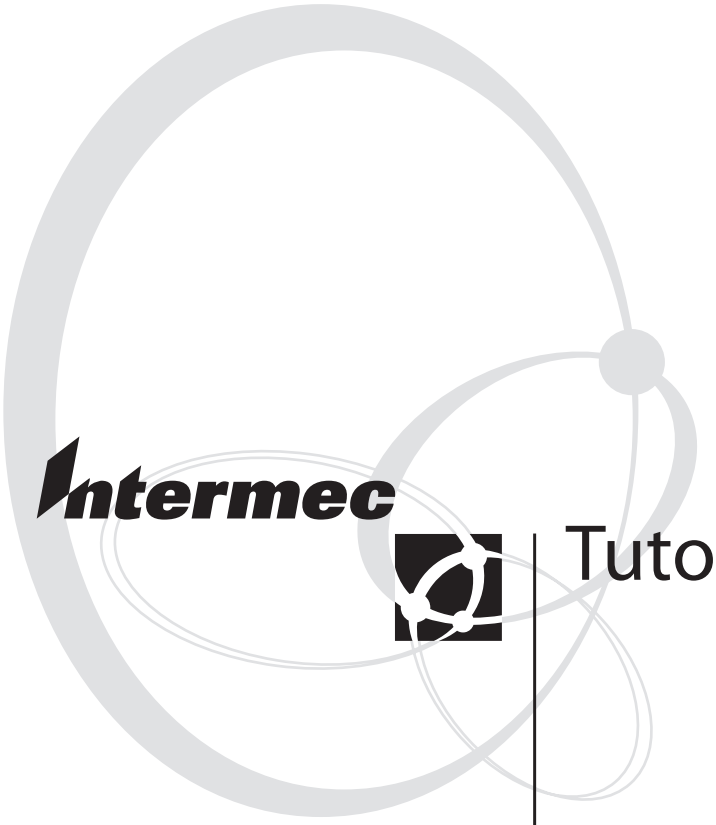


Tutorial

Intermec Fingerprint® v8.00

Intermec Printer AB

Idrottsvägen 10

P.O. Box 123

S-431 22 Mölndal

Sweden

Service support: +46 31 869500

The information contained herein is proprietary and is provided solely for the purpose of allowing customers to operate and/or service Intermec manufactured equipment and is not to be released, reproduced, or used for any other purpose without written permission of Intermec.

Information and specifications in this manual are subject to change without notice.

© 2003 by Intermec Printer AB

All Rights Reserved

EasyCoder, EasyLAN, Fingerprint, and LabelShop are registered trademarks of Intermec Technologies Corp. The word Intermec, the Intermec logo, InterDriver, and PrintSet are trademarks of Intermec Technologies Corp.

Bitstream is a registered trademark of Bitstream, Inc.

Crosstalk and DCA are registered trademarks of Digital Communications Associates, Inc.

The name Centronics is wholly owned by GENICOM Corporation.

HyperTerminal is a trademark of Hilgraeve, Inc.

IBM is a registered trademark of International Business Machines Corporation.

Intel is a registered trademark of Intel Corporation.

Macintosh and TrueType are registered trademarks of Apple Computer, Inc.

Microsoft, MS, and MS-DOS are registered trademarks of Microsoft Corporation.

TrueDoc is a registered trademark of Bitstream, Inc.

Unicode is a trademark of Unicode Inc.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Windows is a trademark of Microsoft Corporation.

Throughout this manual, trademarked names may be used. Rather than put a trademark (™) symbol in every occurrence of a trademarked name, we state that we are using the names only in an editorial fashion, and to the benefit of the trademark owner, with no intention of infringement.

Contents

Introduction viii

1 Getting Started

1.1 Computer Connection 2
1.2 Check Media Supply 2
1.3 Switch On the Printer 3
1.4 Intermec Shell Startup Program 3
1.5 No Startup Program 3
1.6 Custom-Made Startup Program 3
1.7 Breaking a Startup Program 4
1.8 Communications Test 5

2 Creating a Simple Label

2.1 Introduction 8
2.2 Printing a Box 8
2.3 Printing an Image 8
2.4 Printing a Bar Code 9
2.5 Printing Human Readables 9
2.6 Printing Text 10
2.7 Listing the Program 10
2.8 Changing a Program Line 10
2.9 Saving the Program 11
2.10 Error Handling 11
2.11 Renumbering Lines 11
2.12 Merging Programs 12
2.13 Using the Print Key 12

3 Terminology and Syntax

3.1 Lines 16
3.2 Keywords 17
3.3 Statements 17
3.4 Functions 17
3.5 Other Instructions 17
3.6 Expressions 18
3.7 Constants 18
3.8 Variables 18
3.9 Keyword List 19
3.10 Operators 20
Arithmetic Operators 20
Relational Operators 20
Logical Operators 21
3.11 Devices 22

4 Fingerprint Programming

4.1 Introduction 24
4.2 Editing Methods: 24
Line-by Line Method 24
Copy & Paste Method 24
Send Text Method 25
4.3 Immediate Mode 25

4.4 Programming Mode 27
Programming with Line Numbers 27
Programming without Line Numbers 28
Programming Instructions 29
4.5 Conditional Instructions 30
4.6 Unconditional Branching 31
4.7 Branching to Subroutines 32
4.8 Conditional Branching 33
4.9 Loops 37
4.10 Program Structure 49
4.11 Execution 40
4.12 Breaking the Execution 42
4.13 Saving the Program 44
Saving in Printer 44
Naming the Program 44
Protecting the Program 45
Saving Without Line Numbers 45
Making Changes 45
Making a Copy 45
Renaming a Program 46
Saving in DOS-formatted CompactFlash Memory Cards 46
Creating a Startup Program 46
4.14 Rebooting the Printer 48

5 File System

5.1 Printer's Memory 50
Permanent memory ("/rom" and "/c") 50
Temporary Memory ("tmp:") 51
DOS-formatted CompactFlash Memory Cards ("card1:") 52
Non DOS-formatted CompactFlash Memory Cards ("/rom") 52
Other Memory Devices ("storage:") 52
Current Directory 52
Checking Free Memory 52
Providing More Free Memory 53
Formatting the Permanent Memory 53
Formatting CompactFlash Memory Cards 53
5.2 Files System with Directories 54
5.3 Files 55
File Types 55
File Names 55
Listing Files 55
5.4 Program Files 56
Program File Types 56
Instructions 56
5.5 Data Files 57
Data File Types 57
Instructions 57
5.6 Image Files 57
5.7 Font Files 58
5.8 Transferring Text Files 58
5.9 Transferring Binary Files 58
5.10 Transferring Files Between Printers 59
5.11 Arrays 60

6 Input to Fingerprint

6.1 Standard I/O Channel....64
6.2 Input From Host (Std IN Channel only)....64
6.3 Input From Host (Any Channel)....64
6.4 Input From a Sequential File....64
6.5 Input From a Random File....68
6.6 Input From Printer's Keyboard....70
6.7 Communication Control....72
6.8 Background Communication....74
6.9 RS-422 Communication....79
6,10 RS-485 Communication....80
6.11 External Equipment....85
Industrial Interface....85

7 Output from Fingerprint

7.1 Output to Std Out Channel....88
7.2 Redirecting Output from Std Out Channel to File....90
7.3 Output and Append to Sequential Files....91
7.4 Output to Random Files....93
7.5 Output to Communication Channels....96
7.6 Output to Display....96

8 Data Handling

8.1 Preprocessing Input Data....98
8.2 Input Data Conversion....101
8.3 Date and Time....105
8.4 Random Number Generation....108

9 Label Design

9.1 Creating a Layout....110
Field Types....110
Origin....111
Coordinates....111
Units of Measure....111
Insertion Point....111
Alignment....112
Directions....114
Partial Fields....114
XOR Mode....114
Layout Files....115
Checking Current Position....115
Checking the Size and Position of a Field....115
9.2 Single-Line Text Field....116
9.3 Multi-Line Text Field....117
9.4 Bar Code Field....119
9.5 Image Field....121
9.6 Box Field....122
9.7 Line Field....123
9.8 Layout Files....124
Introduction....124
Creating a Layout File....124
Creating a Logotype Name File....127
Creating a Data File or Array....128

Creating an Error File and Array....129
Using the Files in a LAYOUT statement....130

10 Printing Control

10.1 Media Feed....132
10.2 Printing....134
10.3 Length of Last Feed Operation....136
10.4 Batch Printing....137

11 Fonts

11.1 Font Types....142
11.2 Single-byte Fonts....142
11.3 Double-byte Fonts....142
11.4 Font Direction, Size, Slant, and Width....143
11.5 Standard Fonts....143
11.6 Old Font Formats....143
11.7 Adding Fonts....144
11.8 Listing Fonts....144
11.9 Removing Fonts....144
11.10 Font Aliases....145

12 Bar Codes

12.1 Standard Bar Codes....148
12.2 Setup Bar Codes....148

13 Images

13.1 Images vs Image Files....150
13.2 Standard Images....150
13.3 Downloading Image Files....150
13.4 Listing Images....151
13.5 Removing Images and Image Files....151

14 Printer Function Control

14.1 Keyboard....154
Controlling the Printer in the Setup and Immediate Modes....154
Enabling the Keys....154
Key Id. Numbers....155
Key-initiated Branching....155
Audible Key Response....155
Input from Printer's Keyboard....156
Remapping the Keyboard....156
14.2 Display....158
Output to Display....158
Cursor Control....159
14.3 LED Control Lamps....161
14.4 Beeper....162
14.5 Clock/Calendar....163
14.6 Printer Setup....164
Reading Current Setup....164
Creating a Setup File....164
Changing the Setup using a Setup File....165
Changing the Setup using a Setup String....165

Saving the Setup 165
14.7 System Variables 166
14.8 Printhead 168
14.9 Transfer Ribbon 170
14.10 Version Check 170
14.11 Status and Std I/O Check 171

15 Error Handling

15.1 Standard Error-Handling 174
Error Messages 174
15.2 Tracing Programming Errors 175
15.3 Creating an Error-Handling Routine 176
15.4 Error-handling program 178
ERRHAND.PRG Utility Program 178
Listing of ERRHAND.PRG Utility Program 180
Extensions to ERRHAND.PRG Utility Program 183

16 Reference Lists

16.1 Instructions in Alphabetic Order 186
16.2 Instructions by Field of Application 193

17 Keyboards

17.1 Position Numbers 200
17.2 Id. Numbers 202
17.3 ASCII Values 204

Introduction

Intermec Fingerprint v8.00 is a new version of a well-known 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 v8.00 works only with the latest generation of printers, that is, presently the EasyCoder PF2/4i-series and EasyCoder PM4i.

The Intermec Fingerprint firmware is an easy-to-use, intelligent programming tool for label formatting and printer customizing, which allows you to design your own label formats and write your own printer application software. You may easily create a printer program by yourself that exactly fulfils your own unique requirements. Improvements or changes due to new demands can be implemented quickly and without vast expenses.

This tutorial manual describes how to start up Fingerprint programming and how to use the various instructions in their proper context. Programming instructions are only briefly explained.

The *Intermec Fingerprint v8.00, Programmer's Reference Manual* contains comprehensive information on all programming instructions in the Fingerprint programming language in alphabetic order. It also contains other types of program-related information that are common for all printer models from Intermec that uses the corresponding version of Intermec Fingerprint.

The Intermec Direct Protocol v8.00 is a subset of Intermec Fingerprint and is used for combining variable input data from a host with predefined label layouts. It is described in a separate manual called the *Intermec Direct Protocol v8.00, Programmer's Reference Manual*.

All information needed by the operator, like how to run the printer, how to load the media and ribbon supply, and how to maintain the printer, can be found in the *User's Guide* for the printer model in question. The User's Guide for each printer model also provides information on installation, setup, density, media specifications, positioning, and other technical data, which are specific for the printer model in question.

All instructions directly applying to the EasyLAN interface have been excluded from this manual but are available in the *EasyLAN User's Guide*.

A large number of Intermec and third-party software supports or are based on Intermec Fingerprint v8.00, such as Intermec Shell, Intermec LabelShop, Intermec Fingerprint Application Builder (IFAB), Intermec InterDriver, Intermec Printer Network Manager, and Intermec PrintSet.

1 Getting Started

This chapter describes how to start up the printer with Intermec Fingerprint and check if the communication between printer and host is working.

1.1 Computer Connection

The Intermec Fingerprint firmware is stored in a Flash SIMM on the printer's CPU board. No floppy disks or operating system, like MS-DOS or UNIX, is required. The printer only needs to be connected to an AC supply.

Unless the printer is fitted with a program that allows it to be used independently ("stand-alone"), you must also connect it to some kind of device, which can transmit characters in ASCII format. It can be anything from a non-intelligent terminal to a mainframe computer system.

For programming the printer, you need a computer with a screen and an alphanumeric keyboard, that provides two-way serial communication, preferably using RS-232 (for example a PC with Microsoft Windows 2000 or XP). Use a text editor like Windows Notepad for writing programs and a terminal program like Windows HyperTerminal for communication with the printer.

Note: Although most examples in this manual assume a host running MS Windows 2000, other operating systems can also be used, for example Windows 3.1x, Windows 95/98, Windows Me, Windows NT, Windows XP, DOS, Macintosh OS, OS-2, UNIX, etc, as long you have a terminal program that can communicate with the printer and some kind of text editor.

Connect the printer and host as described in the User's Guide for the printer model in question. If the printer has several communication ports, it is recommended to use the serial port "uart1:" for programming. This port is by default is set up for RS-232. Other optional serial communication ports could also be used.

It is possible to set up the printer's communication protocol to fit the host computer. However, until you have become familiar with the Intermec Fingerprint concept, it may be easier to adapt the host to the printer's default setup parameters:

Default communication setup on "uart1:"

- Baud rate: 9600
- Character length: 8
- Parity: None
- No. of stop bits: 1
- Flow control: Disabled
- New line: CR/LF (Carriage Return + Line Feed)

1.2 Check Media Supply

Check that the printer has an ample supply of media and, when applicable, of thermal transfer ribbon. Refer to the User's Guide for loading instructions.

1.3 Switch On the Printer

Check that the printhead is lowered. Switch on the power using the On/Off switch, which is fitted on the printer's rear plate and check that the "Power" control lamp comes on. Then watch the display window. What happens next depends on what kind of startup file there is in the printer.

Make sure that any paper cutter is locked in closed position. In some printer models, the cutter may be activated when the power is switched on!

After a short while, when the printer has performed certain self-diagnostic tests and loaded the startup program, a countdown menu will usually be displayed:

```
ENTER=SHELL
5 sec.     v.8.0
```

1.4 Intermec Shell Startup Program

The countdown menus indicate that the printer is fitted with the Intermec Shell startup program. Wait until the 5 seconds countdown is completed. Then, by default, this menu will be displayed:

```
Fingerprint
8.00
```

This or similar messages indicates that the printer has entered the immediate mode of Intermec Fingerprint, where you can start your programming. Please proceed at Chapter 1.8.

If the Shell countdown menus are shown, but are followed by any other message than "Intermec Fingerprint 8.00", some other application has already been selected in Shell. Refer to the User's Guide for information on how to select the Fingerprint option.

1.5 No Startup Program

If the printer is not fitted with any startup program at all, the display window should show the following message directly after power-up:

```
Fingerprint
8.00
```

This means that the printer has entered the immediate mode of Intermec Fingerprint. Proceed at Chapter 1.8.

1.6 Custom-Made Startup Program

If any other kind of message is displayed than those illustrated above, the printer is provided with some kind of custom-made startup program, which you must break before you can start programming.

1.7 Breaking a Startup Program

Default Method (break from keyboard)

- Press the <**Shift**> key and keep it pressed down while also pressing the <**Pause**> key.

Other Methods

- The program may be provided with other means for breaking the program, for example by sending a certain character from the host or by pressing another key or combination of keys. Break from keyboard may also be disabled completely.

When a break interrupt has been executed and you have entered the immediate mode, there will be no change in the printer's display, but a message should appear on the screen of the host, provided you have a working two-way communication:

```
User break in line XXXX
```

How to go on

- If you have succeeded in breaking the program, proceed at Chapter 1.8.

1.8 Communications Test

Check that you have entered the immediate mode and have a working two-way serial communication by sending a simple instruction from the host to the printer. On the keyboard of the host, type:

```
? VERSION$ ↵                    ( ↵ = Carriage Return key)
```

The printer should respond immediately by returning the version of the installed Intermec Fingerprint firmware to the screen of the host, for example:

```
Fingerprint 8.00
Ok
```

This indicates that the communication is working both ways.

If the communication does not work, switch off the printer and check the connection cable. Also check if the communication setup in the host corresponds to the printer's setup and if the connection is made between the correct ports. Check the verbosity level as described in Chapter 6.7. Then try the communication test again.

Another possible cause of error may be that another communication channel than"auto" or "uart1:" has been selected for Fingerprint in Intermec Shell. Reselect the Fingerprint application for "auto" in Shell as described in the User's Guide.

Once you know that the communication is working, you may go on and make the printer auto-adjust its media feed according to the type of labels loaded. Simultaneously press the <**Shift**> and <**Feed**> keys on the printer's built-in keyboard. The printer will feed out at least one blank label (or the corresponding).

Finally, send a line of text to make sure that characters transmitted from the host are interpreted as expected by the printer's firmware:

```
FONT "Swiss 721 BT" ↵
PRTXT "ABCDEFGHIJKLM" ↵
PRINTFEED ↵
```

Each line will be acknowledged by an "Ok" on the screen, provided that the line has been entered correctly, that there is a working two-way serial communication, and that the verbosity is on. When you press the "Carriage Return" key the third time, the printer will feed out a label, ticket, tag or piece of strip with the text printed near the lower left corner of the printable area.

Try using other characters between the quotation marks in the second line, especially typical national characters like ÅÄÖÜ¿¢•ç etc. Should any unexpected characters be printed, you may need to select another character set, see NASC statement in Chapter 8.1, or switch from 7-bit to 8-bit communication.

2 Creating a Simple Label

This chapter describes how to use Intermec Fingerprint to create a program for printing a simple label, how to modify and save the program, and how to print a copy of the label. It is intended to provide a quick introduction to the main principles of Fingerprint programming and to give a basis for the more elaborate explanations that follow.

For detailed informations on individual instructions, please refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

Note: The examples in this chapter are designed to be run on any present Intermec Fingerprint v8.00-compatible EasyCoder printer connected to a terminal or computer and loaded with media (preferably labels) according to the following specifications (see the User's Guide for media size restrictions).

Label size (minimum):

Width:	58.4 mm	(2.30 inches)
Length:	70.0 mm	(2.75 inches)

2.1 Introduction

Use a simple text editor, for example Windows Notepad, to enter the program lines. Use single space characters to separate the line numbers from the instructions that follow on the same line. Finish each line with a carriage return character (not illustrated in the examples in the remaining part of the manual.

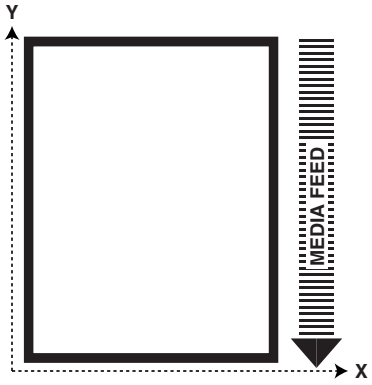
When you have entered a batch of program lines, copy the lines and paste them into a communication program, for example Windows HyperTerminal, which is connected to the printer (see Chapter 1.1).

Note: The printer will not execute the program until you have entered RUN + Carriage Return.

2.2 Printing a Box

Let us start by printing a box 430 dots high and 340 dots wide with a line thickness of 15 dots. The box is inserted at position X=10; Y=10:

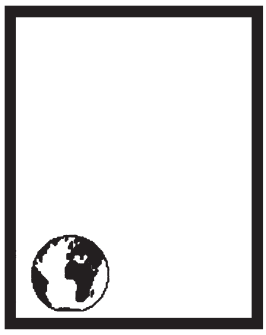
```
NEW
10 PRPOS 10,10
20 PRBOX 430,340,15
200 PRINTFEED
300 END
RUN
```



2.3 Printing an Image

Now we add the image “GLOBE.1” after changing the position coordinates to X=30,Y=30.

```
30 PRPOS 30,30
40 PRIMAGE "GLOBE.1"
RUN
```



2.4 Printing a Bar Code

Before you print a bar code, you need to choose a bar code type. Note there is no blank space in the bartype name.

```
50 PRPOS 75,270
60 BARTYPE "CODE39"
70 PRBAR "ABC"
RUN
```

2.5 Printing Human Readables

To get human readable text printed under the bar code, add these lines:

```
1 BARFONT ON
2 BARFONT "Swiss 721 BT", 6
RUN
```

2.6 Printing Text

Add a line of text at position X=25,Y=220:

```
80 PRPOS 25,220
90 FONT "Swiss 721 BT", 6
100 PRTXT "My FIRST label"
RUN
```

2.7 Listing the Program

To view the whole program, type:

```
LIST
```

The lines will be listed in ascending order on the host screen:

```
1 BARFONT ON
2 BARFONT "Swiss 721 BT", 6
10 PRPOS 10,10
20 PRBOX 430,340,15
30 PRPOS 30,30
40 PRIMAGE "GLOBE.1"
50 PRPOS 75,270
60 BARTYPE "CODE39"
70 PRBAR "ABC"
80 PRPOS 25,220
90 FONT "Swiss 721 BT", 6
100 PRTXT "My FIRST label"
200 PRINTFEED
300 END
ok
```

2.8 Changing a Program Line

If you want to change a program line, simply rewrite the line using the same line number. For example, move the text to the right by rewriting line number 80 with new coordinates:

```
80    PRPOS 75,220
RUN
```

2.9 Saving the Program

If you want to save your first attempt of Fingerprint programming, issue the following instruction:

```
SAVE "LABEL1"
```

Your program will be saved in the printer's memory under the name:

```
LABEL1.PRG
```

2.10 Error Handling

The program above is very simple and there is a very small risk of encountering any errors. When writing more complex programs, you might find use for an errorhandler. For that purpose we have included a program called ERRHAND.PRG in the firmware. Should your printer not contain any errorhandling program, you will find ERRHAND.PRG listed in Chapter 15.4.

ERRHAND.PRG contains subroutines that displays the type of error on the printer's LCD display (for example "Out of paper" or "Head lifted"), prints the error number on your screen, and assigns subroutines to some of the keys on the keyboard. There is also a subroutine that performs a PRINTFEED with error-checking. The ERRHAND.PRG occupies lines 10, 20, and 100000-1900000.

2.11 Renumbering Lines

If ERRHAND.PRG is merged with the program you just wrote, lines 10 and 20 in your program will be replaced with lines 10 and 20 from ERRHAND.PRG. Therefore you have to renumber your program, so that your program begins with an unoccupied number, for example 50, before ERRHAND.PRG is merged:

```
RENUM 50,1,10
Ok
```

```
LIST
50 BARFONT ON
60 BARFONT "Swiss 721 BT",6
70 PRPOS 10,10
80 PRBOX 400,340,15
90 PRPOS 30,30
100 PRIMAGE "GLOBE.1"
110 PRPOS 75,270
120 BARTYPE "CODE39"
130 PRBAR "ABC"
140 PRPOS 75,220
150 FONT "Swiss 721 BT",6
160 PRTXT "My FIRST label"
170 PRINTFEED
180 END
ok
```

2.12 Merging Programs

Now your label-printing program LABEL1.PRG will not interfere with ERRHAND.PRG and you can merge the two programs into a single program. In fact, you will create a copy of ERRHAND.PRG which is merged into LABEL1.PRG. Thus the original ERRHAND.PRG can be merged into more programs later:

```
MERGE "/rom/ERRHAND.PRG"
```

2.13 Using the Print Key

Instead of using a PRINTFEED statement, we will use a subroutine in ERRHAND.PRG. Because ERRHAND.PRG assigns functions to for example the <**Print**> key, you can create a loop in the program so you will get a label every time you press the <**Print**> key.

```
160   GOSUB 500000
170   GOTO 170
RUN
```

Try pressing different buttons on the printer's keyboard. Only those, to which functions been assigned in ERRHAND.PRG (<**Pause**>, <**Print**>, <**Setup**>, and <**Feed**> keys) will work.

You can break the program by simultaneously pressing the <Shift> and <Pause> keys.

Save the program again using the same name as before:

```
SAVE "LABEL1"
```

The previously saved program "LABEL1.PRG" will be replaced by the new version.

With this example, we hope you have gotten a general impression of the basic methods for Intermec Fingerprint programming and that you also see the advantages of using ERRHAND.PRG or a similar program for errorhandling and initiation.

ERRHAND.PRG can easily be modified to fit into more complex programs and we recommend that you use it when writing your programs until you feel ready to create errorhandling programs yourself. Also see Chapter 15 “Error-Handling.”

3 Terminology and Syntax

This chapter explains the terminology used for describing Intermec Fingerprint and explains the syntax used for Fingerprint instructions.

3.1 Lines

You will always use one or several lines to give the instructions to the printer, regardless whether you work in the immediate mode, in the programming mode, or in the Intermec Direct Protocol. The difference is that in the programming mode, the lines are always numbered (visibly or invisibly), whereas in the immediate mode and the Intermec Direct Protocol, they must not be numbered.

A line may contain up to 32,767 characters. A line must always be terminated by a Carriage Return character (ASCII 13 decimal).

Note: If you enter a carriage return on the host, the printer will, by default, echo back a Carriage Return + a Line Feed (ASCII 13 + 10 decimal). Using the setup option "New Line", you may optionally restrict the printer only to echo back either a Carriage Return (ASCII 13 decimal) or a Line Feed (ASCII 10 decimal).

When the line reaches the right edge of the screen of the host, it will usually wrap to the next screen line.

Theoretically, line numbers up to > 2 billion can be used. If you choose to enter the line numbers manually, start by numbering the lines from 10 and upwards with an increment of 10 (10, 20, 30, 40, etc.) That makes it possible to insert additional lines (for example 11,12,13...etc.), when the need arises. However, the line numbers are your own decision, since you must type them yourself.

You can also omit line numbers when you create the program and let Fingerprint provide line numbers automatically. Such line numbers will be invisible until the program is listed.

After having typed the line number, use a space character to separate it from the statement or function that follows. That makes it easier to read the program without having to list it.

Several instruction may be issued on the same line, provided they are separated by colons (:), for example:

```
100 FONT "Swiss 721 BT":PRTXT "HELLO"
```

This is especially useful in the immediate mode (see Chapter 4.3) and in the Intermec Direct Protocol, where you can send a complete set of instructions as a single line, for example:

```
PP100,250:FT"Swiss 721 BT":PT"Text 1":PF ↵
```

It is not possible to alter a line after it has been transmitted to the printer. If you want to change such a line, you must send the whole line again using the same line number, or delete it using a DELETE statement (see Chapter 4.4).

3.2 Keywords

Intermec Fingerprint instructions can always be entered as lowercase characters and in most instances also as uppercase characters. An instruction consists of a keyword and optionally some additional parameters, flags, or input data. Some keywords can be used in an abbreviated form, for example PT instead of PRTXT. You may use a blank space to separate the keyword from the rest of the instruction, which must be entered exactly according to the syntax description in the *Intermec Fingerprint v8.00, Programmer's Reference Manual*. Note that in some cases, a space character is a compulsory part of the keyword, as in LINE↔INPUT. (↔ indicates a compulosy space character). Refer to Chapter 3.9 for a list of all keywords.

3.3 Statements

A statement is an instruction, which specifies an operation. It consists of a keyword, usually followed by one or several parameters, flags, or input data, which further define the statement.

Examples:

```
PRTXT "HELLO"                    Keyword input data
ON BREAK 1 GOSUB 1000            Keywords with parameters
FILES "tmp:",A                   Keyword with parameter and flag
```

3.4 Functions

A function is a procedure, which returns a value. A function consists of a keyword combined with values, flags, and/or operators enclosed by parentheses ( ). Operators will be explained later on.

Examples:

```
CHR$(65)                         Keyword with parameter
TIME$("F")                       Keyword with flag
ABS(20*5)                        Keyword with arithmetic operator (*) and values
IF(PRSTAT AND 1)...              Keywords, logical operator, and value
```

A function can be entered inside a statement or on a line containing other instructions. They are often used in connection with conditional statements, for example:

```
320 IF (PRSTAT AND 1) THEN GOTO 1000
```

Blank spaces may be inserted to separate the function from other instructions and also to separate the keyword from the rest of the statement.

3.5 Other Instructions

In addition to statements and functions, there are a few other types of specialized instructions, such as the DATE$ and TIME$ variables, and the SYSVAR system array, which do not fit into the above-mentioned categories.

3.6 Expressions

In the descriptions of the syntax for the various instructions, the word “Expression” is used to cover both constants and variables.

Expressions are of two kinds:

- String expressions are carriers of alphanumeric text, that is string constants and string variables.
- Numeric expressions contain numeric values, numeric variables and operators only, that is numeric constants and numeric variables

3.7 Constants

Constants are fixed text or values. There are two kinds:

- String constants are sequences of characters (text). If digits or operators are included, they will be considered as text and will not be processed. String constants must always be enclosed by quotation marks (ASCII 34 decimal), for example "TEST.PRG". (If the string constant comes last on a line, the appending quotation mark can be excluded.)
- Numeric constants are fixed numeric values. Only decimal integers are allowed (1, 2, 3, 4, 5 etc.). Values are assumed to be positive unless they are preceded by a minus sign (-). Positive values may optionally be indicated by a leading plus sign (+), whereas a leading minus sign (-) is compulsory for negative values.

Note: Certain characters, for example digits, can be either string constants (text) or numeric constants (numbers). To allow the firmware to detect that difference, string constants must always be enclosed by quotation marks (" ....."), as opposed to numeric constants.

3.8 Variables

Variables are value holders. There are two main types:

- String variables are used to store strings entered as string constants or produced by Fingerprint instructions. Max. size is 64 kbytes (65,535 characters). String variables are indicated by a trailing $ sign. Examples:

```
A$ = "INTERMEC"
A$="INTERMEC"
B$ = TIME$
LET C$ = DATE$
```

- Numeric variables are used to store numbers, entered as numeric constants, or produced by Fingerprint instructions or operations. Numeric variables are indicated by a trailing % sign. Max. value is 2,147,483,647. Examples:

```
A% = 150
B% = DATEDIFF("031201","031230")
LET C% = 2^2
```

The name of a variable may consist of letters, numbers, and decimal points. The first character must always be a letter. No keywords or keyword abbreviations must be used. However, completely embedded keywords are allowed. Examples:

`LOC`	is a keyword
`CLOCK$ = "ABC"`	is OK
`LOC$ = "ABC"`	causes an error
`LOCK$ = "ABC"`	causes an error

The current keywords and keywords reserved for future program enhancement are listed below.

3.9 Keyword List

#	BH	ETUPLE	INT	PB	SETASSOC
'	BLINK	EXECUTE	INVIMAGE	PEC2DATA	SETPFSVAR
(	BM	FF	IP	PEC2LAY	SETSTDIO
)	BR	FIELD	KEY	PECTAB	SETUP
*	BREAK	FIELDNO	KEYBMAP$	PF	SGN
+	BT	FILE&	KILL	PL	SORT
,	BUFFER	FILENAME$	LAYOUT	PM	SOUND
-	BUSY	FILES	LBLCOND	PORTIN	SPACE$
/	CHDIR	FIX	LED	PORTOUT	SPLIT
\	CHECKSUM	FLOTCALC$	LEFT$	PP	STDIO
:	CHR$	FONT	LEN	PRBAR	STEP
;	CLEANFEED	FONTD	LET	PRBOX	STOP
<	CLEAR	FONTDSIZE	LINE INPUT	PRBUF	STORE
<=	CLIP	FONTDSLANT	LIST	PRESCALE	STR$
<>	CLL	FONTNAME$	LISTPFSVAR	PRIMAGE	STRING$
=	CLOSE	FONTS	LOAD	PRINT	SYSTEM
=<	COM ERROR	FONTSIZE	LOC	PRINT USING	SYSVAR
=>	COMBUF$	FONTSLANT	LOF	PRINTFEED	TESTFEED
>	COMSET	FOR	LSET	PRINTONE	THEN
><	COMSTAT	FOR APPEND AS	LTS&	PRLINE	TICKS
>=	CONT	FOR INPUT AS	MAG	PRPOS	TIME$
?	COPY	FOR OUTPUT AS	MAKEASSOC	PRSTAT	TIMEADD$
"	COUNT&	FORMAT	MAP	PRTXT	TIMEDIFF
^	CSUM	FORMAT$	MERGE	PT	TO
:	CURDIR$	FORMFEED	MID$	PUT	TRANSFER
;	CUT	FRE	MKDIR	PX	TRANSFER$
ABS	DATA	FT	MOD	RANDOM	TRANSFERSET
ACTLEN	DATAIN	FUNCTEST	MODE	RANDOMIZE	TROFF
ALIGN	DATE$	FUNCTEST$	NAME	READ	TRON
AN	DATEADD$	GET	NASC	READY	VAL
AND	DATEDIFF	GETASSOC$	NASCD	REBOOT	VERBOFF
AS	DBBREAK	GETASSOCNAME$	NEW	REDIRECT OUT	VERBON
ASC	DBEND	GETPFSVAR	NEXT	REM	VERSION$
BARADJUST	DBSTDIO	GOSUB	NI	REMOVE	WEEKDAY
BARCODENAME$	DBSTEP	GOTO	NORIMAGE	RENDER	WEEKDAY$
BARFONT	DELETE	HEAD	NOT	RENUM	WEEKNUMBER
BARFONTD	DELETEPFSVAR	HEX$	OFF	RESTORE	WEND
BARFONTDSIZE	DEVICES	HOLIDAY$	OFF LINE	RESUME	WHILE
BARFONTDSLANT	DIM	IF	ON	RESUME HTTP	WRITE
BARFONTSIZE	DIR	II	ON BREAK	RESUME NEXT	XOR
BARFONTSLANT	DIRNAME$	IMAGE	ON COMSET	RETURN	XORMODE
BARHEIGHT	ELSE	IMAGENAME$	ON ERROR GOTO	RIBBON	XYZZY
BARMAG	END	IMAGES	ON HTTP GOTO	RIGHT$	
BARRATIO	EOF	IMMEDIATE	ON KEY	RND	
BARSET	ERL	INKEY$	ON LINE	RSET	
BARTYPE	ERR	INPUT	OPEN	RUN	
BEEP	ERR$	INPUT$	OPTIMIZE	SAVE	
BF	ERROR	INSTR	OR	SET FAULTY DOT	

3.10 Operators

There are three main types of operators: arithmetic, relational, and logical:

Arithmetic Operators

+	Addition	(for example 2+2=4)
-	Subtraction	(for example 4-1=3)
*	Multiplication	(for example 2*3=6)
\	Integer division	(for example 6\2=3)
MOD	Modulo arithmetic (results in an integer value which is the remainder of an integer division, for example 5MOD2=1)	
^	Exponent	(for example 5^2=25)

Parentheses can be used to specify the order of calculation, for example:

```
7+5^2\8 = 10
(7+5^2)\8 = 4
```

Relational Operators

<	less than
<=	less than or equal to
<>	not equal to
=	equal to (also used as an assignment operator)
>	greater than
>=	greater than or equal to

Relational operators return:

-1	if relation is TRUE
0	if relation is FALSE

The following rules apply:

- Arithmetic operations are evaluated before relational operations.
- Letters are greater than digits.
- Lowercase letters are greater than their uppercase counterparts.
- The ASCII code "values" of letters increase alphabetically and the leading and trailing blanks are significant.
- Strings are compared by their corresponding ASCII code value.

Logical Operators

`AND` conjunction

`OR` disjunction

`XOR` exclusive or

Logical operators combine simple logical expressions to form more complicated logical expressions. The logical operators operate bitwise on the arguments, for example:

```
1 AND 2 = 0
```

Logical operators can be used to connect relational operators, for example:

```
A%10 AND A%<100
```

The principles are illustrated by the following tables, where A and B are simple logical expressions.

Logical operator: AND

A	B	A **AND** B
1	1	1
1	0	0
0	1	0
0	0	0

Logical operator: XOR

A	B	A **XOR** B
1	1	0
1	0	1
0	1	1
0	0	0

Logical operator: OR

A	B	A **OR** B
1	1	1
1	0	1
0	1	1
0	0	0

3.11 Devices

"Device" is a generic term for communication channels, various parts of the printer's memory, and operator interfaces such as the printer's display and keyboard.

Name	No.	Can be OPENed for...	Remarks
Communication			
console:	0	Input/Output	Printer's display and/or keyboard
uart1:	1	Input/Outpu	Serial communication port
uart2:	2	Input/Output	Serial communication port (option)
uart3:	3	Input/Output	Serial communication port (option)
rs485:	N/A	Input/Output	RS 485 communication
centronics:	4	Input	Parallel communication
net1:	5	Input/Output	EasyLAN communication (option)
usb1:	6	Input/Output	Serial communication port
finisher:	N/A	Input/Output	Printer's finisher interface
Memory			
/rom	N/A	Input (files only)	Printer's firmware (Kernel) plus read-only memory card. Alternative name "rom:".
/c	N/A	Input/Output/Random	Alternative names "c:" or "ram:"
tmp:	N/A	Input/Output/Append/ Random (files only)	Printer's temporary memory
card1:	N/A	Input/Output/Append/ Random (files only)	CompactFlash memory card
Special			
lock:	N/A	Input	Electronic key
storage:	N/A	Input/Output/Random	Electronic key
wand:	N/A	Input	Data from Code 128 bar code via printer's bar code wand interface

The devices can be listed using a DEVICES statement. Devices are referred to by name in connection with instructions concerning directories (for example SAVE, KILL, FORMAT) and with OPEN statements.

Note: The names of all devices should be lowercase characters only and be enclosed by quotation marks, for example "/c". Communication devices must have a trailing colon (:), for example "uart1:" or "centronics:". This is also valid for the "card1:", "finisher:", "lock:", "storage:", and "wand:" devices.

In instructions used in connection with communication (for example BREAK, BUSY/READY, COMSET), the keyboard/display unit and the communication channels are specified by numbers instead of names:

0 = "console:"
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
5 = "net1:"
6 = "usb1:"

4 Fingerprint Programming

This chapter explains how to creating application programs in the various modes of Intermec Fingerprint.

4.1 Introduction

The Intermec Fingerprint firmware works in two main modes, the “Immediate Mode” and the “Programming Mode”. Special cases are the Intermec Direct Protocol and EasyLAN, which are described in separate manuals and will not be explained any further in this manual.

Immediate Mode implies that the instructions are executed at once as soon as a carriage return is received. Most instructions can be used, but the instructions cannot be saved after execution.

Programming Mode is used to enter instructions in the form of program lines. The lines can be manually provided with visible line numbers at editing, or be automatically provided with invisible line numbers by the printer’s firmware. No execution is performed until a RUN statement is issued in the Immediate Mode, that is, on a line without number. The program can be saved in the printer’s memory and used over and over again.

4.2 Editing Methods

To be able to program a printer, you need a terminal or host computer with a screen and a keyboard and a working two-way serial communication between printer and host, preferably RS-232 on communication channel "uart1:". The host must be able to transmit and receive ASCII characters, for example using a communication program like Windows HyperTerminal.

There are three main methods of writing and transmitting a program to the printer:

Line-by-Line Method

If you have an “non-intelligent” terminal that just can transmit and receive ASCII characters, you must write and send each line separately.

Each line will be checked for possible syntax errors as soon as the printer receives it and the printer will return either “Ok” or an error message to the screen of the host, provided verbosity is on.

If you need to correct a mistake, you must rewrite the complete line using the same line number. Thus, this method is not suited for the programming without line numbers.

Note: Even if most examples of computer connection in this manual assumes a PC running under various versions of Microsoft Windows, Intermec Fingerprint is by no means restricted to such computers. Other personal computers and operating systems, such as DOS, Apple Mac-OS, OS-2, UNIX, etc. as well as larger computer systems, can be used following the same principles.

Copy-and-Paste Method

If the host computer is fitted with both a communication program (for example Windows HyperTerminal) and a text editor (for example Windows Notepad), you can write the program, partly or completely, in the text editor and then Copy and Paste it into the communication program.

Each line will be checked for possible syntax errors as soon as the printer receives it and, provided verbosity is on, the printer will return an error message after each line where an error has been detected.

If you need to correct a mistake, you can make the correction in the text editor and then copy and paste the line into the communication program. If you do not use line numbers, you must Copy and Paste the complete corrected program back to the communication program.

Send Text Method

If the host computer is fitted with both a communication program (for example Windows HyperTerminal) and a text editor (for example Windows WordPad), you can write the program, partly or completely, in the text editor and send the whole program as a text file to the printer using the communication program.

Each line will be checked for possible syntax errors as soon as the printer receives it and, provided verbosity is on, the printer will return an error message after each line where an error has been detected.

If you need to correct a mistake, you can make the correction in the text editor and then send the complete program again via the communication program.

4.3 Immediate Mode

The Immediate Mode can be used for three main purposes:

- Printing of labels that you will never need to print again.
- Printing of labels, which have been edited and saved in the host computer and are downloaded as text strings to the printer.
- Executing of instructions outside the editing of programs in the programming mode, for example DELETE, LOAD, MERGE, NEW, REBOOT, or RUN.

Rather than creating programs in the Programming Mode, in some cases you may want to edit the label in your host computer and transmit the printing instructions and data to the printer in the form of text strings. This method resembles the so called “Escape sequences” used in other types of label printers.

To make the strings shorter, use the Intermec Fingerprint abbreviations. Several statements can be issued on the same line separated by colons (:), on separate lines, or using a mix of both methods. Examples:

All intructions can be issued in a single line....

```
PP160,250:DIR3:AN4:FT"Swiss 721 BT":PT"Hello":PF ↵
```

or with each instruction as a separate line...

```
PP160,250 ↵                    (print start position)
DIR3 ↵                         (print direction)
AN4 ↵                          (alignment)
FT"Swiss 721 BT" ↵             (font select)
PT"Hello" ↵                    (text input data)
PF ↵                           (print one copy)
```

As soon as a carriage return is received, the firmware checks the instructions for syntax errors. Provided there is a working two-way communication and the verbosity is on, the printer will either return an error message or "Ok" to the host.

This type of communication works well and is easy to learn, but it does not take full advantage of the flexibility and computing capacity offered by the Intermec Fingerprint printers. For example, you cannot save the labels in the printer but must download each new label, and all error-handling must be taken care of by the host.

Rather than using the Immediate Mode, the Intermec Direct Protocol is usually to prefer, since it allows variable input data to be combined with predefined layouts, handles counters, and contains a flexible error-handler.

Beside printing text, bar codes, and graphics, you can perform other tasks in the Immediate Mode as well, for example calculation. Try typing this instruction on the keyboard of the host:

```
? ((5^2+5)\3)*5 ↵                    (↵ = Carriage Return key)
```

The calculation will be performed immediately and the result will be returned to the screen of the host:

```
50
Ok
```

Seven keys are enabled in the Immediate Mode:
- The <**Print**> key or button produces a FORMFEED.
- The <**Feed**> key produces a FORMFEED.
- The <**Shift**> + <**Feed**> keys produce a TESTFEED.
- The <**Setup**> key switches to the Setup Mode.
- The <**i/F5**> key shows information on the protocol of the communication channels.
- The <⇐> and <⇒> keys are used to browse between communiaction channels after the <**i/F5**> key has been activated.

When the printhead is lowered and the <**Print**> or <**Feed**> keys are pressed, these possible error conditions can cause an error message in English to be displayed:
- "Error 1005 -Press any key!-" (Out of paper)
- "Error 1031 -Press any key!-" (Next label not found)
- "Error 1027 -Press any key!-" (Out of ribbon)
- "Error 1083 -Press any key!-" (Ribbon low)

After the error has been attended to, the error message can be cleared by pressing any of the above-mentioned keys.

When the printhead is lifted, the <**Print**> and <**Feed**> keys will run the printers mechanism in order to facilitate cleaning of the platen roller (the rubber-coated roller that drives the media forward under the printhead). The motor will stop automatically when the platen roller has completed a few rotations.

4.4 Programming Mode

The Programming Mode is used to create programs consisting of one or more program lines. The firmware assumes input to the Programming Mode in two cases:

- When a line starts with a number.
- After an IMMEDIATE OFF statement has been executed. (See "Programming without Line Numbers" later in this chapter.)

One or several lines make up a program, which can be executed as many times as you wish. A program can also be saved, copied, loaded, listed, merged, and killed, see Chapter 5.4. All lines have line numbers, that are either manually entered when the program is edited, or provided automatically and invisibly by the firmware after an IMMEDIATE ON statement has been executed.

Each time the printer receives a program line followed by a Carriage Return character, the firmware checks the line for possible syntax errors. If an error is encountered, an error message will be returned to the host, provided there is a working two-way communication and the verbosity is on.

The program is executed in ascending line number order when a RUN statement followed by a carriage return is entered in the Immediate Mode, that is, on a line without any line number. However, various kinds of branching and loops can be created in the program to make the execution deviate from a strict ascending order.

Often, programs are created as autoexec (startup) files that start up automatically when the printer is switched on, and keeps on running infinitely.

Programming with Line Numbers

In this case you will start each line by manually entering a line number. We recommend that you start with line number 10 and use an increment of 10 between lines to allow additional lines to be inserted later, if necessary. To make the program easier to read, you can use a space character between the line number and the instruction. If not, the firmware will insert a space character automatically when the program is listed. Let us use the calculation example from the Immediate Mode. It would look like this in the Programming Mode:

```
10 ? ((5^2+5)\3)*5
RUN
```

yields:

```
15
Ok
```

Let us have a look at the lines:

- The first line consists of a line number (10) followed by an optional space character and the instruction ? ((5^2+5)\3)*5. (? is a shorthand form for the statement PRINT, which returns the result of the calculation on the standard OUT channel, that is, usually to the screen of the host). The line is terminated by a Carriage Return character.
- Next line has no line number, and contains the statement RUN plus a carriage return. This orders the printer to execute all preceding lines in consecutive ascending order according to their line numbers.

- The result (15) will be displayed on the host screen followed by "Ok" to indicate that execution was successful.

In this manual, the programming examples will generally have line numbers in order to make them easier to understand. For more complex programs, programming without line numbers, as explained below, may be both easier and quicker.

Programming without Line Numbers

You can choose to omit entering line numbers manually when writing a program. Normally, the firmware interprets the lack of line numbers as Immediate Mode. Thus, in order to make the printer understand what you want to do, you must turn off the Immediate Mode using an IMMEDIATE OFF statement. Then you can write the program line by line without having to type a line number at the start of each line. In other respects, you can generally work just as when using line numbers.

However, a major difference is when you want to make the execution branch to a certain line, using for example a GOTO statement. You cannot use a line number to specify the line to be branched to. Instead, there is a feature called "line labels." The line you want to branch to must start with a line label, which is a string of characters appended by a colon (:). The line label must not start with a digit or interfere with any keyword (see Chapter 3.9). When you want to branch to a line marked with a line label, just enter the line label (without any colon), where you otherwise would have entered the line number.

Finish the program by issuing an IMMEDIATE ON statement before you RUN it. The lines will automatically be numbered 10-20-30-40-50, etc., but the line numbers will not be visible until you LIST the program. Line labels will not be replaced by line numbers.

Two simple examples show the difference between using line numbers and line labels:

Line Numbers

```
10 GOSUB 1000
20 END
1000 SOUND 440,50
1010 RETURN

RUN
LIST
10 GOSUB 1000
20 END
1000 SOUND 440,50
1010 RETURN
RUN
```

Line Labels

```
IMMEDIATE OFF
GOSUB Q123
END
Q123:SOUND 440,50
RETURN
IMMEDIATE ON
RUN
LIST
10 GOSUB Q123
20 END
30 Q123: SOUND 440,50
40 RETURN
RUN
```

Programming Instructions

A number of instructions are used in connection with the editing of programs in the Programming Mode:

NEW

Before you enter the first program line, always issue a NEW statement in the Immediate Mode to CLEAR the printer's working memory, CLOSE all files, and CLEAR all variables.

If there already is a program in the working memory, it will be deleted by a NEW statement and cannot be restored unless it has been SAVEd before the NEW statement was executed.

IMMEDIATE OFF

If you want to write the program without entering line numbers manually, issue this statement in the Immediate Mode before the first line is entered.

REM (')

To make the program easier to understand, enter remarks and explanations on separate lines or in lines containing other instructions. Any characters preceded by REM, or its shorthand version ' (apostrophe, ASCII 39 decimal), will not be regarded as part of the program and will not be executed. REM statements can also be used at the end of lines, if they are preceded by a colon (:).

END

Usually, subroutines are entered on lines with higher numbers than the main program. It is a good programming habit to finish the main program with an END statement in order to separate it from the subroutines. When an END statement is encountered, the execution is terminated and all OPENed files and devices are CLOSEd.

STOP/DBBREAK/DBBREAK OFF/DBSTDIO/DBSTEP/DBEND/CONT

Use these instructions to stop and continue the execution of a program in connection with debugging, see Chapter 15.2.

IMMEDIATE ON

If an IMMEDIATE OFF statement has been issued before starting to write the program, turn on the Immediate Mode again using an IMMEDIATE ON statement before starting the execution using a RUN statement.

LIST

You can LIST the entire program to the screen of the host. You can also choose to list part of the program or variables only. If you have edited the program without line numbers, the numbers automatically assigned to the lines at execution will now appear. LIST is issued in the Immediate Mode.

DELETE

Program lines can be removed using the DELETE statement in the Immediate Mode. Both single lines and ranges of lines in consecutive order can be deleted.

RENUM

The program lines can be renumbered, for example to provide space for new program lines, to change the order of execution, or to make it possible to MERGE to programs. Line references for GOSUB, GOTO, and RETURN statements will be renumbered accordingly.

4.5 Conditional Instructions

Conditional instructions control the execution according to whether a numeric expression is true or false. Intermec Fingerprint has one conditional instruction, which can be used in two different ways:

IF...THEN...[ELSE]

If a numeric expression is TRUE, then a certain statement should be executed, but if the numeric expression is FALSE, optionally another statement should be executed. This example allows you to compare two values entered from the keyboard of the host:

```
10    INPUT "Enter first value ", A%
20    INPUT "Enter second value ", B%
30    C$="1:st value > 2:nd value"
40    D$="1:st value <= 2:nd value"
50    IF A%>B% THEN PRINT C$ ELSE PRINT D$
60    END
RUN
```

Another way to compare the two values in the example above is to use three IF...THEN statements:

```
10    INPUT "Enter first value ", A%
20    INPUT "Enter second value ", B%
30    C$="First value > second value"
40    D$="First value < second value"
50    E$="First value = second value"
60    IF A%>B% THEN PRINT C$
70    IF A%<B% THEN PRINT D$
80    IF A%=B% THEN PRINT E$
90    END
RUN
```

IF...THEN...[ELSE]...ENDIF

It is also possible to execute multiple THEN and ELSE statements. Each statement must be entered on a separate line and end of the IF...THEN...ELSE instruction must be indicated by ENDIF on a separate line. Example:

```
10    TIME$ = "121500":FORMAT TIME$ "HH:MM"
20    A%=VAL(TIME$)
30    IF A%>120000 THEN
40    PRINT "TIME IS ";TIME$("F"); ". ";
50    PRINT "GO TO LUNCH!"
60    ELSE
70    PRINT "CARRY ON - ";
80    PRINT "THERE'S MORE WORK TO DO!"
90    ENDIF
RUN
```

yields for example:

```
TIME IS 12:15. GO TO LUNCH!
```

4.6 Unconditional Branching

GOTO

The most simple type of unconditional branching is the "waiting loop." This means that a program line branches the execution to itself, waiting for something to happen, for example a key being pressed or a communication buffer becoming full.

This example shows how the program waits for the key <F1> to be pressed (line 30). Then a signal is emitted by the printer's beeper:

```
10    ON KEY (10) GOSUB 1000
20    KEY (10) ON
30    GOTO 30
40    END
1000  SOUND 880,100
1010  END
RUN
```

It is also possible to branch to a different line. Example:

```
10    INPUT "Enter a number:", A%
20    IF A%<0 THEN GOTO 100 ELSE GOTO 200
30    END
100   PRINT "NEGATIVE VALUE"
110   GOTO 30
200   PRINT "POSITIVE VALUE"
210   GOTO 30
RUN
```

Depending on whether the value you enter from the host is less than 0 or not, the execution branches to one of two alternative lines (100 or 200), which print different messages to the screen. In both cases, the execution branches to line 30, where the program ends.

There are more elegant ways to create such a program, but this example illustrates how GOTO always branches to a specific line. Line 20 is an example of conditional branching, which is explained in Chapter 4.8.

The GOTO statement can also be used to resume program execution at a specified line after a STOP statement.

4.7 Branching to Subroutines

GOSUB and RETURN

A subroutine is a range of program lines intended to perform a specific task, separately from the main program execution. Branching to subroutine can for example take place when:

- An error condition occurs.
- A condition is fulfilled, such as a certain key being pressed or a variable obtaining a certain value.
- A break instruction is received.
- Background communication is interrupted.

Another application of subroutines is branching to the one and same routine from different places in the same program. Thereby, you do not need to write the routine more than once and can make the program more compact.

The instruction for unconditional branching to subroutines is the GOSUB statement. There are also a few instructions for conditional branching to subroutines, which will be explained later in this chapter.

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

The same subroutine can be branched to as many times as you need from different lines in the main program. GOSUB remembers where the last branching took place, which makes it possible to return to the correct line in the main program after the subroutine has been executed. Subroutines may be nested, which means that a subroutine may contain a GOSUB statement for branching to a secondary subroutine, etc.

Subroutines should be placed on 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 illustrating nested subroutines:

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

4.8 Conditional Branching

As the name implies, conditional branching means that the program execution branches to a certain line or subroutine when a specified condition is fulfilled. The following instructions are used for conditional branching:

IF...THEN GOTO...ELSE

If a specified condition is TRUE, the program branches to a certain line, but if the condition is FALSE, something else will be done.

Example:

```
10    INPUT "Enter a value: ",A%
20    INPUT "Enter another value: ",B%
30    IF A%=B% THEN GOTO 100 ELSE PRINT "NOT EQUAL"
40    END
100   PRINT "EQUAL"
110   GOTO 40
RUN
```

ON...GOSUB

Depending on the value of a numeric expression, the execution will branch to one of several subroutines. If the value is 1, the program will branch to the first subroutine in the instruction, if the value is 2 it will branch to the second subroutine and so on.

Example:

```
10    INPUT "Press key 1, 2, or 3 on host: ", A%
20    ON A% GOSUB 1000, 2000, 3000
30    END
1000  PRINT "You have pressed key 1": RETURN
2000  PRINT "You have pressed key 2": RETURN
3000  PRINT "You have pressed key 3": RETURN
RUN
```

ON...GOTO

This instruction is similar to ON...GOSUB but the program will branch to specified lines instead of subroutines. This implies that you cannot use RETURN statements to go back to the main program.

Example:

```
10    INPUT "Press key 1, 2, or 3 on host: ", A%
20    ON A% GOTO 1000, 2000, 3000
30    END
1000  PRINT "You have pressed key 1": GOTO 30
2000  PRINT "You have pressed key 2": GOTO 30
3000  PRINT "You have pressed key 3": GOTO 30
RUN
```

ON BREAK...GOSUB

When a BREAK condition occurs on a specified device, the execution will be interrupted and branched to a specified subroutine. There, you can for example let the printer emit a sound signal or display a message before the program is terminated. You can also let the program execution continue along a different path.

In this example the program is interrupted when the <**Shift**> and <**Pause**> keys on the printer's keyboard are pressed (default). The execution branches to a subroutine, which emits a siren-sounding signal three times. Then the execution returns to the main program, which is indicated by a long shrill signal. You can also issue a break interrupt by transmitting the character "#" (ASCII 35 dec.) from the host on the communication channel "uart1:".

```
10    BREAK 1,35
20    BREAK 1 ON
30    ON BREAK 0 GOSUB 1000:REM Break from keyboard
40    ON BREAK 1 GOSUB 1000:REM Break from host (#)
50    GOTO 50
60    SOUND 800,100
70    BREAK 1 OFF: END
1000  FOR A%=1 TO 3
1010  SOUND 440,50
1020  SOUND 349,50
1030  NEXT A%
1040  GOTO 60
RUN
```

ON COMSET...GOSUB

When one of several specified conditions interrupts the background communication on a certain communication channel, the program branches to a subroutine, for example for reading the buffer. The interrupt conditions (end character, attention string and/or max. number of characters) are specified by a COMSET statement.

Example:

```
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
60    END
1000  END
2000  A$= COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```

Two instructions are used to branch to and from an error-handling subroutine when an error occurs:

ON ERROR GOTO

This statement branches the execution to a specified line when any kind of error occurs, ignoring the standard error-trapping routine. If line number is specified as 0, the standard error-trapping routine will be used.

RESUME

The RESUME statement is used to resume the program execution after an error-handling subroutine has been executed. RESUME is only used in connection with ON ERROR GOTO statements and can be used in five different ways:

`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.

This example shows branching to a subroutine when an error has occurred. The subroutine determines the type of error and takes the appropriate action. In this example only one error; “1019 Invalid font” is checked. After the error is cleared by substituting the missing font, the execution will be resumed.

```
10    ON ERROR GOTO 1000
20    PRTXT "HELLO"
30    PRINTFEED
40    END
1000  IF ERR=1019 THEN FONT "OCR-A BT" ELSE GOTO 2000
1010  PRINT "Substitutes missing font"
1020  FOR A%=1 TO 3
1030  SOUND 440,50
1040  SOUND 359,50
1050  NEXT A%
1060  RESUME
2000  PRINT "Undefined error, execution terminated"
2010  END
RUN
```

ON KEY...GOSUB

All present Intermec Fingerprint v8.00-compatible printers are provided with a built-in keyboard. However, unless there is a program running in the printer, for example Intermec Shell, the keys have no purpose (with the exception of the keys that work in the Immediate Mode, see Chapter 4.3). To make use of the keyboard, each key must be enabled individually using a KEY ON statement and then be assigned to a subroutine using an ON KEY GOSUB statement. The subroutine should contain the instructions you want to be performed when the key is pressed.

In the statements KEY (<id.>) ON, KEY (<id.>) OFF, and ON KEY (<id.>) GOSUB..., the keys are specified by id. numbers enclosed by parentheses, see Chapter 14.1.

Note that ON KEY...GOSUB excludes data input from the printer's keyboard (see Chapter 6.6) and vice versa.

This example shows how the two unshifted keys <**F1**> (id. No. 10) and <**F2**> (id. No. 11) are used to change the printer's setup in regard of printout contrast.

```
10    PRPOS 100,500
20    PRLINE 100,100
30    FONT "Swiss 721 BT"
40    PRPOS 100,300
50    MAG 4,4
60    PRTXT "SAMPLE"
70    ON KEY (10) GOSUB 1000
80    ON KEY (11) GOSUB 2000
90    KEY (10) ON : KEY (11) ON
100   GOTO 70
110   PRINTFEED
120   END
1000  SETUP "MEDIA,CONTRAST,-10%"
1010  PRPOS 100,100 : PRTXT "Weak Print"
1020  RETURN 110
2000  SETUP "MEDIA,CONTRAST,10%"
2010  PRPOS 100,100 : PRTXT "Dark Print"
2030  RETURN 110
RUN
```

4.9 Loops

GOTO

One type of loop has already been described in connection with the GOTO statement in Chapter 4.6, where GOTO was used to refer to the same line or a previous line. There are also two more advanced type of loops:

FOR...NEXT

These statements are to used create loops, where a counter is incremented or decremented until a specified value is reached. The counter is defined by a FOR statement with the following syntax:

FOR<counter>=<start value>TO<final value>[STEP<±interval>]NEXT[<counter>]

All program lines following the FOR statement will be executed until a NEXT statement is encountered. Then the counter (specified by a numeric variable) will be updated according to the optional STEP value, or by the default value +1, and the loop will be executed again. This will be repeated until the final value, as specified by TO <final value>, is reached. Then the loop is terminated and the execution proceeds from the statement following the NEXT statement.

FOR...NEXT loops can be nested, which means a loop can contain another loop, etc. Each loop must have a unique counter designation in the form of a numeric variable. The NEXT statement will make the execution loop back to the most recent FOR statement. If you want to loop back to a different FOR statement, the corresponding NEXT statement must include the same counter designation as the FOR statement.

This example shows how five lines of text entered from the keyboard of the host can be printed with an even spacing:

```
10    FONT "Swiss 721 BT"
20    FOR Y%=220 TO 100 STEP -30
30    LINE INPUT "Type text: ";TEXT$
40    PRPOS 100, Y%
50    PRTXT TEXT$
60    NEXT
70    PRINTFEED
80    END
RUN
```

Here is an example of two nested FOR...NEXT loops:

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

yields:

```
20    1
20    2
40    1
40    2
```

This example shows how an incremental counter can be made:

```
10    INPUT "Start Value: ", A%
20    INPUT "Number of labels: ", B%
30    INPUT "Increment: ", C%
40    X%=B%*C%
50    FOR D%=1 TO X% STEP C%
60    FONT "Swiss 721 BT",24
70    PRPOS 100,200
80    PRTXT "TEST LABEL"
90    PRPOS 100,100
100   PRTXT "COUNTER: "; A%
110   PRINTFEED
120   A%=A%+C%
130   NEXT D%
RUN
```

WHILE...WEND

These statements are used to create loops where series of statements are executed provided a given condition is TRUE.

WHILE is supplemented by a numeric expression, that can be either TRUE (-1) or FALSE (0). If the condition is TRUE, all subsequent program lines will be executed until a WEND statement is encountered. The execution then loops back to the WHILE statement and the process is repeated, provided the WHILE condition still is TRUE. If the WHILE condition is FALSE, the execution bypasses the loop and resumes at the statement following the WEND statement.

WHILE...WEND statements can be nested. Each WEND statement matches the most recent WHILE statement.

This example shows a program that keeps running in a loop (line 20-50) until you press the Y key on the host (ASCII 89 dec.), which makes the WHILE condition become true.

```
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
```

4.10 Program Structure

Although Intermec Fingerprint gives the programmer a lot of freedom in how to compose his programs, based on experience we recommend that the structure below is more or less implemented, with the obvious exception of such facilities that are not needed.

- **Program Information**
  - Program information, for example program type, version, release date, and byline (REM.)
- **Initiation**
  Decides how printer will work and branch to subroutines.
  - References to subroutines using for example ON BREAK GOSUB, ON COMSET GOSUB, ON ERROR GOSUB, ON KEY GOSUB.
  - Printer setup using for example SETUP, OPTIMIZE ON/OFF, LTS& ON/OFF, CUT ON/OFF, FORMAT DATE$, FORMAT TIME$, NAME DATE$, NAME WEEKDAY$, SYSVAR.
  - Character set and map tables (NASC, NASCD, MAP).
  - Enabling keyboard (KEY ON, KEYBEEP, KEYBMAP$).
  - Initial LED setting (LED ON/OFF).
  - Open "console:" for output (OPEN).
  - Assign string variables for each display line (PRINT#).
  - Select current directory (CHDIR).
  - Select standard I/O channel (SETSTDIO).
  - Open communication channels (OPEN).
  - Open files (OPEN).
  - Define arrays (DIM).
- **Main Loop**
  Executes the program and keeps it running in a loop.
  - Reception of input data (INPUT, INPUT#, INPUT$, LINE INPUT#).
  - Printing routine (FORMFEED, PRINTFEED, CUT).
  - Looping instructions (GOTO).
- **Subroutines**
  - Break subroutines (BREAK ON/OFF, BREAK).
  - Background communication subroutines (COM ERROR ON, COM ERROR OFF, COMSET, COMSET ON, COMSET OFF, COMBUF$, COMSTAT).
  - Subroutines for key-initiated actions (ON KEY).
  - Subroutines for display messages (PRINT#).
  - Error handling subroutines (ERR, ERL, PRSTAT).
  - Label layouts subroutines (PRPOS, DIR, ALIGN, FONT, BARSET, PRTXT, PRBAR, PRIMAGE, PRBOX, PRLINE, etc.).

4.11 Execution

To start the execution of the program currently residing in the printer's working memory, issue a RUN statement in the Immediate Mode, that is without any preceding line number. By default, the program will be executed in ascending line number order (with the exception of possible loops and branches) starting from the line with the lowest number. However, you can optionally start the execution at a specified line.

You can also execute a program that is not LOADed and you can execute Fingerprint program from within another Fingerprint program using an EXECUTE statement.

The first program or hardware error that stops the execution will cause an error message to be returned to the screen of the host, provided there is a working two-way communication.

For a working two-way communication, three conditions must be fulfilled:
- Serial communication
- Std IN channel = Std OUT channel
- Verbosity on

In case of program errors, the number of the line where the error occurred will also be reported by default, for example "Field out of label in line 110". After the error has been corrected, the execution must be restarted by means of a new RUN statement, unless a routine for dealing with the error in question is included in the program.

For demonstration purposes, we will now:
- write a short program without line numbers,
- execute it,
- and finally list it.

NEW

Note: For program instructions you can usually use upper- or lowercase characters at will, that is, "NEW" and "new" will work the same way.

```
Ok
IMMEDIATE OFF

Ok
REM This is a demonstration program
PRINT "This is the main program"
GOSUB sub1
END
sub1: PRINT "This is a subroutine":'Line label
RETURN
IMMEDIATE ON

Ok
RUN
```

yields:

```
This is the main program
This is a subroutine
```

```
Ok
LIST
```

yields:

```
10    REM This is a demonstration program
20    PRINT "This is the main program"
30    GOSUB SUB1
40    END
50    SUB1: PRINT "This is a subroutine":'Line label
60    RETURN

Ok
```

4.12 Breaking the Execution

In Chapter 1, “Getting Started”, the methods of breaking a startup program was briefly explained. Startup programs (autoexec files) start up automatically when the printer is switched on and continues to run infinitely using some kind of loop.

You can, by default, break a program by pressing the <**Shift**> key and keep it pressed while you also press down the <**Pause**> key. There is no default break facilities from the host via any communication channel. Therefore, it is strongly recommended always to include some break facilities in startup programs.

If the startup program resides in a memory card, you can of course switch off the printer, remove the card, and start up again.

Four instructions can be used for providing a program with a break interrupt facility:

`BREAK`	Specifies an interrupt character.
`BREAK...ON`	Enables break interrupt.
`BREAK...OFF`	Disables break interrupt.
`ON BREAK...GOSUB...`	Branches the execution to a sub-routine when a break interrupt is executed.

Note: A break interrupt character is saved in the printer’s temporary memory, and will not be removed until the printer is restarted, unless you specifically delete it using a BREAK...OFF statement for the device in question.

In all break-related instructions, the serial communication channels and the keyboard are referred to by numbers:
0 = "console:" (the printer’s keyboard)
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"

BREAK does not work on the following channels:
4 "centronics:"
5 "net1:"
6 "usb1:"

Always specify the interrupt character (BREAK) before enabling it in the program (BREAK...ON).

BREAK

The BREAK statement specifies an interrupt character by its decimal ASCII value. BREAK can be separately specified for each serial communication channel (except "net1:" and "usb1:") and for the printer’s built-in keyboard.

The interrupt character for all serial channels is by default ASCII 03 dec. (ETX) and from the printer’s keyboard ASCII 158 dec. (<**Shift**> + <**Pause**> keys). Also see BREAK...ON.

BREAK...ON

Break interrupt for all serial communication channels is disabled by default, but can be enabled using a BREAK...ON statement for the channel in question. Break interrupt from the keyboard is enabled by default.

BREAK... OFF

The BREAK...OFF statement revokes BREAK...ON for the specified device and deletes the specified break character from the printer's memory.

ON BREAK ...GOSUB...

This instruction is not necessary for issuing a break interrupt, but is useful for making the printer perform a certain task when a break occurs, for example branch the execution to another part of the program, show a message in the display, emit a warning signal, ask for a password, etc.

ON BREAK... GOSUB... can be specified separately for each serial communication channel and for the keyboard.

This example shows how a break interrupt will occur when you press the X-key (ASCII 88 dec.) on the host connected to "uart1:". A signal is emitted and a message appears in the printer's display.

Note: A break interrupt character is saved in the printer's temporary memory, and will not be removed until the printer is restarted, unless you specifically delete it using a BREAK...OFF statement for the device in question.

```
10    BREAK 1,88
20    ON BREAK 1 GOSUB 1000
30    GOTO 80
40    BREAK 1 ON
50    OPEN "console:" FOR OUTPUT AS 1
60    PRINT #1 : PRINT #1
70    PRINT #1, "Press X"
80    PRINT #1, "to break program";
90    BREAK 1 OFF
100   END
1000  SOUND 880,50
1010  PRINT #1 : PRINT #1
1020  PRINT #1, "PROGRAM"
1030  PRINT #1, "INTERRUPTED";
1040  RETURN 90
RUN
```

4.13 Saving the Program

Saving in Printer

Before saving the program, you may need to debug it as described in Chapter 15.2. When you are satisfied with the program, you can SAVE it in the printer's permanent memory ("/c"), in the printer's temporary memory ("tmp:"), or in CompactFlash memory card ("card1:"), see Chapter 5.1. If you save it in "tmp:", it will be lost at power off or at a power failure. We also recommend LISTing the program back to the host in order to make backup copy.

Naming the Program

When you save a program for the first time, you must give it a name consisting of up to 30 characters including possible extension.

You can use upper- or lowercase characters at will, but lowercase characters will—by default—automatically be converted to uppercase, when the program is SAVEd.

If you omit the extension, the firmware will—by default—add the extension ".PRG" automatically. When naming the program, also consider conventions and restrictions imposed by operating system of the host.

The automatic upper-/lower case conversion and adding of extension can be disabled using SYSVAR(43), see Chapter 14.7.

If the program or file name starts with a period character, it will be regarded a system file, see FILES and FORMAT statements in the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

Note: A list of names used for auxiliary Intermec Fingerprint files can be found in Chapter 1 of the *Intermec Fingerprint v8.00, Programmer's Reference Manual*. Make sure not use any of those names when saving your own programs or files.

Examples:

```
SAVE "PROGRAM1"
```
saves the program as PROGRAM1.PRG in the current directory (by default "/c").

```
SAVE "program2"
```
saves the program as PROGRAM2.PRG in the current directory (by default "/c").

```
SAVE "card1:PROGRAM1.TXT"
```
saves the program as PROGRAM1.TXT in a CompactFlash memory card inserted in the printer's memory card adapter.

Protecting the Program

When a program is SAVEd, it can optionally be protected, which means that it cannot be listed after being loaded and program lines cannot be changed, added, or deleted. Once a program has been protected, it cannot be unprotected. Thus, make an non-protected backup copy as a safety measure, should you need to make any changes later.

Example: Saving and protecting the program as PROGRAM1.PRG in the current directory (by default "/c"):

```
SAVE "PROGRAM1.PRG",P
```

Saving Without Line Numbers

A program can also be SAVEd without line numbers to make it easier to MERGE it with another program without risking that the line numbers interfere. Both programs should make use of line labels for referring to other lines, for example in connection with loops and branching instructions.

Example:

Saving the program as PROGRAM1.PRG without line numbers in the current directory (by default "/c"):

```
SAVE "PROGRAM1.PRG",L
```

Making Changes

If you LOAD a program, possibly make some changes and then SAVE the program under the original name and in the original directory, the original program will be replaced.

Example (changes the value of a variable in line 50 of a program and replaces the original version with the changed version):

```
LOAD "PROGRAM1.PRG"
50 A%=300
SAVE "PROGRAM1.PRG"
```

Making a Copy

The easiest way to copy a program is to use a COPY statement. Optionally, you can include directory references in the statement.

Example (copies a program from the permanent memory to a DOS-formatted memory card and gives the copy a new name):

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

If you LOAD a program and then SAVE it under a new name and/ or in another directory, you will create a copy of the original program.

Example (creates a copy of the program LABEL1.PRG and gives the copy the name LABEL2.PRG):

```
LOAD "LABEL1.PRG"
SAVE "LABEL2.PRG"
```

Renaming a Program

To rename a program, LOAD it, SAVE it under a new name, and finally KILL the original program.

Example (renames LABEL1.PRG with the name LABEL2.PRG):

```
LOAD "LABEL1.PRG"
SAVE "LABEL2.PRG"
KILL "LABEL1.PRG"
```

Note: The same general principles also apply to files!

Saving in DOS-formatted CompactFlash Memory Cards

Files can be saved or copied to a DOS-formatted Compact Flash memory card ("card1:"). Directories are not supported.

In addition, Intermec Shell has an application that allows you to download files from the host directly to a CompactFlash memory card using the Zmodem communication tool in order to create firmware upgrade cards. See the User's Guide and the Service Manual for more information.

Note: The printer will not recognize a memory card unless it has been inserted in the memory card slot before the printer is switched on.

Creating a Startup Program

The MKAUTO.PRG program is used to create so called startup programs or autoexec.bat-files, which are programs that will be LOADed and RUN automatically as soon as the power is switched on and the printer has been initialized. Usually, a startup program contains some kind of loop which makes it run infinitely, awaiting some input or action from the operator.

There can be one startup file stored in each of three main parts of the printer's memory. If there are startup files stored in more than one part, only one will be selected with the following priority:

1. An AUTOEXEC.BAT file stored in a CompactFlash memory card, provided the card was inserted in the printer before startup.
2. An AUTOEXEC.BAT file stored in the read/write part of the printer's permanent memory (device "/c").
3. The PUP.BAT file (Intermec Shell) in the read-only part of the printer's permanent memory (device "/rom").

The MKAUTO.PRG program is included in the systems part of the printer's memory ("/rom/MKAUTO.PRG") and consists of the following lines:

```
10    OPEN "AUTOEXEC.BAT" FOR OUTPUT AS 1
20    INPUT "startup file name:",S$
30    PRINT#1,"RUN";CHR$(34);S$;CHR$(34)
40    CLOSE1
```

A startup program can easily be created from an ordinary program using the following method:

- After having written and tested the program, SAVE it.
- Enter the following statement:

  ```
  RUN "/rom/MKAUTO"
  ```

- The following prompt will be displayed on the screen of the host:

  ```
  startup file name:
  ```

- Type the name of the program you just SAVEd (with or without the extension .PRG) and press the Carriage Return key.
- `Ok` on the screen indicates that the operation is completed.
- The startup program will be stored in the printer's current directory (by default "/c", which is the printer's permanent memory).
- When you restart the printer, the new startup program will start running, provided there is no other startup program with higher priority (see above.)

To undo the operation, use the statement:

```
KILL "AUTOEXEC.BAT"
```

This will not erase the original program, but it will no longer be used as a startup program. Note that you cannot KILL startup programs stored in "/rom".

4.14 Rebooting the Printer

Rebooting the printer has the same consequences as switching the power off and then on.

REBOOT

This statement allows you to reboot the printer from the host or as a part of the program execution.

When the printer is rebooted, or the power to the printer is switched on, a number of things happens:

- The printer's temporary memory ("tmp:") is erased, which means that any program not already SAVEd to "/c" or "card1:" will be irrevocably lost, all buffers will be emptied, all files will be closed, all date- and time-related formats will be lost, all arrays will be lost, and all variables will be set to zero. Fonts and images stored in the temporary memory will be erased.
- All parameters in the Fingerprint instructions will be reset to default.
- The printer performs a number of self-diagnostic tests, for example printhead resistance check and memory checksum calculations.
- The printer checks for possible optional devices like interface boards or cutter.
- The various parts of the printer's memory are searched for possible startup programs as described in Chapter 4.13. The first startup program encountered will be executed.
- The printer's internal clock is reset to default or—if a real-time clock circuit (RTC) is installed—updated from the RTC.

Note: Rebooting does not affect the printer's setup, unless any physical changes has been done to the printer during the power-off period, such as a printhead replacement or an installation or removal of an interface board.

5 File System

This chapter describes the printer's file system and the various types of files. It also explains different methods of transferring files and how to create arrays.

The following abbreviations are used:

SIMM	=	Single In-line Memory Module
SDRAM	=	Syncronous Dynamic Random Access Memory
FOS	=	File Operating System
ROM	=	Read Only Memory

5.1 Printer's Memory

The printer's memory consists of a number of parts, some with directories:

Permanent Memory ("/rom" and "/c")

Note: To provide compatibility with earlier versions of Intermec Fingerprint, the device designations "ram:" and "c:" are interpreted as "/c" and "rom:" as "/rom".

The permanent memory resides in a 4MB flash memory SIMM. Optionally, it can be replaced by an 8MB SIMM and futher expanded by a 4MB or 8MB SIMMs. A flash SIMM will keep it contents when the power to the printer is off without the aid of any battery backup system.

A SIMM contains a number of sectors. The "/c" file system uses 1K blocks. This means that a file cannot take less space than 1K. If a file is 2 bytes long, it still takes up 1K room on flash. If a file has 4.5K of data, it takes up 5K of flash. A directory takes 1K, regardless of how many files it contains.

When there are no free blocks left in any sector and at power up, the memory will automatically be reorganized to save space. Before reorganization, the sector is copied to a temporary sector for safety reasons. Then the sector is erased and the content is copied back from the temporary sector. This takes some time and makes the flash memory comparatively slow.

At least one SIMM must always be present. It must have a boot sector and a number of sectors containing the so called "kernel." There is also a temporary area for media feed info and odometer values. Some of these sectors are read-only and are included in the device "/rom".

The remaining part of the same flash memory SIMM contains a number of read/write sectors and is designated as device "/c". If there are additional flash SIMMs for the permanent memory, they are also included in the device "/c".

The following table illustrates the boot flash SIMM for an EasyCoder PF2i/PF4i(PF4i Compact Industrial) or EasyCoder PM4i printer:

Device	Size	Type	Used for
–	128K	Boot	Startup
/c	2048K	User file system or Kernel[1]	Customer's programs, files, images, etc.
/c	1856K	User file system	Customer's programs, files, images, etc.
–	32K	TMP area	Media feed info, odometer value, etc.
–	32K	Parameters	Media feed info, odometer value, etc.

[1]/. This sector can be used for the kernel or for the file system ("/c"). If it is used for the file system, the kernel is placed in the file system under the "/c/boot" directory, which is the normal configuration, enabling the size difference between 2048K and the real kernel size to be used in "/c". Refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual* for a complete list of files stored in /c and /rom by default. The kernel includes Fingerprint firmware, bar codes, standard fonts, standard images, Intermec Shell, auxiliary programs, default setup values, and the EasyLAN home page files.

Temporary Memory ("tmp:")

The temporary memory (device "tmp:") is a read/write SDRAM (Syncronous Dynamic Random Access Memory) and resides in a single 8MB SIMM package (as an option, a 16MB SDRAM can be fitted instead of the standard 8MB SIMM). The temporary memory has no battery backup and will be completely erased at power-off. However, four instructions can be used to prevent valuable variables from being lost at a power failure:

`SETPFSVAR`	Register variable to be saved at power off.
`GETPFSVAR`	Recover saved variable.
`LISTPFSVAR`	List saved variables.
`DELETEPFSVAR`	Delete a saved variable.

The temporary memory is used for the following purposes:

- To execute Fingerprint instructions. At startup, the kernel in the permanent memory is copied to the temporary memory, where all Fingerprint instructions are executed and the print image bitmaps are created.
- For print image buffers. The current image buffer can be saved as a file using the IMAGE BUFFER SAVE statement. The file will automatically converted to an image, that can be used in new label layouts like a preprint or template.
- For the font cache.
- For the Receive/Transmit buffers. Each serial communication channel must have one buffer of each kind. The size of each buffer is decided separately by the setup.
- For communication buffers. In a program, you may set up one communication buffer for each communication channel. This makes it possible to receive data simultaneously from several sources to be fetched at the appropriate moment during the execution of the program.
- To store data that do not need to be saved after power-off.
- To temporarily store data before they are copied to the permanent memory or to a memory card. Because the permanent flash memory has to reorganize itself occasionally, is becomes comparatively slow. Thus, it is more efficient to first create files in the temporary memory and then save them to the permanent memory. When speed is important, also avoid saving such data, that will be of no use after power off any way, in the permanent memory.

Note: There are no fixed partitions in the temporary memory. After the firmware has been copied to it and the Receive/Transmit buffers have been set according to the setup, the remaining memory will be shared between the various tasks.

DOS-Formatted CompactFlash Memory Cards ("card1:")

The built-in memory can be supplemented by a DOS-formatted CompactFlash memory card that is inserted in the printer's memory card slot. Such a card is referred to as "card1:" and can be both read from and written to.

Pre-programmed CompactFlash Cards

A number of CompactFlash cards can be ordered from Intermec to be used for special purposes.

There are four types of pre-programmed CompactFlash cards:

- Font Cards are used to supplement the standard fonts stored in the printer's permanent memory.
- Font Install Cards are used to install additional fonts in the printer's permanent memory.
- Firmware Cards are used to install a new firmware version (kernel) in the printer's permanent memory.
- Configuration Cards are use to configure the printer's CPU board for the characteristics of the printer model in question, especially the distance between the label stop sensor and the printhead's dot line.

Other Memory Devices ("storage:")

The "storage:" device is a small and slow memory device that is used for special applications. It should not be used for normal Fingerprint programming.

Current Directory

"Current directory" means the directory which the Intermec Fingerprint firmware will use unless you specifically instruct it to use another directory. By default, the current directory is "/c".

To appoint another directory as current directory, use a CHDIR statement. The CURDIR$ function returns the current directory.

Example shows changing directory from the default directory ("/c") to "tmp:" and then back to "/c".

```
10 CHDIR "tmp:"
. . . . .
90 CHDIR "/c"
```

Checking Free Memory

You can check the size of the memory in the current directory and see how much free space there is by issuing a FILES statement in the immediate mode.

Another way is to use the FRE function to make a small instruction, that returns the number of free bytes in a specified part of the printer's memory.

Example:

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

yields for example:

```
2382384
```

Providing More Free Memory

In order to free more memory space in the temporary memory, you can use a CLEAR statement to empty all strings, set all variables to zero, and reset all arrays to default. If even more memory is required, you will have to consider either to KILL some programs or files, or to use REMOVE IMAGE to delete some images stored in "/c" and or "tmp:". If the printer is not fitted with the maximum size memory, you could also fit additional or larger Flash or SDRAM SIMM packages (after having made backup copies on the host).

Formatting the Permanent Memory

The printer's permanent memory ("/c") can be formatted either partially or completely.

```
FORMAT "/c",A
```

erases all files in the device "c:" (hard formatting).

```
FORMAT "/c"
```

erases all files, except those starting with a period (.) character (soft formatting). System files are provided with such a period character, for example ".profile".

Formatting CompactFlash Memory Cards

A read/write CompactFlash memory card, inserted in the printer's memory card adapter, can be formatted to MS-DOS format using a FORMAT statement.

Example:

```
FORMAT "card1:",208,512,A
```

5.2 File System with Directories

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

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

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

Note: The convention in Intermec Fingerprint manuals is to use the new directory designations with slashes only in those memory devices that support directories. Thus, "/c" and "/rom" have the new designations, because they presently are the only devices that support directories, whereas "card1:", "tmp:" etc. are retained awaiting coming Fingerprint versions.

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

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

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

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

Example:

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

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

Three instructions helps the programmer to create new directories and see what directories the memory alrady contains:

`MKDIR`	creates a new directory in the printer's permanent memory ("/c").
`CURDIR$`	returns the current directory as the printer stores it.
`DIRNAME$`	returns the names of the directories in a specified part of the printer's memory.

5.3 Files

File Types

Four main types of files can be stored in the various parts of the printer's memory:

- Program Files
- Data Files
- Image Files
- Font Files

Object files, fonts, bar codes, and images are not regarded as files by the Intermec Fingerprint firmware.

File Names

In "/c", there is no restriction regarding the number of characters in a file name, but in "card1:" the name of a file may consist of up to 8+3 characters. Possible restrictions imposed by the operating system of the host should be considered if the file is to be transferred. Refer to *Intermec Fingerprint v8.00, Programmer's Reference Manual* for a list of reserved file names.

Listing Files

The files stored in the printer's memory can be listed using a FILES statement or a FILENAME$ function. Examples:

`FILES,A`	lists all files in the current directory.
`FILES "/c",A`	lists all files in the read/write part of the permanent memory
`FILES "/c",R,A`	lists all files in the read/write part of the permanent memory recursively
`FILES "/rom"`	lists all files stored in the read-only part of the permanent memory, except files preceded by a period character.
`FILES "card1:"`	lists all files stored in any inserted DOS-formatted CompactFlash memory card, except files preceded by a period character.
`FILENAME$("/c")`	returns all files in the read/write part of the permanent memory (wildcards are supported).

You can COPY a file to the standard OUT channel, where it will be printed on the screen of the host, for example:

```
COPY "[device]filename", "uart1:"
```

The FILELIST.PRG program included in the Intermec Fingerprint firmware is used to LIST a line-orientated file to the standard OUT channel:

- On your terminal, enter:

  ```
  RUN "/rom/FILELIST.PRG"
  ```

- The printer will respond by prompting you to enter the name of the file to be listed:

  ```
  Filename?
  ```

- Enter the filename, possibly preceded by a directory reference, for example:

  ```
  "/c/*.*"
  ```

5.4 Program Files

Program File Types

Program files are used to run and control the printer and to produce labels or other printouts. A program file is always composed of numbered lines, although the numbers may be invisible during the editing process (see Chapter 4.4).

A special case of program files is startup files, which means files that automatically start running when the printer is switched on (also called "autoexec-files"). Startup files were explained in Chapter 4.13 "Creating a Startup Program."

Instructions

The following instructions are used for creating and handling program files:

`LOAD`	Copies a specified program file to the printer's working memory.
`LIST`	Lists the program file in the working memory to the standard OUT channel, usually the screen of the host.
`MERGE`	Adds copy of a specified program file to the program file currently residing in the printer's working memory.
`RUN`	Executes the instruction in the program file. Must be issued in the Immediate Mode (not in a numbered line.)
`SAVE`	Saves a copy of the program file in the current directory or, optionally, in another specified directory. If a file with the same name already exists the that directory, it will be replaced by the new file.
`NEW`	Clears the working memory to allow a new program file to be created.
`COPY`	Copies a file to another name and/or directory.
`KILL`	Deletes a file from the printer's permanent memory ("/c"), the printer's temporary memory ("tmp:"), or from a DOS-formatted memory card ("card1:".)

5.5 Data Files

Data File Types

Data files are used by the program files for storing various types of data and can be divided into several subcategories:

- Sequential Input Files — See Chapter 6.4
- Sequential Output Files — See Chapter 7.3
- Sequential Append Files — See Chapter 7.3
- Random Access Files — See Chapters 6.5 and 7.4

Instructions

The following instructions are used in connection with the creation and handling of data files:

`OPEN`	Creates and/or opens a file for a specified mode of access and optionally specifies the record size in bytes.
`CLOSE`	Closes an OPENed file.
`REDIRECT OUT`	Creates a file to which the output data will be redirected (see Chapter 7.2).
`TRANSFERSET`	Sets up the transfer of data between two files.
`TRANSFER$`	Executes the transfer of data between two files according to TRANSFERSET.
`COPY`	Copies a file to another name and/or directory.
`KILL`	Deletes a file.
`LOC`	Returns the position in an OPENed file.
`LOF`	Returns the length in bytes of an OPENed file.

5.6 Image Files

Image files in .PCX format can be downloaded and installed in the printer's memory using the statement IMAGE LOAD.

Image files in .PCX format can be used immediately if they have been transferred using a TRANSFER ZMODEM or TRANSFER KERMIT instruction. The same applies to FTP.

Image files in Intelhex format, or the formats UBI00, UBI01, UBI02, UBI03, or UBI10, can be downloaded and converted to images using the STORE IMAGE and STORE INPUT statements.

Images files can be listed using FILES or FILENAME$ instructions.

Print images downloaded using the PRBUF statement are not saved as files.

The printer's current image buffer can be saved as a file using the IMAGE BUFFER SAVE statement and automatically be installed in the printer as an image in Intermec Fingerprint's internal bitmap format. This also includes print images downloaded to the image buffer using the PRBUF statement.

5.7 Font Files

Font files are files in TrueDoc (*.PFR) or TrueType (*.TTF) format and contain scaleable single or double-byte fonts complying with the Unicode standard. The printer's standard complement of single-byte fonts can be supplemented with additional fonts by downloading font files to the printer using Kermit or Zmodem file transfer protocol (see TRANSFER KERMIT in Chapter 5.9) or using an IMAGE LOAD statement. After a font file has been downloaded, the corresponding font can be used immediately without any need for a reboot.

Additional fonts can also be installed using a Font Install Card or be read from a Font Card. Note that since most double-byte fonts are very large, there may not be enough memory space in the printer to accommodate such fonts. In such a case, use a Font Card.

Font files can be listed using FILES or FILENAME$ instructions.

5.8 Transferring Text Files

Text files, for example program files and data files in ASCII format, can be downloaded via a communication program in the host, such as Windows Hyper Terminal. Text files can also be transferred back to the host, for example for backup purposes, by LOADing the file and LISTing it to a communication program in the host.

5.9 Transferring Binary Files

Font files and some image files come in binary format and can be downloaded from the host to the printer or vice versa using the Kermit or ZModem file transfer protocols, which are commonly used for binary transfer of data and are included in many communication programs, for example MS Windows HyperTerminal. Binary files can also be downloaded to the printer using the FILE& LOAD statement.

TRANSFER KERMIT

The TRANSFER KERMIT statement allows you to specify direction (Send or Receive), file name, input device, and output device. By default, a file name designated "KERMIT.FILE" will be transferred on the standard IN or OUT channel.

Example: The printer is set up to receive a file on the standard IN channel.

```
TRANSFER KERMIT "R"
```

Note: There is a 30 sec. timeout between the issuing of the TRANSFER KERMIT "R" statement and the start of the transmission.

ZMODEM

Files can be sent from host to printer or vice versa using the ZMODEM protocol using the instructions sz (send data from printer), rz (receive data to printer), and TRANSFER ZMODEM (send or transmit data).

TRANSFER STATUS

After a file have been transferred using a TRANSFER KERMIT or TRANSFER ZMODEM statement, the transfer can be checked using the TRANSFER STATUS statement. The statement will place the result into two one-dimensional arrays:

5-element numeric array (requires a DIM statement)

Element 0 returns:	Number of packets
Element 1 returns:	Number of NAKs
Element 2 returns:	ASCII value of last character
Element 3 returns:	Last error
Element 4 returns:	Block check type used

2-element string array (requires no DIM stmt)

Element 0 returns:	Type of protocol: "KERMIT" or "ZMODEM"
Element 1 returns:	Last file name received

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)
RUN
```

5.10 Transferring Files Between Printers

If you want to transfer a file from one printer to another printer, start by transferring the file to the host. Then disconnect the first printer and download the file to the second printer (or have the two printers connected to separate serial ports). After the transfer, check if the transfer was successful by comparing the result of CHECKSUM functions on both printers.

CHECKSUM

The CHECKSUM function uses an advanced algorithm on parts of the printer's internal code. Thus, calculate the CHECKSUM on the program in the transmitting printer before the transfer. After the transfer is completed, LOAD the program in the receiving printer and perform the same calculation. If the checksums are identical, the transfer was successful.

Note: Do not confuse CHECKSUM with CSUM, see Chapter 5.11 "Arrays."

This example calculates the checksum in the lines 10-90,000 in the program "DEMO.PRG."

```
LOAD "DEMO.PRG"
PRINT CHECKSUM (10,90000)
```

5.11 Arrays

Variables containing related data may be organized in arrays. Each value in an array is called an element. The position of each element is specified by a subscript, one for each dimension (max 10.) Each array variable consists of a name and a number of subscripts separated by commas and enclosed by parentheses, for example:

```
ARRAY$(3,3,3)
```

The number of subscripts in an array variable, the first time 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).

Four instructions are specifically used in connection with arrays:

DIM	Specifies the size of an array in regard of elements and dimensions.
SORT	Sorts the elements in a one-dimensional array in ascending or descending order.
SPLIT	Splits a string into an array.
CSUM	Returns the checksum for a string array.

DIM

If more than four elements are needed, or if you want to limit the size of the array, use a DIM statement to specify the size of the array in regard of the number of dimensions as well as the number of elements in each dimension. In most cases, one- or two dimensional arrays will suffice.

This example shows how three 1-dimensional, 5-element arrays can be used to return 125 possible combinations of text strings:

```
10    DIM TYPE$(4),COLOUR$(4),SIZE$(4)
20    TYPE$(0)="SHIRT"
30    TYPE$(1)="BLOUSE"
40    TYPE$(2)="TROUSERS"
50    TYPE$(3)="SKIRT"
60    TYPE$(4)="JACKET"
70    COLOUR$(0)="RED"
80    COLOUR$(1)="GREEN"
90    COLOUR$(2)="BLUE"
100   COLOUR$(3)="RED"
110   COLOUR$(4)="WHITE"
120   SIZE$(0)="EXTRA SMALL"
130   SIZE$(1)="SMALL"
140   SIZE$(2)="MEDIUM"
150   SIZE$(3)="LARGE"
160   SIZE$(4)="EXTRA LARGE"
170   INPUT"Select Type (0-4): ", A%
180   INPUT"Select Colour (0-4): ", B%
190   INPUT"Select Size (0-4): ", C%
200   PRINT TYPE$(A%)+", "+COLOUR$(B%)+", "+SIZE$(C%)
RUN
```

SORT

The SORT statement is used to sort a one-dimensional array in ascending or descending order according the character's ASCII values in the Roman 8 character set. You can also choose between sorting the complete array or a specified interval. For string arrays, you can select by which character position the sorting will be performed.

This example shows how one numeric array is sorted in ascending order and one string array is sorted in descending order according to the fifth character in each element:

```
10 FOR Q%=0 TO 3
20 A$=STR$(Q%)
30 ARRAY%(Q%)=1000+Q%:ARRAY$(Q%)="No. "+A$
40 NEXT Q%
50 SORT ARRAY%,0,3,1
60 SORT ARRAY$,0,3,-5
70 FOR I%=0 TO 3
80 PRINT ARRAY%(I%), ARRAY$(I%)
90 NEXT I%
RUN
```

yields:

```
1000 No. 3
1001 No. 2
1002 No. 1
1003 No. 0
```

SPLIT

The SPLIT function is used to split a string expression into elements in an array and to return the number of elements. A specified character indicates where the string will be split.

In this example a string expression is divided into six parts by the separator character "/" (ASCII 47 dec.) and arranged in a six-element array:

```
10    A$="ONE/TWO/THREE/FOUR/FIVE/SIX"
20    X$="ARRAY$"
30    DIM ARRAY$(6)
40    B%=SPLIT(A$,X$,47)
50    FOR C%=0 TO (B%-1)
60    PRINT ARRAY$(C%)
70    NEXT
RUN
```

yields:

```
ONE
TWO
THREE
FOUR
FIVE
SIX
```

CSUM

The checksum for string arrays can be calculated according to one of three different algorithms and returned using the CSUM statement.

Note! Do not confuse CSUM with CHECKSUM, see Chapter 5.10.

In this example, the checksum of a string array is calculated according both to the LRC (Logitudinal Redundancy Check) and the DRC (Diagonal Redundancy Check) algorithms:

```
10    FOR Q%=0 TO 3
20    A$=STR$(Q%)
30    ARRAY$(Q%)="Element No. "+A$
40    NEXT
50    CSUM 1,ARRAY$,B%:PRINT "LRC checksum: ";B%
60    CSUM 2,ARRAY$,C%:PRINT "DRC checksum: ";C%
RUN
```

yields:

```
LRC checksum: 0
DRC checksum: 197
```

6 Input to Fingerprint

This chapter explains how to provide Fingerprint with various types of input data from the host via a communication channel, from a file stored in the printer, or by typing using the printer's built-in keyboard.

It also deals with methods for controlling the communication and describes the protocol requirted for using RS-485 communication.

Finally, it also describes how to use the Industrial Interface on the Serial/ Industrial interface board.

6.1 Standard I/O Channel

The standard IN and standard OUT channels are the channels for input to the printer and output from the printer respectively (the default setting is "auto", which means that all communication channels are scanned for input). In most instructions, you can override the standard IN or OUT channel by specifying other channels.

SETSTDIO

You can appoint any of the following communication channels as standard IN and/or standard OUT channel using the SETSTDIO statement:

Standard IN channel	Standard OUT channel
0 = "console:"[1]	0 = "console:"[1]
1 = "uart1:"	1 = "uart1:"
2 = "uart2:"	2 = "uart2:"
3 = "uart3:"	3 = "uart3:"
4 = "centronics:"[2]	—
5 = "net1:"	5 = "net1:"
6 = "usb1:"	6 = "usb1:"
100 = "auto" (default)	

[1]/. Do not select "console:" as both std in and out channel, since it would only make characters entered on the printer's keyboard appear in the display.

[2] /. The parallel communication channel "centronics:" can only be used for input..

6.2 Input from Host (Std IN Channel only)

The std IN channel is used for sending instructions and data from the host to the printer in order to control the printer in the immediate mode, to create programs in the programming mode, to download program files, and to transmit input data. Some instructions receives data on the std IN channel only:

`INKEY$`	reads the 1:st character in the receive buffer.
`INPUT`	receives input data during execution of a program.
`LINE INPUT`	assigns one line to a string variable.

6.3 Input from Host (Any Channel)

The following instructions are used to receive input from any communication channel (incl. the std IN channel). The same instructions can also be used to read sequential files, see Chapter 6.4:

`OPEN`	opens a channel for sequential INPUT.
`INPUT#`	receives input data during execution of a program on the specified channel.
`INPUT$`	reads a string of data from the specified channel.
`LINE INPUT#`	assigns an entire line from the specified channel to a string variable.
`CLOSE`	closes the channel.

6.4 Input from a Sequential File

Refer to Chapter 6.3 for a summary of related instructions.

OPEN

Before any data can be read from a sequential file (or a communication channel other than the std IN channel), it must be OPENed for INPUT and assigned a number, which is used when referred to in other instructions. The number mark (#) is optional. Up to 10 files and devices can be open at the same time.

Example: The file "ADDRESSES" is opened for input as number 1:

```
OPEN "ADDRESSES" FOR INPUT AS #1
```

After a file or device has been OPENed for INPUT, you can use the following instructions for reading the data stored in it:

INPUT#

Reads a string of data to a variable. Commas can be used to assign portions of the input to different variables. When reading from a sequential file, the records can be read one after the other by repeated INPUT# statements. The records are separated by commas in the string. Once a record has been read, it cannot be read again until the file has been CLOSEd and then OPENed again.

Example (reads six records in a file and places the data into six variables):

```
10    OPEN "QFILE" FOR OUTPUT AS #1
20    PRINT #1, "Record A","a","b","c"
30    PRINT #1, "Record B",1,2,3
40    PRINT #1, "Record C","x";"y";"z"
50    PRINT #1, "Record D,Record E,Record F"
60    CLOSE #1
70    OPEN "QFILE" FOR INPUT AS #1
80    INPUT #1, A$
90    INPUT #1, B$
100   INPUT #1, C$
110   INPUT #1, D$,E$,F$
120   PRINT A$
130   PRINT B$
140   PRINT C$
150   PRINT D$
160   PRINT E$
170   PRINT F$
180   CLOSE #1
RUN
```

yields:

```
Record A a        b        c
Record B 1        2        3
Record C xyz
Record D
Record E
Record F
```

INPUT$

Reads a specified number of characters from the specified sequential file or channel. (If no file or channel is specified, the data on the standard IN channel will be read.) The execution is held up waiting for the specified number of characters to be received. If a file does not contain as many characters as specified in the INPUT$ statement, the execution will be resumed as soon as all available characters in the file have been received.

Sequential files are read from the start and once a number of characters have been read, they cannot be read again until the file is CLOSEd and OPENed again. Subsequent INPUT$ statements will start with the first of the remaining available characters.

Example (reads portions of characters from a file OPENed as #1):

```
10    OPEN "QFILE" FOR OUTPUT AS #1
20    PRINT #1, "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30    CLOSE #1
40    OPEN "QFILE" FOR INPUT AS #1
50    A$=INPUT$(10,1)
60    B$=INPUT$(5,1)
70    C$=INPUT$(100,1)
80    PRINT "Record 1:",A$
90    PRINT "Record 2:",B$
100   PRINT "Record 3:",C$
110   CLOSE #1
RUN
```

yields:

```
Record1: ABCDEFGHIJ
Record2: KLMNO
Record3: PQRTSUVWXYZ
```

LINE INPUT#

Works similar to INPUT#, but reads an entire line including all punctuation marks to a string variable instead of reading just one record. Note that commas inside a string will be regarded as punctuation marks and will not divide the string into records (compare with INPUT#).

Example (reads a complete line in a file and places the data into a single string variable):

```
10    OPEN "QFILE" FOR OUTPUT AS #1
20    PRINT #1, "Record A,Record B,Record C"
30    CLOSE #1
40    OPEN "QFILE" FOR INPUT AS #1
50    LINE INPUT #1, A$
60    PRINT A$
70    CLOSE #1
RUN
```

yields:

```
Record A,Record B,Record C
```

CLOSE

When a file is no longer used, it can be closed using a CLOSE statement containing the same reference number as the corresponding OPEN statement. An END statement also closes all open files.

A few instructions facilitate the use of files for sequential input:

EOF

The EOF function can connection with the statements INPUT#, LINE INPUT#, and INPUT$ to avoid the error condition "Input past end." 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%=1+1:WEND
120   IF EOF(2) THEN PRINT "End of File"
RUN
```

LOC (Location)

The LOC function returns the number of 128-byte blocks, that have been read or written since the file was OPENed.

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
.....
.....
```

LOF (Length-of-File)

The LOF function returns the length in bytes of an OPENed file.

The example illustrates how the length of the file "PRICELIST" is returned:

```
10    OPEN "PRICELIST" AS #5
20    PRINT LOF(5)
.....
.....
```

6.5 Input from a Random File

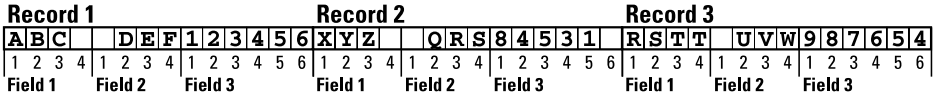
The following instructions are used in connection with input from random files:

`OPEN`	creates and/or opens a file for RANDOM access and optionally specifies the record length in bytes.
`FIELD`	creates a random buffer, divides it into fields and assigns a variable to each field.
`GET`	reads a record from the buffer to the file.
`CLOSE`	closes an OPENed file.
`LOC`	returns the number of the last record read by the use of a GET statements in the specified file.
`LOF`	returns the length in bytes of the specified file.

OPEN

To read the data stored in a random file, you must OPEN it.

The example in this chapter uses the random file created in Chapter 8.4, which can be graphically illustrated like this:




```
10    OPEN "ZFILE" AS #1 LEN=14
```

The appending LEN=14 refers to the length of each record which is 14 bytes (4 + 4 + 6). Do not confuse the LEN parameter in the OPEN statement with the LEN function, see Chapter 8.2.

FIELD

Then enter the same field definitions as when the data was put into the file:

```
20    FIELD#1, 4 AS F1$, 4 AS F2$, 6 AS F3$
```

GET

Use a GET statement to copy the desired record from the file. Note that you can select whatever record you want, as opposed to sequential files, where you reads the records one after the other. In this case, we will copy record No. 1 (compare with the illustration above).

```
30    GET #1,1
```

If you like, you can copy data from other records in the same file by issuing additional GET statements with references to the records in question.

Now you can use the variables assigned to the fields in the record using the FIELD statement to handle the data. Possible numeric expressions converted to string format before being put into the record can now be converted back to numeric format using VAL functions. In our example, we will simply print the data on the host screen:

```
40    PRINT F1$,F2$,F3$
```

CLOSE

Finally, close the file and execute:

```
50    CLOSE #1
RUN
```

yields:

```
ABC       DEF       123456
```

Two instructions facilitate the use of random files:

LOC (Location)

The LOC function returns the number of the last record read by the use of GET statement.

This example closes the file “ADDRESSES” when record No. 100 has been read from the file:

```
10    OPEN "ADDRESSES" AS #1
.....
.....
.....
200   IF LOC(1)=100 THEN CLOSE #1
.....
.....
```

LOF (Length-of-File)

The LOF function returns the length in bytes of an OPENed file.

The example illustrates how the length of the file “PRICELIST” is returned:

```
10 OPEN "PRICELIST" AS #5
20 PRINT LOF(5)
. . . .
. . . .
```

6.6 Input from Printer's Keyboard

All Fingerprint v8.00-compatible printers are provided with a built-in keyboard containing a set of numeric keys supplemented with function keys.

Input from an optional external alphanumeric keyboard is a case of ASCII input on a communication channel, see Chapter 6.1-6.3.

Note: Input from the printer's keyboard excludes the use of ON KEY...GOSUB statements (see Chapter 4.8) and vice versa.

The following instructions are used in connection with input from the printer's keyboard:

`OPEN`	opens the device "console:" for sequential INPUT.
`INPUT#`	reads a string of data to a variable.
`INPUT$`	reads a limited number of characters to a variable.
`LINE INPUT#`	reads an entire line to a variable.
`CLOSE`	closes the device.

The table below shows which ASCII characters the various keys will produce in unshifted and shifted position. However, the keyboard can be remapped (see later in this chapter). Refer to Chapter 17.3 for pictures.

Default ASCII decimal values

Key	Unshifted	Shifted	Notes
Shift	128	–	Adds 128 to the value of an unshifted key
F1/⇐	1	129	
F2/⇑	2	130	
F3/⇒	3	131	
F4/⇓	4	132	
F5/i	5	133	
C	8	136	
Enter	13	141	Unshifted Enter = Carriage Return
Feed	28	156	
Setup	29	157	
Pause	30	158	Shift+Pause is by default Break from keyboard
Print	31	159	
./-	46	174	
0	48	176	
1	49	177	
2	50	178	
3	51	179	
4	52	180	
5	53	181	
6	54	182	
7	55	183	
8	56	184	
9	57	185	

The printable characters actually generated by the respective ASCII value depend on the selected character set (NASC/NASCD) and possible MAP statements, see Chapter 8.1.

In case of INPUT# and LINE INPUT#, the input will not be accepted until a carriage return (<**Enter**>) is issued.

This example demonstrates how the printable character and decimal ASCII value of various keys on the printer's keyboard can be printed to the screen of the host. You can break the program by holding down the <**Shift**> key and pressing <**Pause**>.

```
10    PRINT "Character", "ASCII value"
20    OPEN "console:" FOR INPUT AS 1
30    A$=INPUT$(1,1)
40    B%=ASC(A$)
50    PRINT A$, B%
60    GOTO 30
70    CLOSE 1
RUN
```

6.7 Communication Control

The following instruction are used to control the communication between the printer and the host or other connected devices:

`BUSY/READY`	transmits a busy or ready signal on the specified communication channel.
`ON LINE/OFF LINE`	controls the SELECT signal on the parallel communication channel ("centronics:").
`VERBON/VERBOFF`	turns printer's verbosity on/off.
`SYSVAR(18)`	selects the printer's verbosity level.

BUSY/READY

Using these two statements, you can let the program execution turn a selected communication channel on or off. There is a difference between serial and parallel communication:

- **Serial communication:**

  The type of busy/ready signal is decided in the Setup Mode (Ser-Com; Flowcontrol), see the User's Guide.

  - When a BUSY statement is executed, the printer sends a busy signal, for example XOFF or RTS/CTS low.
  - When a READY statement is executed, the printer sends a ready signal, for example XON or RTS/CTS high.

- **Parallel communication:**

  The parallel Centronics communication channel uses the BUSY/READY statements to control the PE (paper end) signal on pin 12:

  - BUSY = PE high
  - READY = PE low

  The status of the PE signal can be read by a PRSTAT statement, for example:

  ```
  IF (PRSTAT AND 4) GOTO.....ELSE GOTO.....
  ```

Note: Issuing a READY statement is no guarantee that the printer will receive data, since there may be other conditions that hold up the reception, for example a full receive buffer.

ON LINE/OFF LINE

These two statements is only used for the parallel Centronics communication channel and controls the SELECT signal (pin 13 on the parallel interface board):

`ON LINE 4`	sets the SELECT signal high (default)
`OFF LINE 4`	sets the SELECT signal low

VERBOSITY

Three instruction can be used to control the printer's verbosity, which means the response from the printer on the standard OUT channel to instructions received on the standard IN channel:

`VERBON`	switches verbosity on
`VERBOFF`	switches verbosity off
`SYSVAR(18)`	specifies the verbosity level

VERBON/VERBOFF

By default, verbosity is on (VERBON) in Intermec Fingerprint, but off (VERBOFF) in the Intermec Direct Protocol. The verbosity level is controlled by the system variable SYSVAR(18).

All responses will be suppressed when a VERBOFF statement is issued. However, VERBOFF does not suppress question marks and prompts displayed as a result of, for example, an INPUT statement. Instructions like DEVICES, FILES, FONTS, IMAGES, LIST and PRINT will also work normally.

SYSVAR

The system variable SYSVAR is used for many purposes, one of which is to control the verbosity level. The verbosity level can be selected or read by specifying bits in SYSVAR(18):

All levels enabled (default in Fingerprint) ..... -1
No verbosity (default in Direct Protocol).....0
Echo received characters 1
"Ok" after correct command lines 2
Echo INPUT characters from communication port 4
Error after failed lines 8

The levels can be combined, so for example 3 means both "Echo received characters" and "Ok after correct command line."

When the printer receives a character, for example from the keyboard of the host, by default the same character is echoed back on the standard OUT channel, which usually is to the screen of the host. When an instruction has been checked for syntax errors and accepted, the printer returns "Ok". Else an error message is returned.

This example demonstrates how the printer is set to only return "Ok" after correct lines (2) or error messages after failed lines (8):

```
SYSVAR(18) = 10
```

6.8 Background Communication

Background communication means that the printer receives data on an IN channel while the program runs in a loop. The data are stored in a buffer, that can be emptied at an appropriate moment by the running program, which then can use the data. Note that background communication buffers are not the same as the receive buffers. Any input received on a communication channel is first stored in the channel's receive buffer, awaiting being processed. After processing, the data may be stored in the background communication buffer.

The following instructions are used in connection with background communication:

`COMSET`	decides how the background reception will work in regard of: - Communication channel. - Start character(s) of message string. - End character(s) of message string. - Characters to be ignored. - Attention string that interrupt reception. - Maximum number of characters to be received.
`ON COMSET GOSUB`	branches the program execution to a subroutine when background reception on a specified channel is interrupted.
`COMSET ON`	empties the buffer and turns on back-ground reception on the specified channel.
`COMSET OFF`	turns off background reception on the specified channel and empties the buffer.
`COM ERROR ON`	enables error handling on a specified channel.
`COM ERROR OFF`	disables error handling on a specified channel (default).
`COMSTAT`	reads the status of the buffer of a specified channel.
`COMBUF$`	reads data in the buffer of a specified channel.
`LOC`	returns the status of the buffers in a specified channel.
`LOF`	returns the status of the buffers in a specified channel.

Set up the printer for background communication like this:

- Start by enabling the error handling for the desired background communication channel using a COM ERROR ON statement:
  0 = "console:"
  1 = "uart1:"
  2 = "uart2:"
  3 = "uart3:"
  4 = "centronics:"
- It may be useful to create a few messages indicating what have caused the interruption.

Example: Error handling is enabled for communication channel "uart1:" and messages will be printed to the standard out channel for all conditions that can be detected by a COMSTAT function.

```
10    COM ERROR 1 ON
20    A$="Max. number of characters"
30    B$="End char. received"
40    C$="Communication error"
50    D$="Attention string received"
```

- Continue with a COMSET statement specifying:
  - Which communication channel will be used (0–4, see above).
  - Which character, or string of characters, will be used to tell the printer to start receiving data?
  - Which character, or string of characters, will be used to tell the printer to stop receiving data?
  - Which character or characters should be ignored, that is filtered out from the received data?
  - Which character, or string of characters, should be used as an attention string, that is, to interrupt the reception. Start, stop, ignore, and attention characters are selected according to the protocol of the computing device that transmits the data. Non-printable characters, for example STX (Start of Text; ASCII 02 dec.) and ETX (End of Text; ASCII 03 dec.) can be specified using a CHR$ function. To specify no character, use an empty string.
  - How many characters should be received before the transmission is interrupted? This parameter also decides the size of the buffer, that is, how much of the temporary memory will be allocated.

  Example:
  Background reception on the serial channel "uart1:".
  Start character: A
  End character: CHR$ (90), that is the character “Z”.
  Characters to be ignored: #
  Attention string: BREAK
  Max. number of characters in buffer: 20

  ```
  60    COMSET 1,"A",CHR$(90),"#","BREAK",20
  ```

- Decide what will happen, when the reception is interrupted, by specifying a subroutine to which the execution will branch, using an ON COMSET GOSUB statement. Interruption will occur when any of the following conditions is fulfilled:
  - an end character is received.
  - an attention string is received.
  - the maximum number of characters have been received.

  Example: When the reception of data on communication channel 1 ("uart1:") is interrupted, the execution will branch to a subroutine starting on line number 1000.

  ```
  70    ON COMSET 1 GOSUB 1000
  ```

- Turn on the COMSET.

Note: The COMSET interrupt has to be turned on after it has occurred and been taken care of.

```
80    COMSET 1 ON
```

- When the reception has been interrupted, it is time to see what the buffer contains. You can read the content of the buffer, for example to a string variable, using a COMBUF$ function:

```
1000  QDATA$=COMBUF$(1)
```

- The COMSTAT function can be used to detect what has caused the interruption. Use the logical operator AND to detect the following four reason of interruption as specified by COMSET:
  - Max. number of characters received (2).
  - End character received (4).
  - Attention string received (8).
  - Communication error (32).

  Example: The various cases of interruption makes different messages to be printed to the standard OUT channel. By assigning the COMSTAT value to a numeric variable, the execution will be faster than checking the COMSTAT value several times for different values.

```
1010  Q% = COMSTAT (1)
1020  IF Q% AND 2 THEN PRINT A$
1030  IF Q% AND 4 THEN PRINT B$
1040  IF Q% AND 8 THEN PRINT C$
1050  IF Q% AND 32 THEN PRINT D$
```

- If you want to temporarily turn off background reception during some part of the program execution, you can issue a COMSET OFF statement and then turn off the background reception again using a new COMSET ON statement.

Note: Any COMSET ON/OFF statement empties the buffer and the content will be lost if you do not read it first, using a COMBUF$ function.

- After adding a few lines to print the content of the buffer (line 1060) and to create a loop that waits from input from the host (line 90), the entire example will look like this.

  You can run the example by typing RUN and pressing <Enter> on the keyboard of the host. Then enter various characters and see what happens, comparing with the start character, stop character, ignore character, attention string, and max. number of characters parameters in the COMSET statement.

```
NEW
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  Q% = COMSTAT (1)
1020  IF Q% AND 2 THEN PRINT A$
1030  IF Q% AND 4 THEN PRINT B$
1040  IF Q% AND 8 THEN PRINT C$
1050  IF Q% AND 32 THEN PRINT D$
1060  PRINT QDATA$
1070  RETURN
RUN
```

Two instructions facilitate the use of background communication:

LOC (Locate)

The LOC function returns the status of the receive or transmitter buffers in an OPENed communication channel:

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

The number of bytes includes characters that will be mapped as NUL.

This example reads the number of bytes which remains to be received from the receiver buffer of “uart2:”:

```
10    OPEN "uart2:" FOR INPUT AS #2
20    A%=LOC(2)
30    PRINT A%
...
...
```

LOF (Length-of-File)

The LOF function returns the status of the buffers in an OPENed communication channel:

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

The example shows how the number of free bytes in the receive buffer of communication channel "uart2:" is calculated:

```
10    OPEN "uart2:" FOR INPUT AS #2
20    A%=LOF(2)
30    PRINT A%
...
...
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
RUN
```

6.9 RS-422 Communication

As an option, the printers can be fitted with an interface board that provides either RS-422 non-isolated or RS-422 isolated on "uart2:" or "uart3:".

In neither of these protocols, there are any wires for hardware handshake (RTS/CTS).

RS-422 is a point-to-point four-line screened cable connection between a host computer and a printer, or between two printers. Two lines transmit data and the other two receive data. No hardware handshake can be used (4 lines only), but XON/XOFF or ENQ/ACK can be used if so desired.

To set up the printer for RS-422 communication, proceed as follows:

- Fit straps and driver circuits according to the installation instructions for the interface board.
- Set the printer's flowcontrol to:

RTS/CTS:	Always Disable
ENQ/ACK:	Enable or Disable
XON/XOFF, Data to host:	Always Enable
XON/XOFF, Data from host:	Enable or Disable

- Select "uart2:" or "uart3:" as standard I/O channel, for example SETSTDIO 2,2.

6.10 RS-485 Communication

As an option, printers can have a Double Serial Interface Board or an Industrial Interface Board installed that can be fitted with circuits and straps to provide RS-485 on "uart2:".

Note: By the increasing use of local Ethernet and Wireless networks (LAN), RS-485 is becoming somewhat obsolete and we advise our customers to consider the pros and cons of a local area network vs. an RS-485 network before making new installations.

RS-485 is a 2-line screen cable point-to-point or multidrop loop connection, where the two lines switch between transmitting and receiving data according to instructions from the software. By default, the port is set to receive data. Before transmission of data, the port is switched to transmit. After the last character has been transmitted, the port is switched back to receive.

When an interface board configured for RS-485 is fitted in the printer, the serial communication setup for "uart2:" in the Setup Mode will only contain the following parameters:

- Baudrate (300-115200)
- Protocol address (enable/disable)
- New line (CR/LF, LF, CR)
- Receive buffer (30-32767)
- Transmit buffer (30-32767)
- Connected hardware (RS485) Read only info
- Protocol address (0-31)

Terminology

Protocol header

First part of transported information that includes:
- header start (192),
- destination protocol address (0-31),
- source protocol address (0-31),
- length (1-255),
- protocol-type & checksum.

Also see description later in this chapter.

Data area

A number of data-bytes transported with or without a proctocol header.

Choice of Protocol Mode / Raw Mode

There are three main alternatives:

A Protocol Mode = protocol header + data area (incl. substitution)

B Protocol Mode with COMSET control

C Raw Data Mode = No protocol control of data transmission, only data-area is sent, that is, same as ordinary RS-232 or RS-422

Selection of type of mode is made by a combination of device specification in the OPEN statement, selection of protocol address (enable/disable) in the Setup Mode, and specification of protocol address in the Setup Mode.

A: Protocol Mode

```
OPEN "rs485:" FOR INPUT
OPEN "rs485:<prot.addr.>" FOR OUTPUT
```

B: Protocol Mode + COMSET control

```
SETUP "SER-COM,UART2,PROT ADDR ENABLE"
OPEN "uart2:" FOR INPUT or OPEN "rs485:" FOR INPUT
OPEN "rs485:<prot.addr.>" FOR OUTPUT
```

C. Raw Data Mode

```
SETUP "SER-COM,UART2,PROT ADDR DISABLE"
OPEN "uart2:" FOR INPUT
OPEN "uart2:" FOR OUTPUT
```

COMSET Control with Protocol Mode (case B above)

COMSET 2, <sexp$_1$>, <sexp$_2$>, <sexp$_3$>, <sexp$_4$>, <nexp$_2$> is same as ordinary COMSET, but the use of <sexp$_1$>, <sexp$_2$>, <sexp$_3$> and <sexp$_4$> and <nexp$_2$> is done on the received data-area.

COMSET 2 is directly connected to the driver for "uart2:", but use of either "uart2:" and "rs485:" with PROT ENABLE is possible.

Data received on rules setup by COMSET 2, will be stored in COMBUF$(2), even if the device "uart2:" not is opened. If <nexp$_2$> = 0 the first character will be lost, so let <nexp$_2$> be at least 1.

At background reception with COMSET 2.., the rules setup by using <sexp$_1$> to <sexp$_4$> is executed on the data-area with <nexp$_2$> size.

If <nexp$_2$> = 1 and all <sexp$_n$> are set to "", then the first character of data is stored in COMBUF$(2) and the rest of data is available by reading from "uart2:" or "rs485:".

Sending, example: Sending sequence with protocol from unit-address 1 to 5, with data "ABCDE" when key F1 is depressed, is shown below:

```
NEW
IMMEDIATE OFF

REM ############ RS485 SENDING ######################
SETUP "SER-COM,UART2,PROT ADDR,ENABLE"
SETUP "SER-COM,UART2,PROTOCOL ADDR,1"
ON KEY 10 GOSUB XSEND : KEY 10 ON
MAIN:
GOTO MAIN

REM ---------- START SENDING ----------------------
XSEND:
OPEN "rs485:5" FOR OUTPUT AS 1
BUFSEND$="ABCDE"
PRINT# 1, BUFSEND$;
CLOSE 1
RETURN
IMMEDIATE ON
```

Receiving, example: Receiving is controlled by COMSET. Only data to addr. 5 is received.

```
NEW
IMMEDIATE OFF

REM########### RS485 RECEIVING WITH COMSET
############
SETUP "SER-COM,UART2,PROT ADDR,ENABLE"
SETUP "SER-COM,UART2,PROTOCOL ADDR,5"
COMSET 2,"","","","",1
ON COMSET 2 GOSUB RCVNETWORK
COMSET 2 ON
MAIN:
GOTO MAIN

REM -----------  RS485 INTERRUPT ROUTINE -----------
RCVNETWORK:
      TICKEND%=0
      BUFRCV$=COMBUF$(2)
      OPEN "rs485:" FOR INPUT AS #3
      BUFLEN%=LOC(3)
      WHILE ((BUFLEN%>0) OR (TICKEND%>TICKS))
          IF BUFLEN%>0 THEN
                BUFRCV$=BUFRCV$+INPUT$(BUFLEN%,3)
                TICKEND%=TICKS+TICKTIMEOUT%
                BUFLEN%=LOC(3)
          ENDIF
      WEND
      CLOSE 3
      COMSET 2 ON

REM ---------- OUTPUT INFO TO CONSOLE --------------
-
PRINT "LEN="; LEN(BUFRCV$); "MSG="; BUFRCV$
RETURN

IMMEDIATE ON
```

RS 485 Protocol Specification

All packets of data must be preceded by a header record, in which all data are binary:

START	DST	SRC	LEN	PROTO	CRC	<Data record/Request >

Header record: 5 bytes

START indicates the start of the header record using the character ASCII 192 decimal, see note!

DST is the destination protocol address 0–31 (1 byte).

SRC is the source protocol address 0–31 (1 byte).

LEN is the size in bytes of the data record, max. 249 characters (1 byte).

PROTO specifies type of protocols 0 or 1 (1 byte), see below.

CRC is the checksum of the header record (1 byte), that is the inverted sum of DST+SRC+LEN+PROTO bytes.

- **PROTO = 0**

  This protocol is used for ordinary transfer of data between two units. The syntax is:

START	DST	SRC	LEN	PROTO=0	CRC	<Data record>

- **PROTO = 1**

  This protocol is used for communication check from a host computer (cannot be sent from a printer!) Instead of a data record, a REQUEST byte (0 or 1) appends the header record. Any printer in the loop that has an open rs485 file can be checked if it is on-line (REQUEST = 0), or inquired for the number of seconds that have passed since its last startup or reboot (REQUEST = 1). If the printer is online, it will answer by returning the corresponding REQUEST byte, in the latter case followed by the time expressed as a 10-digit value with leading zeros.

  Example 1. The host computer sends:

START	DST	SRC	LEN	PROTO =1	CRC	REQUEST=0

  The printer replies:

START	DST	SRC	LEN	PROTO=1	CRC	REQUEST=0

  Example 2. The host computer sends:

START	DST	SRC	LEN	PROTO =1	CRC	REQUEST=1

  The printer replies:

START	DST	SRC	LEN	PROTO=1	CRC	REQUEST =1+time (nnnnnnnnnn)

Substitution Rules

Type of data	Orginal	Transport byte Substitution	Destination
Start packet mark <START>	(192)	=> (192) =>	(192)
Header & data info (START)	(192	=> (219), (220) =>	(192)
Header & data info (ESCAPE)	(219)	=> (219), (221) =>	(219)
Header info (other than above)	(x)	=> (x) =>	(x)
Data info (LF)	(LF)	=> according to New Line setup	
Data info (other than above)	(x)	=> (x)	

Notes about safe data transport
RS-485 protocols 0 and 1 delivers data from one unit to another with CRC control of the header record:
- destination
- source
- length
- protocol type

There is no check or CRC control of the data packet. If any check fails, no packet will be resent and there is no guarantee that the transmitted packet will be received by the destination unit.

To make sure that data will be transported correctly, you should let each data packet make a reply to the source unit.

6.11 External Equipment

Industrial Interface

The Intermec Fingerprint firmware not only allows you to control the printer, but various types of external equipment, like conveyor belts, gates, turnstiles, control lamps, etc. can also be controlled by the program execution. Likewise, the status of various external devices can be used to control both the printer and other equipment. The computing capacity of the Intermec Fingerprint printer can thus be used to independently control workstations without the requirement of an online connection to a host computer.

What makes this possible is the Serial/Industrial Interface Board, which is available as an option. The board contains a female DB-44pin connector with 8 digital IN ports, 8 digital OUT ports, and 4 OUT ports with relays.

There are two instruction solely used in connection with the Serial/Industrial Interface Board:

PORTOUT ON/OFF

This statement sets one of the four relays OUT ports or one of the digital OUT ports to either on or off.

PORTIN

This function returns the status of a specified port.

Refer to the installation instructions of the Serial/Industrial Interface Board for more details.

7 Output from Fingerprint

This chapter explains how Fingerprint can output information to the host via a communication channel, to the printer's display, and to various types of files.

7.1 Output to Std OUT Channel

The standard OUT channel is used for returning the printer's responses to instructions received from the host. That is why the same device usually is selected both standard IN and OUT channel (see SETSTDIO statement in Chapter 6.1).

After every instruction received on the std IN channel, the printer will either return "Ok" or an error message (for example "Feature not implemented" or "Syntax Error") on the std. OUT channel. If the std. OUT channel is connected to the host computer, this message will appear on the screen.

The response can be turned off/on using VERBOFF/ VERBON statements, the verbosity level can be selected by SYSVAR(18), and the type of error message can be selected by SYSVAR(19).

Some instructions return data on the std OUT channel only:

`DEVICES`	Lists all devices (also see Chapter 3.11).
`FILES`	Lists all files in the current directory or another specified directory (also see Chapter 5.2).
`FONTS`	Lists all fonts in the printer's entire memory (also see Chapter 11.8).
`IMAGES`	Lists all images in the printer's entire memory (also see Chapter 13.4).
`LIST`	Lists the current program in its entity or within a specified range of lines (also see Chapter 4.4).
`PRINT`	Prints the content of numeric or string expressions and the result of functions and calculations (see below).
`PRINTONE`	Prints characters entered as ASCII values (see below).

PRINT (or ?)

The PRINT statement prints a line on the std OUT channel, that is, usually on the screen of the host. The PRINT statement can be followed by one or several expressions (string and/or numeric).

If the PRINT statement contains several expressions, these must be separated by either commas (,) semicolons (;), or plus signs (+, only between string expressions):

- A comma sign (,) places the expression that follows at the start of next tabulating zone (each zone is 10 characters long). Example:

  ```
  PRINT "Price","$10"
  ```

  yields:

  ```
  Price     $10
  ```

- A semicolon sign (;) places the expression that follows immediately adjacent to the preceding expression. Example:

  ```
  PRINT "Price_";"$10"
  ```

  yields:

  ```
  Price_$10
  ```

- A plus sign (+) places the string expression that follows immediately adjacent to the preceding string expression (plus signs can only be used between two string expressions). Example:

```
PRINT "Price_"+"$10"
```

yields:

```
Price_$1
```

- Each line is terminated by a carriage return, as to make the next PRINT statement being started on a new line. However, if a PRINT statement is appended by a semicolon, the carriage return will be suppressed and next PRINT statement will be printed adjacently to the preceding one. Example:

```
10 PRINT "Price_";"$10";
20 PRINT "_per_dozen"
RUN
```

yields:

```
Price_$10_per_dozen
```

- A PRINT statement can also be used to return the result of a calculation or a function. Example:

```
PRINT 25+25:PRINT CHR$ (65)
```

yields:

```
50
A
```

- If the PRINT statement is not followed by any expression, a blank line will be produced.

PRINTONE

The PRINTONE statement prints the alphanumeric representation of one or several characters specified by their respective ASCII values (according to the currently selected character set, see NASC statement in Chapter 8.1) to the standard OUT channel.

The PRINTONE statement is useful, for example when a certain character cannot be produced from the keyboard of the host.

PRINTONE is very similar to the PRINT statement and follows the same rules regarding separating characters (commas and semicolons). Example:

```
PRINTONE 80;114;105;99;101,36;32;49;48
```

yields:

```
Price $ 10
```

7.2 Redirecting Output from Std Out Channel to File

As described in Chapter 7.1, some instructions return data on the standard OUT channel by default. However, it is possible to redirect such output to a file using the REDIRECT OUT statement, as described below.

REDIRECT OUT

This statement can be issued with or without an appending string expression:

`REDIRECT OUT <sexp>`	The string expression specifies the name of a sequential file that will be created and in which the output will be stored. Obviously, in this case no data will be echoed back to the host.
`REDIRECT OUT`	When no file name appends the statement, the output will be directed back to the std. OUT channel.

Example: The output is redirected to the file “IMAGES.DAT”. Then the images in the printer’s memory is read to the file after which the output is redirected back to the standard OUT channel. Then the file is copied to the communication channel "uart1:" and printed on the screen of the host.

```
10    REDIRECT OUT "IMAGES.DAT"
20    IMAGES
30    REDIRECT OUT
RUN
Ok

COPY "IMAGES.DAT","uart1:"
```

yields for example:

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

1695316 bytes free 307200 bytes used
Ok
```

7.3 Output and Append to Sequential Files

The following instructions are used in connection with output to sequential files:

`OPEN`	Creates and/or opens a file for sequential OUTPUT or APPEND and optionally specifies the record length in bytes.
`PRINT#`	Prints data entered as numeric or string expressions to the specified file.
`PRINTONE#`	Prints data entered as ASCII values to the specified file.
`CLOSE`	Closes an OPENed file.
`LOC`	Returns the number of 128-byte blocks, that have been written since the file was OPENed.
`LOF`	Returns the length in bytes of the specified file.

To print data to a sequential file, proceed as follows:

OPEN

Before any data can be written to a sequential file, it must be opened. Use the OPEN statement to specify the name of the file and the mode of access (OUTPUT or APPEND).

- OUTPUT means that existing data will be replaced.
- APPEND means that new data will be appended to existing data.

In the OPEN statement you must also assign a number to the OPENed file, which is used when the file is referred to in other instructions. The number mark (#) is optional. Optionally, the length of the record can also be changed (default 128 bytes.) Up to 10 files and devices can be open at the same time.

Examples:
The file “ADDRESSES” is opened for output and given the reference number 1:

```
OPEN "ADDRESSES" FOR OUTPUT AS #1
```

The file “PRICELIST” is opened for append and is given the reference number 5:

```
OPEN "PRICELIST" FOR APPEND AS #2
```

After a file or device has been OPENed for OUTPUT or APPEND, you can use the following instructions for writing data to it:

PRINT#

Prints data entered as string or numeric expressions to a sequential file. Expressions can be separated by commas or semicolons:

- Commas prints the expression in separate zones.
- Semicolons prints expressions adjacently.

There are two ways to divide the file into records:

- Each PRINT# statement creates a new record (see line 20-40 in the example below).
- Commas inside a string divides the string into records (see line 50 in the example below).

Example:

```
10    OPEN "QFILE" FOR OUTPUT AS #1
20    PRINT #1, "Record A", "a", "b", "c"
30    PRINT #1, "Record B", 1, 2, 3
40    PRINT #1, "Record C", "x"; "y"; "z"
50    PRINT #1, "Record D,Record E,Record F"
```

PRINTONE#

Prints characters entered as decimal ASCII values according to the selected character set to the selected file or device. This statement is for instance useful when the host cannot produce certain characters. Apart from using ASCII values instead of string or numeric expressions, the PRINTONE# works in the same way as the PRINT# statement.

Example (prints two records ”Hello” and “Goodbye” to “FILE1”):

```
10    OPEN "FILE1" FOR OUTPUT AS 55
20    PRINTONE#55,72;101;108;108;111
30    PRINTONE#55,71;111;111;100;98;121;101
```

CLOSE

After having written all the data you need to the file, CLOSE it using the same reference number as when it was OPENed.

Example:

```
10    OPEN "FILE1" FOR OUTPUT AS 55
20    PRINTONE#55,72;101;108;108;111
30    PRINTONE#55,71;111;111;100;98;121;101
40    CLOSE 55
```

LOC (Location)

The LOC function returns the number of 128-byte blocks, that have been written since the file was OPENed.

This example closes the file “ADDRESSES” when record No. 100 has been read from the file:

```
10    OPEN "ADDRESSES" FOR OUTPUT AS #1
.....
.....
200   IF LOC(1)=100 THEN CLOSE #1
.....
```

LOF (Length-of-File)

The LOF function returns the length in bytes of an OPENed file.

The example illustrates how the length of the file “PRICELIST” is returned:

```
10    OPEN "PRICELIST" FOR OUTPUT AS #5
20    PRINT LOF(5)
.....
.....
```

7.4 Output to Random Files

The following instructions are used in connection with output to random files:

`OPEN`	Creates and/or opens a file for RANDOM access and optionally specifies the record length in bytes.
`FIELD`	Creates a random buffer, divides it into fields and assigns a variable to each field.
`LSET/RSET`	Places data left- or right-justified into the buffer.
`PUT`	Writes a record from the buffer to the file.
`CLOSE`	Closes an OPENed file.
`LOC`	Returns the number of the last record written by the use of a PUT statement in the specified file.
`LOF`	Returns the length in bytes of the speci-fied file.

To write data to a random file, proceed as follows:

OPEN

Start by OPENing a file for random input/output. Since random access is selected by default, the mode of access can be omitted from the statement, for example:

```
10    OPEN "ZFILE" AS #1
```

Optionally, the length of each record in the file can be specified in number of bytes (default 128 bytes):

```
10    OPEN "ZFILE" AS #1 LEN=14
```

FIELD

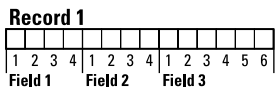
Next action to take is to create a buffer using a FIELD statement. The buffer is given a reference number and divided into a number of fields with individual length in regard of number of characters. To each field, a string variable is assigned.

The buffer specifies the format of each record in the file. The sum of the length of the different fields in a record must not exceed the record length specified in the OPEN statement.

In the example below, 4 bytes are allocated to field 1, 4 bytes to field 2 and 6 bytes to field 3. The fields are assigned to the string variables F1$, F2$, and F3$ respectively.

```
20    FIELD#1, 4 AS F1$, 4 AS F2$, 6 AS F3$
```

Graphically illustrated, the record produced in the line above will look like this:



The file can consist of many records, all with the same format. (To produces files with different record lengths, the file must be OPENed more than once and with different reference numbers.)

Now it is time to write some data to the file. Usually the data comes from the host or from the printer's keyboard. In this example, we will type the data directly on the host and assign the data to string variables:

```
30     QDATA1$="ABC"
40     QDATA2$="DEF"
50     QDATA3$="12345678"
```

Note: Only string variables can be used. Possible numeric expressions must therefore be converted to strings using STR$ functions.

LSET/RSET

There are two instructions for placing data into a random file buffer:

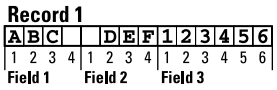
- LSET places the data left-justified.
- RSET places the data right-justified.

In other words, if the input data consist of less bytes that the field into which it is placed, it will either be placed to the left (LSET) or to the right (RSET.)

If the length of the input data exceeds the size of the field, the data will be truncated from the end in case of LSET, and from the start in case of RSET.

```
60     LSET F1$=QDATA1$
70     RSET F2$=QDATA2$
80     LSET F3$=QDATA3$
```

Using the graphic illustration from previous page, the result is meant to be like this:



The first field is left-justified, the second field is right-justified, and the third field is left-justified and truncated at the end. Digits 7 and 8 are omitted since the field is only six bytes long: if the field had been right-justified, digits 1 and 2 had been omitted instead.

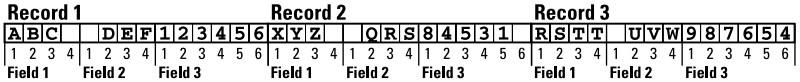
PUT

Next step is to transfer the record to the file. For this purpose we use the PUT statement. PUT is always followed by the number assigned to the file when it was OPENed, and the number of the record in which you want to place the data (1 or larger).

In our example, the file ZFILE was OPENed as #1 and we want to place the data in the first record. You can place data in whatever record you like. The order is of no consequence.

```
90     PUT #1,1
```

If you want, you can continue and place data into other records using additional sets of LSET, RSET and PUT statements. Below is a graphic example of a three-record file:



CLOSE

When you are finished, close the file:

```
100   CLOSE #1
```

Nothing will actually happen before you execute the program using a RUN statement. Then the data will be placed into the fields and records as specified by the program, for example:

```
10    OPEN "ZFILE" AS #1 LEN=14
20    FIELD#1, 4 AS F1$, 4 AS F2$, 6 AS F3$
30    QDATA1$="ABC"
40    QDATA2$="DEF"
50    QDATA3$="12345678"
60    LSET F1$=QDATA1$
70    RSET F2$=QDATA2$
80    LSET F3$=QDATA3$
90    PUT #1,1
100   CLOSE #1
RUN
```

LOC (Locate)

The LOC function returns the number of the last record read or written by the use of GET or PUT statements respectively in an OPENed file.

This example closes the file “ADDRESSES” when record No. 100 has been read from the file:

```
10    OPEN "ADDRESSES" AS #1
.....
.....
200   IF LOC(1)=100 THEN CLOSE #1
.....
.....
```

LOF (Length-of-File)

The LOF function returns the length in bytes of an OPENed file.

The example illustrates how the length of the file “PRICELIST” is returned:

```
10    OPEN "PRICELIST" AS #5
20    PRINT LOF(5)
.....
.....
```

7.5 Output to Communication Channels

Output from a Fingerprint program can be directed to any serial communication channel OPENed for sequential OUTPUT following the same principles as for output to files (see Chapter 7.3).

Note: In this case, the "centronics:" channel cannot be used.

The communication channels are specified by name as follows:

- "uart1:"
- "uart2:"
- "uart3:"
- "net1:"
- "usb1:"

The following instructions are used in connection with output to a communication channel:

`OPEN`	Opens a serial communication channel for sequential output.
`PRINT#`	Prints data entered as numeric or string expressions to the selected channel.
`PRINTONE#`	Prints data entered as ASCII values to the selected channel.
`CLOSE`	Closes an OPENed channel.
`LOC`	Returns the remaining number of free bytes in the transmitter buffer of the se-lected communication channel.
`LOF`	Returns the remaining numbers of char-acters to be transmitted from the transmit-ter buffer is returned.
`COPY`	Copies a file to a communication channel.

Example 1 (prints the records “Record 1” and “Record 2” to the serial communication channel "uart2:"):

```
10    OPEN "uart2:" for OUTPUT AS #1
20    PRINT #1, "Record 1"
30    PRINTONE #1, 82;101;99;111;114;100;32;50
40    CLOSE #1
```

Example 2 (prints the file “datafile” stored in a DOS-formatted Cmpact-Flash memory card to the serial communication channel "uart2:"):

```
COPY "card1:datafile","uart2:"
```

7.6 Output to Display

The only device, other than the serial communication channels, that can be OPENed to receive output from a Fingerprint program, is the printer’s LCD display ("console:"). This is explained in Chapter 14.2 together with other methods for controlling the display.

8 Data Handling

This chapter describes how input data can be preprocessed in order to handle text properly and how to convert input data. It also explains the use of date and time as well generation of random numbers.

8.1 Preprocessing Input Data

All input data to the printer come in binary form via the various communication channels. Text files are transmitted in ASCII format, which upon reception will be preprocessed by the printer's firmware according to two instructions as to provide full compatibility between the printer and the host:

MAP	Remaps the selected character set.
NASC	Selects a single-byte character set.
NASCD	Selects a double-byte character set.

A character received by the printer on a communication channel will first be processed in regard of possible MAP statements. Then the character will be checked for any COMSET or ON KEY... GOSUB conditions. When a character is to be printed, it will be processed into a bitmap pattern that makes up a certain character according to the character set selected using a NASC or NASCD statement.

MAP

The MAP statement is used to modify a character set or to filter out undesired characters on a specified communication channel by mapping them as NUL (ASCII 0 dec.)

If no character set meets your requirements completely (see NASC and NASCD below), select the set that comes closest and modify it using MAP statements. Do not map any characters to ASCII values occupied by characters used in Fingerprint instructions, for example keywords, operators, %, $, #, and certain punctuation marks. Mapped characters will be reset to normal at power-up or reboot.

Example: You may want to use the German character set (49) and 7 bits communication protocol. However, you need to print £ characters, but have no need for the § character. Then remap the £ character (ASCII 187 dec.) to the value of the § character (ASCII 64 dec.) Type a series of § characters on the keyboard of the host and finish with a carriage return:

```
10    NASC 49
20    MAP 64,187
30    FONT "Swiss 721 BT"
40    PRPOS 100,100
50    INPUT "Enter character";A$
60    PRTXT A$
70    PRINTFEED
RUN
```

yields:

```
Enter character?
```
(see note!)

Note: When using 7 bit communication, the printer cannot echo back the correct character to the host if its ASCII value exceeds 127, hence semicolon characters will appear on the screen. Nevertheless, the desired "£" characters will be printed on the label.

NASC

The NASC statement is used to select a single-byte character set that decides how the various characters will be printed. This instruction makes it possible to adapt the printer to various national standards. By default, characters will be printed according to the Roman 8 character set.

Suppose you order the printer to print the character ASCII 124 dec. (We will not concern ourselves with how your computer and its keyboard are mapped. Refer to their respective manuals.) If you check the character set tables in the *Intermec Fingerprint v8.00, Programmer's Reference Manual*, you will see that ASCII 124 will generate the character "|" according to the Roman 8 character set, "ù" according to the French character set, and ñ according to the Spanish set,etc. The same applies to a number of special national characters, whereas digits 0-9 and characters A-Z, a-z plus most punctuation marks are the same in all sets. Select the set that best matches your data equipment and printout requirements.

If none of the sets matches your requirements exactly, select the one that comes closest. Then, you can make final corrections using MAP statements, see above.

A NASC statement will have the following consequences:

- **Text printing**
  Text on labels etc. will be printed according to the selected character set. However, instructions concerning the printable label image, that already has been processed before the NASC statement is executed, will not be affected. This implies that labels may be multilingual.
- **LCD display**
  New messages in the display will be affected by a preceding NASC statement. However, a message that is already displayed will not be updated automatically. The display is able to show most printable latin characters. In the Setup Mode, all characters are mapped according to the US-ASCII standard.
- **Communication**
  Data transmitted from the printer via any of the communication channels will not be affected, as the data is defined by ASCII values, not as alphanumeric characters. The active character set of the receiving unit will decide the graphic presentation of the input data, for example on the screen of the host.
- **Bar code printing**
  The pattern of the bars reflects the ASCII values of the input data and is not affected by a NASC statement. The bar code interpretation (the human readable characters below the bar pattern) is affected by a NASC statement. However, the interpretation of bar codes, that have been processed and are stored in the print buffer, will not be affected.

This example selects the Italian character set:

```
NASC 39
```

NASCD

The NASCD statement works similar to the NASC statement, but is used for double-byte character sets, which means fonts that require 2 bytes to specify a character according to the Unicode standard. This is for example the case with major Asian languages, such as Chinese, Korean, and Japanese.

When a double-byte character set has been selected, the firmware will usually treat all characters from ASCII 161 dec. to ASCII 254 dec. (ASCII A1-FE hex) as the first part of a two-byte character. Next character byte received will specify the second part. However, the selected Unicode double-byte character set may specify some other ASCII value as the breaking point between single and double byte character sets.

There are various ways to produce double-byte characters from the keyboard of the computer. By selecting the proper character set using a NASCD statement, the typed-in ASCII values will be translated to the corresponding Unicode values, so the desired glyph will be printed.

Double-byte fonts and character set tables are available from Intermec on special request, usually in the form of font cards.

Example: The text field in line 50 contains both single- and double-byte fonts. The double-byte font and its character set are stored in a Font Install Card. The program yields a printed text line that starts with the Latin character A (ASCII 65 dec.) followed by the Chinese font that corresponds to the address 161+162 dec. in the character set "BIG5.NCD."

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

Customized Character Sets

Advanced users can compose their own character sets to meet special requirements. Refer to the *Intermec Fingerprint, Font Reference Manual*.

8.2 Input Data Conversion

There are a number of instruction for converting data in numeric or string expressions. You will find them used in many examples in this volume. The instructions will only be described in short terms. For full information, please refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

ABS

The ABS function returns the absolute value of a numeric expression. Absolute value means that the value is either positive or zero.

Example:

```
PRINT ABS (10-15)
```

yields:

```
5
```

ASC

The ASC function returns the ASCII value of the first character in a string expression.

Example:

```
PRINT ASC("HELLO")
```

yields:

```
72
```

CHR$

The CHR$ function returns the readable character from an ASCII value. This function is useful when you cannot produce a certain character from the keyboard of the host.

Example:

```
PRINT CHR$(72)
```

yields:

```
H
```

FLOATCALC$

The FLOATCALC$ function is used to calculate float numbers using the aritmetc operators addition (+), subtraction (-), multiplication (*), and division (/). Precision can also be specified.

Example:

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

yields:

```
1235.90000
```

FORMAT$

The FORMAT$ function is used to format a number represented by a string. It is typically used in connection with FLOATCALC$.

Example:

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

yields:

```
"1235.900000000000000"

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

yields:

```
"1 235.9"
```

INSTR

The INSTR function searches a string expression for a certain character, or sequence of characters, and returns the position.

Example:

```
PRINT INSTR ("INTERMEC","MEC")
```

yields:

```
6
```

LEFT$

The LEFT$ function returns a certain number of characters from the left side of a string expression, that is from the start. The complementary instruction is RIGHT$.

Example:

```
PRINT LEFT$("INTERMEC PRINTER",3)
```

yields:

```
INT
```

LEN

The LEN function returns the number of characters including space characters in a string expression. Example:

```
PRINT LEN ("INTERMEC PRINTER")
```

yields:

```
16
```

MID$

The MID$ function returns a part of a string expression. You can specify start position and, optionally, the number of characters to be returned. Example:

```
PRINT MID$ ("INTERMEC PRINTER",6,3)
```

yields:

```
MEC
```

RIGHT$

The RIGHT$ function returns a certain number of characters from the right side of a string expression, that is, from the end. The complementary instruction is LEFT$. Example:

```
PRINT RIGHT$("INTERMEC PRINTER",7)
```

yields:

```
PRINTER
```

SGN

The SGN function returns the sign (1 = positive, -1 = negative, or 0 = zero) of a numeric expression. Example:

```
PRINT SGN(5-10)
```

yields:

```
-1
```

SPACE$

The SPACE$ function returns a specified number of space characters and is for example useful for creating tables with monospace characters.

Example:

```
10    FONT "Swiss 721 BT"
20    X%=100 : Y%=300
30    FOR Q%=1 TO 5
40    INPUT "Commodity: ", A$
50    INPUT "Price $:", B$
60    C$=SPACE$(15-LEN(A$))
70    PRPOS X%,Y%
80    PRTXT A$+C$+"$ "+B$
90    Y%=Y%-40
100   NEXT
110   PRINTFEED
```

Note: When entering the price in the example for SPACE$, make sure to use a period character (.) to indicate the decimal point.

STR$

The STR$ function returns the string representation of a numeric expression. The complementary instruction is VAL. Example:

```
10    A%=123
20    A$=STR$(A%)
30    PRINT A%+A%
40    PRINT A$+A$
RUN
```

yields:

```
246
123123
```

STRING$

The STRING$ function returns a specified number of a single character specified either by its ASCII value or by being the first character in a string expression. Example:

```
10    A$="*THE END*"
20    FIRST$=STRING$(4,42)
30    LAST$=STRING$(4,A$)
40    PRINT FIRST$+A$+LAST$
RUN
```

yields:

```
*****THE END*****
```

VAL

The VAL function returns the numeric representation of a string expression. The complementary instruction is STR$. VAL is for example used in connection with random files, which only accept strings (see Chapters 6.5 and 7.4). Thus numeric expressions must be converted to string format using STR$ before they are PUT in a random file and be converted back to numeric values using VAL after you GET them back from the file.

Another application is when you want to calculate using data in a string expression, for example when reading the printer's clock (also see Chapter 8.3). Example of how to use the printer as an alarm clock:

```
10    INPUT "Set Alarm"; A%
20    B%=VAL(TIME$)
30    IF B%>=A% THEN GOTO 40 ELSE GOTO 20
40    SOUND 880,100: END
RUN
```

8.3 Date and Time

Depending on model, the printer's CPU board is, or can optionally be, provided with a battery backed-up real-time clock (RTC).

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

The built-in calendar runs from 1980 through 2048 and corrects illegal values automatically, for example 031232 will be corrected to 040101.

The standard formats for date and time are:

Date:

`YYMMDD`, where...

- `YY` are the two last digits of the year
- `MM` are two digits representing the month (01-12)
- `DD` are two digits representing the day (01-28|29|30|31)

Time:

`HHMMSS`, where...

- `HH` are two digits representing the hour (00-23)
- `MM` are two digits representing the minute (00-59)
- `SS` are two digits representing the second (00-59)

In addition to the standard formats, other formats for date and time can be specified by the following instructions:

`FORMAT DATE$`	Specifies the format of date strings returned by DATE$ and DATEADD$.
`FORMAT TIME$`	Specifies the format of date strings returned by TIME$ and TIMEADD$.
`NAME DATE$`	Specifies the names of the months.
`NAME WEEKDAY$`	Specifies the names of the weekdays.

The following instructions are used to read the clock/calendar:

`<svar>=DATE$`	Returns the current date in standard format to a string variable.
`<svar>=DATE$("F")`	Returns the current date in the format specified by FORMAT DATE$ to a string variable.
`<svar>=TIME$`	Returns the current time in standard format to a string variable.
`<svar>=TIME$("F")`	Returns the current time in the format specified by FORMAT TIME$ to a string variable.

WEEKDAY$	Returns the name of the weekday of a specified date in plain text according to the weekday names specified by NAME WEEKDAY$, or—if such a name is missing—the full name in English.
WEEKNUMBER	Returns the week number of a specified date.
TICKS	Returns the time passed since last startup in $^1/_{100}$ seconds.

In most instructions, you can specify the current date or time using DATE$ or TIME$ respectively, for example:

```
WEEKDAY$ (DATE$)
TIMEDIFF (TIME$, "120000")
```

This example shows how the date and time formats are set and the names of months are specified. Finally, a number of date and time parameters printed to the standard OUT channel:

```
10    FORMAT DATE$ "MMM/DD/YYYY"
20    FORMAT TIME$ "hh.mm pp"
30    NAME DATE$ 1, "Jan":NAME DATE$ 2, "Feb"
40    NAME DATE$ 3, "Mar":NAME DATE$ 4, "Apr"
50    NAME DATE$ 5, "May":NAME DATE$ 6, "Jun"
60    NAME DATE$ 7, "Jul":NAME DATE$ 8, "Aug"
70    NAME DATE$ 9, "Sep":NAME DATE$ 10, "Oct"
80    NAME DATE$ 11, "Nov":NAME DATE$ 12, "Dec"
90    A%=WEEKDAY(DATE$)
100   PRINT WEEKDAY$(DATE$)+" "+DATE$("F")+" "+TIME$("F")
110   PRINT "Date:",DATE$("F")
120   PRINT "Time:",TIME$("F")
130   PRINT "Weekday:", WEEKDAY$(DATE$)
140   PRINT "Week No.:",WEEKNUMBER (DATE$)
150   PRINT "Day No.:", DATEDIFF ("030101",DATE$)
160   PRINT "Run time:", TICKS\6000;" minutes"
170   IF A%<6 THEN PRINT "It is ";WEEKDAY$(DATE$);
         ". Go to work!"
180   IF A%>5 THEN PRINT "It is ";WEEKDAY$(DATE$);
         ". Stay home!"
RUN
```

yields for example:

```
Monday Apr/03/2003 08.00 am
Date: Apr/03/2003
Time: 08.00 am
Weekday: Thursday
Week No.: 14
Day No.: 93
Run time: 1 minutes
It is Thursday. Go to work!
```

This example shows how the TICKS function is used to delay the execution for a specified period of time:

```
10    INPUT "Enter delay in sec's: ", A%
20    B%=TICKS+(A%*100)
30    GOSUB 1000
40    END
1000  SOUND 440,50                                    (Start signal)
1010  IF B%<=TICKS THEN SOUND 880,100 ELSE GOTO 1010
1020  RETURN
RUN
```

8.4 Random Number Generation

The Intermec Fingerprint firmware provides two instructions for generating random numbers, for example for use in test programs.

RANDOM

The RANDOM function generates a random integer within a specified interval.

This example tests a random dot on the printhead of a 12 dots/mm printer:

```
10    MIN%=HEAD(-7)*85\100: MAX%=HEAD(-7)*115\100
20    DOTNO%=RANDOM(0,1279)
30    IF HEAD(DOTNO%)<MIN% OR HEAD(DOTNO%)>MAX% THEN
40    BEEP
50    PRINT "ERROR IN DOT "; DOTNO%
60    ELSE
70    BEEP
80    PRINT "HEADTEST: OK!"
90    END IF
RUN
```

RANDOMIZE

To obtain a higher degree of randomization, the random number generator can be reseeded using the RANDOMIZE statement. You can either include an integer with which the generator will be reseeded, or a prompt will appear asking you to do so.

This example prints a random pattern of dots after the random number generator has been reseeded:

```
10    RANDOMIZE
20    FOR Q%=1 TO 100
30    X%=RANDOM(50,400)
40    Y%=RANDOM(50,400)
50    PRPOS X%,Y%
60    PRLINE 5,5
70    NEXT
80    PRINTFEED
RUN
```

yields:

```
Random Number Seed (0 to 99999999) ?          (prompt)
```

9 Label Design

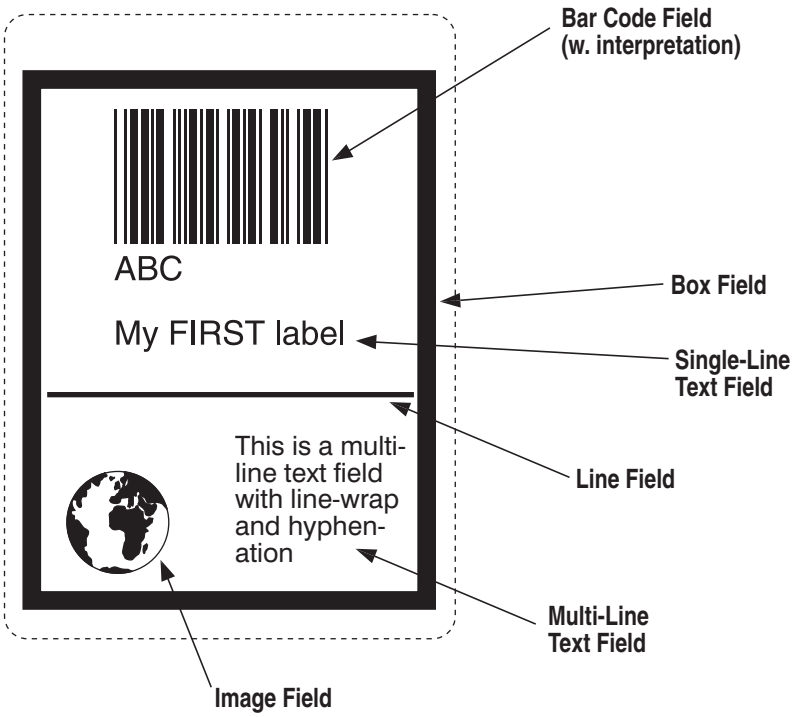
This chapter describes the various types of fields that can be used to create a label layout.

9.1 Creating a Layout

Field Types

A label layout is made up of a number of fields. There are six different types of fields:

- **Single Line Text Field**
  A single line text field consists of a single line of text.
- **Multi-Line Text Field**
  A multi-line text field consists of one or more lines of text with line-wrap and hyphenation, optionally surrounded by a black border line.
- **Bar Code Field**
  A bar code field consists of a single bar code, with or without a bar code interpretation in human readable characters.
- **Image Field**
  An image field is a picture, drawing, logotype, or other type of illustration.
- **Box Field**
  A box field is a rectangular blank area surrounded by a black border line. If the border is sufficiently thick, the whole area may be black.
- **Line Field**
  A line field is a black line that goes either along or across the paper web. A short but thick line can look like a black box.



Origin

The positioning of all fields on the label uses a common system. The starting point is called “origin” and is the point on the media that corresponds to the innermost active dot on the printhead at the moment when the PRINTFEED statement is executed.

The location of the origo is affected by the following factors:

- **Position across the media (X-axis):**
  The position of the origin is determined by the X-Start value in the Setup Mode.
- **Position along the media (Y-axis):**
  The position of the origin is determined by the Feed adjustment in the Setup Mode and any FORMFEED<nexp> statements executed before the current PRINTFEED statement or after the preceding PRINTFEED statement.

Coordinates

Starting from the origin, there is a coordinate system where the X axis runs across the media path from left to right (as seen when facing the printer) and the Y-axis runs along the media path from the printhead and back towards the media supply.

Units of Measure

The unit of measure is always “dots”, which means that all measures depend on the density of the printhead.

In a printer with a 11.81 dots/mm (300 dots/inch) printhead, a dot is 1/11.81 mm = 0.0847 mm = 0.00333 inches or 3.33 mils.

In a printer with an 8 dots/mm printhead (203.2 dots/inch), a dot is 1/8 mm = 0.125 mm = 0.00492 inches or 4.92 mils.

This implies that a certain label, originally designed for 11.81 dots/mm, would be printed larger in a 8 dots/mm printer. However, fonts are specified in points (not dots) and will thus be the same size regardless of printhead density.

A dot has the same size along both the X-axis and the Y-axis.

Insertion Point

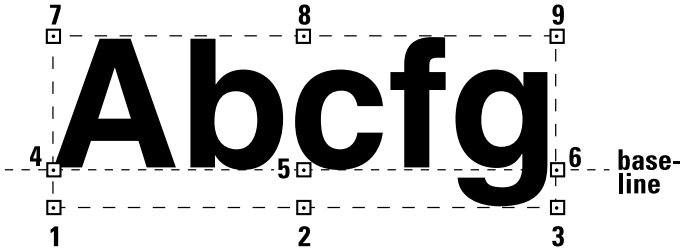
The insertion point of any field is specified within this coordinate system using a PRPOS<x-pos>,<y-pos> statement. For example, the statement PRPOS 100, 200 means that the object will be inserted at a position 100 dots to the right of the origin and 200 dots further back along the media path.

Alignment

Once the insertion point is specified, you must also decided which part of the field should match the insertion point. For example, a single-line text field forms a rectangle. There are 8 anchor points along the borders and one in the center. The anchor points are numbered 1-9 and specified using an ALIGN statement. By specifying for example ALIGN 1, you will place the lower left corner of the text field at the insertion point specified by the PRPOS statement.

The illustrations below and on next page show the anchor points for the various types of fields. Refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual*, ALIGN statement for detailed information on the anchor points of such bar codes, where the interpretation is an integrated part of the bar code pattern, for example EAN and UPC codes.

Single-Line Text Field



Bar Code Field

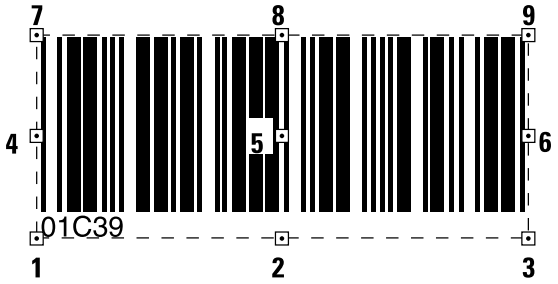
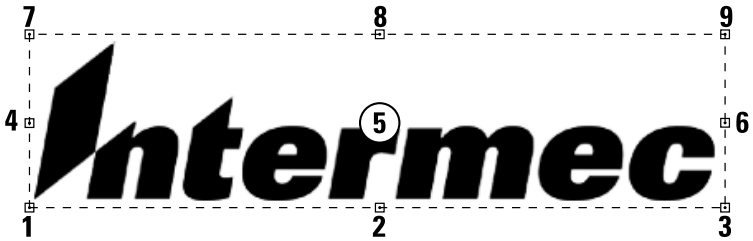
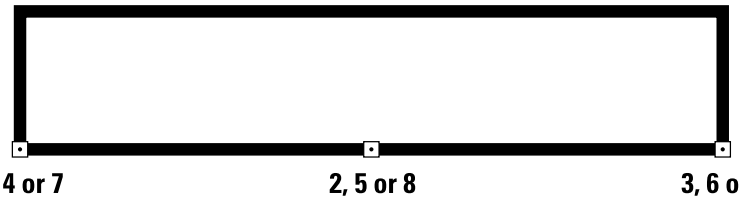


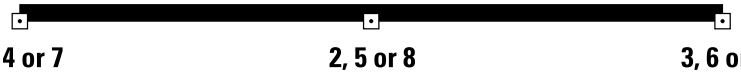
Image Field



Box Field



Line Field

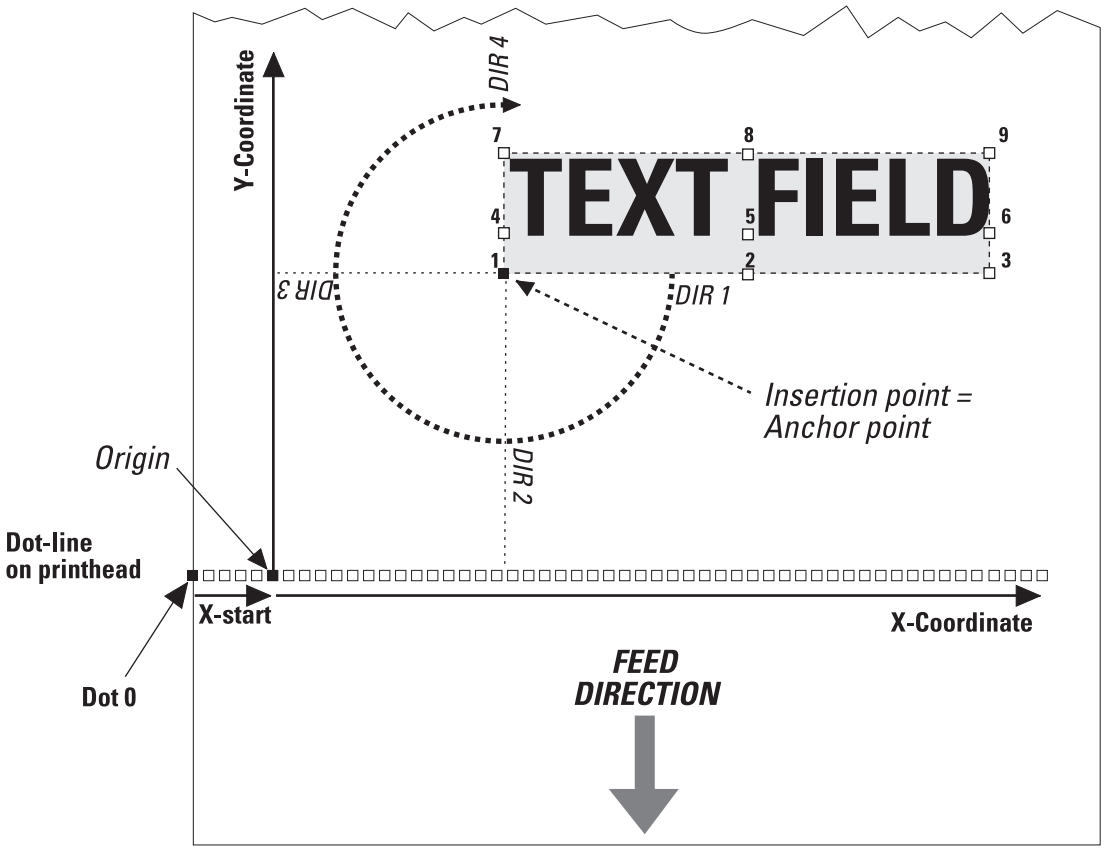


Multi-Line Text Field

Note: In case of multi-line text field, the alignment decides both the anchor point for box that surrounds the field and the alignment of the text lines inside the box. The box can be visible or invisible.

Directions

Intermec Fingerprint allows printing in four different directions. By default, all fields run across the media from left to right (DIR 1). Using a DIR statement, you can rotate the printable object clockwise around the anchor point/insertion point with a 90° increment (0°, 90°, 180°, or 270°), as illustrated below:



Partial Fields

Using the statement CLIP ON, you can make the firmware accept fields extending outside the print window (as specified by the printer's setup in regard of X-Start, Width, and Length). However, all parts of the fields outside the borders of the print window will be excluded (clipped) from the printout.

The clipping of bar codes requires further specification of the CLIP statement (CLIP BARCODE HEIGHT|INFORMATION|X|Y), see the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

The default setting is CLIP OFF, which means that any field extending outside the print window will cause an error condition (Error 1003, "Field out of label").

XOR Mode

Using the statement XORMODE ON/OFF you can control whether crossing lines should result in the intersection being printed black (XORMODE OFF) or white (XORMODE ON). XORMODE OFF is default.

Layout Files

In addition to the method described above, there is an alternative method using files for specifying the various fields and their input data separately as described in Chapter 9.8. However, the various parameters of the layout file are based on the same principles as described in Chapters 9.1-7.

Checking Current Position

After having positioned and specified an object, you can find out the current position of the insertion point by means of a PRSTAT function. This implies that after for example having entered a line of text, you can find out how long it will be and where any new object will be placed unless a new position is specified.

- In print direction 1 or 3, PRSTAT (1) returns the absolute value of the insertion point along the X-axis, whereas PRSTAT (2) returns the Y-value of the last executed PRPOS statement.
- In print direction 2 or 4, PRSTAT (2) returns the absolute value of the insertion point along the Y-axis, whereas PRSTAT (1) returns the X-value of the last executed PRPOS statement.

Example: An unknown number of logotypes will be printed with 10 dots spacing across the media path. The size of the logotype is not known. To avoid an “field out of label” error, a limitation in regard of media width is included (line 80, change if necessary).

```
10    PRPOS 0,50
20    PRIMAGE "GLOBE.1"
30    X%=PRSTAT(1)
40    FOR A%=1 TO 10
50    Z%=PRSTAT(1)
60    PRPOS Z%+10,50
70    PRIMAGE "GLOBE.1"
80    IF Z%>550 THEN GOTO 100
90    NEXT
100   PRINTFEED
110   END
RUN
```

Note: The PRSTAT function can also be used for checking the printer’s status in regard of a number of conditions, see Chapter 15.3.

Checking the Size and Position of a Field

The statements RENDER ON and RENDER OFF enables/disables “rendering”, which means processing Fingerprint instructions to a bitmap pattern that can be printed by the printhead. By combining RENDER OFF with various PRSTAT function, you can find out the size and position of a field without actually printing it on a label.

9.2 Single-Line Text Field

A single-line text field consists of one or several alphanumeric characters on the same line (no practical limit.)

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a single-line text field can contain the following instructions:

FONT (FT) and FONTD

Specifies the single- or double-byte font to be printed respectively. Default choice is the single-byte font Swiss 721 BT in 12 points size, no slant, and the width 100% of the height. Once a font has been specified, it will be used for all text until a new FONT or FONTD statement is executed.

NORIMAGE (NI) / INVIMAGE (II)

Normally, text is printed in black on a transparent background (NORIMAGE). Using INVIMAGE, the printing can be inversed so the text will be transparent, whereas the background will be black. The size of the background is decided by the character cell. A NORIMAGE statement is only needed when changing back from INVIMAGE printing.

PRTXT (PT)

Text can be entered in the form of numeric expressions and/or string expressions. Two or more expression can be combined using semicolons (;) or, in case of string expressions, by plus signs (+). String constants must be enclosed by quotation marks ("..."). Variables are useful for printing for example time, date, or various counters, and when the same information is to appear in several places, for example both as plain text and as bar code input data.

Summary

To print a single-line text field, the following information and instructions must be given (default values will substitute missing parameters):

Purpose	Instruction	Default	Remarks
X/Y Position	PRPOS (PP)	0/0	Number of dots
Alignment	ALIGN (AN)	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Typeface	FONT (FT)	Swiss 721 BT,12,0,100	
	FONTD	–	
Style	INVIMAGE (II)	no	White on black print
	NORIMAGE (NI)	yes	Black on white print
Text	PRTXT (PT)	–	Field input data
Print a label	PRINTFEED (PF)	–	Resets parameters to default

Example:

```
10    PRPOS 100,200
20    ALIGN 7
30    DIR 2
40    FONT "Swiss 721 Bold BT,20,15,80"
50    INVIMAGE
60    PRTXT "HELLO"
70    PRINTFEED
RUN
```

9.3 Multi-Line Text Field

A multi-line text field consists of max. 20 lines of text with max. 300 single-byte characters on each line.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a multi-line text field can contain the following instructions:

FONT (FT)

Specifies the singlebyte font to be printed. Default choice is the font Swiss 721 BT in 12 points size, no slant, and the width 100% of the height. Once a font has been specified, it will be used for all text until a new FONT statement is executed.

NORIMAGE (NI) / INVIMAGE (II)

Normally, text is printed in black on a transparent background (NORIMAGE). Using INVIMAGE, the printing can be inversed so the text will be transparent, whereas the background will be black. The size of the background is decided by the character cell. A NORIMAGE statement is only needed when changing back from INVIMAGE printing.

PRBOX (PX)

Multi-line text fields can be created using an extension of the PRBOX statement. The PRBOX statement allows you to specify the height, width, and line thickness of a box in which the text will be printed. Depending on the line thickness, the box will be invisible (thickness = 0) or get a black border line (thickness >0). Additional parameters allows you to position the text inside the box, decide the line spacing, and control the hyphenation.

Note that the ALIGNment also will affect the positioning of the text insided the box, see Chapter 9.1.

When a text line reaches the border of the box, it will wrap to a new line according to the hyphenation settings.

Refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual* for comprehensive explanations of positioning and hyphenation.

Summary

To print a multi-line text field, the following information and instructions must be given (default values will substitute missing parameters):

Purpose	Instruction	Default	Remarks
X/Y Position	PRPOS (PP)	0/0	Number of dots
Alignment	ALIGN (AN)	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Typeface	FONT (FT)	Swiss 721 BT,12,0,100	
	FONTD	–	
Style	INVIMAGE (II)	no	White on black print
	NORIMAGE (NI)	yes	Black on white print
Text	PRTXT (PT)	–	Field input data
Print a label	PRINTFEED (PF)	–	Resets parameters to default

Example:

```
10    DIR 1
20    ALIGN 8
30    R$="Hyphen&Sated words will be divid&Sed
      into sylla&Sbles."
40    NL$="NEWLINE"
50    S$="&S&Special Cases and EXTRAORDINARY long
      words."
60    T$=R$+NL$+S$
70    PRPOS 300,300
80    PRBOX 700,500,20,T$,25,1,NL$,"&S - +"
90    PRINTFEED
RUN
```

9.4 Bar Code Field

As standard, Intermec Fingerprint v8.00 supports 39 of the most common bar code symbologies including two-dimensional bar codes and dot codes like PDF417 and MaxiCode. Each bar code (optionally including its human readable interpretation) makes up a bar code field.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a bar code field can contain the following instructions:

BARSET

This statement specifies the type of bar code and how it will be printed and can, if so desired, replace the following statements:

`BARHEIGHT (BH)` Height of the bars in the code

`BARRATIO (BR)` Ratio between wide and narrow bars

`BARTYPE (BT)` Bar code type

`BARMAG (BM)` Enlargement

The BARSET statement contains optional parameters for specifying complex 2-dimensional bar or dot codes, for example PDF417 (see *Intermec Fingerprint v8.00, Programmer's Reference Manual)*.

For common one-dimensional bar codes the following parameters should be included in the statement:

- Bar code type — Name must be given according to list in Chapter 12.1 and be enclosed by quotation marks. Default: "INT2OF5"
- Ratio (wide bars) — Default: 3
- Ratio (narrow bars) — Default: 1
- Enlargement — Affects the bar pattern but not the inter-pretation, unless the bar font is an inte-grated part of the code, for example EAN/UPC. Default: 2
- Height — Height of the bars in dots. Default: 100.

BARFONT...ON

Specifies the single-byte font to be used for the bar code interpretation (human readables). In some bar codes (for example EAN/UPC), the interpretation is an integrated part of the code.

The bar font can be specified in regard of:

- Font Default: — Swiss 721 BT
- Size in points — Default: 12 points.
- Slant in degrees — Default: 0.
- Vertical offset — Specifies the distance in dots between the bottom of the bar pattern and the top of the interpretation characters. Default: 6.
- Height Magnification — Default: 1
- Width Magnification — Default: 1
- Width Enlargement — Default: 100 (%)
- ON — Enables the printing of the interpretation.

BARFONT OFF
To disable bar code interpretation printing, use BARFONT OFF.

PRBAR (PB)
Input data to be used to generate the bar code can be entered in the form of numeric and/or string expressions depending on type of bar code. String constants must be enclosed by quotation marks. Variables are useful for printing, for example time, date, or various counters, and when the same information is to appear in several places, for example both as plain text and as bar code input data.

Summary
To print a bar code field, the following information and instructions be must given (in most cases default values will substitute missing information):

Purpose	Instruction	Default	Remarks
X/Y Position	PRPOS (PP)	0/0	Number of dots
Alignment	ALIGN (AN)	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Bar Code Select	BARSET	see above	
Human Readables	BARFONT...ON	see above	Can be omitted
Input Data	PRBAR (PB)	–	Input data to bar code field
Print a label	PRINTFEED (PF)	–	Resets parameters to default

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

9.5 Image Field

An image field is a field containing an image in the internal bitmap format of Intermec Fingerprint.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, an image field can contain the following instructions:

MAG
Images can be magnified 1-4 times independently in regard of height and width.

NORIMAGE (NI) / INVIMAGE (II)
Normally, images are printed as created, that is in black on a transparent background (NORIMAGE). Using INVIMAGE the black and non-printed background can switch colors. The size of the background is decided by the size of the image. A NORIMAGE statement is only needed when changing back from INVIMAGE printing.

PRIMAGE (PM)
Specifies the image by name in the form of a string expression. A string constant must be enclosed by quotation marks. A string variable may be useful when the same image is to appear in several places. The extension indicates the suitable directions:

Extension .1 matches DIR 1 and DIR 3
Extension .2 matches DIR 2 and DIR 4

Summary
To print an image field, the following instructions must be given (in most cases default values will substitute missing information):

Purpose	Instruction	Default	Remarks
X/Y Position	PRPOS (PP)	0/0	Number of dots
Alignment	ALIGN (AN)	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Magnification	MAG	1,1	
Style	INVIMAGE (II)	no	White-on-black
	NORIMAGE	yes	Black-on-white
Image	PRIMAGE (PM)	–	.1 or .2 depending on direction
Print a label	PRINTFEED (PF)	–	Resets parameters to default

Example:

```
10    PRPOS 50,50
20    ALIGN 9
30    DIR 3
40    MAG 2,2
50    INVIMAGE
60    PRIMAGE "GLOBE.1"
70    PRINTFEED
RUN
```

9.6 Box Field

A box is a hollow square or rectangle that can be rotated with an increment of 90° according to the print direction. If the line thickness is sufficiently large, the box will appear to be filled (another method is to print an extremely thick short line).

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a box field can only contain the following instruction:

PRBOX (PX)

Specifies the size of the box in regard of height, width and line weight (thickness) in dots. Also used for multi-line text fields, see Chapter 9.3.

Summary

To print a box, the following information and instructions must be given (in some cases default values will substitute missing information):

Purpose	Instruction	Default	Remarks
X/Y Position	PRPOS (PP)	0/0	Number of dots
Alignment	ALIGN (AN)	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Box specifications	PRBOX (PX)	–	Height, width, and line weight in dots
Print a label	PRINTFEED (PF)	–	Resets parameters to default

Example:

```
10    PRPOS 250,250
20    ALIGN 1
30    DIR 3
40    PRBOX 200,200,10
50    PRINTFEED
RUN
```

9.7 Line Field

A line can be printed in right angles along or across the media path according to the print direction.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a line field can only contain the following instruction:

PRLINE (PL)

Specifies the size of the line in regard of length and line weight (thickness) in dots.

Summary

To print a line, the following information and instructions must be given (in some cases default values will substitute missing information):

Purpose	Instruction	Default	Remarks
X/Y Position	PRPOS (PP)	0/0	Number of dots
Alignment	ALIGN (AN)	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Line specifications	PRLINE (PL)	–	Length and line weight in dots
Print a label	PRINTFEED (PF)	–	Resets parameters to default

Example:

```
10    PRPOS 100,100
20    ALIGN 1
30    DIR 4
40    PRLINE 200,10
50    PRINTFEED
RUN
```

9.8 Layout Files

Introduction

Many application, for example in connection with booking and ticketing, require the label layout as well as variable input data and logotypes to be sent to the printer as files or arrays. This method requires less programming in the printer and less data to be transferred between printer and host, but some kind of overhead program in the host, that handles file transfers as well as the input of data, is of great help.

The program instruction is a statement called LAYOUT. Before using this statement, a number of files or arrays must be created.

Creating a Layout File

The basis of the method is a layout file in random format, that contains a number of records, each with a length of 52 bytes.

Each record starts with a 2-byte hexadecimal element number (bytes 0 and 1) which is used to link the layout record with a variable input record or a record in a layout name file as explained later.

Byte 2 contains a single character that specifies the type of record:

A = Logotype (specified by its name)

B = Bar Code

C = Character (single-line text)

E = Bar Code Extended File
Corresponds to the last six parameters in the BARSET statement. Must have lower element number than the corresponding bar code record (B).

H = Bar Code Font

J = Baradjust (corresponds to BARADJUST statement)

L = Logotype (specified by its number)

S = Separation line

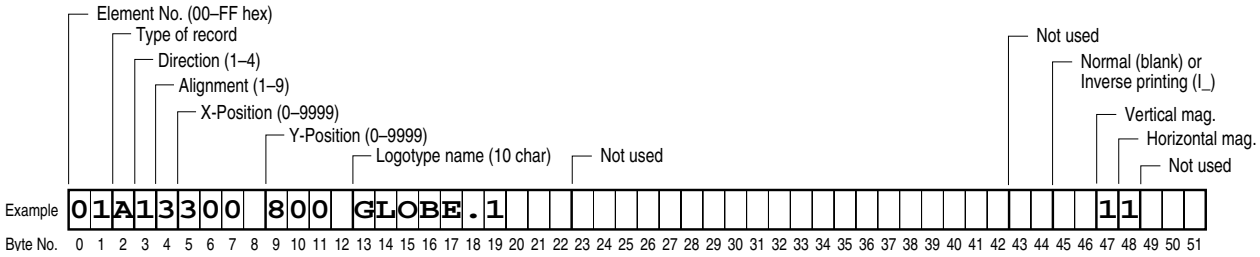
X = Box

The remaining bytes are used differently depending on type of record and may specify direction, position, font, etc. Each such instruction corresponds to a Fingerprint instruction, for example direction corresponds to DIR statement, alignment to ALIGN, x- and y-positions to PRPOS, etc. Note that there are only10 bytes available for the font and bar font names. Since most names of standard fonts are longer, you may need to use font aliases.

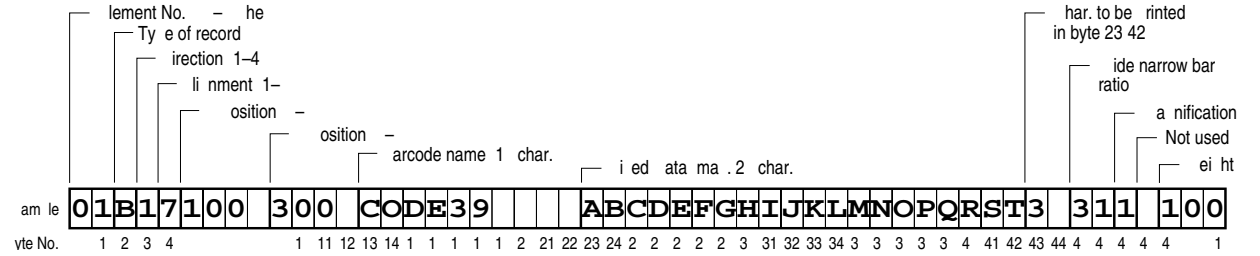
Text and bar code records can contain both fixed and variable data. The fixed data (max. 20 characters) are entered in the layout record. A parameter (bytes 43 and 44) specifies how many characters (starting from the first character) of the fixed data that will be printed or used to generate the bar code. Possible variable data will be appended to the fixed data at the position specified in bytes 43 and 44.

Syntax of layout file records

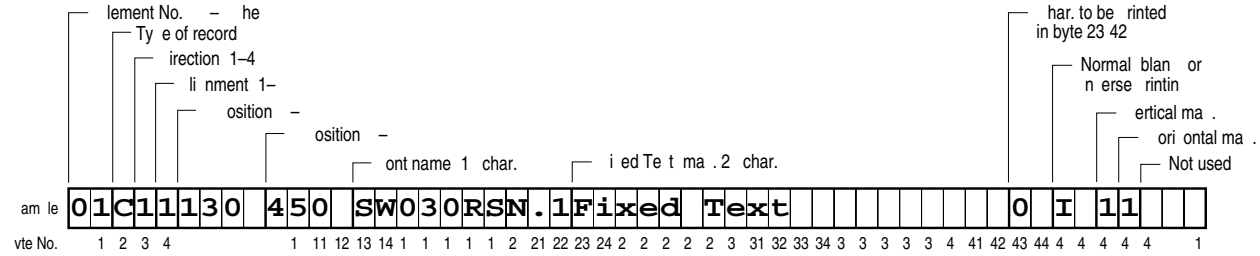
LOGOTYPE RECORD (by name):



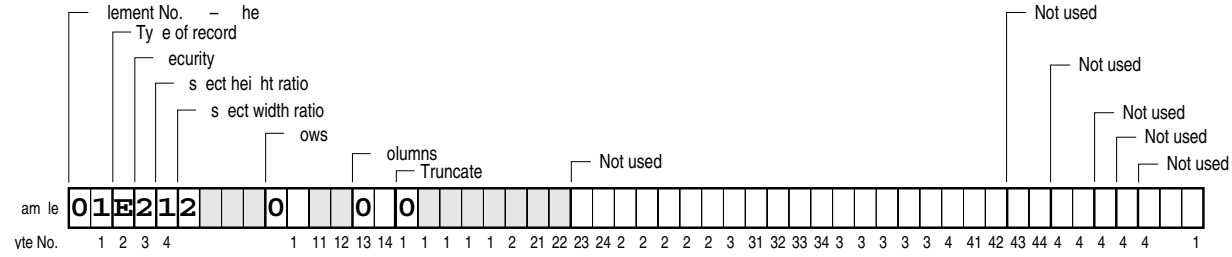
BAR CODE RECORD:



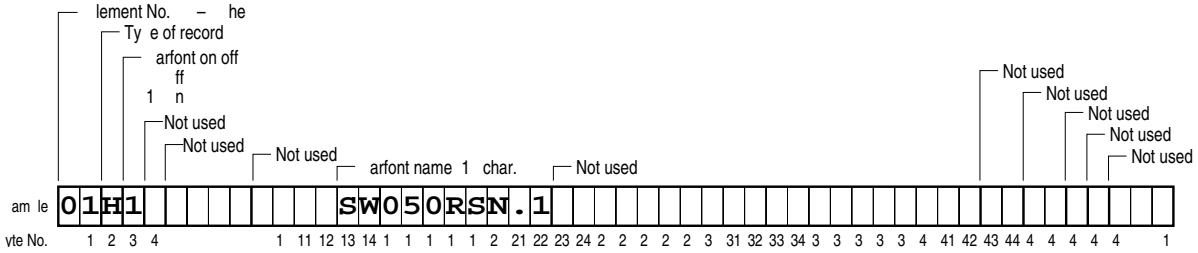
TEXT RECORD:



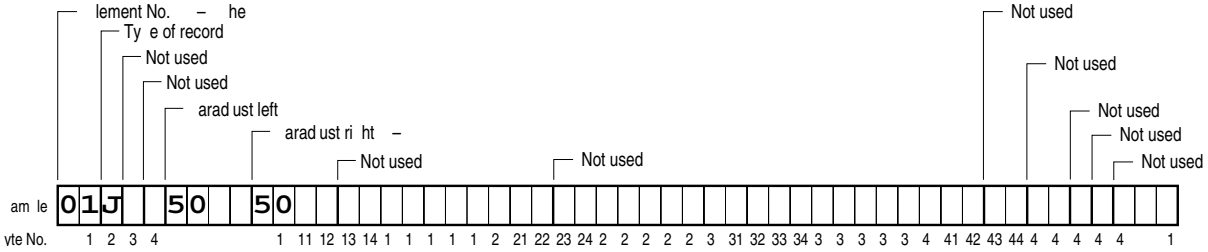
BAR CODE EXTENDED FIELD RECORD:



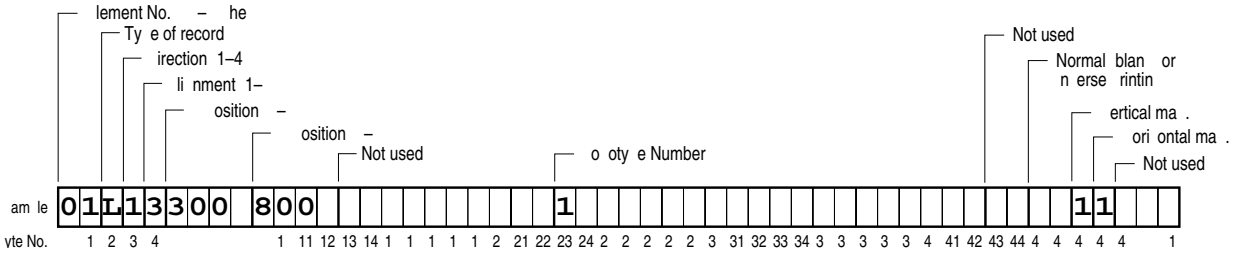
BAR CODE INTERPRETATION RECORD:



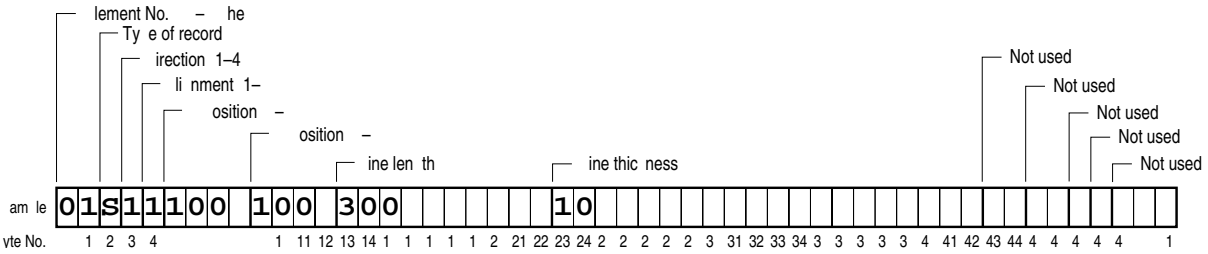
BARADJUST RECORD:



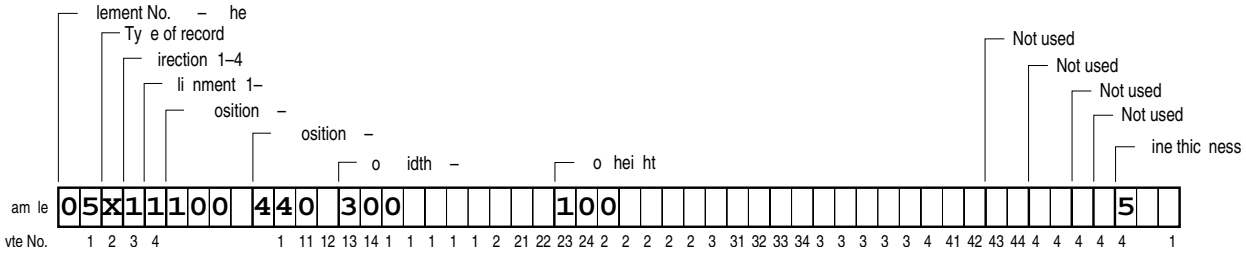
LOGOTYPE RECORD (by number):



LINE RECORD:



BOX RECORD:



This example shows how a small layout file can be composed:

```
10 OPEN "LAYOUT.DAT" FOR OUTPUT AS 2
20 PRINT #2, "01H1         FONT1                                    ";
30 PRINT #2, "02C11100 650 FONT1     Fixed Text          11I 22   ";
40 PRINT #2, "02C11130 450 FONT1     Fixed Text          0   11   ";
50 PRINT #2, "03B17100 300 CODE39    ABC                 3 311 100";
60 PRINT #2, "04A12300 800 GLOBE.1                           11   ";
70 PRINT #2, "05X11100 440 300      100                        5  ";
80 PRINT #2, "06S11100 100 300      10                           ";
90 CLOSE 2
```

There are certain rules that should be observed:

- Each record must be exactly 52 bytes long and be appended by a semicolon (;).
- It is essential that the different types of data are entered exactly in the correct positions. Any input in unused bytes will be ignored.
- The records are executed in the order they are entered. The reference number at the start of each record does not affect the order of execution. This implies that a barfont record will affect all following bar code records, but not those already entered.
- When using bar code interpretation, do not enter a bar code record directly after a record with inverse printing, since the bar code interpretation will be inversed as well. A text or logotype record without inverse printing between the bar code record and the inversed record will reset printing to normal.
- If a magnification larger than 9 is required, you cannot enter it as a digits, because there is only one byte available. Instead, enter the character, the ASCII decimal number of which minus 48 corresponds to the desired magnification. Thus, if magnification 10 is desired, enter the colon character (:), because its ASCII number (58 dec) minus 48 = 10.

Creating a Logotype Name File

Next step is to create a logotype name file. This is a necessary step even if you are not going to use any logotype in your layout (in this case the file can be empty.) In the layout file, you can set a logotype record to use logotypes specified either by name or by number.

- If you specify logotype-by-name (record type A), the printer's entire memory will be searched for an image with the specified name. A logotype-by-name file is composed by a number of records with a length of 10 bytes each that contain the image names, for example:

```
10    OPEN "LOGNAME.DAT" FOR OUTPUT AS 1
20    PRINT#1, "GLOBE.1 "
30    PRINT#1, "GLOBE.2 "
40    PRINT#1, "DIAMONDS.1"
50    PRINT#1, "DIAMONDS.2";
60    CLOSE 1
```

Note: The last record in a sequential file must be appended by a semicolon (;).

- If you specify logotype-by-number (record type L), you must have a logotype name file. A logotype-by-number file is composed by a number of records with a length of 13 bytes each. The first 2 bytes is a reference number (0-99), the third byte is always a colon (:), and the following 10 bytes are used for the image name:

```
10    OPEN "LOGNAME.DAT" FOR OUTPUT AS 1
20    PRINT#1, "0 :GLOBE.1 "
30    PRINT#1, "1 :GLOBE.2 "
40    PRINT#1, "2 :DIAMONDS.1"
50    PRINT#1, "3 :DIAMONDS.2";
60    CLOSE 1
```

Note: The last record in a sequential file must be appended by a semicolon (;).

Creating a Data File or Array

You will also need a data file or data array. If you use a data file, you must use an error file, and if you use a data array, you must use an error array. This file or array contains variable data that will be placed in the position specified by the layout. Each data record starts with a hexadecimal element number (00-FF hex) that links the data to the layout record or records that start with the same element number. Thus you can for example use a single data record to generate a number of text fields with various locations and appearances as well as to generate a bar code.

If you for some reason do not use variable data, you will still need to create either an empty data file or an empty data array.

- Create a data array like this:

```
10    DIM LAYDATA$(7)
20    LAYDATA$(0)="01Mincemeat"
30    LAYDATA$(1)="0AVeal"
40    LAYDATA$(2)="17Roast Beef"
50    LAYDATA$(3)="3FSausages"
60    LAYDATA$(4)="02Venison"
70    LAYDATA$(5)="06Lamb Chops"
80    LAYDATA$(6)="7CPork Chops"
```

- You can create a data file with the same content in a similar way:

```
10    OPEN "LAYDATA.DAT" FOR OUTPUT AS 1
20    PRINT#1,"01Mincemeat"
30    PRINT#1,"0AVeal"
40    PRINT#1,"17Roast Beef"
50    PRINT#1,"3FSausages"
60    PRINT#1,"02Venison"
70    PRINT#1,"06Lamb Chops"
80    PRINT#1,"7CPork Chops";
90    CLOSE 1
```

Note: The last record in a sequential file must be appended by a semicolon (;).

Creating an Error File or Array

The last requirement is an error file or array that can store any errors that may occur. If you use a data array, you must use an error array, and if you use a data file, you must use an error file. The following errors will be stored and presented in said order:

1. If an error occurs in a layout record, the number of the record (1...nn) and the error number is placed in the error array or file.
2. If a data record cannot be used in a layout record, an the index of the unused data record (0...nn) plus the error code -1 is placed in the error array or file.

- Error arrays must be large enough to accommodate all possible errors. Thus, use a DIM statement to specify a one-dimensional array with a number of elements that is twice the sum of all layout records plus twice the sum of all data records. You should also include some routine that reads the array, for example:

```
10    DIM QERR%(28)
20    QERR%(0)=0
.....
190   IF QERR%(1)=0 THEN GOTO 260
200   PRINT "-ERROR- LAYOUT 1"
210   I%=0
220   IF QERR%(I%)=0 THEN GOTO 260
230   PRINT "ERROR ";QERR%(I%+1);" in record"
      ;QERR%(I%)
240   I%=I%+2
250   GOTO 220
260   PRINTFEED
```

- Error files require a little more programming to handle the error message, for example:

```
220   OPEN "ERRORS.DAT" FOR INPUT AS 10
230   IF EOF(10) THEN GOTO 280 ELSE GOTO 240
240   FOR A%=1 TO 28
250   INPUT #10, A$
260   PRINT A$
270   NEXT A%
280   PRINTFEED
```

Note: The loop in line 240 must be large enough to accommodate all possible errors.

Using the Files in a LAYOUT Statement

Now, you have all the files you need to issue a LAYOUT statement. This statement combines the layout file, the logotype file, the data file/array, and the error file/array into a printable image. Depending on whether you have selected to use data and error files or arrays, the statement will have a somewhat different syntax:

Files:
LAYOUT F, <layout file>, <logotype file>,<data file>,<error file>

Arrays:
LAYOUT <layout file>,<logotype file>,<data array>,<error array>

Note: You cannot omit any file or array, since the syntax requires a file name or array designation in each position. If you, for example, do not require any logotype, you must still create an empty logotype file.

Example: The example below shows a simple layout created using the layout statement in combination with data and error arrays:

```
10   DIM QERR%(28)
20   LAYDATA$(0)="02Var. input"
30   LAYDATA$(1)="03 PRINTER"
40   QERR%(0)=0
50   OPEN "LOGNAME.DAT" FOR OUTPUT AS 1
60   PRINT #1, "GLOBE.1";
70   CLOSE 1
80   REM:LAYOUT FILE
90   OPEN "LAYOUT.DAT" FOR OUTPUT AS 2
100  PRINT #2, "01H1         FONT1                                    ";
110  PRINT #2, "02C11100 650 FONT1     Fixed Text          11I 22   ";
120  PRINT #2, "02C11130 450 FONT1     Fixed Text          0   11   ";
130  PRINT #2, "03B17100 300 CODE39    ABC                 3 311 100";
140  PRINT #2, "04A12300 800 GLOBE.1                           11   ";
150  PRINT #2, "05X11100 440 300       100                       5  ";
160  PRINT #2, "06S11100 100 300       10                           ";
170  CLOSE 2
180  LAYOUT "LAYOUT.DAT","LOGNAME.DAT",LAYDATA$,QERR%
190  IF QERR%(1)=0 THEN GOTO 260
200  PRINT "-ERROR- LAYOUT 1"
210  I%=0
220  IF QERR%(I%)=0 THEN GOTO 260
230  PRINT " ERROR "; QERR%(I%+1); " in record "; QERR%(I%)
240  I%=I%+2
250  GOTO 220
260  PRINTFEED
RUN
```

10 Printing Control

This chapter describes the methods for controlling the media feed and for printing single labels as well as batches of labels.

10.1 Media Feed

There are a number of instructions for controlling the media feed without actually printing any labels:

`CLEANFEED`	Runs the printer's media feed mechanism in order to facilitate cleaning of the platen roller.
`FORMFEED`	Feeds out a blank label (or similar) or optionally feeds out or pulls back a specified amount of media without printing.
`TESTFEED`	Adjusts the label stop/black mark sensor while feeding out a number of blank forms or a specified lengh of media.
`LBLCOND`	Overrides the media feed setup.

The media is fed past the printhead by a rubber-coated platen roller driven by a stepper motor controlled by the firmware. The movement of the media is detected by the label stop/black mark sensor (LSS), except when various types of continuous stock are used.

The printer's setup in regard of Media; Media Size; Length and Media; Media Type is essential for how the media feed will work. There are four or five different types of Media Type options (also see the User's Guide):

- Label (w gaps)
- Ticket (w mark)
- Ticket (w gaps)
- Fix length strip
- Var length strip

When a FORMFEED, TESTFEED, or PRINTFEED statement is executed and the media is fed out, the LSS detects the front edge of each new label, the rear edge of each detection gap, or front edge of each black mark.

TESTFEED

By performing a TESTFEED operation after loading a new supply of media, the firmware is able to measure the distance between the forward edges of two consecutive labels, thereby determining the label length, and can adjust the media feed accordingly. The same principle applies to tickets or tags with detection gaps and tickets with black marks. To execute a TESTFEED at media load, simultaneously press <**Shift**> + <**Feed**> keys on the printer's keyboard.

In case of fixed or variable length strip, the LSS will only detect possible out-of-paper conditions. The length of media feed is decided in two different ways:

- **Fixed length strip**
  The amount of media feed for each FORMFEED, TESTFEED, and PRINTFEED operation is decided by the Media; Media Size; Length setup.
- **Variable length strip**
  The size of the print image decides the length of media to be fed out at the execution of a PRINTFEED. Note that a blank space character or a transparent part of an image is also regarded as a part of the print image. The length of TESTFEED and FORMFEED operations is decided by the Media; Media Size; Length setup.

The Feedadjust setup allows you to perform two global adjustments to the media feed described above:

- Start Adjust
- Stop Adjust

By default, both these two parameters are set to 0, which allows for proper tear-off operation when there is no requirement of printing immediately at the forward edge of the label.

- Start Adjust decides how much media will be fed out or pulled back before the FORMFEED, TESTFEED, or PRINTFEED is executed. Usually, there is a small distance between the tear bar and the printhead. Thus, if you for example want to start printing directly at the forward edge of the label, you must pull back the media before printing using a negative start adjust value.
- Stop Adjust decides how much of the media will be fed out or pulled back after the FORMFEED, TESTFEED, or PRINTFEED is executed.

So far we have only discussed how the media feed will work, regardless which program is run or what labels are printed. There are also several ways to let the program control the media feed without changing the setup:

FORMFEED

As already mentioned, if the FORMFEED statement is issued without any specification of the feed length, it will feed out a complete blank label (or the equivalent.) But the FORMFEED statement can also specified as a positive or negative number of dots. However, it is not recommended to use this facility to substitute or modify the global Start Adjust and Stop Adjust setup as a part of the program execution.

LBLCOND

The LBLCOND statement can be used to override the values for the Start Adjust and/or Stop Adjust set in the Setup Mode. It can also be used to disable the LSS/BMS for a specified length of media feed, for example to avoid text or pictures on the backside of a ticket being mistakenly detected as black marks, or when using irregularly shaped labels.

LBLCOND also allows you to choose between three modes for controlling the printing of very short labels, see the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

CLEANFEED

This instruction simply rotates the platen roller forward or backward as specified equivalent to FORMFEED, but it will work regardless of any error conditions, such as "printhead lifted".

It is recommended to use CLEANFEED for removing labels stuck on the platen roller instead of forcing the platen roller to rotate by pulling the media out, which could damage the electronics. Another method is to lift the printhead and press the printer's <**Feed**> key.

10.2 Printing

The following instructions are used in connection with the actual printing:

`CUT`	Activates an optional paper cutter.
`CUT ON/OFF`	Enables/disables automatic cut-off operation in connection with each PRINTFEED statement if an optional paper cutter is fitted.
`LTS& ON/OFF`	Enables/disables an optional label-taken sensor.
`PRINT KEY ON/OFF`	Enables/disables PRINTFEED execution by pressing the <**Print**> key.
`PRINTFEED`	Prints a single label, ticket, tag or piece of strip, or a batch of labels, tickets etc.

CUT

Activates an optional paper cutter. As opposed to the CUT ON/ OFF statement (see below), this statement allows you to control the cutter independently from the PRINTFEED statements. The relation between the media and the cutting edge when a CUT statement is executed decides where the media will be cut off. Since there is a longer distance from the printhead to the cutting edge than to the tear bar, the media feed may need to be adjusted using the Start- and Stopadjust setup.

CUT ON/OFF

Enables/disables automatic cut-off by an optional paper cutter initiated by each PRINTFEED statement and also allows you to decide how many dot the media will be fed out before cutting and be pulled back afterwards.

LTS& ON/OFF

These statements enables or disables an optional label-taken sensor, which is an photo-electrical sensor that detects when a label has not been removed from the printer's outfeed slot, and holds the printing until the label has been removed.

PRINT KEY ON|OFF

These two instructions can only be issued in the Immediate Mode and in the Intermec Direct Protocol and enables/disables a single PRINTFEED operation to be automatically executed each time the <**Print**> key on the printer's built-in keyboard is pressed.

PRINTFEED (PF)

At the execution of a PRINTFEED statement, the firmware renders all previously executed field-related instructions into a bitmap pattern. The bitmap pattern controls the heating of the printhead dots and the stepper motor that feeds the media past the printhead. By default, each PRINTFEED statement produces one single copy, but the size of a batch of copies can optionally be specified. You can also specify the number of copies to be reprinted after an interruption. The relation between media and printhead when the PRINTFEED statement is executed decides the positioning of the printout along the media.

After the execution of a PRINTFEED statement, the following statements are reset to their respective default values:

Statement	Default
ALIGN	1
BARFONT	"Swiss 721 BT", 12, 0, 6, 1,100, OFF
BARFONT ON/OFF	OFF
BARHEIGHT	100
BARMAG	2
BARRATIO	3, 1
BARSET	"INT2OF5", 3, 1, 2, 100, 2, 1, 2, 0, 0
BARTYPE	"INT2OF5"
DIR	1
FONT	"Swiss 721 BT", 12, 0, 100
FONTD	none
INVIMAGE	NORIMAGE
MAG	1, 1
PRPOS	0, 0

This does only affect new statements executed after the PRINTFEED statement, but not already executed statements. The amount of media fed out at the execution of a PRINTFEED statements under various conditions is discussed in Chapter 10.1.

Example (printing five identical labels):

```
10    PRPOS 100,100
20    FONT "Swiss 721 Bold BT",14,10,80
30    PRTXT "TEST LABEL"
40    PRINTFEED 5
RUN
```

Example (printing five copies of the same label layout with consecutive numbering):

```
10    FOR A%=1 TO 5
20    PRPOS 100, 100
30    FONT "Swiss 721 Bold BT",14,10,80
40    PRTXT "LABEL ";A%
50    PRINTFEED
60    NEXT A%
RUN
```

10.3 Length of Last Feed Operation

ACTLEN

This function returns the approximate length in dots of most recently executed media feed operation. It can for example be used to determine the length of the labels before printing a list, so the list can be divided into portions that fit the labels.

Example:

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

10.4 Batch Printing

The term “Batch Printing” means the process of printing several labels without stopping the media feed motor between the labels. The labels may be exact copies or differ more or less in appearance.

When a PRINTFEED is executed, the firmware renders the program instructions into a bitmap pattern and stores it in one of the two image buffers in the printer’s temporary memory. The image buffer compensates for differences between processing time and printing time.

Next step is to use the bitmap pattern to control the heating of the printhead dots while the ribbon and/or media is fed past the printhead. Obviously, a high print speed causes the image buffer to be emptied faster.

Normally, when the first image buffer is emptied and the printing is completed, the firmware can process a new bitmap pattern and store it in the second image buffer. Using an OPTIMIZE "BATCH" ON statement, you can make the firmware start processing next label image and store it in the second image buffer while the first label is still in process of being printed. Thus, by switching between the two image buffers, a high continuous print speed can be maintained.

There are a number of instructions that facilitate batch printing:

`FIELDNO`	Divides the program into portions that can be cleared individually.
`CLL`	Clears part or all of the image buffer.
`OPTIMIZE"BATCH"ON`	Enables optimizing.
`OPTIMIZE"BATCH"OFF`	Disables optimizing.

When using batch printing, consider this:
- The program must be written so as to allow batch printing.
- In case of small differences between labels, make use of CLL and FIELDNO instructions and write the program so the variable data are processed last.
- Always use the OPTIMIZE “BATCH” ON strategy.

Should the printer stop between labels, lower the print speed somewhat. Usually, the overall time to produce a certain number of labels is more important than the actual print speed.

CLL & FIELDNO

The image buffer stores the bitmap pattern of the label layout between processing and printing. The image buffer can be cleared partially or completely using a CLL statement.

- Complete clearing is obtained by a CLL statement without any reference to a field (see below) and is useful to avoid printing a faulty label after certain errors have occurred.
- Partial clearing is used in connection with print repetition when only part of the label should be modified between the copies. In this case, the CLL statement must include a reference to a field, specified by a FIELDNO function. When a CLL statement is executed, the image buffer will be cleared from the specified field to the end of the program.

In this example, the text “Month” is kept in the image buffer, whereas the names of the months are cleared from the image buffer as soon as they are printed, one after the other:

```
10    FONT "Swiss 721 Bold BT",18
20    PRPOS 100,300
30    PRTXT "MONTH:"
40    PRPOS 100,200
50    A%=FIELDNO
60    PRTXT "JANUARY":PRINTFEED
70    CLL A%
80    FONT "Swiss 721 Bold BT",18
90    PRPOS 100,200
100   PRTXT "FEBRUARY":PRINTFEED
110   CLL A%
120   FONT "Swiss 721 Bold BT",18
130   PRPOS 100,200
140   PRTXT "MARCH":PRINTFEED
150   CLL A%
RUN
```

OPTIMIZE "BATCH" ON/OFF

This statement is used to speed up batch printing. The program execution will not wait for the printing of a label to be completed, but proceeds executing next label image into the other image buffer as soon as possible.

The default setting is OPTIMIZE "BATCH" OFF. However, if all the following conditions are fulfilled, OPTIMIZE "BATCH" ON will automatically be invoked:

- A value >1 is entered for the PRINTFEED statement.
- LTS& OFF (default)
- CUT OFF (default)

OPTIMIZE "BATCH" ON revokes OPTIMIZE "BATCH" OFF.

Interrupting Batch Printing

Batch printing is interrupted when an error occurs, but can also be interrupted by pressing either the <**Print**> or the <**Pause**> key on the printer’s front panel. Printing can be resumed by pressing any of those keys again.

To prevent unauthorized use, each of these keys key can be disabled using a MAP or KEYBMAP$ instruction to map it to an ASCII value other than ASCII 30 or 31 dec.

The <**Print**> key can also be enabled/disabled using a PRINT KEY ON/OFF statement, see Chapter 10.2.

Reprint after Interruption

If an error occurs during the printing of a batch or if the printing is otherwise interrupted, it is possible to reprint lost or only partially printed labels without losing variable data, such as counter values. This has been made possible by extensions to the PRINTFEED statement and the PRSTAT function.

In the PRINTFEED statement, you can (as an alternative to specifying the number of copies in a batch) decide how many copies of the last printed label in a batch will be reprinted. The syntax is:

PRINTFEED -1,<number of identical copies to reprint>

If, for example, a 100-label batch print job is interrupted by an out-of-ribbon condition during the printing of label #70 and you have specified that 2 copies should be reprinted, label #70 will be printed twice when the error has been cleared.

Note: You can only reprint the last label!

The PRSTAT function can detect the progress of the printing and report any print-related error conditions. This makes it possible to create an error-handling routine that automatically resumes interrupted print jobs and reprints lost labels.

11 Fonts

This chapter explain briefly how various types of fonts are used in Intermec Fingerprint. More comprehensive explainations can be found in the *Intermec Fingerprint v8.00, Programmer's Reference Manual* and the *Intermec Fingerprint Font Reference Manual*.

11.1 Font Types

Intermec Fingerprint supports scaleable single- and double-byte fonts in TrueDoc (.PFR = Portable Font Resource) and TrueType (.TTF) format that comply with the Unicode standard. TrueDoc fonts in .PFR format can only be obtained from Intermec.

A single .PFR file can contain a number of different fonts. Compared with TrueType fonts, TrueDoc fonts require less memory spaces and work faster.

Intermec Fingerprint v8.00 contains 15 single-byte standard fonts in the systems parts (“Kernel”) of the permanent memory ("/rom").

TrueType fonts from sources other than Intermec could normally be used provided they comply with the Unicode standard. This is usually the case with TrueType fonts for Windows.

Note: For more information on the Unicode standard, visit the UniCode home page at http://www.unicode.org

11.2 Single-byte Fonts

Single-byte fonts are fonts that are mapped in the range of ASCII 0-127 dec (7-bit communication) or ASCII 0-255 dec (8 bit communication). Example of single-byte fonts are Latin, Greek, Cyrillic, Arabic, and Hebrew fonts.

Single-byte fonts are selected using the statements FONT and BARFONT (see Chapter 9.2 and 9.4 respectively) and the corresponding character set using the statement NASC (see Chapter 8.1).

11.3 Double-byte Fonts

Double-byte fonts are fonts that are mapped in the area of ASCII 0-65,536 dec. 8 bit communication must be selected. Any glyph (that is characters, interpunctation marks, symbols, digits, etc.) in the Unicode World Wide Character Standard, can be specified. Presently, Unicode contains about 40,000 glyphs.

Note: For more information on the Unicode standard, visit the Unicode home page at http://www.unicode.org

Example of languages that require double-byte fonts are Chinese, Japanese, and Korean.

Double-byte fonts are selected using the statement FONTD (see Chapter 9.2) and the corresponding character set using the statement NASCD (see Chapter 8.1). Double-byte fonts cannot be used for bar code interpretations (BARFONT) or multi-line text fields (see Chapters 9.3-9.4).

11.4 Direction, Size, Slant, and Width

Fonts can be rotated in four directions using a DIR statement. Using the FONT, FONTD, and BARFONT statements, fonts can be specified in regard of size in points (1 point = 1/72 in. = 0.352 mm) and slant in degrees (clockwise). The width can be set as a percentage value relative the height. Slant and width cannot be used for bitmap fonts (use MAG for bitmap fonts).

11.5 Standard Fonts

As standard, the Intermec Fingerprint v8.00 firmware contains 15 single-byte TrueDoc fonts stored in the systems part (“Kernel”) of the permanent memory. In the FONT and BARFONT statements, the full names according to the list below must be used (case sensitive).

- Century Schoolbook BT
- DingDings SWA (see note)
- Dutch 801 Roman BT
- Dutch 801 Bold BT
- Futura Light BT
- Letter Gothic 12 Pitch BT
- Monospace 821 BT
- Monospace 821 Bold BT
- OCR-A BT (see note)
- OCR-B 10 Pitch BT (see note)
- Prestige 12 Pitch Bold BT
- Swiss 721 BT
- Swiss 721 Bold BT
- Swiss 721 Bold Condensed BT
- Zurich Extra Condensed BT

Note: When selecting DingDings SWA, OCR-A BT, or OCR-B 10 Pitch BT, the printer will automatically switch from the presently selected character set to a special one for the font in question. As soon as any other font is selected again, the printer will automatically return to the previously selected character set.

11.6 Old Font Formats

To maintain compatibility with some earlier versions of Intermec Fingerprint, bitmap fonts in .ATF format can also be used, for example "SW030RSN" or "MS060BMN.2". Extensions (.1 or .2) are of no consequence.

11.7 Adding Fonts

The standard complement of fonts listed in Chapter 11.5 can be supplemented by additional fonts using three different methods:

- **Downloading fonts from a Font Install Card**
  The card must be inserted before the printer is started. At startup the fonts are automatically downloaded, installed, and permanently stored in the printer's memory. The fonts can be used without the card being present.
- **Using fonts from a Font Card**
  The card must be inserted before the printer is started. At startup the fonts are automatically installed, but not copied to the printer's memory which means that the card must always be present for such a font to be used. Font Cards are usually used for double-byte fonts, because of their size.
- **Downloading font files**
  Font files can be downloaded using the statements FILE& LOAD, IMAGE LOAD, TRANSFER KERMIT, or TRANSFER ZMODEM.

11.8 Listing Fonts

Regardless in which parts of the memory the different fonts are stored, they can all be listed to the standard OUT channel by a single statement, namely FONTS. This statement does not list dedicated bar code fonts.

Another method of listing fonts is to use a FONTNAME$ function, which also will list dedicated barcode fonts.

Font files can be listed to the standard OUT channel using the FILES statement.

This example shows how all fonts can be listed:

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

11.9 Removing Fonts

Font files stored in the read/write devices ("/c", "tmp:", and "card1:") can be deleted using KILL statements.

Note: The names of the font files may differ from the names of the corresponding fonts. Make sure to specify the font **file** names in the KILL statement.

11.10 Font Aliases

The names of the standard fonts in Intermec Fingerprint are rather long and may be cumbersome to use. They are also incompatible with the LAYOUT statement, which restricts the font and barfont names to 10 characters and does not allow fonts to be specified in regard of size in dots, slant in degrees, and width in percent of size.

However, it is possible to create a file containing a list of font aliases. The file should be named exactly as shown below (note the leading period character that specifies it as a system file):

```
"/c/.FONTALIAS"
```

The format of the file should be:

"<Alias name #1>","<Name of font>"[,size[,<slant>[,<width>]]]
"<Alias name #2>","<Name of font>"[,size[,<slant>[,<width>]]]
"<Alias name #3>","<Name of font>"[,size[,<slant>[,<width>]]]
.....
"<Alias name #n>","<Name of font>"[,size[,<slant>[,<width>]]]

The file can contain as many fontname aliases as required. The default size is 12 points, the default slant is 0°, and the width is 100%.

A font alias can be used as any other font, but its size, slant, and width can not be changed.

Examples:

```
"BODYTEXT","Century Schoolbook BT",10,80
"HEADLINE","Swiss 721 Bold BT",18,110
"WARNING","Swiss 721 BT",12,10,95
```

12 Bar Codes

This chapter provides a list of the supported bar codes and some brief explanations. For more comprehensive information, refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

12.1 Standard Bar Codes

As standard, Intermec Fingerprint v8.00 supports the following 39 bar codes which are stored in the systems part ("Kernel") of the printer's permanent memory. Some bar codes contain dedicated barcode fonts, for example UPC and EAN bar codes.

Bar codes can be listed using the BARCODENAME$ function.

Bar Code Type	Designation
Codabar	"CODABAR"
Code 11	"CODE11"
Code 16K	"CODE16K"
Code 39	"CODE39"
Code 39 full ASCII	"CODE39A"
Code 39 w. checksum	"CODE39C"
Code 49	"CODE49"
Code 93	"CODE93"
Code 128	"CODE128"
Datamatrix	"DATAMATRIX"
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	"I2OF5C"
Interleaved 2 of 5 A	"I2OF5A"
Matrix 2 of 5	"C2OF5MAT"
MaxiCode	"MAXICODE"
MSI (modified Plessey)	"MSI"
PDF 417	"PDF417"
Plessey	"PLESSEY"
Postnet	"POSTNET"
QR Code	"QRCODE"
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"

12.2 Setup Bar Codes

All present Intermec Fingerprint v8.xx-compatible printers can be set up using special Code 128 bar codes that are read with an EasySet Bar Code Wand or a scanner. Refer to *EasySet Bar Code Wand Setup* manual for information on how to compose such codes.

13 Images

This chapter explains the difference between images and image files, lists the standard images, and describes how to download, list, and remove images.

13.1 Images vs Images Files

There is a distinction between “Images” and “Image Files”:

- “Image” is a generic term for all kinds of printable pictures, for instance symbols, logotypes, or other illustrations, in the internal bitmap format of Intermec Fingerprint.
- “Image Files” are files in various bitmap formats that can be converted to “Images” in the internal bitmap format of Intermec Fingerprint. Images files can be stored in the printer’s memory, but cannot be used for printing before they have been converted to “Images.”

13.2 Standard Images

As standard, the systems part (“Kernel”) of the printer’s permanent memory contains a number of images primarily used for printing test labels and for training purposes:

- CHESS2X2.1
- CHESS4X4.1
- DIAMONDS.1
- GLOBE.1

13.3 Downloading Image Files

Image files in black&white .PCX format can be downloaded to the printer using the Kermit or Zmodem protocols after which they will automatically converted to the internal bitmap format of Intermec Fingerprint and installed.

Image files in .PCX format can also be both downloaded, automatically converted to images and installed using the IMAGE LOAD statement.

The current print buffer can be saved as a file and automatically converted to an image using the IMAGE BUFFER SAVE statement.

Image files in Intel hex formats, or formats according to the transfer protocols UBI00, UBI01, UBI02, UBI03, or UBI10, can be downloaded to the printer using the STORE IMAGE, STORE INPUT, and STORE OFF.

Example:

```
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
RUN
```

The system variable SYSVAR allows you to check the result of an image download using STORE INPUT:

`SYSVAR (16)` reads the number of bytes received.
`SYSVAR (17)` reads the number of frames received.

Both values are reset when a new STORE IMAGE statement is executed.

A special case is print images complying with the PRBUF Protocol. These are not normal pictures or logotypes, but rather complete labels including all kinds of printable objects which have been designed in some application program or printer driver in the host. Using the PRBUF statement, such print images can be downloaded directly to the printer's image buffer and printed. Such print images cannot be saved in the printer.

13.4 Listing Images

The names of all images stored in the various parts of the printer's memory can be listed to the std. OUT channel using an IMAGES statement or a program using the IMAGENAME$ function.

Image files can be listed to the std. OUT channel using a FILES statement.

Example:
This example lists all images the the printer's memory (in this case only the standard images).

```
IMAGES
```

yields:

```
CHESS2X2.1                      CHESS4X4.1
DIAMONDS.1                      GLOBE.1
3568692 bytes free   1717812 bytes used
Ok
```

13.5 Removing Images and Image Files

Images can be removed from the read/write devices ("/c", "tmp:", and "card1:") using REMOVE IMAGE statements.

Image files can be removed from the read/write devices ("/c", "tmp:", and "card1:") using KILL statements.

14 Printer Function Control

This chapter describes how to control various functions in the printer, set up the printer, use system variables, check the versions of firmware, printer family, and hardware, and how to check the printer's status.

14.1 Keyboard

All present Intermec Fingerprint v8.xx-compatible printers are provided with a built-in keyboard containing a set of numeric keys supplemented with a number of function keys. Separate external alphanumeric keyboards are available as options.

The keys have three purposes:

- To control the printer in the Setup Mode, and to some extent also in the Immediate Mode.
- To enter input data in the form of ASCII characters.
- To make the program execution branch to subroutines according to ON KEY...GOSUB statements.

Note:
Input from the printer's keyboard (see Chapter 6.6) excludes the use of ON KEY...GOSUB statements (see Chapter 4.8) and vice versa.
External keyboards do not work in the Setup Mode.

Controlling the Printer in the Setup and Immediate Modes

- The use of the keyboard in the Setup Mode is described in the User's Guide for the printer model in question.
- In a printer running in the Immediate Mode, eight keys are working:
  - The <**Print**> key or button produces a FORMFEED operation, or—if the printhead is lifted—runs the printer's platen roller a number of rotations in order to facilitate cleaning (CLEANFEED). During batch printing, it can be used to interrupt or resume printing.
  - The <**Pause**> key interrupts or resumes batch printing.
  - The <**Feed**> key works the same way as the <Print> ket.
  - The <**Shift**> + <**Feed**> keys pressed simultaneously produce a TEST-FEED operation.
  - The <**Setup**> key gives access to the Setup Mode.
  - The <**i**> key displays information on the communication channels.
  - The ⇐ and ⇒ keys are used to toggle between communcations channels after having pressed the <**i**> key.
- In the Immediate Mode, the printing of labels by means of the print key can be enabled or disabled using a PRINT KEY ON/ OFF statement, also see Chapter 10.2.

Enabling/Disabling the Keys

Before a key can be used to make the execution branch to a subroutine using an ON KEY...GOSUB statement, the key must be enabled using a KEY...ON statement. Enabled keys can also be disabled again using KEY...OFF statements.

The keyboard can be used to enter input data (provided "console:" is OPENed for INPUT), and also be used in the Setup and Test Modes, regardless if the keys are enabled or not.

To prevent unauthorized or accidental use, keys can be mapped to other ("harmless") ASCII values using MAP or KEYBMAP$ instructions.

Key Id. Numbers

The keys are specified by identification (id.) numbers in connection with the following statements:

`KEY...ON`	Enables the specified key.
`KEY...OFF`	Disables the specified key.
`ON KEY..GOSUB..`	Branches the program execution to a subroutine when the specified key is pressed.

Each key has two id. numbers, one for its unshifted position and another for its shifted position. To select the shifted position of a certain key, keep the <**Shift**> key depressed while you press the desired key. The id. number of the shifted key is equal to its unshifted id. number + 100. For example, the <**F1**> key has id. number 10 in unshifted position, but id. number 110 in shifted position. Also see Chapter 17.2 for illustrations.

If the keyboard is remapped (see later in this chapter), the id. numbers will be affected.

Key-initiated Branching

What will happen when an enabled key is pressed can be decided by an ON KEY...GOSUB statement, that branches the program execution to a subroutine, where additional instructions specify the action to be taken. Refer to Chapter 4.8 for further information and additional program example.

Here is an example of how two keys (<**F1**> and <**F2**>) are enabled and used to branch to different subroutines. The keys are specified by their id. numbers (10 and 11 respectively):

```
10    ON KEY (10) GOSUB 1000
20    ON KEY (11) GOSUB 2000
30    KEY (10) ON: KEY (11) ON
40    GOTO 40
50    END
1000  PRINT "You have pressed F1"
1010  RETURN 50
2000  PRINT "You have pressed F2"
2010  RETURN 50
RUN
```

Audible Key Response

Each time a key is pressed, the printer's beeper will, by default, emit a short signal (1200 Hz for 0.030 seconds). The frequency and duration of the signal can be globally changed for all keys using a KEY BEEP statement. Setting the frequency to a value higher than 9999 will turn off the signal for all keys.

Input from Printer's Keyboard

Provided "console:" is OPENed for sequential INPUT, the keys can be used to enter ASCII characters to the program using the following instructions:

`INPUT#`	reads a string of data to a variable.
`INPUT$`	reads a limited number of characters to a variable.
`LINE INPUT#`	reads an entire line to a variable.

Refer to Chapter 6.6 for a table showing the ASCII values that the various keys generate and for a program example. Note that input from keyboard does not require any keys to be enabled.

Remapping the Keyboard

The keyboards of the various printer models are fully remappable (with exception for the <**Shift**> key), so as to allow the printer to be adapted to special applications or national standards using the instruction KEYBMAP$. Thus you can decide which two ASCII characters each key will produce, with and without the <**Shift**> key being activated. The mapping also decides the id. numbers for the keys.

The basis of the remapping process is the position number of each key, see Chapter 17.1 for illustrations.

Note: In the Setup Mode, the keys have fixed positions that are not affected by any KEYBMAP$ instructions.

Note: There is a distinction between id. numbers and position numbers!

The present keyboard mapping can be read to a string variable using the KEYBMAP$ instruction with the following syntax:

<string variable>=KEYBMAP$(n)

n = 0	reads the unshifted characters.
n = 1	reads the shifted characters.

This example reads the unshifted characters on the keyboard of an Easy-Coder PF4i. Non-existing key positions get ASCII value 0:

```
10    PRINT "Pos","ASCII","Char."
20    A$=KEYBMAP$(0)
30    FOR B%=1 TO 64
40    C$=MID$(A$,B%,1)
50    E%=ASC(C$)
60    PRINT B%,E%,C$
70    NEXT
RUN
```

You can also use the KEYBMAP$ instruction to remap the keyboard, using the following syntax:

KEYBMAP$(n) = <string>

n = 0	maps the unshifted characters in ascending position number order.
n = 1	maps the shifted characters in ascending position number order.

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. See table below.

Non-existing key positions are mapped as NUL, that is CHR$(0).

ASCII decimal values for Special Keys

Key	Unshifted	Shifted
F1	1	129
F2	2	130
F3	3	131
F4	4	132
F5	5	133
Pause	30	158
Setup	29	157
Feed	28	156
Enter	13	141
C (Clear)	8	136
Print	31	159

The following example illustrates the mapping of the keyboard for an Easy-Coder PF4i (unshifted keys only).

```
10      B$=CHR$(1)+STRING$(4,0)+CHR$(2)+
        STRING$(4,0)+CHR$(3)
20      B$=B$+STRING$(4,0)+CHR$(4)+STRING$(4,0)+
        CHR$(5)+STRING$(9,0)
30      B$=B$+CHR$(13)+CHR$(28)+CHR$(29)+CHR$(30)+
        STRING$(6,0)
40      B$=B$+".147"+CHR$(0)+"0258"+CHR$(0)+CHR$(8)
        +"369"+CHR$(0)+(CHR$(31)+STRING$(8.0)
50      KEYBMAP$(0)=B$
        RUN
```

14.2 Display

All present Fingerprint 8.xx-compatible printers have a 2 × 16 characters LCD (Liquid Crystal Display). The Intermec Fingerprint firmware uses it to show a number of standard messages, for example in the Setup Mode, but it can also be controlled by programming instructions (see "Output to Display" below). The display is provided with a controllable cursor, as described later in this chapter ("Cursor Control").

When the printer enters the Setup Mode, the character set is automatically switched to US-ASCII. When the printer exits the Setup Mode, it automatically restores the previous mapping.

Output to Display

Before you can print any text to the display, it must be opened for sequential output, for example:

```
10    OPEN "console:" FOR OUTPUT AS 1
```

Then you should clear any previously displayed message by sending two empty PRINT# or PRINTONE# statements:

```
20    PRINT#1:PRINT#1
```

Now you can send a string to each of the two lines. Note the appending semicolon on the second line:

```
30    PRINT#1, "Upper line"
40    PRINT#1, "Lower line";
RUN
```

This will result in the following message being displayed:

```
Upper line
Lower line
```

As a alternative to sending two separate lines, you can also send a single line consisting of max. 33 characters, where:

- Character 1-16 specifies the upper line
- Character No. 17 is not displayed at all
- Character No. 18-33 specifies the lower line
- The line should be appended by a semicolon (;)

Using this method, the example above would look like this (double-headed arrows indicate space characters):

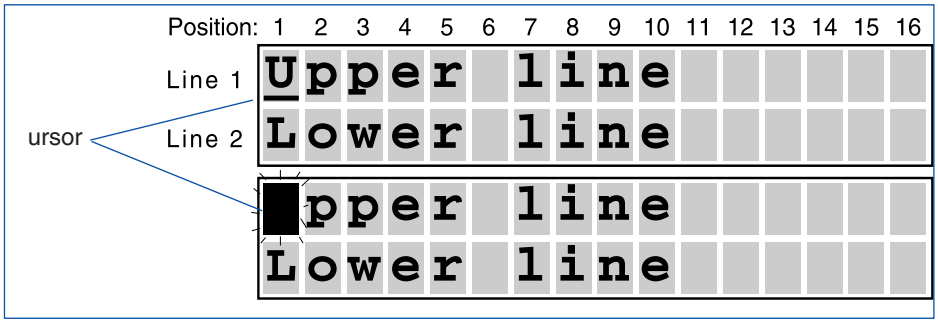
```
10 OPEN "console:" FOR OUTPUT AS 1
20 PRINT#1:PRINT#1
30 PRINT#1,"Upper↔line↔↔↔↔↔↔↔↔Lower↔line";
RUN
```

Cursor Control

The cursor control instructions can be used for four purposes:

- To clear the display from messages as an alternative to a double PRINT# statement, see the example on the previous page.
- To enable or disable the cursor.
- To select cursor type (underscore or block/blink).
- To place the cursor at a specified position or to move it.

The cursor is either a black line under a character position in the display, or a blinking block that intermittently blacks out the character position:



Each cursor control command should start with the character "CSI" (Control Sequence Introducer) = ASCII 155 decimal, or (in case of 7-bit communication) with the characters "ESC" + "[" (ASCII 27 + 91 decimal).

Clearing the Display

Syntax: **<CSI> + <<0|1|2>J>**

CSI	ASCII 155 dec.
0	From active position to end, inclusive (default)
1	From start to active position, inclusive
2	All of the display
J	Must always append the string

Example (clears all of the display):

```
10    OPEN "console:" FOR OUTPUT AS 1
20    PRINT#1, CHR$(155) + "2J";
```

Selecting Cursor Type

Syntax: **<CSI> + <4p|5p>**

CSI	ASCII 155 dec.
4p	Underscore
5p	Block/Blink (default)

Example (selects underscore-type cursor):

```
10    OPEN "console:" FOR OUTPUT AS 1
20    PRINT#1, CHR$(155) + "4p";
```

Enabling/Disabling the Cursor

Syntax: **<CSI> + <2p|3p>**

CSI	ASCII 155 dec.
2p	Cursor On
3p	Cursor Off (default)

Example (enables the cursor):

```
10    OPEN "console:" FOR OUTPUT AS 1
20    PRINT#1, CHR$(155) + "2p";
```

Note: A semicolon should append the PRINT# instructions in order to avoid interfering with existing messages in the display.

Setting the Absolute Cursor Position

Syntax: **<CSI> + <<v>;<h>H>**

CSI	= ASCII 155 dec.
v	Is the line (1 = Upper; 2 = Lower)
h	Is the position in the line (1–16)
H	Must always append the string

If v, h, or both are missing, the default value is 1.

Example (setting the cursor in upper left position):

```
10    OPEN "console:" FOR OUTPUT AS 1
20    PRINT#1, CHR$(155) + "H";
```

Example (setting the cursor in lower right position):

```
10    OPEN "console:" FOR OUTPUT AS 1
20    PRINT#1, CHR$(155) + "2;16H";
```

Move the Cursor Relative Current Position

Syntax: **<CSI><n>A | B | C | D**

CSI	ASCII 155 dec.
n	Is number of steps relative current position (default 1)
A	Is direction Up
B	Is direction Down
C	Is direction Forward
D	Is direction Backward

The relative movement must not place the cursor outside the display area (2 × 16 positions) or the instruction will be ignored.

Example (moving the cursor from the first position in the upper line to the last position in the lower line):

```
10    OPEN "console:" FOR OUTPUT AS 1
20    PRINT#1, CHR$(155) + "1B";
30    PRINT#1, CHR$(155) + "15C";
```

14.3 LED Control Lamps

Beside showing messages in the printer's display window (see Chapter 14.2), a Fingerprint program can use two of the three LED's (Light Emitting Diodes) on the printer's front panel to notify the operator of various conditions.

The statements for control the LED's are:

`LED...ON`	Turns the specified LED on.
`LED...OFF`	Turns the specified LED off.
`LED...BLINK`	Makes the LED blink with an interval of 0.4 seconds.
`LED...BLINK,DATAIN`	Makes the LED blink with an interval of 0.4 seconds when data is received on the standard IN channel.

The printer's front panel contains three LED's labeled "Power", "Ready" (0), and "Error" (1):

- The "Power" LED is connected to the printer's power supply and is lit when the power is on. It cannot be controlled by the program.
- The two other LED's ("Ready" and "Error") can be programmed at will using LED...ON and LED...OFF statements, even though the printed text on the keyboard imposes certain restrictions.

Example: The "Ready" LED (0) is lit until an error occur. Then the "Error" LED (1) is lit instead. The "Error" LED remains lit until the error is cleared. A suitable error can be generated by running the program with the printhead lifted.

```
10    LED 0 ON
20    LED 1 OFF
30    ON ERROR GOTO 1000
40    PRPOS 100,100
50    FONT "Swiss 721 Bold BT",36
60    PRTXT "OK!"
70    PRINTFEED
80    LED 0 ON
90    LED 1 OFF
100   END
1000  LED 0 OFF
1010  LED 1 ON
1020  RESUME
RUN
```

14.4 Beeper

In addition to the visual signals given using the display and the LED control lamps (see Chapters 14.2 and 14.3), audible signals can also be initiated by the program execution in order to notify the operator.

The following instructions can be used:

`BEEP` Initiates a short signal of fixed frequency and duration.

`SOUND` Initiates a signal with variable frequency and duration.

The beeper can be controlled by either a BEEP statement, which gives a short shrill signal , or by a SOUND statement, which allows you to vary both the frequency and duration. You can even compose your own melodies, if your musical ear is not too sensitive!

In this example, a warning signal is emitted from the beeper, for example when the error “printhead lifted” occurs and keeps sounding until the error is cleared. A short beep indicates that the printer is OK.

```
10    ON ERROR GOTO 1000
20    PRPOS 100,100
30    FONT "Swiss 721 Bold BT", 36
40    PRTXT "OK!"
50    PRINTFEED : BEEP
60    END
1000  SOUND 880,25 : SOUND 988,25 : SOUND 30000,10
1010  RESUME
RUN
```

14.5 Clock/Calendar

All present Intermec Fingerprint v8.xx-compatible printers are, or can optionally be, fitted with a real-time clock circuit (RTC). The RTC is battery powered and will keep running even when the printer is turned off.

Refer to Chapter 8.3 for information on how to read the printer's clock/calendar, and on the standard formats for date and time.

The following instructions are to set the clock/calendar:

`DATE$ = <sexp>`	Sets the date (YYMMDD format)
`TIME$ = <sexp>`	Sets the time (HHMMSS format)

Example (setting the clock/calendar to 08.11.30 April 1, 2003):

```
DATE$ = "010403"
TIME$ = "081130"
```

Note: The values must always be entered as string expressions. Possible numeric expressions can be converted to string format using STR$ functions (see Chapter 8.2.)

Note: The internal clock is always used. The internal clock is updated from the RTC at each startup. If no RTC is installed, an error will occur when trying to read the date or time before the internal clock has been manually set using either a DATE$ or a TIME$ variable. If only the date is set, the internal clock starts at 00:00:00 and if only the time is set, the internal clock starts at Jan 01 1980. After setting the internal clock, you can use the DATE$ and TIME$ variables the same way as when an RTC is fitted, until a power off or REBOOT causes the date and time values to be lost. If you do have an RTC installed, it is recommended to reboot the printer now and then to update the clock.

14.6 Printer Setup

The printer's setup can be changed manually in the Setup Mode using the built-in keyboard (see note) or remotely using the Intermec PrintSet application program, or via the printer's home page (requires an EasyLAN connection).

Note: An external keyboard cannot be used in the Setup Mode.

Detailed information on the methods of manual or terminal setup and the various setup parameters can be found in the User's Guide for the printer model in question.

For information on how to set up the printer via an EasyLAN connection, see the EasyLAN User's Guide.

If you want to change some setup parameter either by remote control (other than Terminal Setup) or as a part of the program execution, you can use the SETUP statement.

SETUP

This statement can be used in four different ways:

`SETUP`	Makes the printer enter the Setup Mode.
`SETUP WRITE`	Creates a copy of the printer's current setup and saves it as a file in the printer's memory under a specified name or returns the current setup to the specified communication channel.
`SETUP<file name>`	Changes some or all of the setup parameters in the printer's current setup accord-ing to a setup file.
`SETUP<string>`	Changes a single setup parameter.

Reading the Current Setup

An easy way to read the printer's current setup is to use a SETUP WRITE statement to return the setup to the serial communication channel used for output to the host via a serial channel.

Example:

```
SETUP WRITE "uart1:"
```

Creating a Setup File

Create a setup file using Intermec Fingerprint instructions like this:

- OPEN a file for sequential OUTPUT. See Chapter 8.3.
- Use a PRINT# statement to enter each parameters you want so change. The input must follow the stipulated syntax exactly (see the *Intermec Fingerprint v8.00, Programmer's Reference Manual*, SETUP statement).
- CLOSE the file.

Changing the Setup using a Setup File

Use a SETUP<filename> statement to change the printer's setup. If the setup file is stored in another part of the printer's memory than the current directory, the file name should start with a reference to the memory device in question.

In the following example, we will first save the current setup under a new file name and then make a setup file that changes the size of the transmit buffer on "uart1:" just a little. Finally, we use the setup file to change the printer's setup.

```
10    SETUP WRITE "SETUP1.SYS"
20    OPEN "SETUPTEST.SYS" FOR OUTPUT AS #1
30    PRINT#1,"SER-COM,UART1,TRANS BUF,2000"
40    CLOSE #1
50    SETUP "SETUPTEST.SYS"
RUN
```

Changing the Setup using a Setup String

A single setup parameter can be changed without creating any file. The SETUP statement should be followed by a string following exactly the same syntax as the corresponding parameter in a Setup file, but without any leading PRINT# statement.

The same change as in the example above would look this way when using a setup string:

```
SETUP "SER-COM,UART1,TRANS BUF,2000"
```

Saving the Setup

You can decided whether a change in the printer's setup should be permanent or temporary using the SYSVAR(35). If SYSVAR(35) = 0 (default), the setup values will be saved as a file and will remain effective after a reboot or power down. If SYSVAR(35)=1, the setup will not be saved, and instead the last saved setup values will be effective after a reboot or power down. It is also possible to read how this system variable is set. Also see Chapter 14.7.

14.7 System Variables

Some sensors and other conditions can be read or set using the SYSVAR system variable.

SYSVAR

`SYSVAR(14)`	returns the number of errors since last power on.
`SYSVAR(15)`	returns the number of errors since the previously executed SYSVAR(15) instruction.
`SYSVAR(16)`	returns the number of bytes received at the execution of a STORE INPUT statement.
`SYSVAR(17)`	returns the number of frames received at the execution of a STORE INPUT statement.
`SYSVAR(18)`	returns or sets the verbosity level.
`SYSVAR(19)`	returns or sets the type of error messages transmitted by the printer.
`SYSVAR(20)`	returns 0 if the printer is set up for direct thermal or 1 if set up for thermal transfer printing.
`SYSVAR(21)`	returns the printhead density in dots/mm.
`SYSVAR(22)`	returns the number of dots in the printhead.
`SYSVAR(23)`	returns 1 if a transfer ribbon is detected, else 0.
`SYSVAR(24)`	returns 1 if a power-up has been performed since last SYSVAR(24), else 0.
`SYSVAR(25)`	returns or selects the type of Centronics communication on the parallel communication port "centronics:": `SYSVAR(25)=0` Standard type `SYSVAR(25)=1` IBM/Epson type `SYSVAR(25)=2` Classic type
`SYSVAR(26)`	returns 1 if the ribbon sensor detects that the diameter of the ribbon supply roll is equl or less than the diameter specified in the Setup Mode, else 0.
`SYSVAR(27)`	sets condition for label reprinting at out-of-ribbon error.
`SYSVAR(28)`	decides if the information on the position of the media vs the printhead should be cleared or not when the printhead is lifted.
`SYSVAR(29)`	returns DSR condition on "uart2:".
`SYSVAR(30)`	returns DSR condition on "uart3:".
`SYSVAR(31)`	returns last control character sent from the MUSE protocol (special applications).
`SYSVAR(32)`	returns the length of media that have been fed past the printhead in meters (value is updated in steps of 10 meters).

SYSVAR(33)	returns DSR condition on "uart1:".
SYSVAR(34)	sets or returns TrueType character positioning mode.
SYSVAR(35)	sets or returns permanent or volatile setup saving.
SYSVAR(36)	sets or returns whether changes of program mode should be printed to the Debug Std Out port in connection with debugging.
SYSVAR(37)	sets or returns minimum gap length.
SYSVAR(39)	enables/disables slack compensation.
SYSVAR(41)	sets or returns conditions for overriding error detection at predefined feed length.
SYSVAR(42)	sets or returns conditions for aligning the gaps in the media with the tear bar.
SYSVAR(43)	sets or returns file name conversion enabled/disabled.
SYSVAR(46)	returns 1 if the paper low sensor detects that the diameter of the media supply roll is equal or less than the diameter specified in the Setup Mode, else 0.

For detailed explanations, refer to the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

14.8 Printhead

In addition to the setup, six instructions can be used to check and control the thermal printhead.

SYSVAR

Two parameters in the system variable SYSVAR allows you to check the printhead, also see Chapter 14.7:

`SYSVAR(20)` returns if the printer is set up for direct thermal or transfer printing.

`SYSVAR(21)` returns the printhead density in dots per millimeter.

HEAD

The HEAD function allows you to identify possible faulty dots by detecting abnormal resistance values and optionally mark them as faulty. This application is closely connected to the SET FAULTY DOT and BARADJUST statements, see below.

Note: Some printhead faults, for example cracked or dirty dots, will not be detected by this function, because only the resistance is measured.

SET FAULTY DOT

This statement is used to mark specified dots on the printhead as faulty, a function that also can be performed using the HEAD function. Then, using a BARADJUST statement (see below), you can adjust the lateral location of picket fence bar codes so the dots marked as faulty will not affect the printing, that is the faulty dot(s) will be situated between the bars.

You can also revoke all previous SET FAULTY DOT statements by marking all dots as correct.

BARADJUST

This statement enables automatic lateral relocation of picket fence bar codes within specified limits. The firmware will keep record of all dots marked as faulty (see HEAD and SET FAULTY DOT above) and relocate the bar code as to place the spaces between the bars in line with the faulty dots. Thereby, it will be possible to use the printer pending printhead replacement.

Note: The BARADJUST statement cannot be used for ladder bar codes, stacked bar codes (for example Code 16K), bar codes with horizontal lines (for example DUN-14), EAN/UPC bar codes, or two-dimensional bar codes (for example PDF417).

The example on the next page shows how a program can be made that checks the printhead for faulty dots and warns the operator when a faulty dot is encountered. Pending printhead replacement, the bar code is repositioned to ensure continued readability. Such a program takes a few seconds to execute (there may be more than a thousand dots to check), so it is advisable either to restrict the dot check to the part of the printhead that corresponds to the location of the bar code, or to perform the test at startup only.

```
10    OPEN "console:" FOR OUTPUT AS 10
20    IF HEAD(-1)<>0 THEN GOTO 9000
30    BEEP:D1$="Printhead Error!":D2$="":GOSUB 2000
40    GOSUB 1000
50    BARADJUST 20,20
60    GOTO 9000
1000  FUNCTEST "HEAD",TMP$
1010  A$=":" : TMP%=INSTR(TMP$,A$)+1
1020  RETURN
1030  SET FAULTY DOT -1
1040  QMEAN%=HEAD(-7)
1050  QMIN%=QMEAN%*85\100
1060  QMAX%=QMEAN%*115\100
1070  FOR I%=0 TO WHEAD%-1
1080  QHEAD%=HEAD(I%)
1090  IF QHEAD%>QMAX% OR QHEAD%<QMIN% THEN SET FAULTY DOT I%
1100  NEXT
2000  PRINT #10 : PRINT #10, LEFT$(D1$,16)
2010  PRINT #10, LEFT$(D2$,16);
2020  RETURN
9000  PRPOS 200,20
9010  BARTYPE "CODE39"
9020  BARRATIO 2,1 : BARMAG 2
9030  BARHEIGHT 150
9040  PRBAR "1234567890"
9050  PRINTFEED
9060  END
```

FUNCTEST

The FUNCTEST statement is used to test of the printhead in regard of number of dots, head lifted or possible errors and place the result in a string variable.

Example using FUNCTEST on an EasyCoder PF4i. The program takes a few seconds to execute:

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

yields for example:

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

FUNCTEST$

The FUNCTEST$ function is very similar to the FUNCTEST statement and is used for the same purposes but programming is more simple:

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

yields for example:

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

14.9 Transfer Ribbon

SYSVAR

A number of parameters in the system variable SYSVAR can be used to check the transfer ribbon, also see Chapter 14.7:

`SYSVAR(20)`	returns if the printer is set up for direct thermal or transfer printing.
`SYSVAR(23)`	returns if a transfer ribbon is fitted or not.
`SYSVAR(26)`	returns if the transfer ribbon supply is low or not.
`SYSVAR(27)`	sets or returns conditions for label reprinting at an out-of-ribbon condition.

14.10 Version Check

VERSION$

The VERSION$ function returns one of three characteristics of the printer:

`VERSION$(0)`	returns the firmware version, for example “Fingerprint 8.00”
`VERSION$(1)`	returns the printer family, for example “PF4i”
`VERSION$(2)`	returns the CPU board generation, for example “hardware version 4.0”

This instruction allows you to create programs that will work with several different printer models. For example, you may use the VERSION$ function to determine the type of printer and select the appropriate one of several different sets of setup parameters.

Example (selects a setup file according to the type of printer):

```
10    A$=VERSION$(1)
20    IF A$="PF2i" THEN GOTO 1000
30    IF A$="PF4i" THEN GOTO 2000
40    IF A$="PM4i" THEN GOTO 3000
50    .....
60    .....
70    .....
1000  SETUP "SETUP_PF2i.SYS"
1010  GOTO 50
2000  SETUP "SETUP_PF4i.SYS"
2010  GOTO 50
3000  SETUP "SETUP_PM4i.SYS"
3010  GOTO 50
```

14.11 Status and Std I/O Check

IMMEDIATE MODE/STDIO

In addition to enabling/disabling the Immediate Mode (IMMEDIATE ON|OFF, see Chapter 4.4), the IMMEDIATE statement has two more functions:

`IMMEDIATE MODE`	prints a line to the standard OUT port that shows the status (On or Off) of the following modes: - Execution - Immediate - Input - Layout input - Debug STDIO (DBSTDIO)
`IMMEDIATE STDIO`	prints two lines to the standard OUT port with information on the current settings for the STDIN and STDOUT channels.

15 Error-Handling

This chapter describes the standard error-handling that helps the user during operation as well as various means of debugging a Fingerprint program. It also shows how to create an error-handling routine and lists the ERRHAND.PRG error-handling program which is included it the Intermec Fingerprint v8.00 firmware package.

15.1 Standard Error-Handling

Intermec Fingerprint is intended to be as flexible as possible. Thus, there are very few fixed error-handling facilities, but instead there are a number of tools for designing error-handling routines according to the demands of each application.

The following error-handling facilities are always available:

- **Out-of-Media Detection**
  The firmware will check for a few possible errors when either the <**Print**> or <**Feed**> key on the printer is pressed, provided the printhead is engaged. If an error is detected, a message will appear in the display:
  - Error 1005 “Out of paper”
  - Error 1031 “Next label not found”
  - Error 1027 “Out of transfer ribbon” (thermal transfer printers only)

  After the error has been attended to, the error message can be cleared by pressing any key.
- **Batch Printing Interruption**
  If a batch print job is interrupted by pressing the <**Print**> or <**Pause**> key, the display will show a message telling how many labels remain to be printed and prompt the operator to press the <**Print**> key.
- **Syntax Check**
  Each program line or instruction that is received on the standard IN channel will be checked for possible syntax errors before it is accepted. Provided there is a working two-way communication, possible syntax errors will be transmitted to the host on the standard OUT channel, for example “Feature not implemented” or “Font not found.”

  For a working two-way communication, three conditions must be fulfilled:
  - Serial communication
  - Std IN channel = Std OUT channel
  - Verbosity enabled
- **Execution Check**
  Any program or hardware error that stops the execution will be reported on the standard OUT channel, provided there is a working two-way communication1. In case of program errors, the number of the line where the error occurred will also be reported, for example “Field out of label in line 110.” After the error has been corrected, the execution must be restarted by means of a new RUN statement, unless there is a routine for dealing with the error condition included in the program.

Error Messages

Using the system variable SYSVAR(19), see Chapter 14.7, you can choose between four types of error messages. This is illustrated by the following examples using error #19:

1. “Invalid font in line 10” (default)
2. “Error 19 in line 10: Invalid font”
3. “E19”
4. “Error 19 in line 10”

15.2 Tracing Programming Errors

TRON/TROFF

Large program can be difficult to grasp. If the program does not work as expected, it may depend on some programming error that prevents the program from being executed in the intended order. The TRON (Trace On) statement allows you to trace the execution. When the program is run, each line number will be returned on the standard OUT channel in the order of execution, provided you have a working two-way communication. TROFF (Trace Off) disables TRON.

STOP/CONT

Using the STOP statement, the execution can be temporarily stopped for examining or changing variables. The execution can the be resumed, either from where it was stopped using a CONT statement or from a specified line using a GOTO statement. You cannot use CONT if the program has beed edited during the break.

DEBUGGING

To make it easier to debug a program step by step, you can specify breakpoints in the program. The DBBREAK statement allows you to create or delete a breakpoint where the program execution will halt.

As an alternative, you can use the DBSTEP statement to specify how many lines will be executed before next break.

At a break. the message “break in line nnn” will be transmitted on the Debug STDOUT port, which can be specified by a DBSTDIO statement.

You can resume the execution at next program line using a CONT statement or from the start of the program using a RUN statement.

All breakpoints can be deleted by a single DBBREAK OFF statement.

Using SYSVAR(36) you can choose whether a change of program mode should be printed to the Debug STDOUT port or not.

The statement LIST,B lists all breakpoints to the standard OUT channel.

The statement DBEND terminates the Fingerprint Debugger.

CLIP

If you encounter the error 1003 “Field ot of label”, it can be useful to enable printing of fields that protrude outside the printable area using a CLIP statement. This lets you see how much of the field is missing so you can adjust the layout accordingly.

In most application programs, it is useful to include some kind of error-handler. Obviously, how comprehensive the error-handler needs to be depends on the application and how independent from the host the printer will work. In this chapter, we will explain the general principles and the related instructions and in Chapter 15.4, you will find an example on how an error-handling program can be composed.

15.3 Creating an Error-Handling Routine

ON ERROR GOTO...

This statement is described in more detail in the Chapter 4.8. It is used to branch the execution to a subroutine if any kind of error occurs when a program is run. The major benefit is that the program will not stop, but the error can be identified and dealt with. The execution can then be resumed at an appropriate program line.

ERR

The ERR function returns the reference number of an error that has occurred. The actual meaning of the numbers can be found in Chapter 7, "Error Messages" in the *Intermec Fingerprint v8.00, Programmer's Reference Manual*.

ERL

The ERL function returns the number of the line on which an error has occurred.

RESUME

This statement is used resume the execution after the error has been taken care of in a subroutine. The execution can be resumed at the statement where the error occurred, at the statement immediately following the one where the error occurred, or at any other specified line. Also see Chapter 4.8.

Example

The four instructions described above can be used to branch to a subroutine, identify the error, branch to a secondary subroutine where the error is cleared and resume the execution. In the example only one error condition (Error 1019, "Invalid Font") is taken care of, but the same principles can be used for more errors. You can test the example by either adding a valid font name or lifting the printhead before running the program.

```
10    OPEN "console:" FOR OUTPUT AS 1
20    ON ERROR GOTO 1000
30    PRPOS 50,100
40    PRTXT "HELLO"
50    PRINTFEED
60    A%=TICKS+400
70    B%=TICKS
80    IF B%<A% THEN GOTO 70 ELSE GOTO 90
90    PRINT #1 : PRINT #1
100   END
1000  SOUND 880,50
1010  EFLAG%=ERR : ELINE%=ERL
1020  IF EFLAG%=1019 THEN GOTO 2000 ELSE GOTO 3000
2000  PRINT #1 : PRINT #1
2010  PRINT #1, "Font missing"
2020  PRINT #1, "in line ", ELINE%;
2030  FONT "Swiss 721 BT",24 : INVIMAGE
2040  RESUME
```

```
3000  PRINT #1 : PRINT #1
3010  PRINT #1, "Undefined error"
3020  PRINT #1, "Program Stops!";
3030  RESUME NEXT
RUN
```

PRSTAT

Another instruction that can be used in connection with error handling is the PRSTAT function. In addition to returning the current position of the insertion point, returning the size and position of a field, and check the progress of batch printing (see Chapters 9.1 and 10.4), it can also return the printer's status in regard of several conditions, using a logical operator:

`IF PRSTAT (AND 0)`	Ok
`IF PRSTAT (AND 1)`	Printhead lifted
`IF PRSTAT (AND 2)`	Label not removed (LTS only)
`IF PRSTAT (AND 4)`	Printer out of media
`IF PRSTAT (AND 8)`	Printer out of transfer ribbon (TTR) or ribbon installed (DT)
`IF PRSTAT (AND 16)`	Printhead voltage too high
`IF PRSTAT (AND 32)`	Printer is feeding

Multiple simultaneous errors are indicated by the sum of the values for each error, for example if the printhead is lifted (1) and the printer is out of both media (4) and ribbon (8), it can be detected by:

```
IF PRSTAT (AND 13)
```

To speed up execution when several conditions are to be checked, assign the PRSTAT value to a numeric variable, for example:

```
10    A% = PRSTAT
20    IF A% (AND 1) GOTO 1000
30    IF A% (AND 2) GOTO 2000
```

etc.

15.4 Error-Handling Program

ERRHAND.PRG Utility Program

Originally created in January 1992, the ERRHAND.PRG does not take advantage of a lot of convenient programming features added in Intermec Fingerprint since that time. However, because of Intermec Fingerprint‘s backward compatibility, it will still work even if you surely can improve it a great deal.

The ERRHAND.PRG contains routines for handling errors, managing the keyboard and display, and for printing. Use ERRHAND.PRG to quickly get started with your programming1.

By merging ERRHAND.PRG with your program, the latter can gain access to ERRHAND’s subroutines. Do not use the lines 10-20 and 100,000-1,900,200 in your program, since those line numbers are used by ERRHAND.PRG.

Example:

```
NEW
LOAD "MY PROGRAM.PRG"
MERGE "/rom/ERRHAND.PRG"
RUN
```

If you have more than one application program that requires error-handling in your printer, you will save valuable memory space by keeping ERRHAND.PRG stored separately and merging it with the current program directly after loading, compared with merging ERRHAND.PRG with each program. The approximate size of ERRHAND.PRG is 4 kilobyte.

Variables and subroutines in ERRHAND.PRG that your program can use, or which you can modify, are:

Variables

- NORDIS1$ and NORDIS2$ at line 10 contain the main display texts. You may replace them with your own text.
- DISP1$ and DISP2$ contain the actual text that will appear on the printer’s display on line 1 and 2 respectively.

Subroutines

- At line 160,000

  The errors which normally may occur during printing are taken care of:

Error 1005	Out of paper
Error 1006	No field to print
Error 1022	Head lifted
Error 1027	Out of transfer ribbon
Error 1031	Next label not found

  The subroutine shows the last error that occurred, if any, and the line number where the error was detected. The information is directed to your terminal. Called by the statement GOSUB 160000.

- **At line 400,000**
  The FEED-routine executes a FORMFEED with error-checking. Called by the statement GOSUB 400000.
- **At line 500,000**
  The PRINT-routine executes a PRINTFEED with error-checking. Called by the statement GOSUB 500000.
- **At line 600,000**
  This subroutine clears the printer's display and makes the display texts stored in the variables DISP1$ and DISP2$ appear on the first and second line respectively in the display. Called by the statement GOSUB 600000.
- **At line 700,000**
  The Init routine initiates error-checking, opens the console for output and displays the main display texts (NORDIS1$ and NORDIS2$). It also sets up the some of the keys on the keyboard (if any) and assigns subroutines to each key. Called by the statement GOSUB 700000.
- **At line 1,500,000**
  The <**Pause**> key (key No. 15) interrupts the program until the same key is pressed a second time. Called by the statement GOSUB 1500000.
- **At line 1,700,000**
  Routine for the <**Print**> key (key No. 17), that calls subroutine 500,000. Called by the statement GOSUB 1700000.
- **At line 1,800,000**
  Routine for the <**Setup**> key (key No. 18). Enters the Setup Mode of the printer. Called by the statement GOSUB 1800000.
- **At line 1,900,000**
  Routine for the <**Feed**> key (key No. 19), that calls subroutine 400,000. Called by the statement GOSUB 1900000.

For more information, refer to the complete listing that follows.

Listing of ERRHAND.PRG Utility Program

```
10          PROGNO$ = "Ver. 1.2 92-01-10"
15          NORDIS1$ = "TEST PROGRAM" : NORDIS2$ = "VERSION 1.2"
20          GOSUB 700000 : 'Initiate
100000      'Error routine
100010      EFLAG% = ERR
100050      'PRINT EFLAG%:'Activate for debug
100060      LASTERROR% = EFLAG%
100200      RESUME NEXT
160000      'PRINT "Last error = ";LASTERROR%: 'Activate for debug
160050      'IF LASTERROR% <> 0 THEN PRINT "At line ";ERL
160100      LASTERROR% = 0
160200      RETURN
200000      'Error handling routine
200010      IF EFLAG% = 1006 THEN GOTO 200040:'Formfeed instead of print
200020      LED (1) ON : LED (0) OFF : BUSY
200030      SOUND 400, 10
200040      IF EFLAG% = 1031 THEN GOSUB 300000
200050      IF EFLAG% = 1005 THEN GOSUB 310000
200060      IF EFLAG% = 1006 THEN GOSUB 320000
200070      IF EFLAG% = 1022 THEN GOSUB 330000
200080      IF EFLAG% = 1027 THEN GOSUB 340000
200090      DISP1$ = NORDIS1$ : DISP2$ = NORDIS2$
200100      GOSUB 600000
200110      LED (1) OFF : LED (0) ON : READY
200400      RETURN
300000      'Error 1031 Next label not found
300010      DISP1$ = "LABEL NOT FOUND"
300020      DISP2$ = "ERR NO. " + STR$ (ERR)
300030      GOSUB 600000
300040      EFLAG% = 0
300050      FORMFEED
300060      IF EFLAG% = 1031 THEN GOTO 300040
300200      RETURN
310000      'Error 1005 Out of paper
310010      DISP1$ = "OUT OF PAPER"
310020      DISP2$ = "ERR NO. " + STR$ (ERR)
310030      GOSUB 600000
310040      IF (PRSTAT AND 1)=0 THEN GOTO 310040:'Wait until head lifted
310050      EFLAG% = 0
310060      IF (PRSTAT AND 1) = 0 THEN FORMFEED ELSE  GOTO 310060
310070      IF EFLAG% = 1005 THEN GOTO 310040
310080      IF EFLAG% = 1031 THEN GOSUB 300000
310200      RETURN
320000      'Error 1006 No field to print
320010      GOSUB 400000
320200      RETURN
330000      'Error 1022 Head lifted
```

```
330010      DISP1$ = "HEAD LIFTED"
330020      DISP2$ = "ERR NO. " + STR$ (ERR)
330030      GOSUB 600000
330040      IF (PRSTAT AND 1) THEN GOTO 330040
330050      FORMFEED
330060      IF PCOMMAND% THEN GOSUB 500000
330200      RETURN
340000      'Error 1027 Out of transfer ribbon
340010      DISP1$ = "OUT OF RIBBON"
340020      DISP2$ = "ERR NO. " + STR$ (ERR)
340030      GOSUB 600000
340040      IF (PRSTAT AND 8) THEN GOTO 340040
340050      GOSUB 1500000
340200      IF PCOMMAND% THEN GOSUB 500000
349000      RETURN
400000      'Feed routine
400010      EFLAG% = 0
400020      FORMFEED
400200      IF EFLAG% <> 0 THEN GOSUB 200000
400300      RETURN
500000      'Print routine
500010      EFLAG% = 0
500020      PCOMMAND% = 1
500030      PRINTFEED
500040      IF EFLAG% <> 0 THEN GOSUB 200000
500100      PCOMMAND% = 0
500300      RETURN
600000      'Display handler
600010      PRINT # 10
600020      PRINT # 10
600030      PRINT # 10, DISP1$
600040      PRINT # 10, DISP2$;
600200      RETURN
700000      'Init routine
700010      ON ERROR GOTO 100000
700020      OPEN "console:" FOR OUTPUT AS # 10
700030      DISP1$ = NORDIS1$ : DISP2$ = NORDIS2$
700040      GOSUB 600000
700100      ON KEY (15) GOSUB 1500000 : 'PAUSE
700110      ON KEY (17) GOSUB 1700000 : 'PRINT
700120      ON KEY (18) GOSUB 1800000 : 'SETUP
700130      ON KEY (19) GOSUB 1900000 : 'FEED
700140      KEY (15) ON
700150      KEY (17) ON
700160      KEY (18) ON
700170      KEY (19) ON
700230      LED (0) ON
700240      LED (1) OFF
700300      PAUSE% = 0
```

```
700500      RETURN
1500000     'Pause function
1500010     KEY (15) ON
1500020     PAUSE% = PAUSE% XOR 1
1500030     BUSY : LED (0) OFF
1500040     DISP1$ = "Press <PAUSE>" : DISP2$ = "to   continue"
1500050     GOSUB 600000
1500060     IF PAUSE% = 0 THEN GOTO 1500100
1500070     SOUND 131, 2
1500080     SOUND 30000, 20
1500090     IF PAUSE% THEN GOTO 1500070
1500100     READY : LED (0) ON
1500110     DISP1$ = NORDIS1$ : DISP2$ = NORDIS2$
1500120     GOSUB 600000
1502000     RETURN
1700000     'Printkey
1700010     KEY (17) OFF
1700020     GOSUB 500000
1700030     KEY (17) ON
1700200     RETURN
1800000     'Setup key
1800010     KEY (18) OFF
1800020     LED (0) OFF
1800030     BUSY
1800040     SETUP
1800050     READY
1800060     LED (0) ON
1800080     KEY (18) ON
1800090     DISP1$ = NORDIS1$ : DISP2$ = NORDIS2$
1800100     GOSUB 600000
1800200     RETURN
1900000     'Feed key
1900010     KEY (19) OFF
1900020     GOSUB 400000
1900030     KEY (19) ON
1900200     RETURN
```

Extensions to ERRHAND.PRG Utility Program

The following subroutines are not included in ERRHAND.PRG, but may be added manually to stop new input via the printer's keyboard while a subroutine is executed:

- Enable all keys after having completed a subroutine by issuing the statement GOSUB 800000.

```
800000 'Turn all keys on
800010 FOR I% = 0 TO 21
800020 KEY (I%) ON
800030 NEXT I%
800040 RETURN
```

- Disable all keys before entering a subroutine by issuing the statement GOSUB 900000.

```
900000 'Turn all keys off
900010 FOR I% = 0 TO 21
900020 KEY (I%) OFF
900030 NEXT I%
900040 RETURN
```

16 Reference Lists

This chapter lists the Fingerprint instructions in alphabetic order and in order of purpose.

16.1 Instructions in Alphabetic Order

Instruction	See chapter	Purpose
ABS	8.2	Returning the absolute value of a numeric expression.
ACTLEN	10.3	Returning the length of the most recently executed PRINTFEED, FORMFEED, or TESTFEED statement.
ALIGN (AN)	9.1	Specifying which part (anchor point) of a text field, bar code field, image field, line, or box will be positioned at the insertion point.
ASC	8.2	Returning the decimal ASCII value of the first character in a string expression.
BARADJUST	14.8	Enabling/disabling automatic adjustment of bar code position in order to avoid faulty printhead dots.
BARCODENAME$	12.1	List bar code names.
BARFONT (BF)	9.4. 11.2, 11.4, 11.5	Specifying fonts for the printing of bar code interpretation.
BARFONT (BF) ON/OFF	9.4	Enabling/disabling the printing of bar code interpretation.
BARHEIGHT (BH)	9.4	Specifying the height of a bar code.
BARMAG (BM)	9.4	Specifying the magnification in regard of width of the bars in a bar code.
BARRATIO (BR)	9.4	Specifying the ratio between the wide and the narrow bars in a bar code.
BARSET	9.4	Specifying a bar code and setting additional parameters to complex bar codes.
BARTYPE (BT)	9.4	Specifying the type of bar code.
BEEP	14.4	Ordering the printer to emit a beep.
BREAK	4.12	Specifying a break interrupt character separately for the keyboard and each serial communication channel.
BREAK ON/OFF	4.12	Enabling/disabling break interrupt separately for the keyboard and each serial communication channel.
BUSY	6.7	Ordering a busy signal to be transmitted from the printer on the specified communication channel.
CHDIR	5.1	Specifying the current directory.
CHECKSUM	5.9	Calculating the checksum of a range of program lines in connection with the transfer of programs.
CHR$	8.2	Returning the readable character from a decimal ASCII code.
CLEANFEED	10.1	Running the printer's feed mechanism.
CLEAR	5.1	Clearing strings, variables and arrays to free memory space.
CLIP	9.1, 15.2	Enabling/disabling partial field printing.
CLL	10.4	Partial or complete clearing of the print image buffer.
CLOSE	5.4, 6.3-6.6, 7.3-7.5	Closing one or several files and/or devices for input/output.
COM ERROR ON/OFF	6.8	Enabling/disabling error handling on the specified communication channel.
COMBUF$	6.8	Reading the data in the buffer of the specified communication channel.
COMSET	6.8	Setting the parameters for background reception of data to the buffer of a specified communication channel.
COMSET OFF	6.8	Turning off background data reception and emptying the buffer of the specified communication channel.
COMSET ON	6.8	Emptying the buffer and turning on background data reception on the specified communication channel.

COMSTAT	6.8	Reading the status of the buffer of the specified communication channel.
CONT	15.2	Resuming program execution after a STOP statement.
COPY	4.13, 5.2-5.4, 7.5	Copying files.
CSUM	5.10	Calculating the checksum of an array of strings.
CURDIR$	5.2	Returning the current directory as the printer stores it.
CUT	10.2	Activating an optional paper cutting device.
CUT ON/OFF	10.2	Enabling/disabling automatic cutting after PRINTFEED execution and optionally adjusting the media feed before and after the cutting.
DATE$	8.3, 14.5	Setting or returning the current date.
DATEADD$	8.3	Returning a new date after a number of days have been added to or subtracted from the current date, or optionally a specified date
DATEDIFF	8.3	Returning the difference between two dates as a number of days.
DBBREAK	15.2	Adding or deleting single break points.
DBBREAK OFF	15.2	Deleting all break points
DBEND	15.2	Terminating Fingerprint Debugger.
DBSTDIO	15.2	Selecting standard I/O for debugging.
DBSTEP	15.2	Debugging programs step-by-step.
DELETE	4.4	Deleting one or several consecutive program lines from the printer's working memory.
DELETEPFSVAR	5.1	Deleting a saved variable.
DEVICES	3.10, 7.1	Returning the names of all devices to the standard OUT channel.
DIM	5.10	Specifying the dimensions of an array.
DIR	9.1	Specifying the print direction.
DIRNAME$	5.2	Returning the names of the directories in a specified part of the printer's memory.
END	4.4	Ending the execution of the current program or subroutine and closing all OPENed files and devices.
EOF	6.4	Checking for an end-of-file condition.
ERL	15.3	Returning the number of the line on which an error condition has occurred.
ERR	15.3	Returning the code number of an error that has occurred.
EXECUTE	4.11	Execute a Fingerprint program from another Fingerprint program.
FIELD	6.5, 7.4	Creating a single-record buffer for a random file and dividing the buffer into fields to which string variables are assigned.
FIELDNO	10.4	Getting the current field number for partial clearing of the print buffer by a CLL statement.
FILE& LOAD	5.8, 11.7	Reception and storing of binary files in the printer's memory.
FILENAME$	5.2	List file names.
FILES	5.2, 5,3, 7.1, 11.8, 13.4	Listing the files stored in one of the printer's directories to the standard OUT channel.
FLOATCALC$	8.2	Calculating with float numbers.

FONT (FT)	9.2, 11,2, 11.4, 11.5	Selecting a single-byte font for the printing of the subsequent PRTXT statements.
FONTD	9.2, 11.3, 11.4	Selecting a scaleable TrueType or TrueDoc double-byte font for the printing of the subsequent PRTXT statements.
FONTNAME$	11.8	Returning the names of the fonts stored in the printer's memory.
FONTS	7.1, 11.8	Returning the names of all fonts stored in the printer's memory to the standard OUT channel.
FOR...TO...NEXT	4.9	Creating a loop in the program execution, where a counter is incremented or decremented until a specified value is reached.
FORMAT	5.1	Formatting the printer's permanent memory, or formatting a SRAM-type memory card to MS-DOS format.
FORMAT DATE$	8.3	Specifying the format of the string returned by DATE$("F") and DATEADD$(...,"F") instructions.
FORMAT TIME$	8.3	Specifying the format of the string returned by TIME$("F") and TIMEADD$(...,"F") instructions.
FORMAT$	8.2	Formatting a number represented by a string.
FORMFEED (FF)	10.1	Activating the media feed mechanism in order to feed out or pull back a certain length of media.
FRE	5.1	Returning the number of free bytes in a specified part of the printer's memory.
FUNCTEST	14.8	Performing various hardware tests.
FUNCTEST$	14.8	Performing various hardware tests.
GET	6.5	Reading a record from a random file to a random buffer.
GETPFSVAR	5.1	Recovering a saved variable after a power failure.
GOSUB	4.7	Branching to a subroutine.
GOTO	4.6	Branching unconditionally to a specified line or resuming program execution at a specified line after a STOP statement.
HEAD	14.8	Returning the result of a thermal printhead check or marking bad dots as faulty for a BARADJUST statement.
IF..THEN...[ELSE]...[END IF]	4.5	Conditional execution controlled by the result of a numeric expression.
IMAGE BUFFER SAVE	13.3	Saving the current image buffer as a file.
IMAGE LOAD	5.5, 11,7, 13.3	Receiving, converting, and installing image and font files.
IMAGENAME$	13.4	Returning the names of the images stored in the printer's memory.
IMAGES	7.1, 13.4	Returning the names of all images stored in the printer's memory to the standard OUT channel.
IMMEDIATE	4.4, 14.11	Enabling/disabling the immediate mode of Intermec Fingerprint in connection with program editing without line numbers, printing status of modes, and printing current settings of I/O channels.
INKEY$	6.2	Reading the first character in the receive buffer of the standard IN channel.
INPUT (IP)	6.2	Receiving input data via the standard IN channel during the execution of a program.
INPUT#	6.3, 6.4, 6.6, 14.1	Reading a string of data from an OPENed device or sequential file.
INPUT$	6.2, 6.4, 6.6, 14.1	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.

INSTR	8.2	Searching a specified string for a certain character, or sequence of characters, and returning its position in relation to the start of the string.
INVIMAGE (II)	9.2, 9.3, 9.5	Inversing the printing of text and images from "black-on-white" to "white-on-black."
KEY BEEP	14.1	Resetting/setting the frequency and duration of the sound produced by the beeper, when any key on the printer's keyboard is pressed down.
KEY ON/OFF	14.1	Enabling/disabling a specified key on the printer's front panel to be used in connection with an ON KEY...GOSUB statement.
KEYBMAP$	14.1	Returning or setting the keyboard map table.
KILL	4.13, 5.1, 5.3-5.4, 11.9, 13.5	Deleting a file from the printer's memory or from a DOS-formatted SRAM memory card inserted in the memory card adapter.
LAYOUT	9.8	Handling of layout files.
LBLCOND	10.1	Overriding the media feed setup or selecting mode for short labels.
LED ON/OFF	14.3	Turning a specified LED control lamp on or off.
LEFT$	8.2	Returning a specified number of characters from a given string starting from the extreme left side of the string, that is from the start.
LEN	8.2	Returning the number of character positions in a string.
LET	3.7	Assigning the value of an expression to a variable.
LINE INPUT	6.2	Assigning an entire line, including punctuation marks, from the standard IN channel to a single string variable.
LINE INPUT#	6.3, 6.4, 6.6, 14.1	Assigning an entire line, including punctuation marks, from a sequential file or a device to a single string variable.
LIST	4.4, 5.3, 7.1, 15,2	Listing the current program completely or partially, listing all variables, or listing all breakpoints to the standard OUT channel.
LISTPFSVAR	5.1	Listing saved variables.
LOAD	4.13, 5.3	Loading a copy of a program, residing in the current directory or in another specified directory, into the printer's working memory.
LOC	5.4, 6.4-6.5, 6.8, 7.3-7.5	Returning the current position in an OPENed file or the status of the buffers in an OPENed communication channel.
LOF	5.4, 6.4-6.5, 6.8, 7.3-7.5	Returning the length in bytes of an OPENed sequential or random file or returning the status of the buffers in an OPENed communication channel.
LSET	7.4	Placing data left-justified into a field in a random file buffer.
LTS& ON/OFF	10.2	Enabling or disabling the label taken sensor.
MAG	9.2, 9.3, 9.5, 11.4	Magnifying a font, barfont or image up to four times separately in regard of height and width.
MAP	8.1	Changing the ASCII value of a character when received on the standard IN channel, or optionally on another specified communication channel.
MERGE	5.3	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.
MID$	8.2	Returning a specified part of a string.
MKDIR	5.2	Creating directories.

NAME DATE$	8.3	Formatting the month parameter in return strings of DATE$("F") and DATEADD$(...,"F").
NAME WEEKDAY$	8.3	Formatting the day parameter in return strings of WEEK-DAY$.
NASC	8.1, 11.2	Selecting a single-byte character set.
NASCD	8.1, 11.3	Selecting a double-byte character set according to the Unicode standard.
NEW	4.4, 5.3	Clearing the printer's working memory in order to allow a new program to be created.
NORIMAGE (NI)	9.2, 9.3, 9.5	Returning to "black-on-white" printing after INVIMAGE printing.
ON BREAK GOSUB	4.8, 4.12	Branching to a subroutine, when a break interrupt instruction is received.
ON COMSET GOSUB	4.8, 6.8	Branching to a subroutine, when the background reception of data on the specified communication channel is interrupted.
ON ERROR GOTO	4.8, 15.3	Branching to an error-handling subroutine when an error occurs.
ON GOSUB	4.8	Conditional branching to one or several subroutines.
ON GOTO	4.8	Conditional branching to one of several lines.
ON KEY GOSUB	4.8, 14.1	Branching to a subroutine when a specified key on the printer's front panel is activated.
ON/OFF LINE	6.7	Controlling the SELECT signal on the Centronics communication channel.
OPEN	5.4, 6.3-6.6, 7.3-7.5, 14.2	Opening a file or device–or creating a new file–for input, output, or append, allocating a buffer, and specifying the mode of access.
OPTIMIZE "BATCH" ON/OFF	10.4	Enabling/disabling optimizing for batch printing.
PORTIN	6.11	Reading the status of a port on the Industrial Interface Board.
PORTOUT ON/OFF	6.11	Setting one of four relay port or one of eight optical ports on an Industrial Interface Board to either on or off.
PRBAR (PB)	9.4	Providing input data to a bar code.
PRBOX (PX)	9.3, 9.6	Creating a box or a multi-line text field with line-wrapping.
PRBUF	5.5, 13.3	Transmitting a bitmap print image directly to the image buffer.
PRIMAGE (PM)	9.5	Selecting an image stored in the printer's memory.
PRINT (?)	7.1	Printing data to the standard OUT channel.
PRINT KEY ON/OFF	10.2	Enabling/disabling printing of a label by pressing the Print key.
PRINT#	7.3, 7.5, 14.2	Printing of data to a specified OPENed device or sequential file.
PRINTFEED (PF)	10.2	Printing and feeding out one or a specified number of labels, tickets, tags or portions of strip, according to the printer's setup.
PRINTONE	7.1	Printing characters specified by their ASCII values to the standard OUT channel.
PRINTONE#	7.3, 7.5	Printing characters specified by their ASCII values to a device or sequential file.
PRLINE (PL)	9.7	Creating a line.
PRPOS (PP)	9.1	Specifying the insertion point for a text, bar code, image, box, or line field.

PRSTAT	9.1, 10.4, 15.3	Returning the printer's current status, the current position of the insertion point, the progress of batch printing, or the size and position of the last object.
PRTXT (PT)	19.2	Providing the input data for a single-line text field.
PUT	7.4	Writing a given record from the random buffer to a given random file.
RANDOM	8.4	Generating a random integer within a specified interval.
RANDOMIZE	8.4	Reseeding the random number generator, optionally with a specified value.
READY	6.7	Ordering ready signal to be transmitted from the printer on the specified communication channel.
REBOOT	4.14	Restarting the printer.
REDIRECT OUT	5.4, 7.2	Redirecting the output data to a created file.
REM (')	4.4	Adding headlines and comments to a program without including them in the execution.
REMOVE IMAGE	5.1, 13.5	Removing a specified image from the printer's memory.
RENDER ON/OFF	9.1	Enabling/disabling field rendering.
RENUM	4.4	Renumbering the lines of a program.
RESUME	4.8, 15.3	Resuming program execution after an error-handling sub-routine has been executed.
RETURN	4.7	Returning to the main program after having branched to a subroutine because of a GOSUB statement.
RIGHT$	8.2	Returning a specified number of characters from a given string starting from the extreme right side of the string, that is from the end.
RSET	7.4	Placing data right-justified into a field in a random file buffer.
RUN	4.11, 5.3	Starting the execution of a program.
rz	5.8	Receiving data using the Zmodem protocol.
SAVE	4.13, 5.3	Saving a file in the printer's memory or optionally in a DOS-formatted memory card.
SET FAULTY DOT	14.8	Marking one or several dots on the printhead as faulty, or marking all faulty dots as correct.
SETPFSVAR	5.1	Registering variables to be saved at power failure.
SETSTDIO	6.1	Selecting standard IN and OUT communication channel.
SETUP	14.6	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.
SGN	8.2	Returning the sign (positive, zero, or negative) of a specified numeric expression.
SORT	5.10	Sorting a one-dimensional array.
SOUND	14.4	Making the printer's beeper produce a sound specified in regard of frequency and duration.
SPACE$	8.2	Returning a specified number of space characters.
SPLIT	5.10	Splitting a string into an array according to the position of a specified separator character and returning the number of elements in the array.
STOP	15.2	Terminating program execution.
STORE IMAGE	13.3	Setting up parameters for storing an image in the printer's memory.
STORE INPUT	13.3	Receiving and storing protocol frames of image data in the printer's memory.

STORE OFF	13.3	Terminating the storing of an image and resetting the storing parameters.
STR$	8.2	Returning the string representation of a numeric expression.
STRING$	8.2	Repeatedly returning the character of a specified ASCII value, or the first character in a specified string
SYSVAR	4.13, 6.7, 13.3, 14.6-14.9, 15.1	Reading or setting various system variables.
sz	5.8	Sending data using the Zmodem protocol.
TESTFEED	10.1	Adjusting the label stop sensor while feeding out media.
TICKS	8.3	Returning the time that has passed since the last power-up in the printer, expressed in number of "Ticks" (1 Tick = 0.01 seconds.)
TIME$	8.3, 14.5	Setting or returning the current time.
TIMEADD$	8.3	Returning a new time after a number of seconds have been added to or subtracted from the current time, or optionally a specified time.
TIMEDIFF	8.3	Returning the difference in number of seconds between two specified moments of time in number of seconds.
TRANSFER KERMIT	5.8, 11.7	Transferring of data files using Kermit communication protocol.
TRANSFER STATUS	5.8	Checking last TRANSFER KERMIT operation.
TRANSFER ZMODEM	5.10	Transferring of data files using ZMODEM communication protocol.
TRANSFER$	5.4	Executing a transfer from source to destination as specified by a TRANSFERSET statement.
TRANSFERSET	5.4	Entering setup for the TRANSFER$ function.
TRON/TROFF	15.2	Enabling/disabling tracing of the program execution.
VAL	8.2	Returning the numeric representation of a string expression.
VERBON/VERBOFF	6.7	Specifying the verbosity level of the communication from the printer on the standard OUT channel (serial communication only.)
VERSION$	14.10	Returning the version of the firmware, printer family, or type of CPU board.
WEEKDAY	8.3	Returning the weekday of a specified date.
WEEKDAY$	8.3	Returning the name of the weekday from a specified date.
WEEKNUMBER	8.3	Returning the number of the week for a specified date.
WHILE...WEND	4.9	Executing a series of statements in a loop providing a given condition is true.
XORMODE ON/OFF	9.1	Enabling/disabling the xor/flip mode in connection with graphical operations.

16.2 Instructions by Field of Application

Instruction	Abbr.	Type	Purpose
SETUP AND PREFERENCES			
General Intermec Fingerprint Control:			
CHDIR<scon>		Stmt	Change current directory
MAP[<nexp>,]<nexp>,<nexp>		Stmt	Remapping
NASC<nexp>		Stmt	Select single-byte character set
NASCD<nexp>		Stmt	Select double-byte character set
REBOOT		Stmt	Restart printer
SETUP [[WRITE<sexp>] | [<sexp>]| [<sexp>]]		Stmt	Printer setup
Setting the Clock/Calendar:			
DATE$=<sexp>		Var	Set the date
TIME$=<sexp>		Var	Set the time
OPERATOR INTERFACE			
Keyboard Setup:			
KEY(<nexp>)ON|OFF		Stmt	Enable/disable key on printer's keyboard
ON KEY(<nexp>)GOSUB<ncon>|<line label>		Stmt	Key-initiated branching
KEY BEEP<nexp>,<nexp>		Stmt	Set frequency and duration of key response
KEYBMAP$(<nexp>)=<sexp>		Var	Set the keyboard map table
Output to Display:			
OPEN "console:" FOR OUTPUT AS[#]<nexp>		Stmt	Open display for output
PRINT#<nexp>[,<<nexp>|<sexp>>[<,|;><<nexp|<sexp>>...][;]]		Stmt	Print data to display
CLOSE [#]<nexp>		Stmt	Close display for output
LED Control Lamps:			
LED<nexp>ON|OFF|BLINK[,DATAIN]		Stmt	Turn LED on or off
Audible Signals:			
BEEP		Stmt	Emit a beep
SOUND<nexp>,<nexp>		Stmt	Produce sound
Breaking Program Execution:			
BREAK<nexp>,<nexp>		Stmt	Specify break interrupt character
BREAK <nexp> ON|OFF		Stmt	Enable/disable break interrupt
ON BREAK<nexp>GOSUB<ncon>|<line label>		Stmt	Branching at break interrupt
PRINTER CHECKOUT AND CONTROL			
Modes:			
IMMEDIATE MODE		Stmt	Print status of modes to std OUT
STD I/O:			
IMMEDIATE STDIO		Stmt	Print settings for std IN and std OUT
Keyboard:			
<svar> = KEYBMAP$(<nexp>)		Var	Read keyboard mapping
Memory:			
CLEAR		Stmt	Clear strings, variables and arrays
DELETEPFSVAR<sexp>		Stmt	Delete saved variable
FORMAT<sexp>[,<nexp>[,<nexp>]][,A]		Stmt	Format "/c", "temp:", or "card1:"
FRE(<<nexp>|<sexp>>)		Func	Return number of free bytes in part of memory
GETPFSVAR(<sexp<[,D])		Func	Recover saved variable after power failure
IMAGE BUFFER SAVE<sexp>		Stmt	Save image buffer as a file
KILL<sexp>[,R[,A]]		Stmt	Delete file or directory
LISTPFSVAR		Stmt	List saved variables
REMOVE IMAGE<sexp>		Stmt	Remove image from memory
SETPFSVAR<sexp>[,<nexp>]		Stmt	Register variable to be saved at power failure
Odometer:			
SYSVAR(32)		Array	Read odometer
Printhead:			
BARADJUST<nexp>,<nexp>		Stmt	Enable/disable automatic bar code repositioning
HEAD(<nexp>) | <nexp> = HEAD(<sexp>)		Func	Checking printhead dots
FUNCTEST<sexp>,<svar>		Stmt	Checking printhead
FUNCTEST$(<sexp>)		Func	Checking the printhead
SET FAULTY DOT<nexp>[,<nexp>...]		Stmt	Marking dots as faulty for BARADJUST
SYSVAR(21)		Array	Read printhead density
SYSVAR(22)		Array	Read number of printhead dots

Instruction	Abbr.	Type	Purpose
PRINTER CHECKOUT AND CONTROL, cont.			
Transfer Ribbon:			
SYSVAR(20)		Array	Read DT/TTP mode
SYSVAR(23)		Array	Read ribbon end sensor
SYSVAR(26)		Array	Read ribbon low sensor
SYSVAR(27)		Array	Set print repeat at out-of-ribbon
PROGRAMMING			
Managing Programs and Files:			
CHECKSUM(<nexp>,<nexp>)		Func	Calculate checksum at program transfer
COPY<sexp>[,<sexp>]		Stmt	Copy file
KILL<sexp> [,R[,A]]		Stmt	Delete file or directory
LOAD<scon>		Stmt	Load program
MERGE<scon>		Stmt	Merge programs
MKDIR<sexp>		Stmt	Create directory
NEW		Stmt	Clear the working memory
SAVE<scon>[,P|L]		Stmt	Save program
SYSVAR(43)		Array	Enable/disable file name conversion
Listings:			
BARCODENAME$(<nexp>)		Func	Return names of bar codes in printer's memory
CURDIR$[(<sexp>)]		Func	Return current directory
DEVICES		Stmt	List devices to standard I/O channel
DIRNAME$[(<sexp>)]		Func	Return directory names
FILENAME$[(<sexp>)]		Func	Return names of files in printer's memory
FILES[<sexp>][,R] [,A]		Stmt	List files to standard I/O channel
FONTNAME$(<nexp>)		Func	Return names of fonts in printer's memory
FONTS		Stmt	List all fontnames to standard I/O channel
IMAGENAME$(<nexp>)		Func	Return names of images in printer's memory
IMAGES		Stmt	List all imagenames to standard I/O channel
LIST[[<ncon>[- <ncon>]]|,V]		Stmt	List current program or all variables to std I/O
VERSION$[(<nexp>)]		Func	Returns F/W or H/W version or printer model
Program Editing and Execution:			
DELETE<ncon>[-<ncon>]		Stmt	Delete program lines
END		Stmt	Terminate program execution
EXECUTE<sexp>		Stmt	Execute program from another program
IMMEDIATE ON|OFF		Stmt	Start/stop writing program w/o line numbers
LIST[[<ncon>[- <ncon>]]|,V]		Stmt	List current program or all variables to std I/O
NEW		Stmt	Clear the working memory
REM<remark>	'	Stmt	Remark
RENUM[<ncon>][,[<ncon>][,<ncon>]]		Stmt	Renumber program lines
RUN[<<scon>|<ncon>>]		Stmt	Execute program
SAVE<scon>[,P|L]		Stmt	Save program
Data Manipulation:			
ABS(<nexp>)		Func	Return the absolute value of an expression
ASC(<sexp>)		Func	Return ASCII code for 1:st char. in string
CHR$(<nexp>)		Func	Convert ASCII code
FLOATCALC$(<sexp>,<sexp>,<sexp>[,<nexp>]		Func	Calculate with float numbers
FORMAT$(<sexp>,<sexp>)		Func	Format number represented by a string
INSTR([<nexp>,]<sexp>,<sexp>]		Func	Return position of character in string
LEFT$(<sexp>,<nexp>)		Func	Return characters from left side of string
LEN(<sexp>)		Func	Return number of characters in string
[LET]<<nvar>=<nexp>>|<<svar>=<sexp>>		Stmt	Assign a value to a variable
MID$(<sexp>,<nexp>[,<nexp>])		Func	Return part of string
RANDOM (<nexp>,<nexp>)		Func	Generate a random integer
RANDOMIZE[<nexp>]		Stmt	Reseed random number generator
RIGHT$(<sexp>,<nexp>)		Func	Return characters from right side of string
SGN(<nexp>)		Func	Return sign of numeric expression
SPACE$(<nexp>)		Func	Return specified number of space characters
STR$(<nexp>)		Func	Return string representation of num. expr.
STRING$(<nexp>,<<nexp>|<sexp>>)		Func	Return a number of repeated characters
VAL(<sexp>)		Func	Return numeric representation of string expr.

Instruction	Abbr.	Type	Purpose
PROGRAMMING, cont.			
Branching and Conditionals:			
FOR<nvar>=<nexp>TO<nexp>[STEP<nexp>]NEXT[<nvar>]		Stmt	Creating a program loop
GOSUB<ncon>|<line label>		Stmt	Branch to subroutine
GOTO<ncon>|<line label>		Stmt	Unconditional branching
IF<nexp>[,]THEN<stmt>[ELSE<stmt>]		Stmt	Conditional execution
ON <nexp>GOSUB<ncon>|<line label>[,<ncon>|<line label>...]		Stmt	Cond. branching to one of many subroutines
ON <nexp>GOTO<ncon>|<line label>[,<ncon>|<line label>...]		Stmt	Conditional branching to one of several lines
RETURN[<ncon>|<line label>]		Stmt	Return from subroutine
WHILE<nexp>↵ <stmt>↵ [...<stmt>]↵ WEND		Stmt	Conditional execution of loop of statements
Arrays:			
CSUM<ncon>,<svar>,<nvar>		Stmt	Calculate checksum of array of strings
DIM<<nvar>|<svar>>(<nexp>[,<nexp>...])...[,<<nvar>|<svar>>(nexp>[,<nexp>...])]		Stmt	Set array dimensions
SORT<<nvar>|<svar>>,<nexp>,<nexp>,<nexp>		Stmt	Sort a one-dimensional array
SPLIT(<sexp>,<sexp>,<nexp>)		Func	Split a string into an array
Clock/Calendar Facilities:			
<svar>=DATE$[("F")]		Var	Read the date
<svar>=TIME$[("F")]		Var	Read the time
DATEADD$[(<sexp>,]<nexp>["F"])		Func	Add days to a date
TIMEADD$[(<sexp>,]<nexp>["F"])		Func	Add seconds to a time
DATEDIFF(<sexp>,<sexp>)		Func	Calculate difference between dates
TIMEDIFF(<sexp>,<sexp>)		Func	Calculate difference between times
FORMAT DATE$<sexp>		Stmt	Specify date format
FORMAT TIME$<sexp>		Stmt	Specify time format
NAME DATE$<nexp>,<sexp>		Stmt	Specify names of the months
NAME WEEKDAY$<nexp>,<sexp>		Stmt	Specify names of the weekdays
WEEKDAY(<sexp>)		Func	Return weekday of a date
WEEKDAY$(<sexp>)		Func	Return name of the weekday for a date
WEEKNUMBER(<sexp>[,<nexp>])		Func	Return weeknumber for a date
TICKS		Func	Return time passed since startup
Errorhandling and Debugging:			
CONT		Stmt	Resume program execution after STOP stmt
DBBREAK<nexp>|<sexp>[ON|OFF]		Stmt	Add or delete a single breakpoint
DBBREAK OFF		Stmt	Delete all breakpoints
DBEND		Stmt	Terminate Fingerprint Debugger
DBSTDIO<nexp>,<nexp>[,<sexp>,<sexp>] | [<nexp>,<nexp>,]<sexp>,<sexp>		Stmt	Select std I/O for debugging
DBSTEP<ncon>		Stmt	Debug step-by-step in specified intervals
ERL		Func	Return number of line with error
ERR		Func	Return error code number
GOTO<ncon>|<line label>		Stmt	Resume program execution after STOP stmt
LIST,B		Stmt	List all break points to std OUT
ON ERROR GOTO<ncon>|<line label>		Stmt	Branch at error
PRSTAT[(<nexp>)]		Func	Returns various conditions
RESUME[<<ncon>|<line label>|<NEXT>|<0>>]		Stmt	Resume program execution after error
STOP		Stmt	Stop program execution
SYSVAR(19)		Array	Set or return type of error message
SYSVAR(36)		Array	Enable/disable print of program mode changes
TRON/TROFF		Stmt	Enable/disable tracing of program execution
COMMUNICATION:			
Communication Control:			
BUSY[<nexp>]		Stmt	Send busy signal on communication channel
ON|OFF LINE<nexp>		Stmt	SELECT signal high/low (Centronics)
READY[<nexp>]		Stmt	Send ready signal on communication channel
REDIRECT OUT[<sexp>]		Stmt	Redirect output data to file
SETSTDIO<nexp>[,<nexp>]		Stmt	Set standard I/O channels
SYSVAR(18)		Array	Set verbosity level
SYSVAR(25)		Array	Select type of Centronics communication
SYSVAR (29)|(30)|(33)		Array	Read DSR condition
SYSVAR (31)		Array	Read last control character sent from MUSE
VERBOFF/VERBON		Stmt	Set verbosity off/on

Instruction	Abbr.	Type	Purpose
COMMUNICATION, cont.			
Background Communication:			
COM ERROR<nexp>ON|OFF		Stmt	Enable/disable error handling
COMBUF$(<nexp>)		Func	Read communication buffer
COMSET<nexp>,<sexp>,<sexp>,<sexp>,<sexp>,<nexp>		Stmt	Set communication parameters
COMSET<nexp>ON|OFF		Stmt	Turn on/off background data reception
COMSTAT(<nexp>)		Func	Read communication buffer status
ON COMSET<nexp>GOSUB<nexp>|<line label>		Stmt	Branch at background comm. interrupt
FILE TRANSFER:			
General:			
PRBUF<nexp>[,<nexp>]<new line><image data>		Stmt	Receive and print PRBUF Protocol file
FILE& LOAD[<nexp>,]<sexp>,<nexp>[,<nexp>]		Stmt	Receive and store binary files
RUN "rz [<switches>][<filename>]"		Ext	Receive data using ZMODEM protocol
RUN "sz [<switches>][<filename>]"		Ext	Send data using ZMODEM protocol
TRANSFER K[ERMIT]<sexp>[,<sexp>[,<sexp>[,<sexp>]]]		Stmt	Data transfer using Kermit protocol
TRANSFER S[TATUS]<nvar>,<svar>		Stmt	Check last TRANSFER KERMIT execution
TRANSFER Z[MODEM]<sexp>[,<sexp>[,<sexp>[,<sexp>]]]		Stmt	Data transfer using ZMODEM protocol
Data Files:			
TRANSFER$(<nexp>)		Func	Execute transfer and set time-out
TRANSFERSET[#]<nexp>,[#]<nexp>,<sexp>[,<nexp>]		Stmt	Enter setup for file transfer using TRANSFER$
Font Files:			
FILE& LOAD[<nexp>,]<sexp>,<nexp>[,<nexp>]		Stmt	Receive and store font files (installed after restart)
IMAGE LOAD[<nexp>,]<sexp>,<nexp>[,<sexp>[,<nexp>]]		Stmt	Receive, convert and install fonts
TRANSFER K[ERMIT]<sexp>[,<sexp>[,<sexp>[,<sexp>]]]		Stmt	Transfer, convert and install fonts
TRANSFER S[TATUS]<nvar>,<svar>		Stmt	Check last TRANSFER KERMIT execution
Image Files:			
IMAGE LOAD[<nexp>,]<sexp>,<nexp>,<sexp>[,<nexp>]		Stmt	Receive, convert and install .PCX images
STORE IMAGE[RLL][KILL]<sexp>,<nexp>,<nexp>,[<nexp>],<sexp>		Stmt	Set up image storage parameters
STORE INPUT<nexp>[,<nexp>]		Stmt	Receiving and storing image data
STORE OFF		Stmt	End storing of image data
SYSVAR(16)		Array	Read number of bytes received
SYSVAR(17)		Array	Read number of frames received
INPUT TO INTERMEC FINGERPRINT			
Input from Standard IN Channel:			
INKEY$		Func	Read 1:st character from std IN channel
INPUT[<scon><;|,>]<<nvar>|<svar>>[,<<nvar>|<svar>>...]	IP	Stmt	Input to variables
INPUT$(<nexp>[,<nexp>])		Func	Input, limited number of characters
LINE INPUT[<scon>;]<svar>		Stmt	Input, entire line
Input from Host on Any Channel:			
CLOSE[[#]<nexp>[,[#]<nexp>...]]		Stmt	Close device
INPUT#<nexp>,<<nvar>|<svar>>[,<<nvar>|<svar>...]		Stmt	Input to variables
INPUT$(<nexp>[,<nexp>])		Func	Input, limited number of characters
LINE INPUT#<nexp>,<svar>		Stmt	Input, entire line
LOC(<nexp>)		Func	Remaining number of characters in receive buffer
LOF(<nexp>)		Func	Remaining free space in receive buffer
OPEN<sexp>FOR INPUT AS[#]<nexp>		Stmt	Open device
Input from Sequential File:			
CLOSE[[#]<nexp>[,[#]<nexp>...]]		Stmt	Close file
EOF(<nexp>)		Func	End of file
INPUT#<nexp>,<<nvar>|<svar>>[,<<nvar>|<svar>...]		Stmt	Input to variables
INPUT$(<nexp>[,<nexp>])		Func	Input, limited no. of characters
LINE INPUT#<nexp>,<svar>		Stmt	Input, entire line
LOC(<nexp>)		Func	Return current position in file
LOF(<nexp>)		Func	Return length of file
OPEN<sexp>FOR INPUT AS[#]<nexp>		Stmt	Open file
Input from Random File:			
CLOSE[[#]<nexp>[,[#]<nexp>...]]		Stmt	Close file
FIELD[#]<nexp>,<nexp>AS<svar>[,<nexp>AS<svar>...]		Stmt	Create a buffer for a random file
GET[#]<nexp>,<nexp>		Stmt	Read rec. from random file to random buffer
LOC(<nexp>)		Func	Return current position in file or buffer
LOF(<nexp>)		Func	Return length of file
OPEN<sexp>AS[#]<nexp>[LEN=<nexp>]		Stmt	Open a random file

Instruction	Abbr.	Type	Purpose
INPUT TO INTERMEC FINGERPRINT, cont.			
Input from Printer's Keyboard:			
CLOSE [#]<nexp>		Stmt	Close keyboard for input
INPUT#<nexp>,<<nvar>|<svar>>[,<<nvar>|<svar>...]		Stmt	Input to variables
INPUT$(<nexp>[,<nexp>])		Func	Input, limited no. of characters
LINE INPUT#<nexp>,<svar>		Stmt	Input , entire line
OPEN "console:" FOR INPUT AS[#]<nexp>		Stmt	Open keyboard for input
Industrial Interface:			
PORTIN(<nexp>)		Func	Reading status of a specified port
PORTOUT(<nexp>)ON|OFF		Stmt	Set the relay on a specified port
OUTPUT FROM INTERMEC FINGERPRINT			
Output to Standard OUT Channel :			
PRINT[<<nexp>|<sexp>>[<,|;><<nexp>|<sexp>>...][;]]	?	Stmt	Print data to standard I/O channel
PRINTONE[<nexp>[<,|;><nexp>...][;]]		Stmt	Print ASCII characters to std I/O channel
Output to Any Communication Channel:			
CLOSE[[#]<nexp>[,[#]<nexp>...]]		Stmt	Close device
PRINT#<nexp>[,<<nexp>|<sexp>>[<,|;><<nexp|<sexp>>...][;]]		Stmt	Print data to device
PRINTONE#<nexp>[,<nexp>[<,|;><nexp>...][;]]		Stmt	Print ASCII characters to device
LOC(<nexp>)		Func	Remaining free bytes in transmitter buffer
LOF(<nexp>)		Func	Remaining no. of char. in transmitter buffer
OPEN<sexp>[FOR <OUTPUT|APPEND>]AS[#]<nexp>		Stmt	Open device
Output to a Sequential File:			
CLOSE[[#]<nexp>[,[#]<nexp>...]]		Stmt	Close file
PRINT#<nexp>[,<<nexp>|<sexp>>[<,|;><<nexp|<sexp>>...][;]]		Stmt	Print data to sequential file
PRINTONE#<nexp>[,<nexp>[<,|;><nexp>...][;]]		Stmt	Print ASCII characters to sequential file
LOC(<nexp>)		Func	Current position in file
LOF(<nexp>)		Func	Length of file
OPEN<sexp>[FOR <OUTPUT|APPEND>]AS[#]<nexp>		Stmt	Open file
Output to Random File:			
CLOSE[[#]<nexp>[,[#]<nexp>...]]		Stmt	Close file
FIELD[#]<nexp>,<nexp>AS<svar>[,<nexp>AS<svar>...]		Stmt	Create a buffer for a random file
LOC(<nexp>)		Func	Current position in file
LOF(<nexp>)		Func	Length of file
LSET<svar>=<sexp>		Stmt	Place data in random file buffer (left justified)
PUT[#]<nexp>,<nexp>		Stmt	Write rec. from random buffer to random file
OPEN<sexp>AS[#]<nexp>[LEN=<nexp>]		Stmt	Open a random file
RSET<svar>=<sexp>		Stmt	Place data in random file buffer (right justified)
FORMATTING AND PRINTING			
General Formatting Instructions:			
ALIGN<nexp>	AN	Stmt	Alignment
CLIP [BARCODE[HEIGHT|INFORMATION|X|Y]][ON|OFF]		Stmt	Enable/disable partial field printing
DIR<nexp>		Stmt	Select print direction
PRPOS<nexp>,<nexp>	PP	Stmt	Set coordinates for insertion point
LAYOUT[F,]<sexp>,<sexp>,<svar>|<sexp>,<nvar>|<sexp>		Stmt	Creating and using layout files
XORMODE ON/OFF		Stmt	Enable/disable xor mode
Text Printing:			
INVIMAGE	II	Stmt	Inverse image printing
MAG<nexp>,<nexp>		Stmt	Magnification of font (obsolete)
NORIMAGE	NI	Stmt	Return to normal image printing
FONT<sexp>[,<nexp>[,<nexp>[,<nexp>]]]	FT	Stmt	Select single-byte font
FONTD<sexp>[,<nexp>[,<nexp>[,<nexp>]]]		Stmt	Select double-byte font
PRBOX<nexp>,<nexp>,<nexp>[,<sexp> [,<nexp> [,<nexp>[,<sexp>[,<sexp>]]]]]	PX	Stmt	Print multi-line text field
PRTXT<<nexp>|<sexp>>[;<<nexp>|<sexp>>...][;]	PT	Stmt	Input data to text field
SYSVAR (34)		Array	Select TrueType character positioning mode
Bar Code Printing:			
BARFONT[#<ncon>,]<sexp>[,nexp>[,<nexp>[,<nexp>[,nexp> [,nexp>[,nexp>]]]]][ON]	BF	Stmt	Specify bar code interpretation fonts
BARFONT ON	BF ON	Stmt	Enable bar code interpretation
BARFONT OFF	BF OFF	Stmt	Disable bar code interpretation
BARHEIGHT<nexp>	BH	Stmt	Bar code height
BARMAG<nexp>	BM	Stmt	Bar code magnification

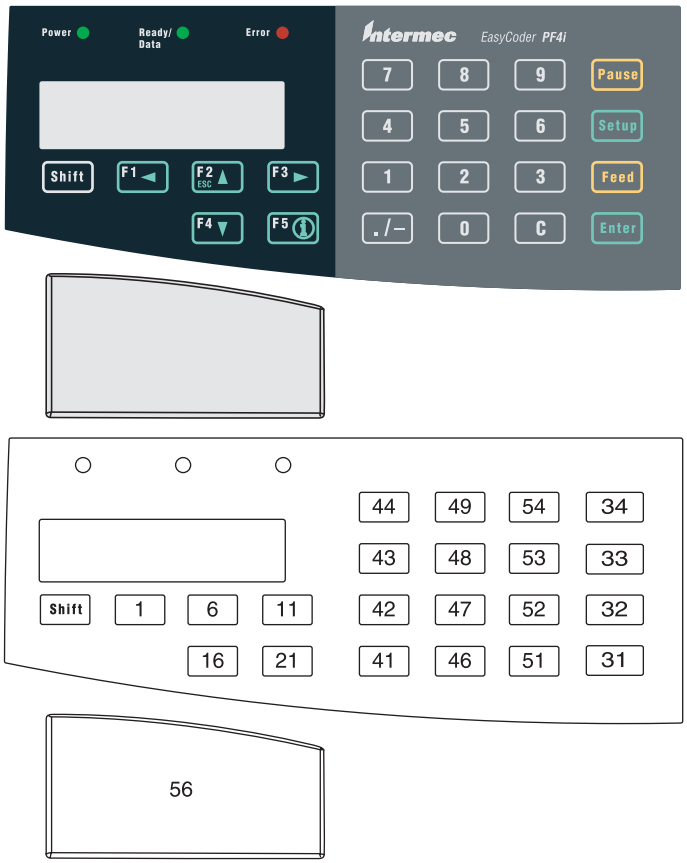
Instruction	Abbr.	Type	Purpose
FORMATTING AND PRINTING, cont.			
Bar Code Printing, cont:			
BARRATIO<nexp>,<nexp>	BR	Stmt	Wide/narrow bar ratio
BARSET[#<ncon>,][<sexp>[,<nexp>[,<nexp>[,<nexp>[,<nexp>[,<nexp>[,<nexp>[,<nexp>[,<nexp>[,<nexp>]]]]]]]]]		Stmt	Specifying complex bar codes
BARTYPE<sexp>	BT	Stmt	Bar code type
MAG<nexp>,<nexp>		Smt	Magnification of barfont (obsolete)
PRBAR<<sexp>|<nexp>>	PB	Stmt	Input data to bar code field
Image and Graphics Printing:			
INVIMAGE	II	Stmt	Inverse image printing
MAG<nexp>,<nexp>		Smt	Magnification of image
NORIMAGE	NI	Stmt	Return to normal image printing
PRBOX<nexp>,<nexp>,<nexp>	PX	Stmt	Create a box
PRIMAGE<sexp>	PM	Stmt	Select a preprogrammed image
PRLINE<nexp>,<nexp>	PL	Stmt	Create a line
Printing and Media Feed Control:			
ACTLEN		Func	Read length of last paper feed
CLEANFEED<nexp>		Stmt	Running the printer's feed mechanism
CLL[<nexp>]		Stmt	Clear print buffer
CUT		Stmt	Activate optional cutting device
CUT <nexp> ON|OFF		Stmt	Enable/disable automatic cut-off
FIELDNO		Func	Get current field number for CLL
FORMFEED[<nexp>]	FF	Stmt	Paper feed
LBLCOND<nexp>,<nexp>|<nexp>		Stmt	Overriding media feed setup
LTS& ON|OFF		Stmt	Enable/disable label taken sensor
OPTIMIZE "BATCH" ON|OFF		Stmt	Enable/disable optimizing for batch printing
PRINT KEY ON|OFF		Stmt	Enable/disable PRINTFEED using Print key
PRINTFEED[<nexp>]|[-1,<nexp>]	PF	Stmt	Print labels or reprint last label after interruption
PRSTAT[(<nexp>)]		Func	Return position and size of last object
RENDER ON|OFF		Stmt	Enable/disable rendering
SYSVAR(28)		Array	Erase media feed data
SYSVAR(37)		Array	Set minimum gap length
SYSVAR(39)		Array	Enable/disable slack compensation
SYSVAR(41)		Array	Override error detection at predefined feed length
SYSVAR(42)		Array	Align gaps with tear bar
SYSVAR(46)		Array	Detect media low conditions
TESTFEED[<nexp>]		Stmt	Auto adjustment of label stop sensor

17 Keyboards

This chapter illustrates how the keyboards of EasyCoder PF2i, PF4i, PF4i Compact Industrial, and PM4i har numbered in regard of position numbers and id. numbers, and which ASCII values the various keys will produce by default.

17.1 Position Numbers

EasyCoder PF-series

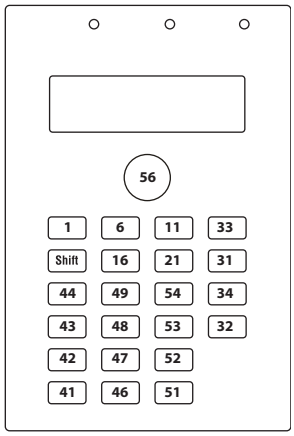


Actual keyboard apperance

Position numbers

EasyCoder PM4i

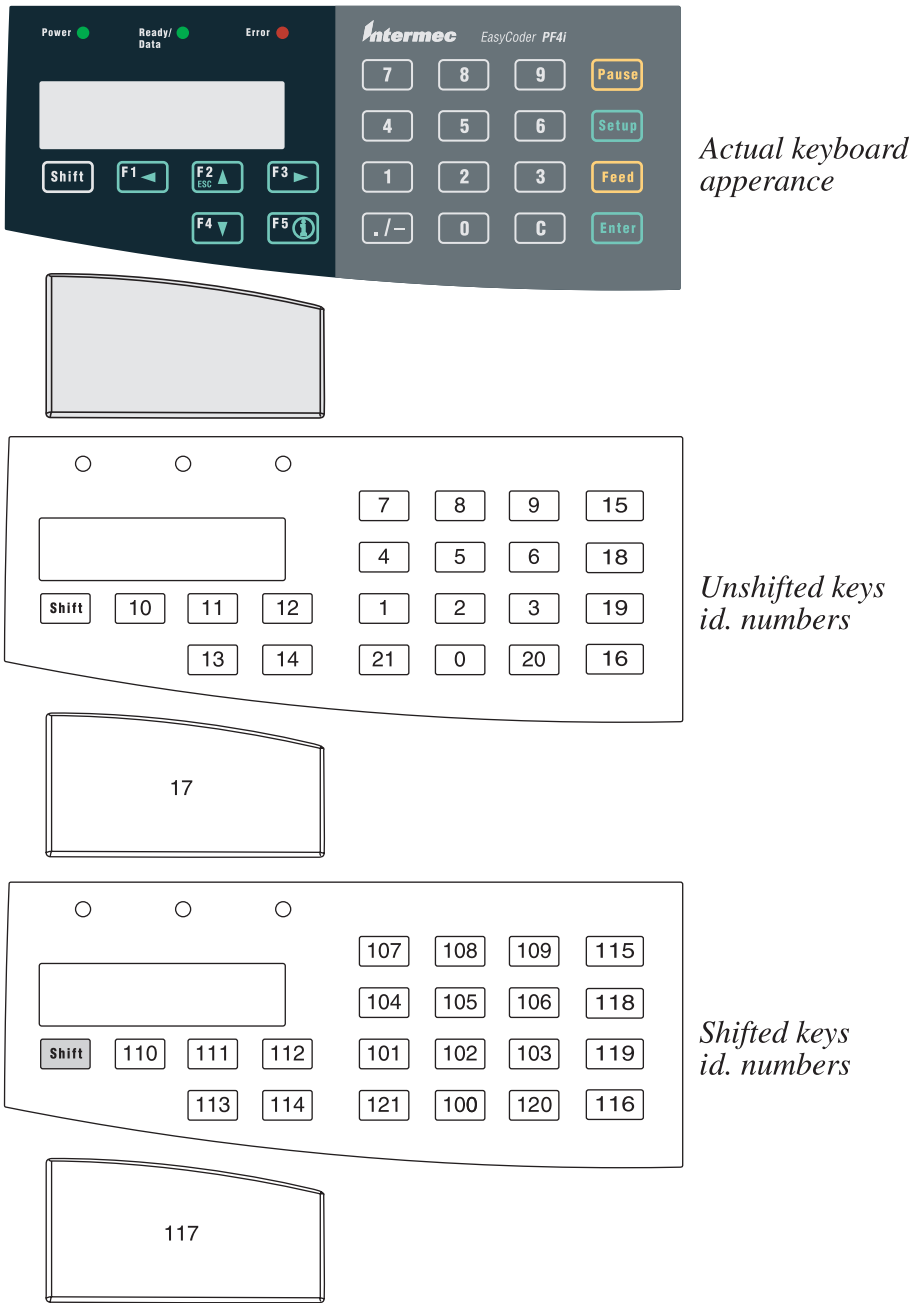
Actual keyboard appearance



Position numbers

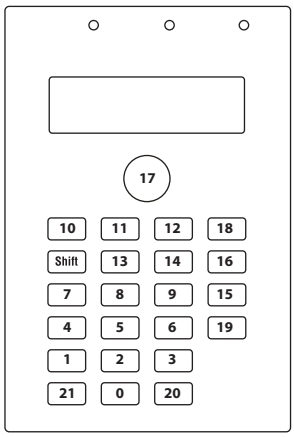
17.2 Id. Numbers

EasyCoder PF-series

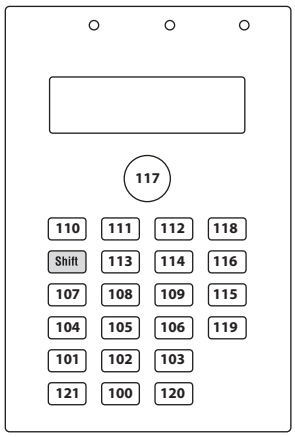


EasyCoder PM4i

Actual keyboard appearance



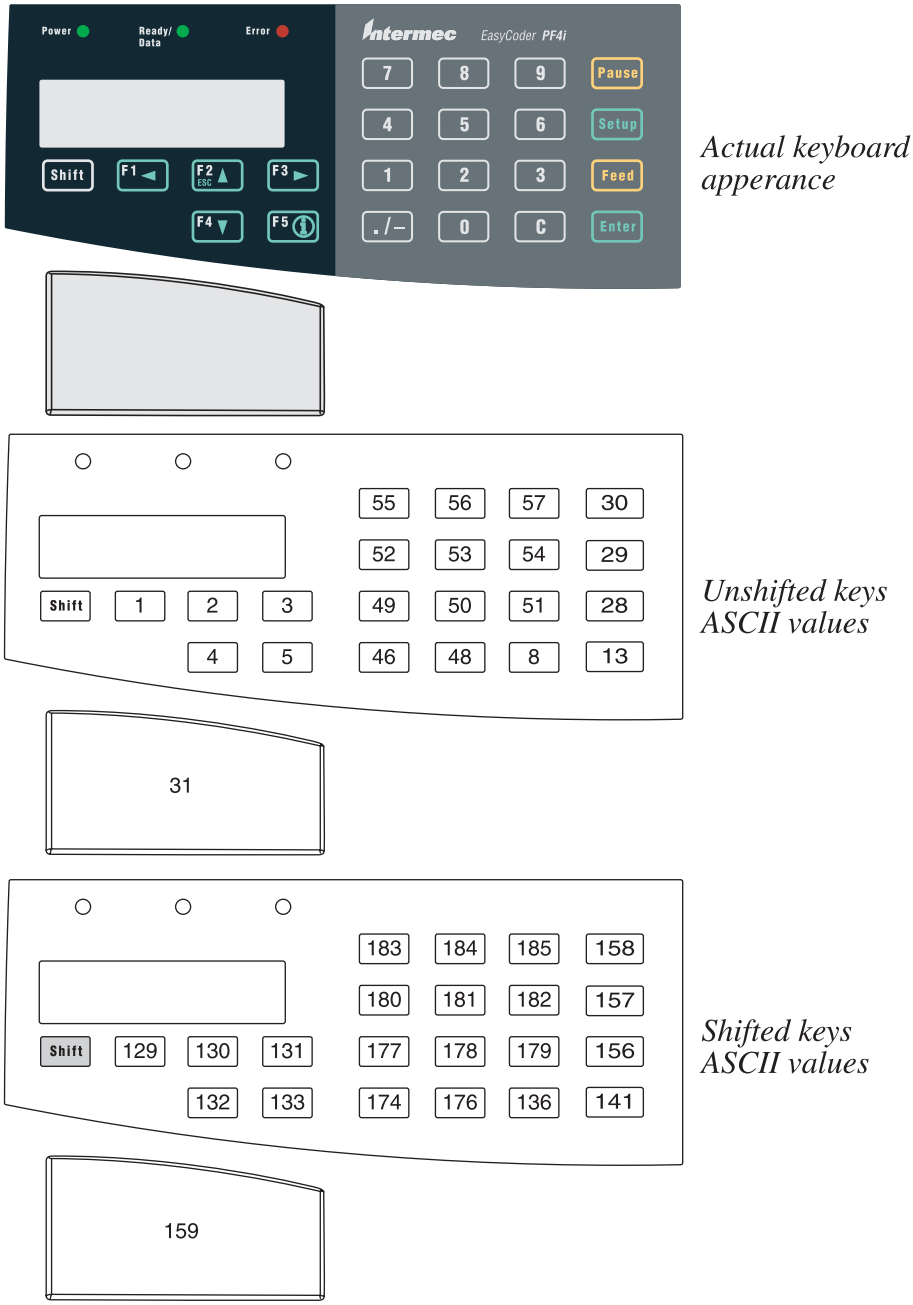
Unshifted keys; id. numbers



Shifted keys; id. numbers

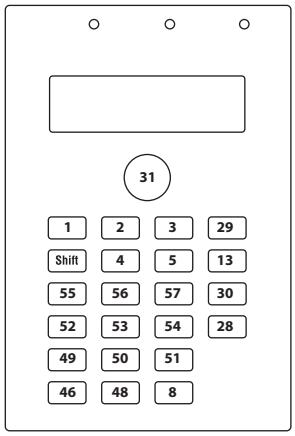
17.3 ASCII Values

EasyCoder PF-series

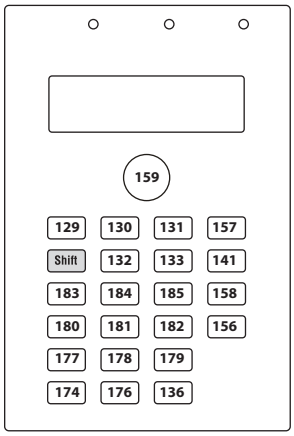


EasyCoder PM4i

Actual keyboard appearance



Unshifted keys; ASCII values



Shifted keys; ASCII values

Intermec Printer AB
Idrottsvägen 10, P.O. Box 123
S-431 22 Mölndal, Sweden
tel +46 31 869500
fax +46 31 869595
www.intermec.com

Intermec Fingerprint v8.00 Tutorial

1-960608-00