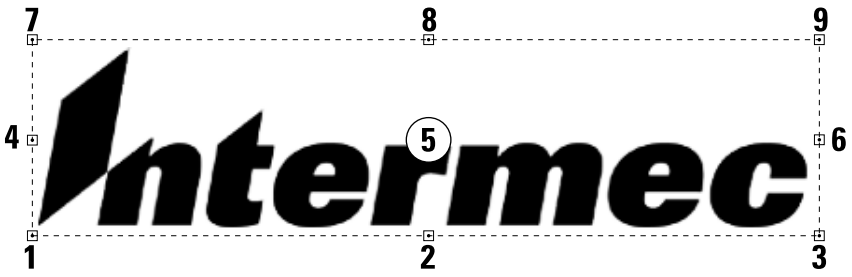


Image field:



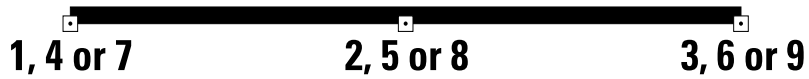
The size of an image field is decided when the field is created. Note that an image field consists of the entire area of the original image, even possible white or transparent background.

Continued!

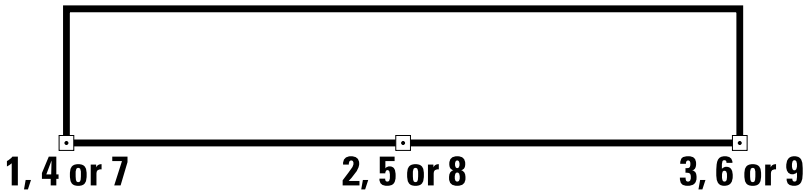
ALIGN (AN), cont'd. *STATEMENT*

Remarks, cont'd.

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.

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 "SW030RSN"
50    PRTXT "Hello!"
60    PRINTFEED
RUN
```

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

ASC FUNCTION

Field of Application

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 STATEMENT

Field of Application

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 (No. of dots).*
<nexp$_2$> *is the maximum **right** offset (No. of dots).*

Default: *0,0 (i.e. 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 paper web. 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 – if not the cause – 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 (e.g. Code 16K)
- Bar codes with horizontal lines (e.g. 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 "SW030RSN" ON
50    PRBAR "ABC"
60    BARADJUST 0,0
70    PRINTFEED
```

BARFONT (BF) *STATEMENT*

Field of Application **Specifying fonts for the printing of bar code interpretation.**

Syntax

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

#<ncon>	*is, optionally, the start parameter in the syntax above.*
<sexp$_1$>	*is the designation of the first font selected for the bar code interpretation , optionally including extension.*
<sexp$_2$>	*is the designation of the second font selected for the bar code interpretation , optionally including extension.*
<nexp$_1$>	*is the distance in dots between bar code and bar font.*
<nexp$_2$>	*is the magnification in regard of height.*
<nexp$_3$>	*is the magnification in regard of width.*
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

Bar Code Interpretation Font:
The selected bitmap font must exist in the printer's memory and be specified by a string expression. Standard *Intermec* bitmap fonts are available in two version, one for printing across the paper and another for printing along the paper web, which is indicated by an extension to the font name:

- Extension **.1** means that the font can be used for horizontal printing, i.e. in print direction 1 and 3 (see DIR statement).
- Extension **.2** means that the font can be used for vertical printing, i.e. in print direction 2 and 4 (see DIR statement).

With the introduction of *Intermec Fingerprint 6.0*, you do not have to specify an extension in the BARFONT and FONT statements. The firmware keeps record of the print direction and selects the font with the specified name and the correct extension, provided such a font exists in the printer's memory.

The BARFONT statement allows you to select two different bar fonts, one for horizontal printing and another for vertical printing. If you want to use the same font for all directions, you could either enter the same name twice (optionally with different extensions), or change the start parameter value to 2 and enter the name once and without extension.

Default bar code interpretation font: None.

Continued!

BARFONT (BF), cont'd. *STATEMENT*

Remarks, cont'd.

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. (Refer to FONT statement for definition of the character cell).

Default value: 6

Magnification:
The bar code font can be magnified up to 4 times in regard of height and/or width. The two last 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

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.

Also see page 9 for remaining bugs and limitations.

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 #2,"SW030RSN",5,1,1 ON
60   PRBAR "ABC"
70   PRINTFEED
80   END
```

BARFONT ON/OFF (BF ON/OFF) *STATEMENT*

Field of Application **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 stmt.

Also see page 9 for remaining bugs and limitations.

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 #2,"SW030RSN",5,1,1
60    BARFONT ON
70    PRBAR "ABC"
80    PRINTFEED
90    END
```

BARHEIGHT (BH) *STATEMENT*

Field of Application **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, e.g. 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 "SW030RSN" ON
90   PRBAR "ABC"
100  PRINTFEED
```

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

BARMAG (BM) *STATEMENT*

Field of Application **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, as 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 ($2 \times 3:1 = 6:2$).

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 "SW030RSN" ON
90   PRBAR "ABC"
100  PRINTFEED
```

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

BARRATIO (BR) *STATEMENT*

Field of Application

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 absolute terms (i.e. 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
i.e. the wide bars are 6 dots wide and the narrow bars are 2 dots wide.

Note that certain bar codes have a fixed ratio, e.g. EAN and UPC codes. In those cases, any BARRATIO statement will be ignored. Refer to the chapter *"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 "SW030RSN" ON
90   PRBAR "ABC"
100  PRINTFEED
```

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

BARSET *STATEMENT*

Field of Application

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> *is the barcode type.*
<nexp$_1$> *is the ratio of the large bars.*
<nexp$_2$> *is the ratio of the small bars.*
<nexp$_3$> *is the enlargement.*
<nexp$_4$> *is the height of the code in dots.*
<nexp$_5$> *is the security level according to bar code specification.*
<nexp$_6$> *is the aspect height ratio.*
<nexp$_7$> *is the aspect width ratio.*
<nexp$_8$> *is the number of rows in the bar code.*
<nexp$_9$> *is the number of columns in the bar code.*
<nexp$_{10}$> *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 (optional), it can be used for any type of bar code if non-relevant parameters are left out (e.g. <nexp$_5$> – <nexp$_{10}$>).

Start Parameter:
Start parameter specifies which parameter in the syntax above should be the first parameter (#1–11). Thereby you may bypass some of the initial parameters, e.g. barcode type, ratio, and enlargement.

Default value: #1

Bar Code Type:
The bar code type parameter corresponds to the BARTYPE statement.

Default bar code: "INT2OF5"

Bar Code Ratio:
The two ratio parameters correspond to the BARRATIO statement.

Default value: 3:1

Enlargement:
The enlargement parameter corresponds to the BARMAG statement.

Default value: 2

Continued!

BARSET, cont'd. *STATEMENT*

Remarks, cont'd.

Bar Code Height:
The height parameter corresponds to the BARHEIGHT statement.

Default value: 100 dots

Security Level:
The security level is only used in some complex bar codes, e.g. PDF417, and should be used according to the specifications of the bar code in question.

Default value: 2

Aspect Ratios:
The aspect height ratio and aspect width ratio is used for complex bar codes, e.g. PDF417, 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 specifications of the bar code for allowed input.

Default values:
1 for aspect ratio height
2 for aspect ratio width.

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

Truncate Flag:
The truncate flag is used in some complex bar codes, e.g. PDF417, to omit parts of the code pattern. Refer to the specifications of the bar code for allowed input.

Default value: 0

Example

This example shows how a BARSETstatement is used to specify a Code 39 bar code (compare e.g. 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 #2,"SW030RSN",5,1,1 ON
60   PRBAR "ABC"
70   PRINTFEED
```

BARTYPE (BT) *STATEMENT*

Field of Application **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 the chapter *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 "SW030RSN" ON
90   PRBAR "ABC"
100  PRINTFEED
```

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

BEEP STATEMENT

Field of Application **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 STATEMENT

Field of Application

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:"/"rs485:"
3 = "uart3:"

<nexp$_2$> *is the decimal ASCII value for the desired break interrupt character .*

Default: *Comm. channels:* *ASCII 03 decimal*
Console: *ASCII 158 decimal(< Pause >+< C >)*

Remarks

The execution of a program can be interrupted. The same applies to the printing of a batch of labels in the Immediate Mode or in the *Intermec Direct Protocol*. Batch printing initiated by a PRINTFEED <nexp> statement in a program cannot be interrupted.

To issue a break interrupt, **by default**, hold down the **C**-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 chapter *"Printer Function Control; Keyboard"* in *"Intermec Fingerprint, Programmer's Guide"* for more information.

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 (i.e. INPUT [#], LINE INPUT [#] and INPUT$).

The BREAK statement allows you to specify other ways of interrupting the execution, e.g. 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 no-save area of the RAM memory until the printer is restarted or REBOOTed, which may be confusing e.g. when changing 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.

Continued!

BREAK, cont'd. *STATEMENT*

Remarks, cont'd.

It is strongly recommended to include some facility for issuing a break interrupt from the host computer in startup (autoexec) files, especially when using a printer model without any keyboard. If not, you may find yourself with an erroneous program running in a loop without being able to break it, except for removing the RAM packages and thereby erasing the complete RAM memory!

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 key No. 10 (usually marked "F1") on the printer's keyboard, or by typing an uppercase A on the keyboard of the host:

```
10    BREAK 0,1:BREAK 1,65
20    BREAK 0 ON:BREAK 1 ON
30    GOTO 30
RUN
```

In the Intermec Direct Protocol or the Immediate Mode, you can break the printing of a batch of labels. This example shows how the <Pause> key on the printer's built-in keyboard is used for that purpose:

```
BREAK 0,30:BREAK 0 ON:FT "SW030RSN":PT "HELLO":PF 10↵
```

BREAK ON/OFF *STATEMENT*

Field of Application

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:"/"rs485:"	
	3 = "uart3:"	
Default:	*Comm. ports:*	*Disabled*
	Console:	*Enabled*

Remarks

By default, the execution of a program (or the printing of a batch of labels in the Immediate Mode or the *Intermec Direct Protocol*) can be interrupted by simultaneously pressing down the <**Pause**> and <**C**> keys on the printer's keyboard, if any. Other ways of issuing a break interrupt can also be specified, see BREAK statement.

The use of the break interrupt 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 no-save area of the printer's RAM 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 STATEMENT

Field of Application

Ordering a busy signal, e.g. 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:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

Remarks

The selected communication protocol usually contains 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, as described in the *Technical Manual*. (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:

```
10    FONT "SW030RSN"
20    PRTXT "HELLO!"
30    BUSY2
40    PRINTFEED
50    READY2
RUN
```

CHDIR STATEMENT

Field of Application

Specifying the current directory.

Syntax

CHDIR<scon>

<scon> *specifies the current directory as "rom:", "ram:", or "card1:".*
Default: *"ram:"*

Remarks

By default, the RAM memory ("ram:") is the current directory, i.e. the directory that is selected if the *Intermec Fingerprint* instruction does not contain any reference to a directory, e.g. FILES. This implies that to access the ROM memory ("rom:") or an optional DOS-formatted memory card ("card1:"), you must include such a reference in your instructions, e.g. FILES "rom:".

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

Example

In this example, the current directory is changed to "rom:", all files in "rom:" are listed and finally the current directory is changed back to "ram:". (Normally, use FILES "rom:" for this purpose).

```
10    CHDIR"rom:"
20    FILES
30    CHDIR"ram:"
RUN
```

yields e.g.:

```
Files on rom:

MKAUTO.PRG    124 FILELIST.PRG117

71776 bytes free   1252 bytes used
```

CHECKSUM *FUNCTION*

Field of Application

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 *Intermec Fingerprint* 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.

Note:
This function was modified in Intermec Fingerprint 4.0. Calculating the checksum of program lines created in earlier versions of Intermec Fingerprint, may result in other checksums than before.

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$ FUNCTION

Field of Application **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

Only integers between 0 and 255 are allowed. Input less than 0 or larger than 255 will result in an error condition (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 *STATEMENT*

Field of Application **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 (the rubber roller beneath the printhead) and other roller's connected to the stepper motor of the paper feed mechanism. In case of thermal transfer printers, it also often drives the ribbon mechanism. The motor will run regardless of possible error conditions, e.g. if the printhead is lifted or not, or if there is no ribbon or paper 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, i.e. as when feeding out a label.

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

The execution of a CLEANFEED statement 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, must be specified in regard of feed length.

Example

In order to pull a cleaning card back and forth under the printhead in an 8 dots/mm printer, a 100 mm long negative CLEANFEED and then the same amount of positive CLEANFEED is performed. The operation is repeated twice:

```
10    FOR A%=1 TO 3
20    CLEANFEED -800
30    CLEANFEED 800
40    NEXT
RUN
```

CLEAR *STATEMENT*

Field of Application **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(9)
90   CLEAR
100  PRINT "String A after:     ";A$
110  PRINT "String B after:     ";B$
120  PRINT "Free memory after:  ";FRE(9)
RUN
```

yields: e.g.

```
String A before:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
String B before:    abcdefghijklmnopqrstuvwxyz
Free memory before: 377808
String A after:
String B after:
Free memory after:  386640

Ok
```

CLL STATEMENT

Field of Application **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, i.e. 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.

  An example of using CLL in connection with errors, is error 43 (*"Memory overflow"*). Each time a new font or image is used, it will be added to an internal font or image table, that can contain max. 40 fonts or images respectively. The tables are cleared automatically at power-up or REBOOT, but should any of these tables become overfilled, error 43 occurs. Instead of turning the printer off and then on, or performing a REBOOT, a CLL statement can be used to clear the tables.

Continued!

CLL, cont'd. *STATEMENT*

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 "SW030RSN"
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 no. 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 *STATEMENT*

Field of Application **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 *STATEMENT*

Field of Application

Enabling/disabling error handling on the specified communication channel.

Syntax

COM↔ERROR<nexp>ON|OFF

<nexp> *is one of the following comm. channels:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
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, COM STAT 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.

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$ *FUNCTION*

Field of Application **Reading the data in the buffer of the specified communication channel.**

Syntax

COMBUF$(<nexp>)

<nexp> *is one of the following comm. channels:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

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.

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 STATEMENT

Field of Application

Setting the parameters for background reception of data to the buffer of a specified communication channel.

Syntax

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

<nexp$_1$>	*is one of the following comm. channels:* *0 = "console:"* *1 = "uart1:"* *2 = "uart2:"/"rs485:"* *3 = "uart3:"* *4 = "centronics:"*
<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.*

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 condition 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 max. number of characters also decides how much of the memory will be allocated to the buffer.

Continued!

COMSET, cont'd. *STATEMENT*

Remarks, cont'd.

The reception of data to the buffer can be interrupted by four conditions:
- An end string being encountered.
- An attention string being encountered.
- 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, i.e. 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.

Note that COMSET filters out out any incoming ASCI 00 dec. characters (NUL).

Also see page 9 for remaining bugs and limitations.

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 *STATEMENT*

Field of Application

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

Syntax

COMSET<nexp>OFF

<nexp> *is one of the following comm. channels:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

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 — *STATEMENT*

Field of Application

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

Syntax

COMSET<nexp>ON

<nexp> *is one of the following comm. channels:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

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 *FUNCTION*

Field of Application **Reading the status of the buffer of the specified communication channel.**

Syntax

COMSTAT(<nexp>)

<nexp> *is one of the following comm. channels:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

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 ..... 0 or 1
• Interruption: Max. number of characters received 2
• Interruption: End character received 4
• Interruption: Attention string received 8
• Interruption: Communication error 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
```

COPY STATEMENT

Field of Application

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 device for the copy.*

Remarks

This statement allows you to copy a file to another name and/or device 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 "ram:", i.e. the printer's internal RAM memory. If the file is to be 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 RAM 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, "ram:" 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 "rom:" to the current directory and changing the file name:

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

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

```
COPY "rom: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& STATEMENT

Field of Application

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, i.e. 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, e.g. 1, 2 or 3. When used for printing, they are referred to by "**CNT<ncon>$**" variables, where <ncon> is the number of the counter as specified by COUNT&, e.g. CNT**5**$.

A counter variable without a matching counter will be regarded as a common string variable.

The parameter 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)

Continued!

COUNT&, cont'd *STATEMENT*

Remarks, cont'd.

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 (i.e. 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 the value by which 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.

Default: 2,147,483,647 (numeric) or Z (alpha)

RESTART:
Decides the value at which 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 STATEMENT

Field of Application

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)* *2: Diagonal redundancy check (DRC)*
<svar>	*is the array of strings of which the checksum is to be calculated.*
<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.

LRC:
The even parity of all character bits in the array columnwise.
Algorithm: LRC = LRC XOR (next character)
(Initial value of LRC is 1:st character in the array.)

DRC:
The even parity of all character bits in the array diagonally.
Algorithm: DRC = (Rotate Right DRC) XOR (next character)
(Initial value of DRC is 1:st character in the array.)

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
```

CUT STATEMENT

Field of Application **Activating an optional paper-cutting device.**

Syntax

```
CUT
```

Remarks

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

When a PRINTFEED statement is executed, the printer feeds out a certain amount of the web according to the printer's setup in regard of startadjust and stopadjust, as explained in its *Technical Manual*. The paper feed can be further adjusted by a FORMFEED statement appended by a positive or negative value, which specifies an additional amount of paper to be fed out or withdrawn. Then the cutter can be activated by a CUT statement.

Example

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

```
10   PRPOS 250,250
20   DIR 1
30   ALIGN 4
40   FONT "SW030RSN"
50   PRTXT "Hello everybody!"
60   PRINTFEED
70   FORMFEED 280
80   CUT
90   FORMFEED -280
RUN
```

CUT ON/OFF *STATEMENT*

Field of Application

Enabling or disabling automatic cutting after PRINTFEED execution and optionally adjusting the paper feed before and after the cutting.

Syntax

CUT [<nexp>] ON|OFF

<nexp> *is optionally the amount of paper 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 paper feed in connection with the cutting operation is required, use startadjust and stopadjust setup, FORMFEED statements, or specify the desired amount of paper to be be fed out before the cutting is performed and pulled back afterwards in the CUT ON statement.

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 web. The paper is then pulled back the same distance. Compare with the example for the CUT statement:

```
10    CUT 280 ON
20    PRPOS 250,250
30    DIR 1
40    ALIGN 4
50    FONT "SW030RSN"
60    PRTXT "Hello everybody!"
70    PRINTFEED
RUN
```

DATE$ VARIABLE

Field of Application **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.*

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	(e.g. 1998 = 98)
MM	=	Month	Two digits	(01–12)
DD	=	Day	Two digits	(01–28|29|30|31)

Example: = October 25, 1998 is entered as "981025".

The built-in calendar corrects illegal values for the years 1980 – 2048, e.g. the illegal date 981232 will be corrected to 990101.

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$ = "981025"                    (sets date)
20    FORMAT DATE$ "DD/MM/YY"             (sets date format)
30    PRINT DATE$                         (returns unformatted date)
40    PRINT DATE$("F")                    (returns formatted date)
RUN
```

yields:

```
981025
25/10/98
```

DATEADD$ *FUNCTION*

Field of Application

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, i.e. in the order YYMMDD, where:

YY	=	Year	Last two digits	(e.g. 1998 = 98)
MM	=	Month	Two digits	(01–12)
DD	=	Day	Two digits	(01–28|29|30|31)

Example: October 25, 1998 is entered as "981025".

The built-in calendar corrects illegal values for the years 1980 – 2048, e.g. the illegal date 981232 will be corrected to 990101.

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$ = "981025"
20    A%=15
30    B%=-10
30    FORMAT DATE$ "DD/MM/YY"
40    PRINT DATEADD$("971025",A%)
50    PRINT DATEADD$("971025",A%,"F")
60    PRINT DATEADD$(B%,"F")
RUN
```

yields:

```
981109
09/11/98
25/10/98
```

DATEDIFF *FUNCTION*

Field of Application **Returning the difference between two dates as a number of days.**

Syntax

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

<sexp$_1$> *is one of two dates.*
<sexp$_2$> *is the other of two dates.*

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 (date1) 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, i.e. in the following order:

Year	Last two digits	(e.g. 1997 = 98)
Month	Two digits	(01–12)
Day	Two digits	(01–28|29|30|31)

Example: October 25, 1998 is entered as "981025".

The printer's calendar corrects illegal values for the years 1980 – 2048, e.g. the illegal date 981232 will be corrected to 990101.

Examples

Calculation of the difference in days between the dates October 1, 1998 and November 30, 1998:

```
10    A%=DATEDIFF("981001","981130")
20    PRINT A%
RUN
```

yields:

```
60
```

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

```
10    A%=DATEDIFF("981130","981001")
20    PRINT A%
RUN
```

yields:

```
-60
```

DELETE *STATEMENT*

Field of Application

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.*

DEVICES STATEMENT

Field of Application

Returning the names of all devices to the standard OUT channel.

Syntax

DEVICES

Remarks

All devices available in the *Intermec Fingerprint* firmware will be listed, regardless if they are installed or not. 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 appended by a colon (:).

Device	Explanation	Can be OPENed for...
uart1:	Serial communication port	Input/Output
uart2:	Serial communication port	Input/Output
uart3:	Serial communication port	Input/Output
centronics:	Parallel communication port	Input
console:	Printer's display and/or keyboard	Input/Output
ram:	Printer's internal RAM memory	Input/Output/Append/Random
rom:	Printer's ROM memory[1]	Input
cutter:	Optional paper-cutting device	n.a.
prel:	Reliable protocol (RS 485 only)[2]	Input/Output
rs485:	RS 485 protocol[2]	Input/Output
card1:	Optional DOS-formatted card	Input/Output/Append/Random
msg:	Implementation of SITA CUTE 2 [3]	Input/Output
par:	Implementation of SITA CUTE 2 [3]	Input/Output
bscrypt:	Internal use only	–
null:	Internal use only	–
ind:	Internal use only	–

[1]/. EPROM's and possible non DOS-formatted memory card.
[2]/. Presently not available for Intermec EasyCoder 101
[3]/. All instructions for the SITA CUTE 2 protocol are excluded from this manual.

Example

DEVICES *yields:*

```
uart3:
ind:
uart2:
rs485:
centronics:
par:
prel:
msg:
card1:
null:
console:
uart1:
bscrypt:
rom:
ram:
cutter:
```

DIM STATEMENT

Field of Application

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).

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 STATEMENT

Field of Application **Specifying the print direction.**

Syntax

DIR<nexp>

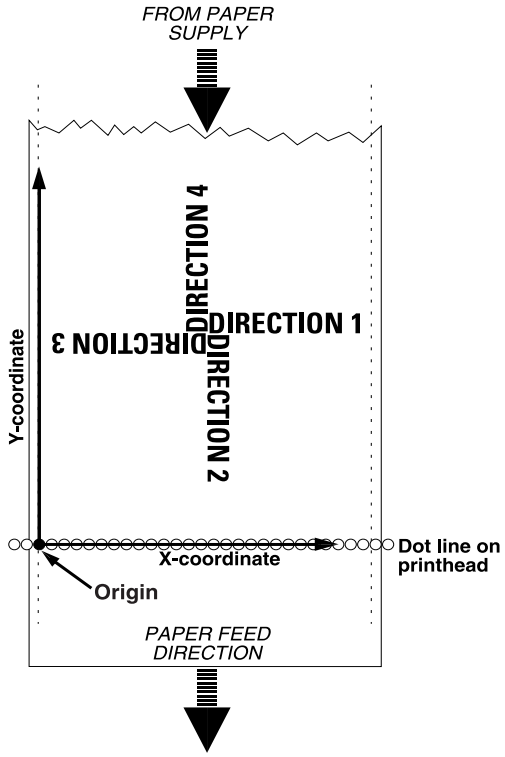
<nexp> *is the print direction (1, 2, 3, or 4).*
Default value: *1*
Reset to default by: *PRINTFEED execution or SETUP files*

Remarks

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

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

Text:

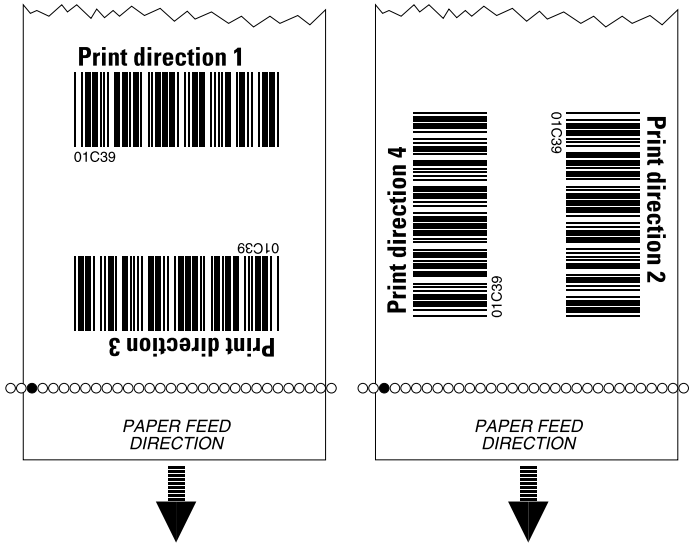


Continued!

DIR, cont'd. *STATEMENT*

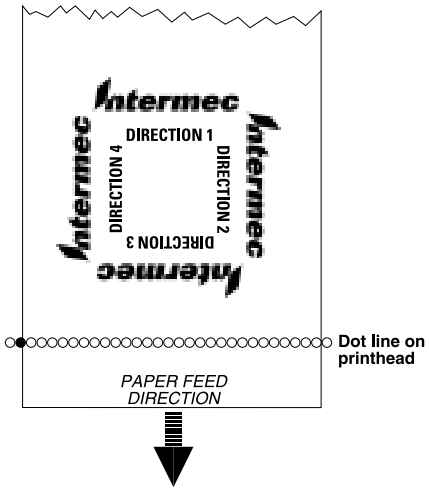
Remarks, cont'd.

Bar Codes:



Horizontal "picket fence" printing. *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:

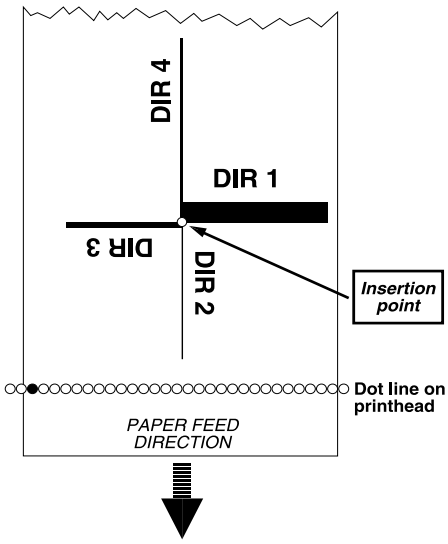
DIR 1 & 3, use extension **.1**
DIR 2 & 4, use extension **.2**

Continued!

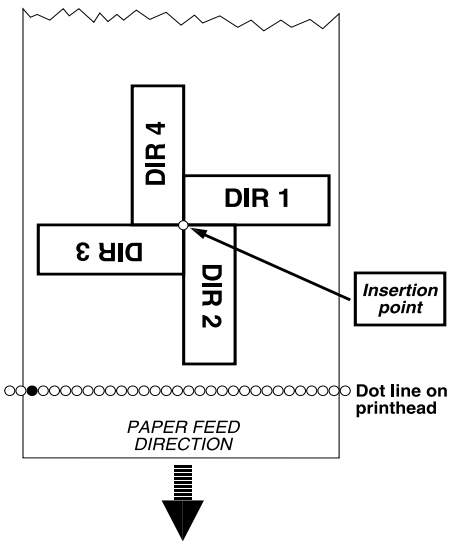
DIR, cont'd. *STATEMENT*

Remarks, cont'd.

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    MAG 3,3
50    FONT "SW030RSN"
60    PRTXT "TEXT PRINTING"
70    PRPOS 30,290
80    PRLINE 550,5
90    PRINTFEED
RUN
```

Printing the same information vertically necessitates new positioning to avoid a "Field out of label" error condition (Error 1003):

```
10    PRPOS 300,30                    (new position)
20    DIR 4                           (new direction)
30    ALIGN 4
40    MAG 3,3
50    FONT "SW030RSN"
60    PRTXT "TEXT PRINTING"
70    PRPOS 310,30                    (new position)
80    PRLINE 550,5
90    PRINTFEED
RUN
```

END *STATEMENT*

Field of Application

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"SW030RSN"
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.

END IF *STATEMENT*

Field of Application **Ending multiple IF...THEN...ELSE statements**

Syntax

```
END IF
```

Remarks See IF...THEN...ELSE statement. The space character between END and IF is optional.

EOF *FUNCTION*

Field of Application **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 *FUNCTION*

Field of Application **Returning the number of the line on which an error condition has occurred.**

Syntax

```
ERL
```

Remarks Useful in connection with an ON ERROR GOTO statement.

Examples *In this example, the line number of the line, where an error has occurred, decides the action to be taken:*

```
10   ON ERROR GOTO 1000
.....
.....
.....
100  PRTXT "HELLO"
110  PRINTFEED
120  END
.....
.....
.....
1000 IF ERL=110 THEN PRINT "PRINT ERROR"
1010 RESUME NEXT
```

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

```
PRINT ERL
```

yields e.g.

```
1010
```

ERR *FUNCTION*

Field of Application **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 the list *"Error Messages"* at the end of this manual. 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 e.g.

```
1022
```

ERROR STATEMENT

Field of Application

Setting an error (all kinds of *Intermec Fingerprint* programming), or 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

This statement can be used in two ways:

- **ERROR<nexp>** calls the error handler with the specified error code number, thereby simulating the error. This facility is available in both common *Intermec Fingerprint* programming and in the *Intermec Direct Protocol*.
- **ERROR<nexp>,<sexp>** 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 errorhandler of the *Intermec Direct Protocol* will always handle the following error conditions (also see *Intermec Direct Protocol*, Programmer's Guide):
- Out of paper
- No field to print
- Head lifted
- Out of transfer ribbon
- Next label not found

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 chapter *"Printer Function Control; Display"* in the *Intermec Fingerprint* Programmer's Guide. An empty string removes any previously entered error message for the error in question. Likewise, a previously entered messages string can be replaced by a new one.

Continued!

ERROR, cont'd. *STATEMENT*

Remarks, cont'd.

When an error defined by an ERROR statement is detected, the printer sets its standard IN port BUSY and displays the error messages. The error message will be cleared and the standard IN port will be set READY when the printer's < **Print** > key is pressed. However, in case of the standard errors, the error condition must also be physically dealt with, e.g. by loading a fresh stock of labels or lowering the printhead.

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.

Note that the ERROR statements affects 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, i.e. 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 at the end of this manual).

Examples

In this example, error No. 1005 "Out of paper" is simulated:

```
ERROR 1005
```

yields :

```
Out of paper
```

In this example, just a few errors are specified. Note the blank spaces for character position 17 in each message (space characters marked by bullets):

```
ERROR 43,"MEMORY••••••••••OVERFLOW" ↵
ERROR 1003,"FIELD• OUT • OF••••• LABEL" ↵
ERROR 1010,"HARDWARE •••••••••ERROR" ↵
ERROR 1029,"PRINTHEAD•VOLT-•• AGE • TOO • HIGH" ↵
```

FIELD *STATEMENT*

Field of Application

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$_{2\text{-}n}$>	*is the number of bytes to be reserved for the string variable that follows.*
<svar$_{1\text{-}n}$>	*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 by means of 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 *FUNCTION*

Field of Application

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 "SW030RSN"
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 STATEMENT

Field of Application

Reception and storing of binary files in the printer's RAM 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 RAM 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, and is useful for e.g. downloading outline font files.

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

The optional first parameter makes it possible to use this statement in MS-DOS (CR/LF problem), while retaining the compatibility with *Intermec Fingerprint 6.0*.

The name of the file, when stored in the printer's RAM 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, CTS/RTS 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 "ARIAL.TTF",65692,5
30   CLOSE 5
```

FILES STATEMENT

Field of Application

Listing the files stored in one of the printer's directories to the standard OUT channel.

Syntax

FILES[<scon>]

<scon> *optionally specifies the directory as "rom:", "ram:", or "card1:".*

Remarks

If no directory is specified, the files in the printer's *current* directory will be listed. As default, the current directory is the printer's RAM memory ("ram:"), but it can be changed to the EPROM memory ("rom:") or an optional DOS-formatted memory card ("card1:") by the use of a CHDIR statement.

FILES lists all files stored in the current directory.

FILES "rom:" lists all files stored in EPROM, including any inserted preprogrammed non DOS-formatted memory card.

FILES "ram:" lists all files stored in RAM.

FILES "card1:" lists all files stored in any inserted DOS-formatted memory card.

FILES also lists possible scalable outline fonts files.

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

Example

The presentation may look like this on the screen:

```
FILES
Files on ram:

LISTFONT.PRG  132     LABEL4.PRG   345
LABEL1.PRG    204     LABEL5.PRG   421
LABEL2.PRG     96     LABEL6.PRG    86
LABEL3.PRG    138     LABEL7.PRG   120

91958 bytes free      1542 bytes used
Ok
```

Note that all programs automatically get the extension .PRG, unless you manually give a program another extension.

FONT (FT) STATEMENT

Field of Application

Selecting a font for the printing of the subsequent PRTXT statements, and optionally generating a bitmap font from a scalable outline font file in Speedo or TrueType format.

Syntax

FONT|FT<sexp$_1$>	*(bitmap fonts)*
FONT|FT<sexp$_1$>[,<sexp$_2$>[,<nexp$_1$>[,<nexp$_2$>[,<nexp$_3$>[,<nexp$_4$>[,<sexp$_4$>[,<nexp$_5$>[,<nexp$_6$>[,<sexp$_5$>]]]]]]]]]	*(font scaling)*
FONT|FT<sexp$_1$>[,<sexp$_2$>[,<nexp$_1$>[,<sexp$_3$>[,<nexp$_4$>[,<sexp$_4$>[,<nexp$_5$>[,<nexp$_6$>[,<sexp$_5$>]]]]]]]]	*(font scaling)*

The first syntax applies to bitmap fonts, whereas the two latter syntaxes applies to generation of bitmap fonts from scalable outline fonts.

<sexp$_1$> *is the font name of an existing bitmap font or, in case of conversion of outline fonts, the desired name of the generated bitmap font. Max. 60 different fonts can be used.*
Default: No font selected.

<sexp$_2$> *is the file name of the outline font file.*

<nexp$_1$> *is the height in dots (default) or points (option) of the generated bitmap font (baseline to top of ascenders).*
Default: 40 dots.

<nexp$_2$> *is the decimal ASCII value of the first character in a range to be generated. Also see <sexp$_3$>. Default: 32.*

<nexp$_3$> *is the decimal ASCII value of the last character in a range to be generated. Also see <sexp$_3$>. Default: 126.*

<sexp$_3$> *is a string of characters to be generated. This is an alternative to defining characters by their ASCII values (see <nexp$_2$> and <nexp$_3$>).*
Default: All (ASCII 32 –126 dec. in Roman 8)

<nexp$_4$> *is the direction (1 or 2) of the generated font. Default: 1.*

<sexp$_4$> *is a flag (A, S, P, and/or B) that specifies the generated font to be:*
A: Added to an existing bitmap font.
S: Saved in the printer's RAM memory.
P: Defined in regard of height in points.
B: Height = the size of an uppercase A character (ascenders excluded).
Default: No flag.

<nexp$_5$> *is the slant of the generated bitmap font in degrees.*
Default: 0 degrees.

<nexp$_6$> *is the rotation of the generated bitmap font in degrees. Default: 0 degrees.*

<sexp$_5$> *is the name of an optional map file.*

Reset to default by: PRINTFEED execution or SETUP files

Continued!

FONT (FT), cont'd. STATEMENT

Remarks

Bitmap Fonts

Before any text can be printed, a suitable bitmap font must be selected and, optionally, generated. If not, an error condition will occur. There is no default font. If the font name is given as a string constant, it must be enclosed by double quotation marks ("....").

Since a large number of fonts are available on special request, and you in some printer models can generate your own bitmap fonts from scalable outline fonts in *Speedo* and *TrueType* formats, it is quite possible that your printer is fitted with a number of non-standard fonts. Use a FONTS statement to list the names of all fonts installed in your own printer to the standard OUT channel.

Usually, the fonts stored in the printers memory are organized in couples, where one font is used for vertical printing (i.e. DIR 2 & 4) and the other is used for horizontal printing (i.e. DIR 1 & 3). The extension, which appends the font name, indicates in which directions the font can be used (also see DIR statement):
.1 indicates that the font can be used in DIR 1 and 3, i.e. horizontal printing across paper web.
.2 indicates that the font can be used in DIR 2 and 4, i.e. vertical printing along the paper web.

A new feature introduced with *Intermec Fingerprint 6.0* is that it is no longer necessary to include the extension in the font name. If no extension is given, the firmware will search the memory for a font with the specified name and correct direction, regardless of extension. However, if an extension is given in the font name, the firmware will only search for a font with the correct name and extension.

Even if you no longer need to specify the extension in your FONT and BARFONT statements, you will still need each bitmap font in two versions for both horizontal and vertical printing, unless you consciously choose not to use the font in one direction. Note that it is not the extension itself that specifies the direction of the font. It is just a convention to make it easier to keep tabs on the font. We strongly recommend that you stick to that convention also when generating your own bitmap fonts.

A bitmap font may be magnified up to 4 times separately in regard of height and width by means of a MAG statement.

Note that both “text” bitmap fonts and barfonts come from the same sets of bitmap character generators. The only difference is how they are used.

The firmware will start searching for a specified bitmap font in the printer's memory. If the font is not found there, the current directory (see CHDIR statement) will be searched for a bitmap font file with the same name. If such a file is found, it will be copied and used as a font. If there is not enough memory left to hold the copy, old font file copies will be deleted until a sufficient amount of memory becomes available.

Continued!

FONT (FT), cont'd. STATEMENT

Remarks, cont'd.

Scalable Outline Fonts
Some printers models can be fitted with a special firmware package for generating bitmap fonts from scalable outline font files in *Speedo* and *TrueType* format for PC. (*TrueType* fonts in *Macintosh* format will not work).

• *Speedo* font files usually have the extension .SPD.

• *TrueType* font files usually have the extension .TTF.

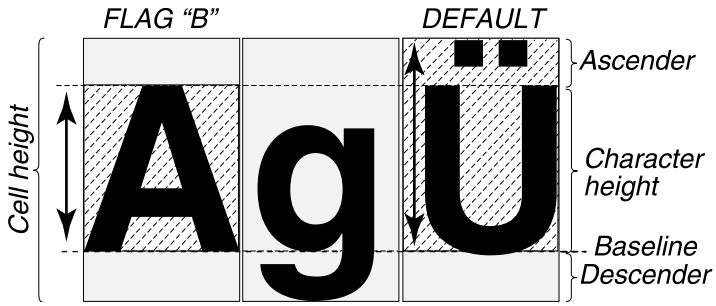
Intermec Fingerprint generates bitmap fonts faster from from *Speedo* fonts than from *Truetype* fonts.

Outline font files can be stored in the printer's ROM memory or in a non DOS-formatted memory card ("rom:"), in a DOS-formated memory card "card1:", or be downloaded to the printer's RAM memory ("ram:") using e.g. the *Kermit* communication protocol on a serial channel (see TRANSFER KERMIT statement). Below, the various types of input data to the FONT statement in connection with font scaling are explained:

Font Name:
Start by giving the new bitmap font a name (max. 10 characters incl. possible extension). We recommend that you include the extension **.1** or **.2** according to the direction of the font.

Outline Font File:
State the name, and optionally the directory (see CHDIR statement), of the outline font file you want to use.

Font Height:
By default, the height is the distance from the baseline to the top of the character cell, including ascenders. Descenders are not included. Optionally, the height can be defined in regard to the actual size of an uppercase A character (i.e. the spaces for ascenders and descenders are excluded), see "Flag B" in the illustration below. This feature provides compatibility to the size specification in *Intermec Fonts*, which is included in the *Intermec Toolbox* set of programming tools.



By default, the height is measured in dots which implies that the size of the printed characters depend on the printhead density of the printer (e.g. 6, 8, or 11.81 dots/mm). Optionally, the size can be set to an absolute value (points), see "Flags" parameter below.

Continued!

FONT (FT), cont'd. STATEMENT

Remarks, cont'd.

Character Range:
You can specify which characters you want to include in the generated bitmap font by entering either a range of characters by their decimal ASCII values according to the Roman 8 character set, or a string of characters (e.g. “ABCDEFGHIJKLMNOPQRSTUVWXYZ").

Direction:
The direction decides if bitmap font can be used for printing across the paper web (1) or along the web (2). This parameter corresponds to the extension .1 and .2 described above. If you want to use such an extension, you must manually include it in the name of the generated bitmap font.

Flags:
The optional flag characters decide how the generated bitmap font will be saved and also provide options in regard of how the character height can be specified. One, two, three or four flag characters can be used in the string expression. The order between the flag characters and upper-/lowercase is of no consequence.

A The generated bitmap font will be added to an existing bitmap font with the same name. A prerequisite is that the new and the existing bitmap font have the same cell height. It is possible to mix different typefaces, slanting and rotation as long as the cell height is the same. If the same character is included in both the existing and the new font, the existing character will be retained and the new one ignored. This option is useful for creating bitmap fonts where some characters in the middle of the ASCII range should be left out.

S The generated bitmap font will be saved in the printer's battery backed-up RAM memory. If this option is not used, the bitmap font will stored in the no-save area of the RAM memory and thus be erased at next power-up or reboot.

P The height for the generated bitmap font will get an absolute value in points (1 point = 1/72 inch ≈ 0.352 mm) instead of dots. Thus, the density of the printhead will not affect the size of the printed text.

B The height is specified as the actual size of an uppercase A to provide compatibility with Intermec Fonts (also see “Font Height” parameter above).

Slanting:
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.

Slanting value: 10° ***ABCDEFGH***

Slanting value: 20° ***ABCDEFGH***

Continued!

FONT (FT), cont'd. *STATEMENT*

Remarks, cont'd.

Rotation:
Rotation rotates each character separately clockwise, e.g.:

Rotation value: 350° **ABC**

The firmware does not support kerning.

Map Table:
By default, the outline fonts are automatically remapped according to the Roman 8 character set (see later in this manual). If this character set does not meet your requirements, you can specify a map table of your own. Only differences from the Roman 8 character set need to be included in the map table.

- The map table is made as a line-orientated file, which can be given any name (max. 30 characters).
- Lines starting with a "**#**" character are ignored (can be used for remarks).
- Lines starting with a "**$**" character are command lines. Command lines are presently only used in connection with *TrueType* fonts.
- To ensure that a specific map table in a *TrueType* font file is used, a command line starting with a "**$**" character followed by the command "**m**" can be included in the map file. The command is followed by the platform id (pid) and the specific id (sid), following the syntax: **$m<pid>,<sid>**↵
- Presently, the following map tables are used:

  Microsoft UGL Character Set with Unicode indexing scheme:
  <pid> = 3 **<sid> = 1**

  Macintosh Roman Character Set:
  <pid> = 1 **<sid> = 0**

- Each line remaps a single character, and starts with the decimal ASCII number of the character according to the Roman 8 character set, followed by a "=" (equal to) sign, and finally the internal id. number of the character according to the indexing scheme of the outline font:
  <ASCII number> = <internal font id. number>↵

More information on fonts can be found in the chapter *Fonts* later in this manual.

Continued!

FONT (FT), cont'd. *STATEMENT*

Examples

Printing a label with one line of text in print direction 1:

```
10    PRPOS 30,300
20    DIR 1
30    ALIGN 4
40    MAG 3,3
50    FONT "SW030RSN"
60    PRTXT "HELLO"
70    PRINTFEED
RUN
```

Generating the uppercase characters A–Z (incl. space) from a Speedo font called "UBI0010.SPD" residing in the ROM memory and printing the same label as above. In this case magnification can be omitted since the font can be scaled during generation. No slanting or rotation is performed. The height is specified in points and the font is saved in the printer's RAM memory:

```
10    PRPOS 30,300
20    DIR 1
30    ALIGN 4
40    FONT "SWISS.1","rom:UBI0010.SPD",30,65,90,1,"PS"
50    PRTXT "HELLO"
60    PRINTFEED
RUN
```

Suppose you want to add an exclamation mark (! = ASCII 33 dec.) in the text. Then you must add that character to the font:

```
10    PRPOS 30,300
20    DIR 1
30    ALIGN 4
40    FONT "SWISS.1","rom:UBI0010.SPD",30,33,33,1,"PSA"
50    PRTXT "HELLO!"
60    PRINTFEED
RUN
```

Another way of editing line 40 in the example above is:

```
40    FONT "SWISS.1","rom:UBI0010.SPD",30,"!",1,"PSA"
```

Example of a map table file that replaces the $ sign with a £ sign:

```
# Force use of Microsoft UGL↵
$m 3,1 ↵
36=163 ↵
```

FONT LOAD STATEMENT

Field of Application

Downloading and converting .ATF fonts to the printer's internal font format.

Syntax

FONT 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 font data.*
<sexp$_1$>	*is the desired name of the font after downloading and conversion.*
<nexp$_2$>	*is the number of bytes to download.*
<sexp$_2$>	*is one or more of the following flags:* *"S" = Save font in battery backed-up RAM area* *"1" = Create font printable in horizontal direction (DIR 1 & 3)* *"2" = Create font printable in vertical direction (DIR 2 & 4)*
<nexp$_3$>	*is, optionally, the number of an OPENed communication channel.*

Remarks

This statement requires the two EPROMs of the *Scalable Fonts Kit* to work, i.e. it can only be used in printers with six EPROM sockets, such as *EasyCoder 401/501/601*. If these EPROMs are missing, error No. 19 (*"Error in filename"*) occurs. However, the GAL included in the *Scalable Fonts Kit* is not required.

The optional first parameter is used to filter out undesired characters added to the file by the operation system during the transfer (e.g. CR/LF problems in MS-DOS).

The file name should preferably comply with the *Intermec Fingerprint* font name convention of 10 characters including the extension ".1" or ".2" to indicate printable directions.

The number of bytes to download is the actual size of .ATF font file according to the host.

The "S" flag saves the font in the battery-backed up part of the printer's RAM memory. If no "S" flag is included in the statement, the font will be deleted at next power-up or REBOOT.

The flags "1" and "2" decides the printable direction of the font. If both flags are included in the same statement, the one last specified will be decide the direction.

By default, the font file is received on the standard IN channel. You can redirect the input to any other serial channel OPENed FOR INPUT by stating its reference number according to the OPEN statement.

Continued!

FONT LOAD, cont'd. *STATEMENT*

Remarks, cont'd.

Before FONT LOAD can be used on a serial channel, the setup must be changed to 8 characters, CTS/RTS handshake. When a FONT 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 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

This example skips the two first bytes in the transmission and downloads the file MS050RMN.ATF to the printer on communication channel "uart2:". The font will be adapted for horizontal printing and saved in the printer's battery backed-up RAM memory as MS050RMN.1. The actual downloading procedure depends on the communication program of the host and is omitted from this example:

```
10    OPEN "uart2:" FOR INPUT AS #1
20    FONT LOAD 2,"MS050RMN.1",972,S1,1
30    CLOSE #1
RUN
```

FONTNAME$ *FUNCTION*

Field of Application **Returning the names of the bitmap 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

This function can be used to produce a list of all bitmap fonts including dedicated bar code fonts (another method is to use the FONTS statement, which however does not include bar code fonts).

Font files downloaded by means of a TRANSFER KERMIT statement will not be returned, since the firmware will regard them as files, not as fonts. This also applies to all scalable outline font files.

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 e.g.:

```
-SUPFNT.1
-UPC11.1
-UPC11.2
-UPC21.1
-UPC21.2
-UPC31.1
-UPC31.2
-UPC51.1
-UPC51.2
-USD5FNT12DOT.1
-USD5FNT12DOT.2
-USD5FNT6DOT.1
-USD5FNT6DOT.2
-USD5FNT8DOT.1
-USD5FNT8DOT.2
MS030RMN.1
```

etc, etc.

FONTS *STATEMENT*

Field of Application

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

Syntax

```
FONTS
```

Remarks

This statement can be used to list all fontnames, except dedicated bar code fonts. (Another method is to use a FONTNAME$ function).

Font files downloaded by means of a TRANSFER KERMIT statement will not be printed, since the firmware will regard them as files rather than fonts.

Example

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

```
FONTS
```

yields e.g.:

```
MS030RMN.1                MS030RMN.2
MS050RMN.1                MS050RMN.2
MS060BMN.1                MS060BMN.2
OB035RM1.1                OB035RM1.2
SW020BSN.1                SW020BSN.2
SW030RSN.1                SW030RSN.2
SW050RSN.1                SW050RSN.2
SW060BSN.1                SW060BSN.2
SW080BSN.1                SW080BSN.2
SW120BSN.1                SW120BSN.2

392440 bytes free   648 bytes used
Ok
```

FOR *STATEMENT*

Field of Application

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$_3$>]

<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.

FOR...NEXT loops can be nested, i.e. 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.

Example

The counter A% is incremented from 20 to 100 in steps of 20 by means of a FOR...NEXT loop:

```
10    FOR A%=20 TO 100 STEP 20
20    PRINT A%
30    NEXT
RUN
```

yields

```
20
40
60
80
100
```

FORMAT STATEMENT

Field of Application

Formatting the printer's RAM memory, or formatting a RAM-type memory card to MS-DOS format.

Syntax

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

<sexp> *specifies the device to be formatted either as "ram:" or "card1:"*

<nexp$_1$> *Specifies the number of entries in the root directory (only applicable when <sexp> = "card1:"). Default: 208 entries.*

<nexp$_2$> *Specifies the number of bytes per sector (only applicable when <sexp> = "card1:"). Default: 512 bps.*

Remarks

FORMAT "ram:"

Formats the RAM memory of the printer, i.e. all **files** in the device "ram:" will be erased. In this context, “files” refers to such files that can be listed by a FILES statement. No other data in RAM, such as image, fonts, setup parameters, date- and time formats, counters, variables etc., will be affected. Be careful. There is no way to undo a FORMAT operation.

FORMAT "card1:"

Formats a JEIDA-4 memory card of RAM-type, which is inserted in the printer's optional memory card adapter, to MS-DOS format. Optionally, the number of entries in the root directory (i.e. number of files on the card) and the number of bytes per sector can be specified.

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:".

Continued!

FORMAT, cont'd. *STATEMENT*

Example

Issuing the statement FILES before and after a FORMAT "ram:" statement shows how the RAM memory is affected:

```
FILES
Files on ram:

LISTFONT.PRG  132     LABEL4.PRG   345
LABEL1.PRG    204     LABEL5.PRG   421
LABEL2.PRG     96     LABEL6.PRG   86
LABEL3.PRG    138     LABEL7.PRG   120

90672 bytes free      2828 bytes used
Ok

FORMAT "ram:"
Ok

FILES
Files on ram:

92848 bytes free       652 bytes used
Ok
```

In the following statement, an SRAM-type 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.

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

FORMAT DATE$ *STATEMENT*

Field of Application

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.

Example (the year is 1998):

Y	generates	8
YY	generates	98
YYY	generates	998
YYYY	generates	1998
YYYYY	generates	_1998

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

The date format is not saved in the printer's memory, but 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 *STATEMENT*

Field of Application

Specifying separators for the LAYOUT RUN statement used in the *Intermec Direct Protocol*.

Syntax

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

<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)*

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.

Always execute the FORMAT INPUT statement in the Immediate Mode. If you are using the *Intermec Direct Protocol*, exit it by means of an INPUT OFF statement before changing the separators using a FORMAT INPUT statement. Then you can enter the *Intermec Direct Protocol* again by means of 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 no-save area of the printer's RAM 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$ *STATEMENT*

Field of Application

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. 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	_08	**p**	generates	a

In order 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 not saved in the printer's memory, but 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 German standard with seconds omitted:

```
FORMAT TIME$ "HH:MM Uhr"
```

Changing the date format to British standard:

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

FORMFEED (FF) *STATEMENT*

Field of Application

Activating the paper feed mechanism in order to feed out or pull back a certain length of the paper web.

Syntax

FORMFEED|FF[<nexp>]

<nexp> *is, optionally, the paper 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 strip according to the printer's setup. See start- and stop-adjustments and media type (i.e. label w gap, ticket w mark, ticket w gap, fix length strip or var length strip) in the Technical Manual.

If a value (positive or negative) is entered after the FORMFEED statementthe paper 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 the web. (The plus sign is optional. All numbers not preceded by a minus sign are presumed to be positive).
- A **negative** number of dots makes the printer **pull back** the specified length of the web. In this case, be careful not to enter a value larger than the length of the label to avoid the risk of causing a paper jam.

It is of importance whether a FORMFEED statement is issued **before** or **after** a PRINTFEED statement:
- FORMFEED statement issued **before** PRINTFEED affects the position of the origin in relation to the paper web on the first copy to be printed, i.e. it has the same effect as the *start adjustment* in the setup.
- FORMFEED statement issued **after** PRINTFEED does not affect the position of the origin on the first copy, but next copy will be affected, i.e. it has the same effect as the *stop adjustment* in the setup.

In many cases, you may want to adjust the paper feed length. There is normally a difference in feed length when self-adhesive labels are to be dispensed as opposed to the web being torn off between labels. You may also want to feed back a certain length of the web before printing in order to make use of the first part of a label. Note that the CUT statement allows you to feed out and then pull back a specified amount of paper without having to use startadjust, stopadjust or FORMFEED.

This can be done in two ways, possibly in combination:
- During the setup procedure you may specify a certain adjustment of the feed length, which will work all the time, regardless of which program you run.
- You may specify the adjustment within each program or subroutine using a FORMFEED statement.

Refer to the Technical Manual for the printer model in question for a list of suitable adjustments for dispensing, tearing off etc.

Also see page 9 for remaining bugs and limitations.

Continued!

FORMFEED (FF), cont'd. *STATEMENT*

Examples

Printing a line of text and feeding out an extra length (60 dots) of the web after printing:

```
10    FONT "SW030RSN"
20    PRPOS 30,200
30    PRTXT "HELLO"
40    PRINTFEED
50    FORMFEED 60
RUN
```

Pulling back the paper 20 dots of the web before printing:

```
10    FORMFEED -20
20    FONT "SW030RSN"
30    PRPOS 30,200
40    PRTXT "HELLO"
50    PRINTFEED
RUN
```

Note that in this case, the positioning of the text line will be performed after the paper has been pulled back.

FRE *FUNCTION*

Field of Application **Returning the number of free bytes in the printer's RAM memory.**

Syntax

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

<nexp>|<sexp> *is a dummy argument.*

Remarks

The expression <sexp> or <nexp> is a dummy argument. It does not matter what you enter, but you must enter something within the parentheses.

Example

```
PRINT FRE(1)
```

yields e.g.:

```
80836
```

FUNCTEST STATEMENT

Field of Application **Performing various hardware tests.**

Syntax

FUNCTEST<sexp>,<svar>

<sexp> *is the type of test to be performed:*
- *RAM* *Test of internal RAM.*
- *ROMn* *Test of the EPROM package in the socket specified by n (n=1–4 , or n=1–6).*
- *CARD* *Test of memory card inserted in the optional memory card adapter.*
- *HEAD* *Test the integrity of head shift hardware.*

<svar> *is the variable in which the result will be placed.*

Remarks

Each of the FUNCTEST hardware tests has a number of possible responses:

<sexp> = "RAM"

The complete RAM memory is tested, i.e. all RAM packages in sockets IC-5 to IC-8 on the CPU-board. The response may be:

RAM OK	Test was successful
FAIL,x	An error was detected. (**x** is the hexadecimal address of the first faulty memory byte).

<sexp> = "ROMn"

The EPROM package fitted in the socket on the CPU-board specified by **n** is tested. (n=1: IC-1, n=2: IC-2 etc.). The response may be:

XXXX	**xxxx** is a 4-digit hexadecimal checksum.
NO ROM	An illegal ROM socket was specified.

<sexp>="CARD"

If a RAM-type card (not write-protected) is fitted, a RAM test is performed (see above).

If a ROM-type card (write-protected) is fitted, a ROM test is performed, see above).

If no card is fitted, `NO CARD` is returned.

<sexp> = "HEAD"

The printhead is checked for the number of dots and possible faults. There may be three different responses:

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 24V voltage for the printhead is not checked. Use the HEAD function for additional printhead tests.*

Continued!

FUNCTEST, cont'd. *STATEMENT*

Example

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

```
10   FUNCTEST "RAM", A$
20   FUNCTEST "ROM1", B$
30   FUNCTEST "ROM2", C$
40   FUNCTEST "ROM3", D$
50   FUNCTEST "ROM4", E$
60   FUNCTEST "HEAD", F$
70   PRINT "RAMTEST:", A$
80   PRINT "IC-1:", B$
90   PRINT "IC-2:", C$
100  PRINT "IC-3:", D$
110  PRINT "IC-4:", E$
120  PRINT "HEADTEST:", F$
RUN
```

yields e.g.:

```
RAMTEST:  RAM OK
IC-1: 92C9
IC-2: C9C0
IC-3: 8000
IC-4: C000
HEADTEST: HEAD OK,SIZE:640 DOTS

Ok
```

FUNCTEST$ *FUNCTION*

Field of Application **Returning the result of various hardware tests.**

Syntax

FUNCTEST$(<sexp>)

<sexp> *is the type of test to be performed:*

RAM	*Test of internal RAM.*
ROMn	*Test of the EPROM package in the socket specified by n (n=1–4 , or n=1–6).*
CARD	*Test of memory card inserted in the optional memory card adapter.*
HEAD	*Test the integrity of head shift hardware.*

Remarks

The hardware tests correspond to those in the FUNCTEST statement and have a number of possible responses:

<sexp> = "RAM"
The complete RAM memory is tested, i.e. all RAM packages in sockets IC-5 to IC-8 on the CPU-board. The response may be:

RAM OK	Test was successful
FAIL,x	An error was detected. (**x** is the hexadecimal address of the first faulty memory byte).

<sexp> = "ROMn"
The EPROM package fitted in the socket on the CPU-board specified by **n** is tested. (n=1: IC-1, n=2: IC-2 etc.). The response may be:

XXXX	**xxxx** is a 4-digit hexadecimal checksum.
NO ROM	An illegal ROM socket was specified.

<sexp>="CARD"
If a RAM-type card (not write-protected) is fitted, a RAM test is performed (see above).
If a ROM-type card (write-protected) is fitted, a ROM test is performed, see above).
If no card is fitted, `NO CARD` is returned.

<sexp> = "HEAD"
The printhead is checked for the number of dots and possible faults. There may be three different responses:

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 24V voltage for the printhead is not checked. Use the HEAD function for additional printhead tests.*

Continued!

FUNCTEST$, cont'd. *FUNCTION*

Example

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

```
10   PRINT "RAMTEST:", FUNCTEST$ ("RAM")
20   PRINT "IC-1:", FUNCTEST$ ("ROM1")
30   PRINT "IC-2:", FUNCTEST$ ("ROM2")
40   PRINT "IC-3:", FUNCTEST$ ("ROM3")
50   PRINT "IC-4:", FUNCTEST$ ("ROM4")
60   PRINT "HEADTEST:", FUNCTEST$ ("HEAD")
RUN
```

yields e.g.:

```
RAMTEST:  RAM OK
IC-1:     9C8D
IC-2:     08F6
IC-3:     1D09
IC-4:     B2B3
HEADTEST: HEAD OK,SIZE:832 DOTS

Ok
```

GET STATEMENT

Field of Application

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 ≠ 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
```

GOSUB STATEMENT

Field of Application **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, i.e. a subroutine may contain a GOSUB statement for branching to a secondary subroutine and so on until the printer runs out of memory.

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
```

Continued!

GOSUB, cont'd. *STATEMENT*

Examples, cont'd. *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
IMMEDIATE ON
```

GOTO STATEMENT

Field of Application **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.

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 FUNCTION

Field of Application **Returning the result of a thermal printhead check.**

Syntax

HEAD(<nexp>)

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

Remarks

This function allows you to examine the printhead in regard of dot resistance. A prerequisite is that the printer is fitted with a CPU-board which supports dot sensing, i.e. presently the *EasyCoder 201 II* and *EasyCoder 401/501/601* printer families.

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. Obviously, the definition of what resistance values will indicate a defect dot must be set according to the characteristics of the brand of printhead in question.

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 themselves the right of physical examination of the printhead before any replacement free of charge can be discussed.

<nexp> ≥ 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 dots may be faulty. You can use the statement SET FAULTY DOT and BARADJUST to avoid printing bar codes in a position, where a faulty dot may impair the readability of the bar code. The dot numbering starts at 0 (zero). This implies that in e.g. a 640 dots printhead, the dots are numbered 0 – 639.

<nexp> = -1

A check of the complete printhead is performed.

HEAD(-1) = -1 The printhead is within the allowed limits, i.e. 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.

Continued!

HEAD, cont'd. *FUNCTION*

Remarks, cont'd.

<nexp> = -7
The mean resistance value in ohms of all dots of the printhead is returned. This is the resistance value for which the printer should be set up. It can be used in combination with a SETUP statement to set up the printhead resistance automatically (standard feature in some models). After replacing a printhead, no printing should be performed before the resistance value has been reset, manually or automatically.

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)
```

IF...GOTO...(ELSE) *STATEMENT*

Field of Application **Conditional branching controlled by the result of a numeric expression.**

Syntax

IF<nexp>[,][THEN]GOTO<ncon>|<line label>[ELSE<stmt>]

<nexp> *is a numeric expression, which is either true or false.*
<ncon>/<line label> *is the number or label of the line to which the program should branch, when the IF-condition is true.*
<stmt> *is an optional statement or list of statements which tells the program what to do, should the IF-condition be false.*

Remarks

If THEN is omitted when the statement is entered, it will be assumed and added when the program is listed.

ELSE statements may be nested.

Examples

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% GOTO 50 ELSE PRINT "NOT EQUAL"
40    END
50    PRINT "EQUAL":END
RUN
```

yields:

```
NOT EQUAL
```

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

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

yields:

```
NOT EQUAL
```

IF...THEN...(ELSE) *STATEMENT*

Field of Application **Conditional execution controlled by the result of a numeric expression.**

Syntax

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

IF<nexp>[,]THEN ↵
<stmt$_1$> ↵
[...<stmt$_{1+n}$>] ↵
[ELSE ↵
<stmt$_2$> ↵
[...<stmt$_{2+n}$>]] ↵
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 telling the program what to do, 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.

Example

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 e.g.:

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

IMAGE LOAD *STATEMENT*

Field of Application

Reception and conversion of image files in .PCX format to images in the *Intermec Fingerprint* internal bitmap format.

Syntax

IMAGE LOAD[<nexp$_1$>,]<sexp$_1$>,<nexp$_2$>,<sexp$_3$>[,<nexp$_3$>]

<nexp$_1$>	*is optionally the number of bytes to skip before starting to read the image data.*
<sexp$_1$>	*is the desired name of the image to be created, optionally including the extension .1 or .2.*
<nexp$_2$>	*is the size of the .PCX file in number of bytes.*
<sexp$_2$>	*is an optional flag:* *"S "specifies that the image will be saved in RAM* *An empty string ("") specifies that the image will be stored in the no-save area of the RAM memory and thus be deleted at next power up.*
<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 file on the standard IN channel (see SETSTDIO statement) or on another communication channel OPENed for INPUT. When the file is received, it will automatically be converted to an image in the internal bitmap format of *Intermec Fingerprint* (compare with the PCX2BMP external command).

The optional first parameter makes it possible to use this statement in MS-DOS (CR/LF problem), while retaining the compatibility with *Intermec Fingerprint 6.0*.

The name of the image to be created by consist of max. 30 characters including possible extension. The image will have the same direction as the original .PCX file and can only be rotated 180° by means of a DIR statement. We therefore recommend that you include an extension to indicate for which print directions the image is intended, according to the *Intermec Fingerprint* convention:

.1 indicates that the image is intended for print directions across the paper feed direction, i.e. DIR 1 & 3.

.2 indicates that the image is intended for print directions along the paper feed direction, i.e. DIR 2 & 4.

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

Continued!

IMAGE LOAD, cont'd. *STATEMENT*

Remarks, cont'd.

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 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

```
IMAGE LOAD "Logotype.1",400,"S"
```

IMAGENAME$ *FUNCTION*

Field of Application **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 firmware 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 e.g.

```
CHESS2X2.1
CHESS4X4.1
DIAMONDS.1
GLOBE.1
```

IMAGES *STATEMENT*

Field of Application

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 e.g.:

```
CHESS2X2.1          CHESS4X4.1
DIAMONDS.1          GLOBE.1

2008664 bytes free    288 bytes used
```

IMMEDIATE ON/OFF *STATEMENT*

Field of Application

Enabling or disabling the immediate mode of *Intermec Fingerprint* in connection with program editing without line numbers.

Syntax

IMMEDIATE ON|OFF

Remarks

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 between lines are done by means of "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. When a line is labelled, the line must start with the line label. When a line label is used as a reference to another line, e.g. within a GOTO statement, the colon should be omitted.

Before the program is RUN, it 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, i.e. 10-20-30-40-50.... The line numbers will not appear on the screen before 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.

Example

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
```

yields:

```
Ok

RUN
```

yields:

```
LINE 1
LINE 2
Ok
```

INKEY$ FUNCTION

Field of Application

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, e.g. "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 std. IN channel will be printed on the screen of your terminal 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 your terminal's keyboard. They will not be printed on the 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) *STATEMENT*

Field of Application

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 300 characters.

Note that INPUT filters out any incoming ASCII 00 dec. characters (NUL).

Continued!

INPUT (IP), cont'd. *STATEMENT*

Example

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, e.g.

```
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 double quotation marks (".."), e.g.:

```
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 — STATEMENT

Field of Application

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*. Also refer to *Intermec Direct Protocol* Programmer's Guide.

INPUT ON Enables the *Intermec Direct Protocol*, i.e.:
- Enables reception of input data to a stored layout
- Starts the error handler
- Sets the verbosity to off (SYSVAR (18) = 0)
- Shows *"Direct Protocol 6.13"* in the display

INPUT OFF Disables the *Intermec Direct Protocol*, i.e.:
- 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 6.13"* in the display

The following instructions will only work with the *Intermec Direct Protocol*, i.e. after a INPUT ON statement has been executed:

COUNT&	ERROR	FORMAT INPUT	INPUT OFF
LAYOUT END	LAYOUT INPUT	LAYOUT RUN	PRINTKEY ON|OFF

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 "LABEL1"↵
FT "SW030RSN" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$↵
LAYOUT END ↵
LAYOUT RUN "LABEL1"↵
#Line number 1&Line number 2&@↵
PF ↵
INPUT OFF ↵
```

INPUT# STATEMENT

Field of Application **Reading a string of data from an OPENed 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 300 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$ *FUNCTION*

Field of Application

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.*
<ncon$_2$> *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 *FUNCTION*

Field of Application

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) *STATEMENT*

Field of Application

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 and SETUP files*

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 paper therefore will retain its original colour, whereas "black" means the colour 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.

The INVIMAGE will be revoked by a NORIMAGE statement.

Example

```
10   PRPOS 30,300
20   DIR 1
30   ALIGN 4
40   INVIMAGE
50   FONT "SW030RSN"
60   PRTXT "Inverse printing"
70   PRINTFEED
RUN
```

KEY BEEP STATEMENT

Field of Application

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↔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 sec. each.*

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 and/or the duration to 0 (zero).

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).

Note	Hz	Note	Hz	Note	Hz	Note	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 STATEMENT

Field of Application

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

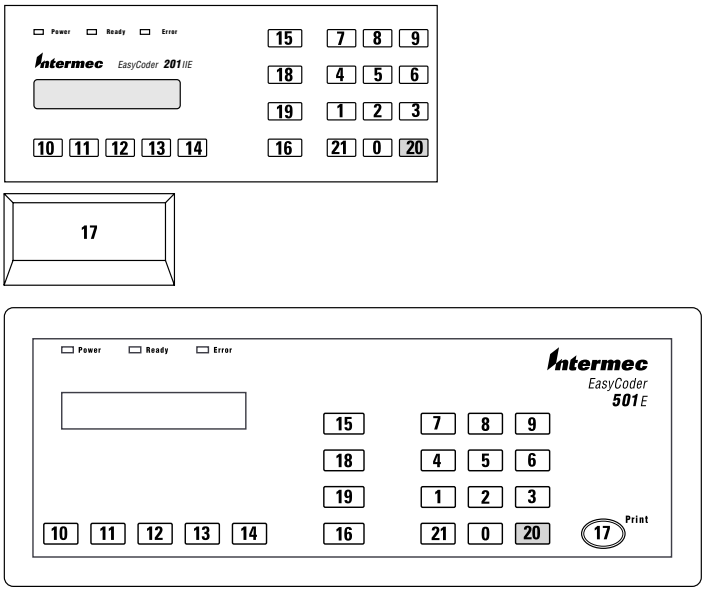
<nexp> *is the i.d. number of one of the keys on the printer's front panel (see illustration below).*

OFF/ON *disables/enables the specified key.*

Remarks

All *Intermec Fingerprint* printer models are fitted with a "Print" key or button. In addition, some models are fitted with a membrane keyboard. Using an ON KEY... GOSUB statement, any key can be assigned, alone or in combination with **C**-key, to make the program branch to a subroutine. The keys are enabled/disabled individually using their respective i.d. numbers as shown below.

Please note the difference between the i.d. numbers of the keys and the ASCII values they are able to produce (see e.g. BREAK).



Default I.d. numbers of the most common keyboard types in the *EasyCoder* printer line.
(Some printers only have a Print key or button)

The **C** or Clear key (i.d. No. 20) works as a Shift key. When pressed in connection with another key, it adds 100 to the i.d number of the other key.

Example

In this example, the F1 key (i.d. 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$ *VARIABLE*

Field of Application **Returning or setting the keyboard map table.**

Syntax

To 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
2 = Position No. of Shift key.

To set the map table:	**KEYBMAP$(<nexp>) = <sexp>**

<nexp> *is the type of string to be returned:*
0 = Unshifted 64 characters
1 = Shifted 64 characters
2 = Position No. of Shift key.
<sexp> *is the string specifying the keyboard mapping in the selected type of string.*

Remarks

The keyboard can be remapped in order to better suit special applications or national standards.

The string that contains the desired keyboard map should contain the desired character for each of 64 key positions (in ascending order) regardless if the keyboard contains that many keys.

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.

Please refer to the chapter *"Printer Function Control; Keyboard"* in *Intermec Fingerprint* Programmer's Guide for information on keyboard position numbers and a table of the ASCII values for special characters.

Non-existing key positions are mapped as Null, i.e. CHR$(0).

The key appointed as < **Shift** > key is specified by its keyboard position number in a separate string.

The single < **Print** > key of *EasyCoder 401/501* cannot be remapped.

The current keyboard mapping can also be read back to the host.

Continued!

KEYBMAP$, cont'd. *VARIABLE*

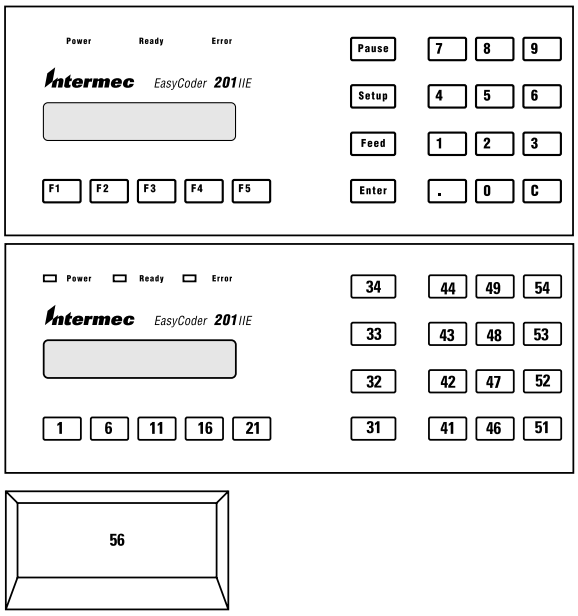
Examples

In this example, the unshifted keyboard map is read back to the host. The string is modified (F1 is replaced by Feed) and used to change the keyboard map.

```
10    A$ = KEYBMAP$(0)
20    B$ = CHR$(28) + MID$(A$,2)
30    KEYBMAP(0)=B$
40    END
```

The following example illustrates the mapping of the keyboard for Easy-Coder 201 IIE (unshifted keys only). Note the limit of max. 300 characters per program line

```
10    A$=CHR$(1)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(2)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(3)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(4)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(5)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(13)
20    A$=A$+CHR$(28)+CHR$(29)+CHR$(30)+CHR$(0)
      +CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
      +"."+"1"+"4"+"7"+CHR$(0)+"0"+"2"+"5"+"8"
      +CHR$(0)+CHR$(8)+"3"+"6"+"9"+CHR$(0)
      +CHR$(31)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
      +CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
30    KEYBMAP$(0)= A$
40    END
```



Key designations and keyboard position numbers of an EasyCoder 201 II E printer.

KILL *STATEMENT*

Field of Application

Deleting a file from the printer's RAM memory or from a DOS-fomatted memory card inserted in an optional memory card adapter.

Syntax

KILL<sexp>

<sexp> *is the name , **including extension**, of the file, which is to be deleted .*

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), a reference to the directory in question must be included in the file name, e.g. "card1:<filename>.XYZ".

KILL can not be used for files residing in "rom:".

Examples

```
KILL "LABEL14.PRG"
KILL "ram:LABEL14.PRG
KILL "card1:LABEL7.PRG"
```

Startup files and outline font files have other extensions:

```
KILL "AUTOEXEC.BAT"      (startup file)
KILL "NNNNNNNN.SPD"      (Speedo font file)
KILL "NNNNNNNN.TTF"      (TrueType font file)
```

LAYOUT STATEMENT

Field of Application

Handling of layout files.

Syntax

LAYOUT[F,] <sexp$_1$>,<sexp$_2$>,<svar>|<sexp$_3$>,<nvar>|<sexp$_4$>

F, *optionally allows handling of data and error files instead of data and error arrays (see <sexp$_3$> and <sexp$_4$> below).*
<sexp$_1$> *is the layout file.*
<sexp$_2$> *is the logotype name file.*
<svar>|<sexp$_3$> *is the data array (svar>) or file (<sexp$_3$).*
<nvar>/<sexp$_4$> *is the error array (<nvar>) or file (<sexp$_4$).*

Remarks

The layout file should have the following syntax:

<sexp$_1$>: Layout file format *(sorted in ascending order)*
Records 1–n, 52 bytes each

Byte #	*Parameter*	*Layout Type*	*Input*	*Notes*
0–1	Element number		HH	
2	Layout type		C	See table
3	Direction	A,B,C,L,S,or X	D	
	Barfont on/off (0=off; 1=on)	H	D	
	Security	E	D	
4	Alignment	A,B,C,L,S,X	D	
	Aspect height ratio	E	D	
5–8	X-position	A,B,C,L,S,or X	DDDD	
	Aspect width ratio	E	D	
	Baradjust left	J	DDDD	
9-12	Y-position	A,B,C,L,S,or X	DDDD	
	Rows in bar code	E	DD	
	Baradjust right	J	DDDD	
13–22	Font name	C	C_1–C_{10}	
	Logotype name	A	C_1–C_{10}	
	Bar code name	B	C_1–C_{10}	
	Barfont name	H	C_1–C_{10}	
	Line length	S	DDDD	
	Box width	X	DDDD	
	Columns in bar code	E	DD	Byte 13-14
	Truncate according to code spec:s	E	D	Byte 15
23–42	Fixed text or alphanumeric data	B or C	C_1–C_{20}	
	Fixed numeric data	B	D_1–D_{20}	
	Logotype number	L	DD	
	Box height	X	DDDD	
	Line thickness	S	DDDD	
43–44	No of char. to print (of byte 23-42)	B or C	DD	
45–46	Image type (I = inverse image)	A,C, or L	C	
	Bar code ratio (wide/narrow bars)	B	DD	
47	Vertical magnification	A,C, or L	D	
	Bar code magnification	B	D	
48	Horizontal magnification	A,C, or L	D	
49–51	Bar code height	B	DDD	
	Line thickness	X	DDD	

Continued!

Abbreviations:
H = hex digit.
D = numeric digit.
C = alpha character.

Layout types:	
Logotype by name	A
Bar code	B
Text	C
Bar code extended field[1]	E
Barfont on/off	H
Baradjust[2]	J
Logotype by number	L
Line	S
Box	X

[1]/. Corresponds to the 6 last parameters in the BARSET statement. Must have a lower element number than the corresponding bar code layout record (B), where the bar code field is otherwise defined.
[2]/. Corresponds to the BARADJUST statement.

LAYOUT, cont'd. *STATEMENT*

Remarks, cont'd.

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

Also refer to the chapter *"Label Design; Layout Files"* in *Intermec Fingerprint* Programmer's Guide.

Do not confuse this statement with the statements LAYOUT INPUT, LAYOUT END and LAYOUT RUN, which can only be used in the Direct Mode.

Continued!

LAYOUT, cont'd. *STATEMENT*

Examples

```
10   DIM QERR%(10)
20   LAYDATA$(0)="01DAY"
30   LAYDATA$(1)="04123456789012"
40   QERR%(0)=0
50   OPEN "LOGNAME.DAT" FOR OUTPUT AS 19
60   PRINT# 19,"DIAMONDS.1";
70   CLOSE 19
80   OPEN "LAYOUT.DAT" FOR OUTPUT AS 6
90   PRINT# 6,"01C11100 10  SW030RSN.1                    00I 11   ";
100  PRINT# 6,"01C11100 40  SW030RSN.1                    00  22   ";
110  PRINT# 6,"01C11100 100 SW030RSN.1WEDNES              06I 11   ";
120  PRINT# 6,"01C11100 130 SW030RSN.1SATURNUS            05I 11   ";
130  PRINT# 6,"02L11300  70           1                       33   ";
140  PRINT# 6,"03S11100 210 300       3                           ";
150  PRINT# 6,"04B14100 300 EAN13                        0 312 100";
160  CLOSE 6
170  LAYOUT "LAYOUT.DAT","LOGNAME.DAT",LAYDATA$,QERR%
180  IF QERR% (1) = 0 THEN GOTO 250
190  PRINT "-ERROR- LAYOUT 1"
200  I%=0
210  IF QERR%(I%)=0 THEN GOTO 250
220  PRINT "    ERROR  ";QERR%(I%+1);" in record ";QERR%(I%)
230  I%=I%+2
240  GOTO 210
250  PRINTFEED
```

LAYOUT END *STATEMENT*

Field of Application

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.

The layout will be saved in the printer's RAM memory ("ram:") and 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 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 "LABEL1"↵
FT "SW030RSN" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$↵
LAYOUT END ↵
LAYOUT RUN "LABEL1"↵
#Line number 1&Line number 2&@↵
PF ↵
INPUT OFF ↵
```

LAYOUT INPUT *STATEMENT*

Field of Application

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)*

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 std IN channel **after** a LAYOUT INPUT statement and **before** a LAYOUT END statement, will be included in the layout.

Variable input data to text, bar code and image fields (PRTXT, PRBAR, PRIMAGE) can be provided separately by means of 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

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 "LABEL1" ↵
FT "SW030RSN" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$↵
LAYOUT END ↵
LAYOUT RUN "LABEL1" ↵
#Line number 1&Line number 2&@↵
PF ↵
INPUT OFF ↵
```

LAYOUT RUN *STATEMENT*

Field of Application

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 the printer's memory (see LAYOUT INPUT and LAYOUT END statements) and provide input to string variables in the layout. Such variables can be included in PRTXT, PRBAR and PRIMAGE statements in the layout and 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 FORMAR INPUT statement).

<STX><input to VAR1$><CR><input to VAR2$><CR>.....<input to VARn$><CR><EOT>

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 "LABEL1" ↵
FT "SW030RSN" ↵
PP 100,250 ↵
PT VAR1$ ↵
PP 100,200 ↵
PT VAR2$↵
LAYOUT END ↵
LAYOUT RUN "LABEL1" ↵
#Line number 1&Line number 2&@↵
PF ↵
INPUT OFF ↵
```

LBLCOND STATEMENT

Field of Application

Overriding the paper feed setup.

Syntax

LBLCOND<nexp$_1$>,<nexp$_2$>

<nexp$_1$> *specifies the type of action:*
0 = Overriding the stop adjust.
1 = Overriding the start adjust.
2 =Turning off the LSS or Black Mark Sensor.

<nexp$_2$> *specifies <nexp$_1$> as a number of dots.*

Remarks

This instruction allows you to override the printer's feed-adjust setup or to temporarily turn off the label stop or black mark sensor:

<nexp$_1$> = 0 sets the **stop adjust** to the value specified by <nexp$_2$>.

<nexp$_1$> = 1 sets the **start adjust** to the value specified by <nexp$_2$>.

<nexp$_1$> = 2 makes the **label stop sensor** (LSS) or **black mark sensor** ignore any gaps or marks detected within the length of paper feed specified by <nexp$_2$>. 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 paper that would interfere with the detection of the black mark.

Verifying a start adjust or stop adjust value in the Setup Mode by pressing key No. 16, or setting the value by means of a setup file, 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, i.e. 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) of paper feed:

```
10   LBLCOND 1,5: LBLCOND 2,160
20   FONT "SW030RSN.1"
30   PRTXT "Hello"
40   PRINTFEED
```

LED ON/OFF — STATEMENT

Field of Application

Turning a specified LED control lamp on or off.

Syntax

LED<nexp>ON | OFF

<nexp> *is the LED which is to be turned on or off.*
0 is the "Ready" LED.
1 is the "Error" LED.

Remarks

All present *Intermec Fingerprint* printers are equipped with three LED (Light Emitting Diode) control lamps on the front panel. Two of the LED's can be used to indicate e.g. when an error occurs or when the printer is ready. It is up to the programmer to decide how they will be used, but, since the LED's are marked with text, some restriction is recommended.

The “Power” LED is connected to the mains switch and cannot be program-controlled.

Example

In this example, the “error” LED will be lighted if you e.g. attempt to run the program with a lifted printhead. Lower the printhead and a label will be fed out. The “error” LED goes out and the “ready” LED comes on.

```
10    LED 0 ON
20    LED 1 OFF
30    ON ERROR GOTO 1000
40    PRPOS 30,300
50    FONT "SW030RSN"
60    MAG 3,3
70    PRTXT "OK!"
80    PRINTFEED
90    LED 0 ON
100   LED 1 OFF
110   END
.....
.....
.....
1000  LED 0 OFF
1010  LED 1 ON
1020  RESUME
```

LEFT$ *FUNCTION*

Field of Application

Returning a specified number of characters from a given string starting from the extreme left side of the string, i.e. 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, i.e. 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 *FUNCTION*

Field of Application **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, e.g. space characters, are included in the value returned by the LEN function:

```
PRINT LEN("INTERMEC THERMAL PRINTERS")
                                                         yields:
25
```

LET STATEMENT

Field of Application **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 optional. The equal sign (=) is sufficient to make the assignment. The expression and the variable most be of the same type, i.e. both must 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 *STATEMENT*

Field of Application

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. 300 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 screen of the terminal or host computer 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 business 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 "SW030RSN"
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# *STATEMENT*

Field of Application

Assigning an entire line, including punctuation marks, from a sequential file or a device to a single string variable.

Syntax

LINE↔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. 300 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 STATEMENT

Field of Application

Listing the current program completely or partially, or listing all variables, to the standard OUT channel.

Syntax

LIST[[<ncon$_1$>[–<ncont$_2$>]]|,V]

<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.*

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 unneccessary characters and adds assumed keywords. The instruction is usually given in the immediate mode, i.e. on a line without any preceding line number.

The LIST statement can be used in six 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 number10, i.e. 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.

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.*

LOAD STATEMENT

Field of Application

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, e.g. EPROM ("rom:") or an optional DOS-formatted memory card ("card1:"). Start the file name by specifying the directory, e.g.:

```
LOAD "rom:MKAUTO"
```

or

```
LOAD "card1:PROGRAM1.PRG"
```

This will create a copy, which you can list or change and then save under a new name.

LOC FUNCTION

Field of Application

Returning the current position in an OPENed file or the status of the buffers in an OPENed 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 receiver or transmitter buffer of the specified communication channel:

- If the channel is OPENed for INPUT, the remaining number of characters (bytes) to be read from the receiver buffer is returned.
- If the channel is OPENed for OUTPUT, the remaining free space (bytes) in the transmitter buffer is returned.

The number of bytes includes characters that will be MAPped as NULL.

Also see page 9 for remaining bugs and limitations.

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 receiver buffer of "uart2:":

```
100  OPEN "uart2:" FOR INPUT AS #2
110  A%=LOC(2)
120  PRINT A%
```

LOF *FUNCTION*

Field of Application

Returning the length in bytes of an OPENed sequential or random file or returning the status of the buffers in an OPENed 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 receiver or transmitter buffer of the specified communication channel:

- If a channel is OPENed for INPUT, the remaining free space (bytes) in the receiver buffer is returned.
- If a channel is OPENed for OUTPUT, the remaining number of characters to be transmitted from the transmitter 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 receiver buffer of communication channel "uart2:" is calculated:

```
100  OPEN "uart2:" FOR INPUT AS #2
110  A%=LOF(2)
120  PRINT A%
```

LSET STATEMENT

Field of Application **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 — STATEMENT

Field of Application **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 fed out so it still sticks a little to the backing paper without falling off).

Using the LTS ON statement, you can order the printer to stop the execution at next PRINTFEED statement until the LTS no longer can detect any label. Then the PRINTFEED is executed. This is must useful when printing batches of labels or tickets. As soon as a label is taken, 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 "SW030RSN"
50    MAG 4,4
60    PRPOS 200,200
70    PRTXT B$
80    PRINTFEED
90    NEXT
RUN
```

MAG STATEMENT

Field of Application

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** (1–4).*
<nexp$_2$> *is the magnification in regard of **width** (1–4).*
Default value: *1,1*
Reset to default by: *PRINTFEED execution, SETUP files.*

Remarks

Magnification makes the object grow in directions away from the selected anchor point, see ALIGN statement.

Note that the MAG statement cannot be used for bar code patterns (use BARHEIGHT and BARMAG statement for that purpose).

Example

This program will print a 2-line label, where the first line is printed in a double-sized version of the font used for the second line. Note that the change back to the default magnification in line 70 is needed only because the magnification was changed from the default value in line 40.

```
10    PRPOS 160,150
20    ALIGN 5
30    FONT "SW030RSN"
40    MAG 2,2
50    PRTXT "Intermec"
60    PRPOS 160,120
70    MAG 1,1
80    PRTXT "Printers"
90    PRINTFEED
RUN
```

MAP STATEMENT

Field of Application

Changing the ASCII value of a character when received on the std IN channel, or optionally on another specified communication channel.

Syntax

MAP[<nexp$_1$>,]<nexp$_2$>,<nexp$_3$>

<nexp$_1$>	*optionally specifies a communicationchannel :* *0 = "console:"* *1 = "uart1:"* *2 = "uart2:"/"rs485:"* *3 = "uart3:"* *4 = "centronics:"* *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 statement) or to filter out undesired character . If you e.g. 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., i.e. 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”, since that character has not been remapped.

To reset the mapping, map the character back to its original ASCII value, e.g.:

```
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, e.g. by mapping them as NULL (ASCII 0 decimal).

After processing, the selected character set (see NASC statement) 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 statements will be processed last.

Do not map any characters to ASCII values occupied by characters used in *Intermec Fingerprint* instructions, e.g. keywords, operators, %, $, # and certain punctuation marks. Mapping will be reset to normal at power-up or reboot.

Continued!

MAP, cont'd. STATEMENT

Examples

You can check what characters the host produces by means of a simple program. Pressing different keys should produce the corresponding characters both on the label and on the screen of the host. Else, 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 computer or terminal 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 "SW030RSN"
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 re-map the character Q to be printed as Z, as in the explanation on the previous page. After printing, we map the character Q back as before.

```
5     MAP 81,90
10    FONT "SW030RSN"
20    PRPOS 30,100
30    INPUT "Enter character";A$
40    PRTXT A$
50    PRINTFEED
60    MAP 81,81
```

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 NULL (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 *STATEMENT*

Field of Application

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 incl. 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.

Important:
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, i.e. by 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.

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.

Examples

The program "ERRHAND.PRG" will be merged with the current program. If there are identical line numbers in both programs, the lines from "ERRHAND.PRG" will replace those in the current program.

```
MERGE "ERRHAND.PRG"          (from current directory)
MERGE "ram:ERRHAND.PRG"      (from RAM)
MERGE "rom:ERRHAND.PRG"      (from ROM)
MERGE "card1:ERRHAND.PRG"    (from DOS-formatted memory card)
```

MID$ *FUNCTION*

Field of Application **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$_1$> 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
```

NAME DATE$ *STATEMENT*

Field of Application

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 e.g.:
98.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$="980601"
20 NAME DATE$ 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:

```
JUN 01, 1998
```

NAME WEEKDAY$ *STATEMENT*

Field of Application **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, i.e. 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$="981015"
30 NAME WEEKDAY$ 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:

```
Wed, 10/15/98
```

NASC STATEMENT

Field of Application

Selecting a 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 (for Microsoft Windows v. 3.0, 3.1)*

Remarks

Please refer to the end of this section for complete character set tables. In case of national character sets, the reference numbers coincide with the telephone country codes.

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 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 operation system of the host.

That implies that a certain ASCII code received by the printer may result in different characters being 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.

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.

Continued!

NASC, cont'd. *STATEMENT*

Remarks, cont'd.

- **Communication:**
  Data transmitted via any of the communication channels will not be affected as the data is defined as ASCII values, not than alpha-numeric characters. The active character set of the **receiving** unit will decide the graphic presentation of the input data, e.g. 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 (i.e. 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
```

NEW STATEMENT

Field of Application

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. 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 terminal's 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
```

NEXT *STATEMENT*

Field of Application

Creating a loop in the program execution, where a counter is incremented or decremented according to a FOR statement.

Syntax

NEXT[<nvar>]

<nvar> *is optionally the variable used as a counter in the FOR statement.*

Remarks

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

The counter's designation, its initial and final values and, optionally, its incrementation value are specified by a FOR...TO...STEP statement. Each time the statement NEXT is encountered, the loop will be executed again until the final value is reached. Then the execution will proceed from the first line after the NEXT statement.

If the optional variable is omitted, the NEXT statement will make the program execution 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, i.e. 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.

Example

The counters A% and B% are incremented by means of two nested FOR...NEXT loops:

```
10    FOR A%=20 TO 80 STEP 20
20    FOR B%=1 TO 2
30    PRINT A%,B%
40    NEXT B% : NEXT A%
RUN
```

yields:

```
20    1
20    2
40    1
40    2
60    1
60    2
80    1
80    2
```

NORIMAGE (NI) *STATEMENT*

Field of Application

Returning to normal printing after an INVIMAGE statement has been issued.

Syntax

NORIMAGE|NI

Remarks

Normal image is the default type of printing, i.e. text and images will be printed in black on a paper-coloured background.

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 "SW030RSN"
50    PRTXT "INVERSE PRINTING"
60    PRPOS 30, 200
70    NORIMAGE
80    PRTXT "NORMAL PRINTING"
90    PRINTFEED
RUN
```

ON BREAK GOSUB *STATEMENT*

Field of Application **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:"/"rs485:"
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 *STATEMENT*

Field of Application

Branching to a subroutine, when the background reception of data on the specified communication channel is interrupted.

Syntax

ON$_{\leftrightarrow}$COMSET<nexp$_1$>GOSUB<nexp$_2$>|<line label>

<nexp$_1$> *is one of the following communication channels:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

<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
```

Continued!

ON COMSET GOSUB, cont'd. *STATEMENT*

Examples, cont'd. *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 *STATEMENT*

Field of Application

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 given as **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 lifted (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 "SW030RSN"
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
```

Continued!

ON ERROR GOTO, cont'd. *STATEMENT*

Examples, cont'd. *The same example written without line numbers would look like this:*

```
IMMEDIATE ON
LED 0 ON:LED 1 OFF
ON ERROR GOTO QQQ
FONT "SW030RSN"
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 OFF
```

ON GOSUB *STATEMENT*

Field of Application **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. E.g., 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: "You have pressed key 2"
RETURN
ZZZ: "You have pressed key 3"
RETURN
IMMEDIATE ON
```

ON GOTO *STATEMENT*

Field of Application **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 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 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: "You have pressed key 2"
GOTO YYY
ZZZ: "You have pressed key 3"
GOTO YYY
IMMEDIATE ON
```

ON KEY GOSUB STATEMENT

Field of Application

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 i.d. number of one of the keys on the printer's front panel (see illustration below).*

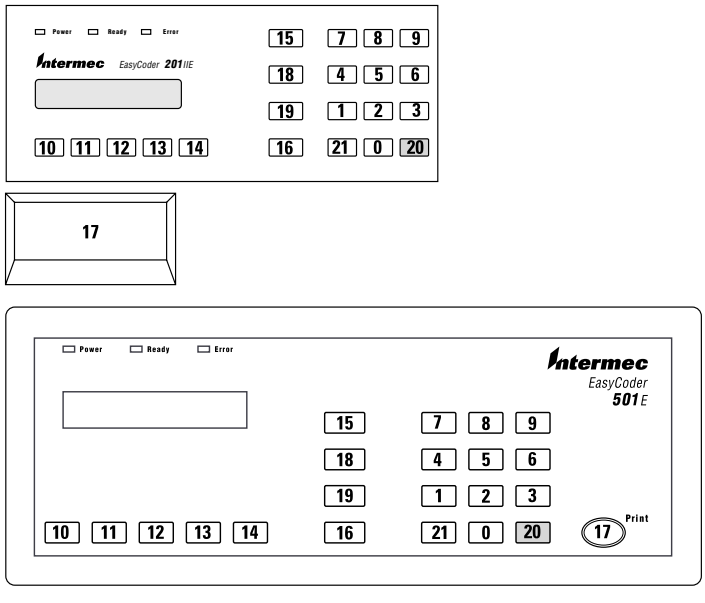
<ncon>/<line label> *is the number or label of the line to which the program will branch when the specified key is pressed down.*

Remarks

All *Intermec Fingerprint* printer models are fitted with a "Print" button or key. In addition, some models are fitted with a membrane keyboard. Each key can be enabled individually using its i.d. number in a KEY ON statement. Then the key can be assigned, alone or in combination with the **C**-key, to make the program branch to a subroutine using an ON KEY... GOSUB statement.

Please note the difference between the i.d. numbers of the keys and the ASCII values they are able to produce (see e.g. 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.



Default I.d. numbers of the most common keyboard types in the *EasyCoder* printer line.
(Some printers only have a Print key or button)

The **C** or Clear key (i.d. No. 20) works as a Shift key. When pressed in connection with another key, it adds 100 to the i.d number of the other key.

Continued!

ON KEY GOSUB, cont'd. *STATEMENT*

Examples

*This example illustrates how activating the **F1** key (i.d. 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 "SW030RSN"
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 "SW030RSN"
PRPOS 30,100
PRTXT "HELLO"
PRINTFEED
END
IMMEDIATE ON
RUN
```

ON/OFF LINE *STATEMENT*

Field of Application

Controlling the SELECT signal on the optional Centronics communication channel.

Syntax

ON | OFF↔LINE<nexp>

<nexp> *specifies the communication channel as 4 (= "Centronics:").*

Remarks

Pin 13 in the Centronics 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, i.e. the Centronics channel will be ON LINE.

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 STATEMENT

Field of Application

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↔<INPUT|OUTPUT|APPEND>↔]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 10 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.

Continued!

OPEN, cont'd. *STATEMENT*

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
```

OPTIMIZE ON/OFF — STATEMENT

Field of Application

Enabling/disabling optimizing strategies for batch printing.

Syntax

OPTIMIZE [<sexp>]↔ON | OFF

<sexp>	*optionally specifies the optimizing strategy as "PRINT", STRING" or "BATCH".*
ON/OFF	*enables/disables optimizing strategy respectively.*

Remarks

This facility is intended to speed up batch printing, i.e. the uninterrupted printing of large numbers of identical or very similar labels. OPTIMIZE is not recommended for the printing of labels with frequently varying content.

"**PRINT**" optimizing strategy implies that the processing, which is performed before the printing starts, is minimized on the basis of an analysis of the preceding label's appearance.

"**STRING**" optimizing strategy implies that all printable strings are converted to bitmap format, which makes the printing faster, provided the strings are not altered between copies. However, this requires more RAM memory. Should any difficulties be encountered during printing, disable the "STRING" optimizing strategy and try again.

"**BATCH**" optimizing strategy implies that the program execution will not wait for the label to be printed, but proceeds as soon as the print image has been transferred to the image buffer.

If no strategy is specified in the statement, the "PRINT" and "STRING" optimizing strategies will be enabled/disabled simultaneously.

By default, all three strategies are disabled. However, if the following conditions are all fulfilled, the BATCH optimizing strategy is automatically enabled:
- A value larger than 1 has been entered for the PRINTFEED statement.
- LTS& OFF
- CUT OFF

Also see page 9 for remaining bugs and limitations.

Example

This example enables both PRINT and STRING optimizing strategies before a batch printing job is started and disables the PRINT strategy of them afterwards. By entering a value for the PRINTFEED statement, the BATCH strategy is automatically enabled too:

```
10   OPTIMIZE ON
20   FONT "SW030RSN"
30   PRPOS 30,300
40   PRTXT "NO SMOKING"
50   PRINTFEED 10
60   OPTIMIZE "PRINT" OFF
. . .
```

PCX2BMP EXTERNAL COMMAND

Field of Application

Converting image files in .PCX format to the internal bitmap format of *Intermec Fingerprint*.

Syntax

RUN "pcx2bmp$_{\leftrightarrow}$<scon$_1$>$_{\leftrightarrow}$<scon$_2$>"	**Converts file**
RUN "pcx2bmp$_{\leftrightarrow}$-i$_{\leftrightarrow}$<scon$_1$>"	**Dumps short file info**
RUN "pcx2bmp$_{\leftrightarrow}$-i$_{\leftrightarrow}$-v<scon$_1$>"	**Dumps comprehensive file info**

-i	*dumps short information on the .PCX file without any conversion.*
-i -v	*dumps comprehensive information on the .PCX file without any conversion.*
<scon$_1$>	*is the name, and optionally the device, of the original .PCX file.*
<scon$_2$>	*is the name, and optionally the device, of the new bitmap file (file name max. 30 characters).*

Remarks

.PCX a bitmap format used in e.g. *Windows* applications. BMP is an internal bitmap format used in *Intermec Fingerprint* and must not be confused with .BMP, which is a bitmap format in *Windows*.

An image file in .PCX format cannot be used in a *Intermec Fingerprint* program before it has been converted to a file in BMP format. The file .PCX can be downloaded to the printer via e.g. Kermit (see TRANSFER KERMIT stmt), or be copied into a DOS-formatted memory card.

Before starting to download and covert an PCX image file, check that you have a suffient amount of free memory in the printer using a FILES statement.

There must be a free space corresponding to **twice** the size of image plus the size of the downloaded PCX file. The size of the image is roughly the same as the uncomressed image file (some image-creating programs have facilities for determining the exact size of the image). Once the conversion is completed, only one single copy of image and the PCX file will remain stored in the memory. To save space, the PCX file can be removed using a KILL statement.

The conversion command **pcx2bmp** must be entered in lowercase characters, be enclosed by double quotation marks, and be preceded by a RUN statement, i.e. **`RUN "pcx2bmp <pcx-file> <bmp-file>"`**.

When the file is used for printing the first time (see PRINT IMAGE statement), an image with the same name as the file will be created. This implies that there will be one file and one image with the same name.

Continued!

PCX2BMP, cont'd. *EXTERNAL COMMAND*

Example

Conversion of a .PCX file in a memory card to the printer's permanent memory (the device name is optional for the current directory, see CHDIR statement):

```
RUN "pcx2bmp card1:LOGO.PCX c:LOGOTYPE.1"
```

yields:

```
Ok
```

A shorthand information dump of the same file may look like this on the screen:

```
RUN "pcx2bmp -i rom:17n.pcx"
```

yields:

```
PCX Version 5, RLL, 17x17, 1 plane(s)
```

A comprehensive information dump of the same file may look like this on the screen:

```
RUN "pcx2bmp -i -v rom:17n.pcx"
```

yields:

```
Manufacturer   : 10
                 Zsoft .pcx
Version        : 5
                 Version 3.0 and > of PC Paintbrush
Encoding       : 1
                 .PCX run length encoding
Bits per pixel : 1
Xmin           : 0
Ymin           : 0
Xmax           : 16
Ymax           : 16
HDpi           : 1024
VDpi           : 768
Colormap [48]
       0   0   0 255 255 255 148 132  71etc.
      79  90  44 111  71 113 236 107  14etc.
       0   0  12   0 148 132  44 111  71etc.
NPlanes        : 1
Bytes per line : 4
Palette info   : 1143
                 Unknown
HScreen size   : 32512
VScreen size   : 0
*** Warning! 44 (of 54) non-zero filler ***
```

PORTIN *FUNCTION*

Field of Application **Reading the status of a port on the Industrial Interface Board.**

Syntax

PORTIN(<nexp>)

<nexp> *is the number of the port to be read. Pin numbers refer to the IN/OUT connector on the Industrial Interface Board:*

IN ports:
101 = Pin 9 & 10
102 = Pin 11 & 12
103 = Pin 13 & 15
104 = Pin 14 & 15

OUT ports:
201 = Pin 1 & 2
202 = Pin 3 & 4
203 = Pin 5 & 6
204 = Pin 7 & 8

Remarks

The Industrial Interface Board is available as an option for some *Intermec Fingerprint*-compatible printers. It contains a connector with 4 IN and 4 OUT ports. For information on how to set the OUT ports, please refer to the PORTOUT statement.

A current can be lead through an opto-coupler 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.

Please refer to the *Technical Manual* or *Service Manual* of the printer model in question for more information on the Industrial Interface Board.

Example

The status of IN port 101 on the Industrial Interface decides when a label is to be printed. The printing will be held until the current comes off:

```
10    FONT "SW030RSN"
20    PRTXT "POWER IS OFF"
30    IF PORTIN (101) THEN GOTO 30
40    PRINTFEED
50    END
```

PORTOUT ON/OFF *STATEMENT*

Field of Application

Setting one of four relays on the Industrial Interface Board to either Open or Closed.

Syntax

PORTOUT (<nexp>) ON|OFF

<nexp> *is the number of the port for which the relay is to be set. Pin numbers refer to the IN/OUT connector on the Industrial Interface Board:*
201 = Pin 1 & 2
202 = Pin 3 & 4
203 = Pin 5 & 6
204 = Pin 7 & 8

Remarks

The Industrial Interface Board is available as an option for some *Intermec Fingerprint*-compatible printers. It contains a connector with 4 IN and 4 OUT ports. For information on the IN ports, please refer to the PORTIN function.

A current can be lead through a relay in each OUT port. Straps on the Industrial Interface Board decides whether PORTOUT ON/PORTOUT OFF should result is Open/Closed port or vice versa.

This feature is intended to allow the execution of the *Intermec Fingerprint* program to control various external units like gates, lamps or conveyor belts.

Please refer to the *Technical Manual* or *Service Manual* of the printer model in question for information on straps and connections.

Example

The OUT port 201 on the Industrial Interface is Opened and then Closed like this (provided that the Industrial Interface Board is fitted with strap between pin 1–2 on P2):

```
. . . . .
. . . . .
1000 PORTOUT (201) ON
. . . . .
. . . . .
2000 PORTOUT (201) OFF
. . . . .
. . . . .
```

PRBAR (PB) *STATEMENT*

Field of Application **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.

Example

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 #2,"SW030RSN" 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
```

PRBOX (PX) *STATEMENT*

Field of Application **Creating a box.**

Syntax

PRBOX|PX<nexp$_1$>,<nexp$_2$>,<nexp$_3$>

<nexp$_1$> *is the **height** of the box in dots (max. 6000).*
<nexp$_2$> *is the **width** of the box in dots (max. 6000).*
<nexp$_3$> *is the **line weight** in dots (max. 6000).*

Remarks

A 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 centre along its baseline.

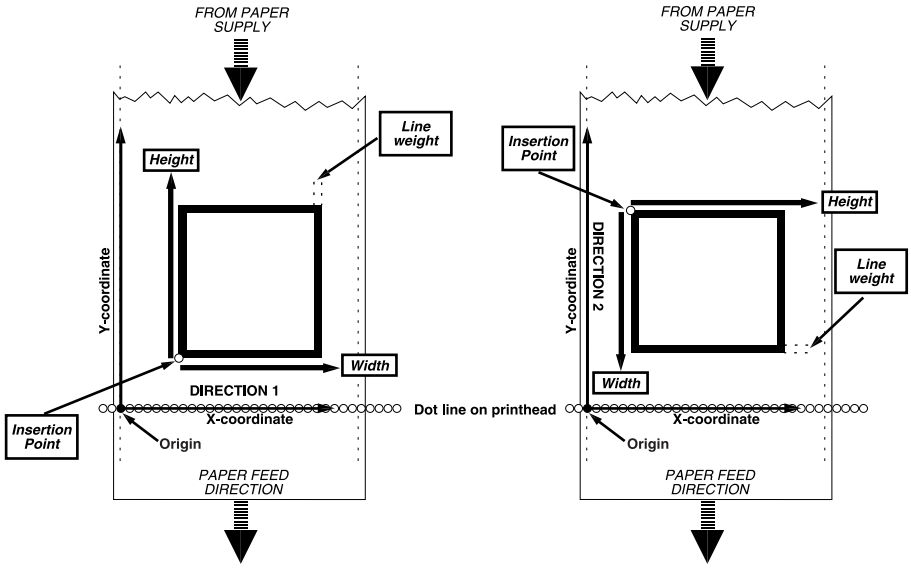
The print direction specifies how the box is rotated in relation to its anchor point. The baseline is in parallel with text lines in the selected print direction. The height of the box goes the same way as the height of characters and bar codes in the selected print direction.

The line weight (i.e. the thickness of the line) grows inward from the anchor point, i.e. the heavier the line, the less white area inside the box. Thus, it is possible to create a black field using a box with very heavy lines. The white area inside a box can be used for printing. Boxes, lines and text may cross.

Example

This examples draws a rectangle:

```
10    PRPOS 50,50
20    PRBOX 100, 200, 3
30    PRINTFEED
RUN
```



Left:
A box aligned left in print direction 1.

Right:
A box aligned left in print direction 2.

PRIMAGE (PM) *STATEMENT*

Field of Application

Selecting an image stored in the printer's memory.

Syntax

PRIMAGE|PM<sexp>

<sexp> *is the full name of the desired image including extension. Max. 60 different images can be used.*

Remarks

An image is positioned according to the preceding PRPOS, DIR and ALIGN statements. It is magnified according to the currently valid MAG statement, if any.

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 currect image.

The firmware will start searching for the specified image in the printer's memory. If the image is not found there, the current directory (see CHDIR statement) will be searched for an image file with the same name. If such a file is found, it will be copied and used as an image. If there is not enough memory left to hold the copy, copies of old image files will be deleted until a sufficient amount of memory becomes available.

Example

This example illustrates the printing of a label containing an image upside down and magnified 2x:

```
10    PRPOS 200,200
20    DIR 3
30    ALIGN 5
40    MAG 2,2
50    PRIMAGE "GLOBE.1"
60    PRINTFEED
RUN
```

PRINT (?) *STATEMENT*

Field of Application **Printing of 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

Do not confuse this statement with the PRINTFEED statement.

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), e.g. usually on the screen of the host. The shorthand form of PRINT is a question mark (?).

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 firmware 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 *STATEMENT*

Field of Application

Enabling or disabling printing of a label by pressing the Print key.

Syntax

PRINT KEY ON|OFF

Default: *PRINT KEY OFF*

Remarks

All *Intermec Fingerprint*-compatible printers are provided with at least one "Print" button or key. In the Immediate Mode and in the *Intermec Direct Protocol*, this 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 strip will be printed and fed out.

Note that this 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 when available):

```
INPUT ON ↵
PRINT KEY ON ↵
PP 100,100 ↵
FT "SW030RSN" ↵
PT "TEST LABEL" ↵

<Press Print Key>

INPUT OFF ↵
```

PRINT# STATEMENT

Field of Application **Printing of 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$_{2\text{-}n}$>|<sexp$_{1\text{-}n}$>> *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 e.g. be cleared by sending PRINT#<ncon> 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!

```
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) *STATEMENT*

Field of Application

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>]

<nexp> *is, optionally, the number of copies to be printed.*

Remarks

The PRINTFEED statement is used for printing labels etc. Each time a PRINTFEED statement, without any appending value, is executed, one new copy 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 Immediate and Programming Modes, the copies will be identical, whereas 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, i.e. OPTIMIZE BATCH ON. When theses conditions are no longer fulfilled, BATCH optimizing strategy is automatically disabled, i.e. OPTIMIZE BATCH OFF.

The execution of a PRINTFEED statement clears the following statements to their default values:

ALIGN	BARFONT	BARFONT ON/OFF	INVIMAGE
BARHEIGHT	BARMAG	BARRATIO	MAG
BARTYPE	DIR	FONT	PRPOS
BARSET			

Fields defined by statements that already have been executed before the PRINTFEED statement are not affected. Note that, when using a PRINTFEED statement 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.

The amount of web to be fed out when executing 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). Please refer to the *Technical Manual* for more information. The amount of paper to be fed out after a PRINTFEED can be further modified by an additional positive or negative FORMFEED statement, either before or after the PRINTFEED statement.

Also see page 9 for remaining bugs and limitations.

Continued!

PRINTFEED (PF), cont'd. *STATEMENT*

Examples

Printing a single label with one line of text:

```
10    FONT "SW030RSN"
20    PRTXT "Hello!"
30    PRINTFEED
RUN
```

Printing five identical labels with one line of text:

```
10    FONT "SW030RSN"
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 "SW030RSN"
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 time is updated between labels, provided a predefined layout is used:

```
INPUT  ON  ↵
FORMAT INPUT "#","@","&"↵
LAYOUT INPUT "LABEL1"↵
FT "SW030RSN" ↵
PP 100,100 ↵
PT TIME$ ↵
PP 100,200 ↵
PT VAR1$↵
LAYOUT END ↵
LAYOUT RUN "LABEL1"↵
#See how time flies&@↵
PF 5 ↵
INPUT OFF ↵
```

PRINTFEED NOT (PF NOT) *STATEMENT*

Field of Application **Preparing the printing.**

Syntax

PRINTFEED|PF↔ NOT

Remarks

PRINTFEED NOT prepares the printing in order to shorten the time between the execution of a PRINTFEED statement and the actual printing by preprocessing as much of the label layout as the size of the image buffer allows. Thereby the time between the execution of the PRINTFEED statement and the actual start of the printing can be minimized. This facility is useful for applications where the PRINTFEED execution is triggered by some external device and quick delivery of a large printed label with lots of information is required.

A PRINTFEED NOT statement bust be followed by a PRINTFEED statement before a new PRINTFEED NOT statement can be executed.

Example

A label will be printed as soon as a current to one of the ports of the Industrial Interface Board is switched on. Warning, do not run this example, since is contains a loop which you may not be able to break!

```
10   PRPOS 50, 50
20   MAG 4, 4
30   PRIMAGE "Intermec.1"
40   PRINTFEED NOT
50   IF PORTIN (101) THEN PRINTFEED ELSE GOTO 50
.....
.....
```

PRINTONE *STATEMENT*

Field of Application

Printing of 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, i.e. 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# STATEMENT

Field of Application

Printing of 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$_{2\text{-}n}$> *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#<ncon> 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:

PRLINE (PL) *STATEMENT*

Field of Application **Creating a line.**

Syntax

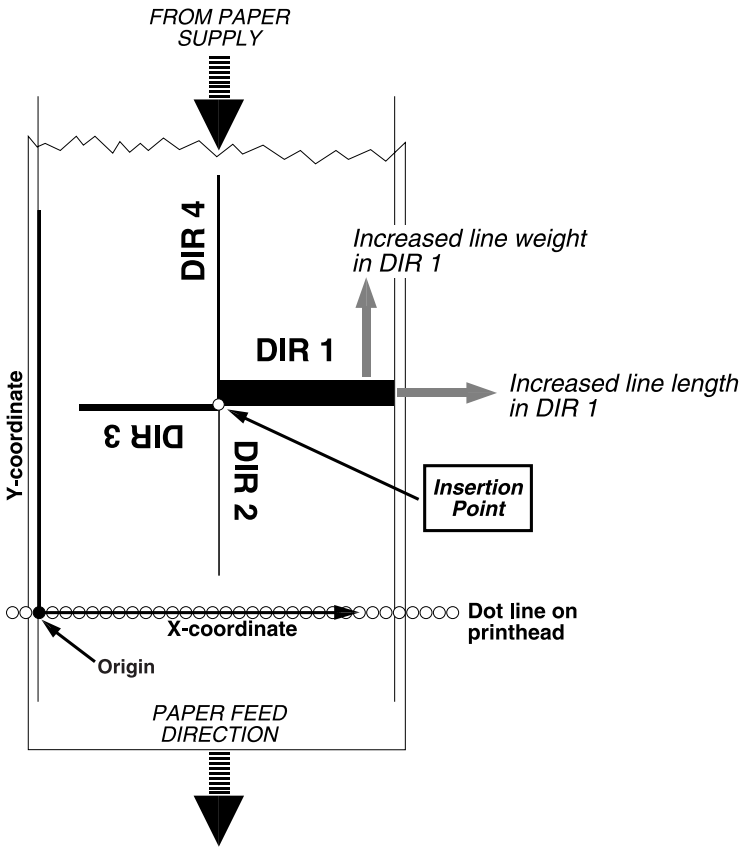
PRLINE|PL<nexp$_1$>,<nexp$_2$>

<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. The line runs in parallel with text printed in the selected direction.

A line can be ALIGNed left, right or centre. The anchor points are situated at the bottom of the line, i.e. with an increased line weight (thickness), the line will the grow upward in relation to the selected direction. In the illustration below, all lines are aligned left.



Example

This example draws a 2.5 cm (1") long and 2 dots thick line across the paper in an 8 dots/mm printer:

```
10    PRPOS 50,100
20    PRLINE 200,2
30    PRINTFEED
RUN
```

PRPOS (PP) STATEMENT

Field of Application

Specifying the insertion point for a line of text, a bar code, an image, a box, or a line.

Syntax

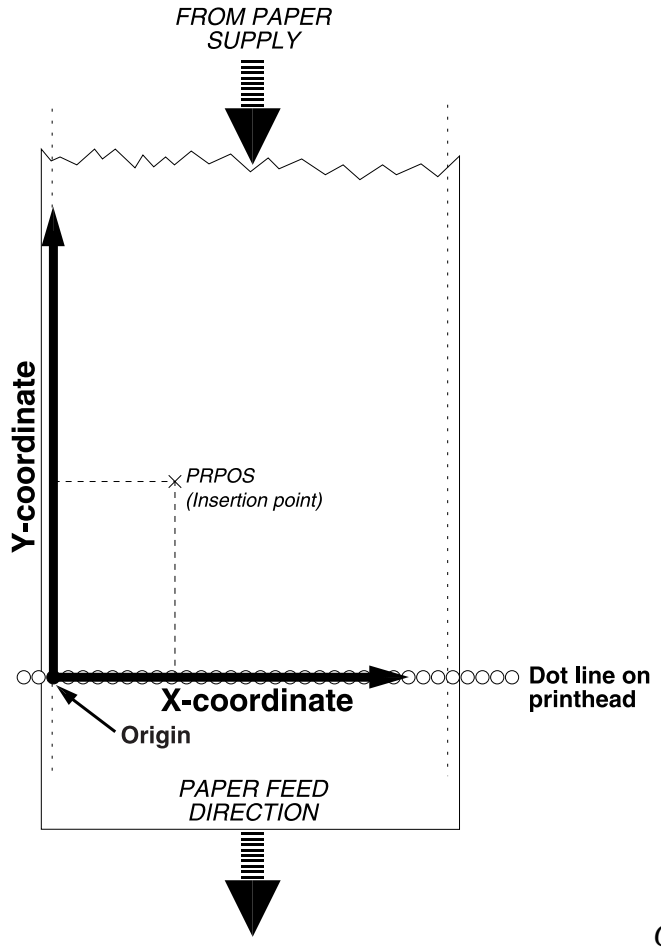
PRPOS|PP<nexp$_1$>,<nexp$_2$>

<nexp$_1$> *is the X-coordinate (number of dots).*
<nexp$_2$> *is the Y-coordinate (number of dots).*
Default value: *0,0*
Reset to default by: *PRINTFEED execution or SETUP files.*

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 the origin and setting the max. print length along the Y-axis from the origin.

The X-coordinate goes across the paper and the Y-coordinate along the paper feed direction, as illustrated below. They are set in relation to the origin on the printhead, not in relation to the paper. Thus, the position where an object actually will be printed depends on the relation between printhead and paper at the moment when the printing starts.



Continued!

PRPOS (PP), cont'd. *STATEMENT*

Remarks, cont'd.

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 e.g. an image, the alignment and other formatting instructions must be considered. A field that do not fit entirely inside the print window will cause an error (Error 1003 *"Field out of label"*).

Examples

Programming and printing a line of text:

```
10    FONT "SW030RSN"
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 "SW030RSN"
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:

```
10    FONT"SW030RSN"
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, i.e. 10 lines will be printed.

PRSTAT *FUNCTION*

Field of Application

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.
2 Returns the Y-position for the insertion point at DIR 2&4.

Remarks

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

OK	0	
Printhead lifted	1	
Label not removed	2	*See note 1.*
Printer out of paper	4	
Printer out of transfer ribbon	8	*See note 2.*
Printhead voltage too high	16	
Printer is feeding	32	

Note 1: Always 0 in printers not fitted with a label-taken sensor.
Note 2: Always 0 in direct thermal printers.

If two error conditions occur at the same time, e.g. the printhead is lifted and the printer is out of paper, 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.

If the PRSTAT function is appended by a numeric expression (= 1 or 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 e.g. measuring the length of a text or a bar code.

Examples

This examples shows how two error conditions are checked:

```
10   IF (PRSTAT AND 1) THEN GOSUB 1000
20   IF (PRSTAT AND 4) THEN GOSUB 1010
30   END
.....
1000 PRINT "Printhead is lifted":RETURN
1010 PRINT "Printer out of paper":RETURN
```

This example illustrates how you can check the length of a text:

```
10   PRPOS 100,100: FONT "SW030RSN"
20   PRTXT "ABCDEFGHIJKLM"
30   PRINT PRSTAT(1)
RUN
```

yields:

```
305
```

PRTXT (PT) STATEMENT

Field of Application

Providing the input data for a text field, i.e. one line of text.

Syntax

PRTXT|PT<<nexp>|<sexp>>[;<<nexp>|<sexp>>...][;]

<<nexp>|<sexp>> specifies one line of text (max. 300 characters)

Remarks

The text field must be defined in regard of FONT 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 double quotation marks, whereas numeric constants or any kind of variables must not.

Examples

Programming and printing a line of text:

```
10    FONT "SW030RSN"
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 "SW030RSN"
20    PRPOS 30,300
30    PRTXT "SUN";"SHINE"
40    A$="MOON"
50    B$="LIGHT"
60    PRPOS 30,260
70    PRTXT A$+B$
80    PRINTFEED
RUN
```

yields a label with the text:

```
SUNSHINE
MOONLIGHT
```

Continued!

PRTXT (PT), cont'd. *STATEMENT*

Examples, cont'd.

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 "SW030RSN"
20    PRPOS 30,300
30    PRTXT 123;456
40    A%=222
50    B%=555
60    PRPOS 30,260
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, e.g.:

```
10    FONT "SW030RSN"
20    PRPOS 30,300
30    A$="September"
40    B%=27
50    PRTXT A$;" ";B%;" ";"1998"
80    PRINTFEED
RUN
```

yields a label with the text:

```
September_27_1998
```

Two program lines of text will be printed on the same line if the first program line is appended by a semicolon:

```
10    FONT "SW030RSN"
20    PRPOS 30,300
30    PRTXT "HAPPY"+" ";
40    PRTXT "BIRTHDAY"
50    PRINTFEED
RUN
```

yields a label with the text:

```
HAPPY BIRTHDAY
```

PUT STATEMENT

Field of Application **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 *FUNCTION*

Field of Application **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$_2$>

I.e. 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 e.g.:

```
31
45
82
1
13
16
41
77
20
70
```

RANDOMIZE *STATEMENT*

Field of Application

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 99999999.

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% = RANDOM (1,100)
30    PRINT A%
RUN
Random Number Seed (0 to 99999999) ?
```

Enter 555 *yields e.g.:*

```
36
```

When the reseeding integer is specified, no prompt will appear:

```
10    RANDOMIZE 556
20    A% = RANDOM (1,100)
30    PRINT A%
RUN
```

yields e.g.:

```
68
```

A higher degree of randomization will be obtained in the random integer generator is reseeded with a more or less random integer, e.g. provided by a TICKS function:

```
10    A% = TICKS
20    RANDOMIZE A%
30    B% = RANDOM (1,100)
40    PRINT B%
RUN
```

yields e.g.:

```
42
```

READY STATEMENT

Field of Application

Ordering a ready signal, e.g. XON, CTS/RTS or PE, to be transmitted from the printer on the specified communication channel.

Syntax

READY[<nexp>]

<nexp> *optionally specifies a comm. channel:*
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

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, e.g. because of a full receive buffer.

For the "centronics:" optional communication channel, BUSY/READY controls the PE (paper end) signal on pin 12 according to an error-trapping routine, as described in the *Technical Manual*. (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:

```
10    FONT "SW030RSN"
20    PRTEXT "HELLO!"
30    BUSY2
40    PRINTFEED
50    READY2
RUN
```

REBOOT STATEMENT

Field of Application

Restarting the printer.

Syntax

```
REBOOT
```

Remarks

This statement has exactly the same effect as turning off and on the power.

Using SYSVAR(24), the printer can be polled by the host to see if the printer has been REBOOTed or otherwise restarted, e.g. manually or because of a power failure.

When the printer starts up, the following happens:

- RAM checksums are checked.
- RAM memory is cleared, partially or completely:
  - All of RAM memory, if a checksum error occurs
  - Part of RAM memory, if checksums are OK:
    All data, variables and program lines, which have not been SAVEd will be lost, fonts and images stored in the no-save area of the RAM memory will be lost, and all buffers will be emptied.
    All defined errors, counters, time formats, date formats, and input separators will be lost, see ERROR, COUNT&, FORMAT TIME$, FORMAT DATE$ and FORMAT INPUT statements.
- Printer type is tested by shifting through printhead to check head width.
- The printer starts up in the Immediate Mode.
- Verbosity level is set to default, see SYSVAR(18).
- Type of error message is set to default, see SYSVAR(19).
- Setup values are not affected by a REBOOT or power-up. If any part of the printer's setup is lost, the message *"Setup lost – Press any key"* will be displayed and the program waits for a key to be pressed, after which the printer will be set up with default setup values.
- Memory allocation is reset to default.
- The "console:" device is initiated.
- The cutter, if any, is rotated one cycle to home position. If the cutter is not in home position, a new attempt to perform a cut cycle will be performed.
- Bar codes are initiated (in all memory parts).
- Standard IN and OUT channels are initiated.
- MAP table is initiated.
- Use of COMSET is initiated.
- BREAK handling is initiated.
- Ribbon Save is initiated, if ribbon save hardware is found.
- Font and Image handlers are initiated (in all memory parts).
- Default error messages for *Intermec Direct Protocol* are set up.
- If there is an AUTOEXEC.BAT file anywhere in the printer's memory, it will be executed. Otherwise, the printer waits for input on the standard IN channel.

REDIRECT OUT *STATEMENT*

Field of Application **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, that means the screen of the host computer or terminal. 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 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 RAM memory is redirected. The redirection is then terminated and the file is OPENed for input.

```
10   REDIRECT OUT "LIST.DAT"
20   FILES "ram:"
30   REDIRECT OUT
40   OPEN "LIST.DAT" FOR INPUT AS #1
. . . . .
. . . . .
. . . . .
```

REM (') STATEMENT

Field of Application

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. 300 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 should be preceded by a colon (:REM).

A shorthand form for REM is a single quotation mark ('), i.e. 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 "SW030RSN"
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/FONT *STATEMENT*

Field of Application **Removing a specified image or bitmap font from the printer's memory.**

Syntax

REMOVE↔IMAGE|FONT<sexp>

<sexp> *is the full name incl. extension of the image or bitmap font to be removed.*

Remarks

Useful for removing obsolete or faulty images or bitmap fonts from the RAM memory in order to save valuable memory space.

Images or bitmap fonts in any other part of the memory cannot be removed.

Note that there is a distinction between on one hand images and fonts and on the other hand image files and font files (compare with IMAGES, FONTS and FILES statements). This implies that REMOVE statements can only be used for images downloaded by means of a STORE statement (See STORE and STORE IMAGE) or bitmap fonts. Image files or scaleable outline font files downloaded by means of e.g. a TRANSFER KERMIT statement should be removed the same way as other files, i.e. by means of a KILL statement.

Be careful, REMOVE IMAGE|FONT is irreversible!

Example

```
10    REMOVE IMAGE "LOGOTYPE.1"
20    REMOVE FONT "XY050BMN.2"
RUN
```

RENUM STATEMENT

Field of Application

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, e.g. 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 "SW030RSN"
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 "SW030RSN"
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.

RESUME *STATEMENT*

Field of Application

Resuming program execution after an error-handling subroutine has been executed.

Syntax

RESUME[<<ncon>|<line label>|<NEXT>|<0>>]

<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.

Examples

This short program is the basis for two examples of alternative subroutines:

```
10    ON ERROR GOTO 1000
20    PRTXT "HELLO"
30    PRIMAGE "GLOBE.1"
40    PRINTFEED
50    END
```

1. A font is selected automatically and execution is resumed from the line where the error occurred. If another error than the specified error condition occurs, the execution is terminated.

```
1000 IF ERR=1019 THEN FONT "SW030RSN":RESUME
1010 RESUME 50
```

2. An error message is displayed and the execution goes on from the line following the one where the error occurred.

```
1000 IF ERR=1019 THEN PRINT "Invalid font"
1010 RESUME NEXT
```

RETURN *STATEMENT*

Field of Application

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, an error condition will occur (Error 28 *"Return without Gosub"*).

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 at subroutine 2
You're back at subroutine 1
You're back at the main program
```

RIBBON SAVE ON/OFF *STATEMENT*

Field of Application **Enabling/disabling the optional Ribbon Save device.**

Syntax

RIBBON SAVE ON|OFF

Default (if Ribbon Save is fitted): *RIBBON SAVE ON*
Default (if Ribbon Save is not fitted): *RIBBON SAVE OFF*

Remarks

This statement can only be used for thermal transfer printers run in thermal transfer mode and fitted with an optional ribbon save device.

When RIBBON SAVE ON is effective, the transfer ribbon will be stopped while blank parts of the label is fed out. Thereby the consumption of transfer ribbon will be reduced to a minimum. For specifications, please refer to the *Technical Manual* of the model in question.

RIBBON SAVE ON/OFF affects all of the immediate, programming and direct modes.

The Ribbon Save device does not support batch printing, i.e. the printing will stop between labels regardless of optimizing strategy (see OPTIMIZE ON/OFF statements). This does not restrict the capability of executing multiple PRINTFEED or PRINTFEED <nexp> statements. No ribbon will be saved at the execution of startadjust.

When the printer is fitted with a Ribbon Save device, that is **disabled** by means of a RIBBON SAVE OFF statement, it is recommended not to pull back the paper, e.g. by setting up a negative startadjust value or issuing a negative FORMFEED statement. Pulling back the paper increases the risk of ribbon wrinkling and unsatisfactory printout quality. In case pull back cannot be avoided, test first! Note that this restriction does **not** apply when the Ribbon Save Device is **enabled**.

Also see page 9 for remaining bugs and limitations.

Example

The Ribbon Save mechanism is turned on by means of the F1 key and turned off by means of F2:

```
10   KEY 10 ON: KEY 11 ON
20   ON KEY (10) GOSUB 1000
30   ON KEY (20) GOSUB 2000
.....
.....
1000 RIBBON SAVE ON
1010 RETURN
2000 RIBBON SAVE OFF
2010 RETURN
```

RIGHT$ *FUNCTION*

Field of Application

Returning a specified number of characters from a given string starting from the extreme right side of the string, i.e. from the end.

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, i.e. 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 STATEMENT

Field of Application **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 STATEMENT

Field of Application **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 ("ram:", "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 an error will occur (Error 40 *"Run statement in program"*).

A RUN statement executed in the *Intermec Direct Protocol* will make the printer switch to the Immediate Mode, i.e. 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 ROM memory, from its first line.

SAVE STATEMENT

Field of Application

Saving a file in the printer's RAM memory or optionally in a DOS-formatted memory card.

Syntax

SAVE<scon>[,P|L]

<scon>	*is the name of the file, optionally starting with a reference to a directory.* *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 add the extension .PRG. The name must not contain any double quotation marks (") and the extension must always start with a period (.) character.

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 RAM memory ("ram:") or in an optional DOS-formatted JEIDA-4 RAM-type 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.

Files **cannot** be SAVEd in the printer's EPROM memory, in an OTPROM-type memory card, or in a **non DOS-formatted** RAM-type memory card.

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.

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.

Continued!

SAVE, cont'd. *STATEMENT*

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 "card1:LABEL14.PRG"

saves the file in an optional DOS-formatted memory card.

SET FAULTY DOT *STATEMENT*

Field of Application

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> *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. In printers with dot-sensing, i.e. the *EasyCoder 201 II* and *EasyCoder 401/501/601* printer families, 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. By means of the BARADJUST statement, you can allow the firmware to automatically reposition horizontal bar codes sideways 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.

SET FAULTY DOT can also be used on printers without dot-sensing, but then you have to identify the faulty dots manually, since the HEAD function will not work.

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 "Intermec"
110  SET FAULTY DOT -1
120  PRINTFEED
RUN
```

SETSTDIO STATEMENT

Field of Application **Selecting standard IN and OUT communication channel.**

Syntax

SETSTDIO<nexp$_1$>[,<nexp$_2$>]

<nexp$_1$> *is the desired input channel:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"
4 = "centronics:"

<nexp$_2$> *optionally specifies a different output channel:*
0 = "console:"
1 = "uart1:"
2 = "uart2:"/"rs485:"
3 = "uart3:"

Remarks

The printer is usually controlled from its host computer or terminal via the standard communication channel. By default, the serial communication channel "uart1:" is used for both input and output. If only one channel is specified, it will serve as both input and output channel.

For programming, it is recommended to use "uart1:" as standard input **and** output 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.

The five possible IN/OUT channels are:

"console:"	Printer's keyboard (input) and display (output)
"uart1:"	Serial, standard
"uart2:"/"rs485:"	Serial, optional.
"uart3:"	Serial, optional.
"centronics:"	Parallel, optional (input only).

("uart2:" & "uart3:" excludes "centronics:" and vice versa; "rs485 excludes "uart2:" and "centronics:" and vice versa).

Example

*This example selects the "uart2:" communication channel as the standard input **and** output channel:*

```
10   SETSTDIO 2
. . . .
. . . .
```

SETUP STATEMENT

Field of Application

Entering the printer's Setup Mode, changing the setup by means of a setup file or setup string, or creating a setup file containing the printer's current setup values.

Syntax

SETUP [[WRITE<sexp$_1$>] | [<sexp$_2$>] | [<sexp$_3$>]]

<sexp$_1$> *is the desired name of a new file containing a copy of the printer's current setup.*

<sexp$_2$> *is the name of an existing setup file that will be used to change the printer's current setup.*

<sexp$_3$> *is a string used to change a parameter in the printer's current setup.*

Remarks

The SETUP statement can be used in four ways:

SETUP makes the printer enter the Setup Mode.

SETUP WRITE<file name> is used to automatically create a setup file with the assigned name containing a copy of the printer's **current** setup.

SETUP<file name> is used to change the printer's setup according to a setup file with a special syntax as described below.

SETUP<setup string> is used to change a parameter in the printer's current setup using a special syntax described below.

SETUP:

Never use the pure SETUP statement in a printer without keyboard, since once you have entered the Setup Mode there is no way of using or leaving the Setup Mode when the keyboard is missing.

SETUP WRITE:

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 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.

Another application of SETUP WRITE is printing the printer's current setup to a serial communication channel, e.g. SETUP WRITE "uart1:".

Continued!

SETUP, cont'd. STATEMENT

Remarks, cont'd.

SETUP FILES & SETUP STRINGS:

The methods of manual setup via the printer's built-in keyboard is discussed in the tutorial manual *"Intermec Fingerprint 6.13 Programmer's Guide"* and in the *Technical Manuals* for the various printer models. In printers, that do not have a built-in keyboard, the use of setup files or setup strings are the only ways to change most setup parameters, unless a special application program is used. Setup files and strings can also be used to change the setup as a part of the program execution, or to change the setup by remote control from the host.

A setup file may contain new values for one or several setup parameters, whereas as 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 in the Immediate and Direct Modes, i.e. in *Intermec Fingerprint Direct*.

There are some restrictions to the use of setup files and setup strings:

- In printers featuring automatic head resistance adjustment (dot sensing), any attempt to change the head resistance – manually or by means of setup files – will be ignored.
- In some printer models, certain functions (e.g. high or ultra high print speed or additional interface boards) may be disabled and any attempt to change the corresponding setup parameters will result in an error condition.
- The combination parity none/1 stop bit does not work with communication channel "uart1:" in some printer models. This restriction does not apply to "uart2:" and "uart3:". See Technical Manual for the printer model in question.
- In the setup, the device "rs485:"is refer to as UART2.

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 will be reset to their default values, when a SETUP statement is executed (see ALIGN, DIR, FONT, INVIMAGE, MAG and PRPOS).

The content of setup files can be listed by the use of the program FILELIST.PRG stored in the printer's ROM memory, and in the *Intermec Shell* startup program.

Continued!

SETUP, cont'd. *STATEMENT*

Remarks, cont'd.

SETUP FILES & SETUP STRINGS, cont'd.:
When creating setup files or setup strings, there is 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 underscore spaces:

"CONTRAST,5"
"SER-COM,**UART1|UART2|UART3**,BAUDRATE,**300|600|1200|2400|4800|9600|19200|38400**" *("uart1:" max. 19200)*
"SER-COM,**UART1|UART2|UART3**,PARITY,**NONE|EVEN|ODD|MARK|SPACE**"
"SER-COM,**UART1|UART2|UART3**,CHAR_LENGTH,**7|8**"
"SER-COM,**UART1|UART2|UART3**,STOPBITS,**1|2**"
"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,RTS/CTS,**ENABLE|DISABLE**" *(not RS422/RS485)*
"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,ENQ/ACK,**ENABLE|DISABLE**"
"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,XON/XOFF,DATA_TO_HOST,**ENABLE|DISABLE**"
"SER-COM,**UART1|UART2|UART3**,FLOWCONTROL,XON/XOFF,DATA_FROM_HOST,**ENABLE|DISABLE**"
"SER-COM,UART2,FLOWCONTROL,PROT_ADDR,**ENABLE|DISABLE**" *(RS485 only)*
"SER-COM,**UART1|UART2|UART3**,NEW_LINE,**CR/LF|LF|CR**"
"DETECTION,LSS_ADJUST,n|nnn" *(number of digits depend on model)*
"DETECTION,FEEDADJ,STARTADJ,nnnn" *(negative value allowed)*
"DETECTION,FEEDADJ,STOPADJ,nnnn" *(negative value allowed)*
"SERVICE,MEDIA_SIZE,XSTART,nnn"
"SERVICE,MEDIA_SIZE,WIDTH,nnnn"
"SERVICE,MEDIA_SIZE,LENGTH,nnnn"
"SERVICE,MEDIA_TYPE,**LABEL_(w_GAPS)|TICKET_(w_MARK)|TICKET_(w_GAPS)|FIX_LENGTH_STRIP|VAR_LENGTH_STRIP**"
"SERVICE,PRINT_DEFS,HEAD_RESIST,nnn" *(some models have automatic head resistance setup)*
"SERVICE,PRINT_DEFS,PAPER_TYPE,(name of paper or transfer ribbon)"
"SERVICE,PRINT_DEFS,NEW_SUPPLIES,nnnnnnnnnnnnn" *(some models only)*
"SERVICE,PERFORMANCE,**NORMAL|HIGH|ULTRA HIGH**" *(limited number of options in some models)*
"SERVICE,MEMORY_ALLOC,IMAGE_BUFF_SIZE,nnnn"
"SERVICE,MEMORY_ALLOC,REC_BUF_**UART1|UART2|UART3**,nnnnn"
"SERVICE,MEMORY_ALLOC,TRANS_BUF_**UART1|UART2|UART3**,nnnnn"

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
```

In this example, the current setup is saved in the printer's RAM memory under the name "SETUP1.SYS" (line 10). Then the start adjustment is changed to "200" by the creation of a new setup file named "SETUP2.SYS" (line 20–40). The setup file is finally used to change the printer's setup (line 50).

```
10   SETUP WRITE "SETUP1.SYS"
20   OPEN "SETUP2.SYS" FOR OUTPUT AS #1
30   PRINT#1,"DETECTION,FEEDADJ,STARTADJ,200"
40   CLOSE
50   SETUP "SETUP2.SYS"
```

Continued!

SETUP, cont'd. *STATEMENT*

Examples, cont'd. *This example 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 "Setup.sys" FOR OUTPUT AS #1
20 PRINT#1,"CONTRAST,7"
30 PRINT#1,"SER-COM,UART1,BAUDRATE,4800"
40 PRINT#1,"SER-COM,UART1,PARITY,EVEN"
50 PRINT#1,"SER-COM,UART1,CHAR LENGTH,8"
60 PRINT#1,"SER-COM,UART1,STOPBITS,1"
70 PRINT#1,"SER-COM,UART1,FLOWCONTROL,RTS/CTS,DISABLE"
80 PRINT#1,"SER-COM,UART1,FLOWCONTROL,ENQ/ACK,ENABLE"
90 PRINT#1,"SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA TO HOST,DISABLE"
100 PRINT#1,"SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA FROM HOST,DISABLE"
110 PRINT#1,"SER-COM,UART1,NEW LINE,CR"
120 PRINT#1,"DETECTION,LSS ADJUST,1"
130 PRINT#1,"DETECTION,FEEDADJ,STARTADJ,150"
140 PRINT#1,"DETECTION,FEEDADJ,STOPADJ,50"
150 PRINT#1,"SERVICE,MEDIA SIZE,XSTART,300"
160 PRINT#1,"SERVICE,MEDIA SIZE,WIDTH,300"
170 PRINT#1,"SERVICE,MEDIA SIZE,LENGTH,800"
180 PRINT#1,"SERVICE,MEDIA TYPE,TICKET (w GAPS)"
190 PRINT#1,"SERVICE,PRINT DEFS,HEAD RESIST,676"
200 PRINT#1,"SERVICE,PRINT DEFS,PAPER TYPE,RICOH 130LAB/LAM"
210 PRINT#1,"SERVICE,PERFORMANCE,HIGH"
220 PRINT#1,"SERVICE,MEMORY ALLOC,IMAGE BUFF SIZE,55"
230 PRINT#1,"SERVICE,MEMORY ALLOC,REC BUF UART1,600"
240 PRINT#1,"SERVICE,MEMORY ALLOC,TRANS BUF UART1,800"
250 CLOSE
260 SETUP "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"SERVICE,MEMORY ALLOC,IMAGE BUFF SIZE,100"↵
```

This method can also be used in the Programming Mode, e.g.:

```
10 SETUP"SERVICE,MEMORY ALLOC,REC BUF UART1,600"
.....
.....
```

SGN FUNCTION

Field of Application **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 STATEMENT

Field of Application **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 according to the Roman 8 ASCII table.

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 an error condition (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 STATEMENT

Field of Application

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 in Hz. (must be >0).*
<nexp$_2$> *is the duration of the sound in periods of 0.020 sec. each (must be >0).*

Remarks

This statement allows you include significant sound signals in your programs, e.g. to notify the operator that various errors have occurred. A sound 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 a very high value (>30,0000) which is in audible to the human ear.

Note	Hz	Note	Hz	Note	Hz	Note	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 392,15
20   SOUND 330,20
30   SOUND 330,15
40   SOUND 349,15
50   SOUND 392,15
60   SOUND 659,25
70   SOUND 659,25
80   SOUND 523,25
```

SPACE$ *FUNCTION*

Field of Application **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, e.g. 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 *FUNCTION*

Field of Application

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 according to the Roman 8 character set.*

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, an error will occur (Error 57 *"Subscript out of range"*). 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
```

STORE STATEMENT

Field of Application **Storing protocol frames of image data in RAM.**

Syntax

STORE<sexp>

<sexp> *contains the protocol frame(s) of image data .*

Remarks

This statement is obsolete and is retained for compatibility reasons only. For most applications, the STORE INPUT statement is more convenient.

Protocol frames can also be received using INPUT or INPUT$ statements and be assigned to string variables, or be received via a communication buffer (see COMSET, COMBUF$ etc.).

The STORE operation must be set up by a STORE IMAGE statement and be concluded by a STORE OFF statement.

Example

This example shows how an Intelhex file is received via the standard input channel and stored in the printer's RAM 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 IMAGE *STATEMENT*

Field of Application **Setting up parameters for storing an image in RAM.**

Syntax

STORE↔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 RAM at printer startup.*
<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 ("INTELHEX"). Upper- or lowercase letter can be used at will. Refer to the chapter *"Image Transfer Protocols"* for more information on the syntax of the 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 a nosave area of the RAM memory, i.e. an area which is erased at power up or REBOOT.

A STORE IMAGE statement must precede any STORE or STORE INPUT statement.

Also see page 9 for remaining bugs and limitations.

Example

This example shows how an Intelhex file is received via the standard input channel and stored in the printer's RAM 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 *STATEMENT*

Field of Application **Receiving and storing protocol frames of image data in RAM.**

Syntax

STORE$_{\leftrightarrow}$INPUT<nexp$_1$>[,<nexp$_2$>]

<nexp1> *is the timeout in ticks (0.01 sec.)*
<nexp2> *is, optionally, the number assigned to a device when it was OPENed for INPUT (default: Std 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 is usually more convenient than the STORE statement.

The STORE INPUT statements 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. Compare with the example for STORE stmt. The number of input parameters may vary depending on type of protocol, see STORE INPUT stmt.

```
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 stmt.

```
10   STORE OFF
20   OPEN "uart3:" 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 #3
100  STORE OFF
```

STORE OFF *STATEMENT*

Field of Application **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 RAM 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$ FUNCTION

Field of Application **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$ *FUNCTION*

Field of Application

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**********
```

SYSVAR — SYSTEM ARRAY

Field of Application **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 Read LSS receiver*
- *2 Not intended for public use*
- *3 Not intended for public use*
- *4 Not implemented*
- *5 Not intended for public use*
- *6 Not intended for public use*
- *7 Not intended for public use*
- *8 Read or set LSS emitter*
- *9 Reserved special applications*
- *10 Reserved special applications*
- *11 Reserved special applications*
- *12 Read paper counter*
- *13 Read ribbon counter*
- *14 Read errors since power on*
- *15 Read errors since last SYSVAR(15)*
- *16 Read number of bytes received at execution of a STORE or STORE INPUT statement*
- *17 Read number of frames received at execution of a STORE or STORE INPUT statement*
- *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 new startup has been performed since last SYSVAR(24)*
- *25 Not intended for public use*

Remarks

1. LSS receiver:
Reads the photoelectric receiver in the label stop sensor. A value between 0–255 will be returned. A small value indicates a label being detected, whereas a high value indicates the detection of a semi-transparent backing paper or that no web was detected. Exact limits between web, backing paper and no web must be determined empirically according to the transparency of each brand of backing paper, the condition of the LSS and the selected level (parameter 8).

Continued!

SYSVAR, cont'd. *SYSTEM ARRAY*

Remarks, cont'd.

2–7. Not for public use or not implemented.

8. LSS emitter:
Reads or sets the level of the light-emitter in the label stop sensor. The value depends on printer model (0–3, or 0–127). See *"LSS Setup"* in the *Technical Manual* of the printer model in question.

9–11. Reserved for special applications.

12. Paper counter:
Reads the value of the paper counter (certain models only).

13. Ribbon counter:
Reads the value of the ribbon counter (certain models only).

14. Errors since power up.
Reads number of errors detected since last power up.

15. Errors since last SYSVAR(15).
Reads number of errors detected since last executed SYSVAR(15).

16. Number of bytes received.
Reads the number of bytes received after the execution of a STORE or STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

17. Number of frames received.
Reads the number of frames received after the execution of a STORE or STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

18. Verbosity level.
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 communication port	
SYSVAR (18) = 8	Error after failed lines	

The levels can be combined, so e.g. 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, by e.g. PRINT SYSVAR(18).

Continued!

SYSVAR, cont'd. SYSTEM ARRAY

Remarks, cont'd.

19. Type of error message.
Four types of error messages can be selected:

SYSVAR(19) = 1	<string> in line <line>	(default)
	e.g. "Invalid font in line 10"	
SYSVAR(19) = 2	Error <number> in line <line>: <string>	
	e.g. "Error 19 in line 10: Invalid font"	
SYSVAR(19) = 3	E<number>	
	e.g. "E19"	
SYSVAR(19) = 4	Error <number> in line <line>	
	e.g. "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), by e.g PRINT SYSVAR(19).

20. Direct or transfer mode.
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.
SYSVAR(21) allows you to read the density of the printer's printhead, expressed as number of dots per millimetre.

22. Number of dots.
SYSVAR(22) allows you to read the number of dots in the printer's printhead.

23. Transfer ribbon sensor.
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

24. Power up since last SYSVAR(24).
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, e.g. 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)

Continued!

SYSVAR, cont'd. *SYSTEM ARRAY*

Remarks, cont'd.

25. Not intended for public use.

Examples

Reading the value of a system variable, in this case the LSS emitter:

```
PRINT SYSVAR(8)
```

Setting the value of a system variable. In this case the LSS emitter isset from the keyboard of the host:

```
10    INPUT "LSS Adjust",A%
20    SYSVAR(8)= A%
```

TESTFEED *STATEMENT*

Field of Application

Performing a formfeed to allow the label stop sensor to adjust itself according to the presently loaded paper web.

Syntax

TESTFEED

Remarks

The TESTFEED statement makes the printer feed out a blank copy (label, ticket or piece of strip according to the setup) while automatically adjusting the label stop sensor or black mark sensor for the paper web presently loaded. This procedure corresponds to the adjustment done when setting up the LSS or Black Mark Sensor (see the *Technical Manual*), but does **not** substitute the type of adjustment done by means of the potentiometer or a SYSVAR instruction.

TESTFEED is useful when switching between various types or brands of print media.

To ascertain that the adjustment will be correctly performed, it is recommended that the TESTFEED statement is issued at least twice. Should the printer still not feed out the paper as expected, readjust the potentiometer as described in the *Technical Manual*.

Example

This program performs a double TESTFEED statement when the key No. 10 (usually marked "F1") on the printer's keyboard is activated:

```
10   ON KEY (10) GOSUB 1000
20   KEY (10) ON
30   GOTO 30
40   END
. . . . .
. . . . .
. . . . .
1000 TESTFEED:TESTFEED
1010 END
```

TICKS *FUNCTION*

Field of Application

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 e.g.:

```
1081287
```

The time which has passed since the printer was started is 10812.87 seconds, i.e. 3 hours 12.87 seconds.

TIME$ VARIABLE

Field of Application **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** 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.

The time is always entered and, by default, returned in the following 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, e.g. 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, e.g. 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$("F") variable can be changed by means of a FORMAT TIME$ 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$ *FUNCTION*

Field of Application

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$_1$>.*

<sexp2> *1s an optional flag "F", indicating that the time will be returned according to the format specified by FORMAT TIME$.*

Remarks

The original time (<sexp>) should always be entered according to the TIME$ format, i.e. 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, e.g. 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 p"
30    A% = -40
40    PRINT TIMEADD$(A%,"F")
RUN
```

yields:

```
01.30.10 pm
```

TIMEDIFF *FUNCTION*

Field of Application

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.*
<sexp$_2$> *is the other of the two moments.*

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, i.e. 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, e.g. 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 *STATEMENT*

Field of Application **Transferring of data files using KERMIT communication protocol.**

Syntax

TRANSFER↔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:", "rs485:", or "uart3:" (default: std IN channel).*

<sexp$_4$> specifies, optionally, the output device as "uart1:", "uart2:", "rs485:", or "uart3:" (default: std OUT channel).

Remarks

Kermit is a protocol for serial binary transfer of a complete file between e.g. a PC and a printer. *Kermit* is included in *DCA Crosstalk*, *Microsoft Windows Terminal* 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).

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 "ram:" 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.

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 STATUS *STATEMENT*

Field of Application **Checking last TRANSFER KERMIT 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.
1: Number of NAK's.
2: ASCII value of last status character.
3: Last error.
4: Block check type used.

<svar> *is a two-element one-dimensional string array where the elements will return:*
0: Type of protocol, i.e. "KERMIT".
1: Last file name received.

Remarks

After a file transfer using the *Kermit* protocol has been performed (see TRANSFER KERMIT statement), 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.

For comprehensive information on *Kermit*, please refer to the reference volume *"Kermit – A File Transfer Protocol"* by Frank da Cruz (Digital Press 1987, ISBN 0-932376-88-6).

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$ *FUNCTION*

Field of Application

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 mS).*

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 *STATEMENT*

Field of Application **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, i.e. the file or device OPENed for input.*
<nexp$_2$>	*is the number of the destination file, i.e. 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 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

Field of Application

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 round brackets (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 *FUNCTION*

Field of Application **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 STATEMENTS

Field of Application

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 , i.e. SYSVAR(18) = -1 (default).*
VERBOFF	*disables all verbosity levels, i.e. SYSVAR (18) = 0.*

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, i.e. verbosity has no meaning in case of the parallel "centronics:" communication protocol.

Other verbosity levels can be selected by means of the system variable SYSVAR(18), and the type of error message by SYSVAR (19).

VERBOFF:
All responses will be suppressed, i.e. 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.

Important:
*If RS 485 is used and the setup option “Prot addr enable” is selected in the printer, the verbosity **must** be turned off (VERBOFF).*

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$ *FUNCTION*

Field of Application

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 version number of the firmware consists of two parts separated by a period (.) character. The first part (e.g. **6**.1) indicates a major upgrading level, whereas the last part (e.g. 6.**13**) indicates a new sublevel caused by minor changes and/or debugging.

The printer family is returned as a 3-digit number:
201 = *EasyCoder 201 II*
401 = *EasyCoder 401*
501 = *EasyCoder 501* or *EasyCoder 401 Linerless*
601 = *EasyCoder 601*

The type of CPU-board is returned as a 1-digit number:
#2 = CPU-board 971,700,28 or later (*EasyCoder 201 II*)
#3 = CPU-board 040,700,28 (*early versions of EasyCoder 501*)
#4 = CPU-board 040,700,29 or 1-040700-30 (*EasyCoder 401/501/601*)

There is no distinction made between various models within the same family, e.g. *Intermec EasyCoder 501*, *501 E* and *501 SA* are indicated by the same number.

Examples

```
PRINT VERSION$(0)
```

yields e.g.:

```
Intermec Fingerprint 6.13
```

```
PRINT VERSION$(1)
```

yields e.g.:

```
201
```

```
PRINT VERSION$(2)
```

yields e.g.:

```
hardware version #2
```

WEEKDAY *FUNCTION*

Field of Application **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, i.e. in the following order:

Year	Last two digits	(e.g. 1998 = 98)
Month	Two digits	(01–12)
Day	Two digits	(01–28|29|30|31)

Example: June 1, 1998 is entered as 980601.

The built-in calendar corrects illegal values for the years 1980 – 2048, e.g. the illegal date 981232 will be corrected to 990101.

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 e.g.:

```
THURSDAY
```

WEEKDAY$ *FUNCTION*

Field of Application **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, e.g. “Friday”.

The date should be entered according to the syntax for the DATE$ variable, i.e. in the order YYMMDD, where:

YY	=	Year	Last two digits	(e.g. 1998 = 98)
MM	=	Month	Two digits	(01–12)
DD	=	Day	Two digits	(01–28|29|30|31)

Example: October 25, 1998 is entered as 981025.

The built-in calendar corrects illegal values for the years 1980 – 2048, e.g. the illegal date 981232 will be corrected to 990101.

Example

This example shows how to make the printer return the name of the weekday as a 3-letter English abbreviation in connection with a formatted date:

```
10   FORMAT DATE$ ", MM/DD/YY"
20   DATE$="981025"
30   NAME WEEKDAY$ 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:

```
Sun, 10/25/98
```

WEEKNUMBER *FUNCTION*

Field of Application **Returning the number of the week for a specified date.**

Syntax

WEEKNUMBER(<sexp>)

<sexp> *is the date for which the week number will be returned.*

Remarks

This function returns the number of the week as a numeric value between 1 – 53.

The date should be entered according to the syntax for the DATE$ variable, i.e. in the following order:

Year	Last two digits	(e.g. 1998 = 98)
Month	Two digits	(01–12)
Day	Two digits	(01–28|29|30|31)

Example: October 25, 1998 is entered as 981025.

The built-in calendar corrects illegal values for the years 1980 – 2048, e.g. the illegal date 981232 will be corrected to 990101.

Examples

In this example the week number for the Christmas eve 1998 is printed on the screen of the host:

```
10    B$="981224"
20    A% = WEEKNUMBER (B$)
30    PRINT A%
RUN
```

yields:

```
52
```

This example shows how the week number for the current date is returned:

```
PRINT WEEKNUMBER (DATE$)
```

yields e.g.:

```
27
```

WHILE...WEND *STATEMENT*

Field of Application

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 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
```

IMAGE TRANSFER PROTOCOLS

The following 5 image transfer file protocols are used in connection with the STORE IMAGE statement and use a common format for the image data, as described 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.*

Continued!

IMAGE TRANSFER PROTOCOLS, cont'd.

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 (i.e. printed upsidedown). 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, i.e. first row of defined data is the first thing out. (This may be somewhat confusing. The only result, if you missinterpretate this, is that your picture will come out upside down.)

Bitmap pattern, bit representation
The bitmap picture is encoded word oriented (ie 16 bits), low byte first. The bits in each byte is read from lsb first (ie 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 (i.e. 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...127 (0..7f hex). To get a row longer then 127, concatenate two rows with zero, e.g. 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...-128 (ff..80 hex).

Continued!

IMAGE TRANSFER PROTOCOLS, cont'd.

Image Format, cont'd.

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 (i.e. 16bit).*
- *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.XXXXXXXXXXXXXXXXX.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
```

Continued!

IMAGE TRANSFER PROTOCOLS, cont'd.

Image Format, cont'd.

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 repeated by the data in row 17.*

```
row  0 XXXXXXXXXXXXXXXXXXXXX       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.XXXXXXXXXXXXXXXXX.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 XXXXXXXXXXXXXXXXXXXXX       0,22,0
    28 ..........X...........       -4,11,1,10,-4
    29 ..........X...........
    30 ..........X...........
    31 ..........X...........
    32 ......XXXXXXXXX.......       7,9,6
```

Continued!

IMAGE TRANSFER PROTOCOLS, cont'd.

UBI10

UBI10 is a combined protocol/file format for image transfer, as opposed to Intelhex and UBI00 –UBI03 protocols described earlier in this chapter. UBI10 is used in the various Intermec Windows Drivers.

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:

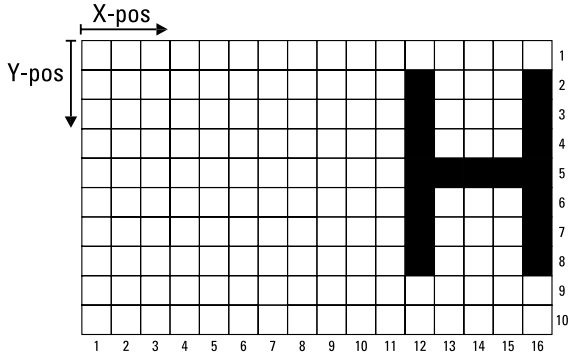
The width of the image in the STORE IMAGE statement should be given as a multiple of 16 bits.

!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.
!Y<pos>A	Set absolute y position <pos>. Default value is 0.
!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.

Continued!

IMAGE TRANSFER PROTOCOLS, cont'd.

UBI10, cont'd.



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, i.e. 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 exeption 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
```

Continued!

IMAGE TRANSFER PROTOCOLS, cont'd.

UBI10, cont'd.

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)*

ROMAN 8 – ASCII decimal Character Set NASC: 1

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

FRENCH National Character Set NASC: 33

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	£	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

SPANISH National Character Set NASC: 34

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	£	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	¾	–	¼	½	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

ITALIAN National Character Set NASC: 39

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	£	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

ENGLISH (UK) National Character Set NASC: 44

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	£	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	¾	–	¼	½	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

SWEDISH (Names) National Character Set NASC: 46

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	¾	–	¼	½	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

NORWEGIAN National Character Set NASC: 47

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

GERMAN National Character Set NASC: 49

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

JAPANESE Latin Character Set (romají) NASC: 81

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

PORTUGUESE National Character Set NASC: 351

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	¶	3/4	–	1/4	1/2	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

PCMAP Character Set NASC: –1

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	•	µ	¶	¾	–	¼	½	ª
250	º	«	■	»	±					

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

ANSI Character Set NASC: –2

ASCII	0	1	2	3	4	5	6	7	8	9
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	space	!	"	#	$	%	&	'
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	ª	«		—		¯	°	±		
180	´	µ	¶	•			º	»	1/4	1/2
190	3/4	¿	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				

- *Characters between ASCII 0 decimal and ASCII 31 decimal are unprintable control characters.*
- *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 above ASCII 127 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.*

BAR CODES

General Information

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 web (ladder style) than along the web (picket fence style). Therefore, to ensure a highly readable printout, we must recommend that you, when printing bar codes with the bars across the web (ladder style), do not use narrow bars less than 3 dots. Also use normal or high speed only. It is possible to use more narrow bars and/or higher speed, but we are not prepared to guarantee the result. You are, of course, free to do your own tests.

No such restrictions apply for bar codes with the bars along the paper web (picket fence style).

Standard Bar Codes

Bar Codes	Designation
Codabar	"CODABAR"
Code 11	"CODE11"
Code 39	"CODE39"
Code 39 full ASCII	"CODE39A"
Code 39 w. checksum	"CODE39C"
Code 93	"CODE93"
Code 128	"CODE128"
DUN-14/16	"DUN"
EAN-8	"EAN8"
EAN-13	"EAN13"
EAN-128	"EAN128"
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"
MSI (modified Plessey)	"MSI"
Plessey	"PLESSEY"
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"

Continued!

BAR CODES, cont'd.

Optional Bar Codes

Bar Codes	Designation
Code 16K	"CODE16K"
Code 49	"CODE49"
LEB	"LEB"
MaxiCode	"MAXICODE"
PDF 417	"PDF417"
Philips	"PHILIPS"
Philips (alternative designation)	"DOT CODE A"
USD5 (Intermec Fingerprint ≥ 5.1)	"USD5"
USD5 (Intermec Fingerprint < 5.1)	"USD5_239"

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.

BAR CODES, cont'd.

EAN 8

BARTYPE: "EAN8"
BARRATIO: Fixed ratio.
BARRATIO statement ignored.
BARMAG: 2 or 3. See note.
BARHEIGHT: No restriction.
BARFONT: Barfont generated automatically.
BARFONT statement ignored.

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

Note:
On a printer fitted with an 8 dots/mm prinhead, this bar code can only be printed within specification when BARMAG 3 is used. However, other magnifications will be acceptable for most applications. This restriction does not apply to printers with a 6 dots/mm or 11.81 dots/mm printhead.

EAN 13

BARTYPE: "EAN13"
BARRATIO: Fixed ratio.
BARRATIO statement ignored.
BARMAG: 2 or 3. See note.
BARHEIGHT: No restriction.
BARFONT: Barfont generated automatically.
BARFONT statement ignored.

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

Note:
On a printer fitted with an 8 dots/mm prinhead, this bar code can only be printed within specification when BARMAG 3 is used. However, other magnifications will be acceptable for most applications. This restriction does not apply to printers with a 6 dots/mm or 11.81 dots/mm printhead.

BAR CODES, cont'd.

UPC-E

BARTYPE: "UPCE"
BARRATIO: Fixed ratio.
BARRATIO statement ignored.
BARMAG: 2 or 3. See note.
BARHEIGHT: No restriction.
BARFONT: Barfont generated automatically.
BARFONT statement ignored.

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

Note:
On a printer fitted with an 8 dots/mm prinhead, this bar code can only be printed within specification when BARMAG 3 is used. However, other magnifications will be acceptable for most applications. This restriction does not apply to printers with a 6 dots/mm or 11.81 dots/mm printhead.

UPC-A

BARTYPE: "UPCA"
BARRATIO: Fixed ratio.
BARRATIO statement ignored.
BARMAG: 2 or 3. See note.
BARHEIGHT: No restriction.
BARFONT: Barfont generated automatically.
BARFONT statement ignored.

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

Note:
On a printer fitted with an 8 dots/mm prinhead, this bar code can only be printed within specification when BARMAG 3 is used. However, other magnifications will be acceptable for most applications. This restriction does not apply to printers with a 6 dots/mm or 11.81 dots/mm printhead.

BAR CODES, cont'd.

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.

BAR CODES, cont'd.

Code 128

BARTYPE: "CODE128"
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[1]
FNC2: ASCII 129 decimal[1]
FNC3: ASCII 130 decimal[1]
FNC4: ASCII 131 decimal[1]
Start characters: Added automatically[2].
Stop character: Added automatically[2].

[1]/. Function characters FNC1–4 require either an 8-bit communication protocol, remapping to an ASCII value between 0–127 dec., or the use of an CHR$ function.

[2]/. The Intermec Fingerprint firmware automatically calculates and inserts the start, code and shift characters that are required to optimize the code according to the Code 128 specifications.

EAN128

BARTYPE: "EAN128"
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: Added automatically[1].
Stop character: Added automatically[1].

[1]/. The Intermec Fingerprint firmware automatically calculates and inserts the start, code and shift characters that are required to optimize the code according to the EAN 128 specifications.

This bar code is identical to Code 128 with the exception that the initial FNC1 function character is generated automatically.

FONTS

Font Designations

At delivery, the printer's firmware contains a number of standard character generators that provide the bitmap fonts. In addition, there is an increasing number of bitmap fonts available on special request. The character generators are derived from scalable typefaces developed by *Bitstream Inc*.

The *Intermec Fingerprint* font designation system is made up by 10 characters that provide information on the characteristics of the font:

VVnnnXYZ.d, where...

VV is a two-character abbreviation of the font name, e.g. SW for Swiss.

nnn is the height of the font matrix in dots incl. ascenders and descenders.

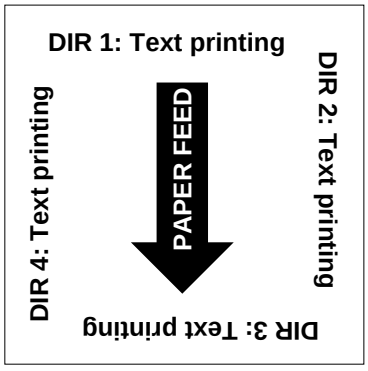
X is the style:
R = Regular (Roman)
I = *Italic*
B = **Bold**
O = ***Bold Italic***

Y is the letter spacing:
S = Proportionally spaced.
M = Monospaced

Z is the character width:
N = Normal
C = Compressed
E = Extended

. is a separating period character between name and extension.

d is an extension, which indicates in which print directions the font can be used:
1 = DIR 1 & DIR 3
2 = DIR 2 & DIR 4



Print directions for text.

Example:

SW030RSN.1 means "**S**wiss, **030** dots high, **R**egular, Proportionally **s**paced, **N**ormal width, Direction **1** & 3".

FONTS, cont'd.

Standard Fonts

Printers using Intermec Fingerprint ≤6.13 **always** contain the two character generators stored in the *Intermec Fingerprint* EPROM:s (IC 1-2 in *Easy-Coder 201 II* or IC 100-101 in *EasyCoder 401/501/601*):

"SW030RSN.1" **"SW030RSN.2"**

In addition to the standard font SW030RSN, the following 18 character generators are included in configuration EPROM's fitted in **standard** *EasyCoder* printers using Intermec Fingerprint ≤6.13 (IC 3-4 in *EasyCoder 201 II* or IC 102-103 in *EasyCoder 401/501/601*):

Monospaced:

MS030RMN.1"	**MS030RMN.2"**
MS050RMN.1"	**MS050RMN.2"**
MS060BMN.1"	**MS060BMN.2"**

Swiss:

"SW020BSN.1"	**"SW020BSN.2"**
"SW050RSN.1"	**"SW050RSN.2"**
"SW060BSN.1"	**"SW060BSN.2"**
"SW080BSN.1"	**"SW080BSN.2"**
"SW120BSN.1"	**"SW120BSN.2"**

OCR:

"OB035RM1.1" "OB035RM1.2"

Stand-Alone printers and printers fitted with some kind of custom-made application program may have other sets of character generators. Please refer to the documentation of the program in question.

Some *Intermec EasyCoder* printers can be fitted with an optional *Scalable Fonts Kit* that allows bitmap fonts to be generated from scalable outline font files in *Speedo* and *TrueType* format.

FONTS, cont'd.

Printout Samples at 6 dots/mm (153.9 dpi)

OB035RM1 **Derived from:** **Bitstream No. 0646** **OCR-B**	abcdefgijklmnopqrsABCDEFGHIJKLMNOPQR The Quick Brown Fox Jumps Over The L
MS030RMN **Derived from:** **Bitstream No. 0596** **Monospace 821 Text™**	abcdefghijkåäöABCDEFGHIJKÄÅÖ 12345#&)= The quick Brown Fox Jumped Over The Lazy
MS050RMN **Derived from:** **Bitstream No. 0596** **Monospace 821 Text™**	abcdefghijkåäABCDEFGHIJK The Quick Brown Fox Jump
MS060BMN **Derived from:** **Bitstream No. 0598** **Monospace 821 Bold/Text™**	abcdefghijåABCDEFGHI The Quick Brown Fox
SW020BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	abcdefghijklmnåäöABCDEFGHIJKLMNÅÄÖ 12345#&)= The Quick Brown Fox Jumped Over The Lazy Dog
SW030RSN **Derived from:** **Bitstream No. 0003** **Swiss 721 Roman™**	***Standard font*** abcdefghijklmnåäöABCDEFGHIJKLMNÅÄÖ 12345#&)= The Quick Brown Fox Jumped Over The Lazy Dog
SW050RSN **Derived from:** **Bitstream No. 0003** **Swiss 721 Roman™**	abcdefghijkåABCDEFGHIJKÅ 12345# The Quick Brown Fox Jumped Over
SW060BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	abcdefghijklåäABCDEFGHIJKL The Quick Brown Fox Jumped
SW080BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	abcdefghijåABCDEFGHI The Quick Brown Fox Ju

Note: Due to the method of reproduction, the printout quality is not representative of what you can expect from your Intermec printer.

Continued!

FONTS, cont'd.

Printout Samples at 6 dots/mm (153.9 dpi), cont'd.

SW120BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	

Note: Due to the method of reproduction, the printout quality is not representative of what you can expect from your Intermec printer.

FONTS, cont'd.

Printout Samples at 8 dots/mm (203.2 dpi)

OB035RM1 **Derived from:** **Bitstream No. 0646** **OCR-B**	abcdefgijklmnopqrsABCDEFGHIJKLMNOPQRS12345#&)= The Quick Brown Fox Jumps Over The Lazy Dog
MS030RMN **Derived from:** **Bitstream No. 0596** **Monospace 821 Text™**	abcdefghijkåäABCDEFGHIJKÅÄ 12345#&)= The quick brown Fox Jumped Over The Lazy Dog
MS050RMN **Derived from:** **Bitstream No. 0596** **Monospace 821 Text™**	abcdefghijkåäABCDEFGHIJKÅÄ 12345#& The Quick Brown Fox Jumped Over Th
MS060BMN **Derived from:** **Bitstream No. 0598** **Monospace 821 Bold/Text™**	**abcdefghijåABCDEFGHIJÅ 1234** **The Quick Brown Fox Jumped**
SW020BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	abcdefghijklmnåäABCDEFGHIJKLMNÅÄO 12345#&)= The Quick Brown Fox Jumped Over The Lazy Dog
SW030RSN **Derived from:** **Bitstream No. 0003** **Swiss 721 Roman™**	***Standard font*** abcdefghijklmnåäABCDEFGHIJKLMNÅÄO 12345#&)= The Quick Brown Fox Jumped Over The Lazy Dog
SW050RSN **Derived from:** **Bitstream No. 0003** **Swiss 721 Roman™**	abcdefghijkåABCDEFGHIJKÅ 12345#& The Quick Brown Fox Jumped Over The
SW060BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	**abcdefghijklääABCDEFGHIJKL** **The Quick Brown Fox Jumped**
SW080BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	**abcdefghijåABCDEFGHIJÅ 1234** **The Quick Brown Fox Jumped**

Note: Due to the method of reproduction, the printout quality is not representative of what you can expect from your Intermec printer.

Continued!

FONTS, cont'd.

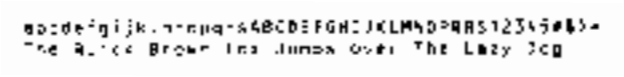
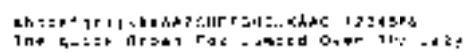
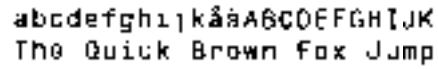
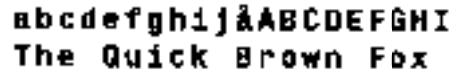
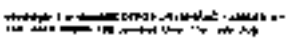
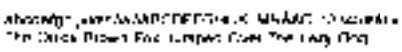
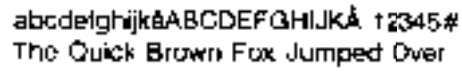
Printout Samples at 8 dots/mm (203.2 dpi), cont'd.

SW120BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	abcdjåABCDJÅ 1234 The Quick Brown Fox

Note: Due to the method of reproduction, the printout quality is not representative of what you can expect from your Intermec printer.

FONTS, cont'd.

Printout Samples at 11.81 dots/mm (300 dpi)

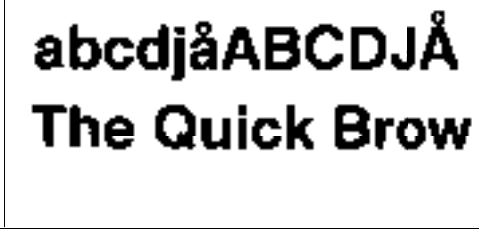
OB035RM1 **Derived from:** **Bitstream No. 0646** **OCR-B**	
MS030RMN **Derived from:** **Bitstream No. 0596** **Monospace 821 Text™**	
MS050RMN **Derived from:** **Bitstream No. 0596** **Monospace 821 Text™**	
MS060BMN **Derived from:** **Bitstream No. 0598** **Monospace 821 Bold/Text™**	
SW020BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	
SW030RSN **Derived from:** **Bitstream No. 0003** **Swiss 721 Roman™**	***Standard font*** 
SW050RSN **Derived from:** **Bitstream No. 0003** **Swiss 721 Roman™**	
SW060BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	
SW080BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	

Note: Due to the method of reproduction, the printout quality is not representative of what you can expect from your Intermec printer.

Continued!

FONTS, cont'd.

Printout Samples at 11.81 dots/mm (300 dpi), cont'd.

SW120BSN **Derived from:** **Bitstream No. 0005** **Swiss 721 Bold™**	

Note: Due to the method of reproduction, the printout quality is not representative of what you can expect from your Intermec printer.

ERROR MESSAGES

Interpretation Table

Code	Message/Explanation	Code	Message/Explanation
0	No error	42	Illegal bar code ratio.
1	Syntax error.	43	Memory overflow.
2	Unbalanced parenthesis.	44	File is write protected.
3	Feature not implemented.	45	Unknown store option.
4	Evaluation syntax error.	46	Store already in progress.
5	Unrecognized token.	47	Unknown store protocol.
6	Tokenized line too long.	48	No store defined.
7	Evaluation stack overflow.	49	NEXT without FOR
8	Error in exectab.	50	Bad store record header.
9	Undefined token.	51	Bad store address.
10	Non-executing token.	52	Bad store record.
11	Evaluation stack underflow.	53	Bad store checksum.
12	Type mismatch.	54	Bad store record end.
13	Line not found.	55	Remove in ROM.
14	Division with zero.	56	Illegal communication channel.
15	Font not found.	57	Subscript out of range.
16	Bar code device not found.	58	Field overflow.
17	Bar code type not implemented.	59	Bad record number.
18	Disk full.	60	Too many strings.
19	Error in file name.	61	Error in setup file.
20	Input line too long.	62	File is list protected.
21	Error stack overflow.	63	ENTER function.
22	RESUME without error.	64	FOR without NEXT
23	Image not found.	65	Evaluation overflow.
24	Overflow in temporary string buffer.	66	Bad optimizing type.
25	Wrong number of parameters.	67	Error from communication channel.
26	Parameter too large.	68	Unknown execution entity.
27	Parameter too small.	69	Not allowed in immediate mode.
28	RETURN without GOSUB	70	Line label not found.
29	Error in startup file.	71	Line label already defined.
30	Assign to a read-only variable.	72	IF without ENDIF.
31	Illegal file number.	73	ENDIF without IF.
32	File is already open.	74	ELSE without ENDIF.
33	Too many files open.	75	ELSE without IF.
34	File is not open.	76	WHILE without WEND.
37	Cutter device not found.	77	WEND without WHILE
38	User break.	78	Not allowed in execution mode.
39	Illegal line number.	79	Not allowed in a layout.
40	Run statement in program.	80	Download timeout
41	Parameter out of range.		

Continued!

ERROR MESSAGES, cont'd.

Interpretation Table, cont'd.

Code	Message/Explanation	Code	Message/Explanation
1001	Not implemented.	1045	Media was removed.
1002	Memory too small.	1046	Memory checksum error.
1003	Field out of label.	1047	Interrupted system call.
1004	Wrong font to chosen direction.	1051	Dot resistance measure out of limits.
1005	Out of paper.	1052	Error in printhead.
1006	No field to print.	1053	Unable to complete a dot measurement.
1007	Lss too high.	1054	Error when trying to write to device.
1008	Lss too low.	1055	Error when trying to read from device.
1009	Invalid parameter.	1056	O_BIT open error.
1010	Hardware error.	1057	File exists.
1011	I/O error.	1058	Transfer ribbon fitted.
1012	Too many files opened.	1059	Cutter does not respond.
1013	Device not found.	1060	DC motor to ribbon save did not start/stop.
1014	File not found.	1061	Wrong type of media.
1015	File is read-only.	1101	Illegal character in bar code.
1016	Illegal argument.	1102	Illegal bar code font.
1017	Result too large.	1103	Too many characters in bar code.
1018	Bad file descriptor.	1104	Bar code too large.
1019	Invalid font.	1105	Bar code parameter error.
1020	Invalid image.	1106	Wrong number of characters.
1021	Too large argument for MAG.	1107	Illegal bar code size.
1022	Head lifted.	1108	Number or rows out of range.
1023	Incomplete label.	1109	Number of columns out of range.
1024	File too large.	1201	Insufficient font data loaded.
1025	File does not exist.	1202	Transformation matrix out of range.
1026	Label pending.	1203	Font format error.
1027	Out of transfer ribbon.	1204	Specifications not compatible with output module.
1028	Paper type is not selected.	1205	Intelligent transform not supported.
1029	Printhead voltage too high.	1206	Unsupported output mode requested.
1030	Character is missing in chosen font.	1207	Extended font not supported.
1031	Next label not found.	1208	Font specifications not set.
1032	File name too long.	1209	Track kerning data not available.
1033	Too many files are open.	1210	Pair kerning data not available.
1034	Not a directory.	1211	Other Speedo error.
1035	File pointer is not inside the file.	1212	No bitmap or outline device.
1036	Subscript out of range.	1213	Speedo error six.
1037	No acknowledge received within specified timeout.	1214	Squeeze or clip not supported.
1038	Communication checksum error.	1215	Character data not available.
1039	Not mounted.	1301	Index outside collection bounds.
1040	Unknown file operating system.	1302	Collection could not be expanded.
1041	Error in fos structure.	1303	Ptr ptr is not a collection.
1042	Internal error in mcs.	1304	Item not a member of the collection.
1043	Timer table full.	1305	No compare function, or compare returns faulty value.
1044	Low battery in memory card.	1306	Tried to insert a duplicate item.