Fingerprint

Developer's Guide

Intermec Technologies Corporation

Worldwide Headquarters
6001 36th Ave.W.
Everett, WA 98203
U.S.A.

www.intermec.com

Contents

Before You Begin

This section provides you with safety information, technical support information, and sources for additional product information.

Safety Information

This section explains how to identify and understand the notes that are in this document.

A caution alerts you to an operating procedure, practice, condition, or statement that must be strictly observed to prevent equipment damage or destruction, or corruption or loss of data.

Note: Notes either provide extra information about a topic or contain special instructions for handling a particular condition or set of circumstances.

Global Services and Support

Warranty Information

To understand the warranty for your Intermec product, visit the Intermec web site at **www.intermec.com** and click **Support** > **Returns and Repairs** > **Warranty**.

Disclaimer of warranties: The sample code included in this document is presented for reference only. The code does not necessarily represent complete, tested programs. The code is provided "as is with all faults." All warranties are expressly disclaimed, including the implied warranties of merchantability and fitness for a particular purpose.

Web Support

Visit the Intermec web site at **www.intermec.com** to download our current manuals (in PDF).

Visit the Intermec technical knowledge base (Knowledge Central) at **www.intermec.com** and click **Support** > **Knowledge Central** to review technical information or to request technical support for your Intermec product.

Send Feedback

Your feedback is crucial to the continual improvement of our documentation. To provide feedback about this manual, please contact the Intermec Technical Communications department directly at **TechnicalCommunications@intermec.com**.

Telephone Support

In the U.S.A. and Canada, call **1-800-755-5505**.

Outside the U.S.A. and Canada, contact your local Intermec representative. To search for your local representative, from the Intermec web site, click **About Us** > **Contact Us**.

Who Should Read This Manual

This document is written for the person who is responsible for developing applications in the Intermec Fingerprint programming language. You need to be familiar with operating, maintaining, and troubleshooting your Intermec printer. You should also be familiar with networking terms, such as IP address.

Related Documents

This table contains a list of related Intermec documents and their part numbers.

Document Title
Fingerprint Command Reference Manual

The Intermec web site at **www.intermec.com** contains our documents (as PDF files) that you can download for free.

To download documents

1. Visit the Intermec web site at **www.intermec.com**.
2. Click the **Products** tab.
3. Using the **Products** menu, navigate to your product page. For example, to find the PC23d printer product page, click **Printers** > **Desktop Printers** > **PC23d**.
4. Click the **Manuals** tab.

If your product does not have its own product page, click **Support** > **Manuals**. Use the **Product Category**, the **Product Family**, and **Product** to find your documentation.

1

Introduction to Fingerprint

This chapter introduces Intermec Fingerprint and includes these topics:

What Is Fingerprint?

Fingerprint is a programming language you use to create custom label formats and printer application software. Fingerprint firmware is stored in the printer memory. Intermec Direct Protocol is a subset of Intermec Fingerprint and is used for combining variable input data with predefined label layouts.

This guide includes information on using Fingerprint to develop applications for your Intermec printer. For information on specific Fingerprint or Direct Protocol commands, see the ***Fingerprint Command Reference Manual.***

To locate the latest Fingerprint firmware for your printer:

1. Visit the Intermec web site at **www.intermec.com**.
2. Click **Support** > **Downloads**.
3. Use the **Product Category**, the **Product Family**, and **Product** to find your printer.

For more information on printer-specific features, such as setting up the printer, loading printer firmware, or loading media, see your printer user manual.

Note: Depending on your printer and hardware options, some Fingerprint commands may not be supported. For more information, see the ***Fingerprint Command Reference Manual.***

Learning the Structure of Fingerprint Commands

Fingerprint commands are text strings that instruct the printer to perform a variety of operations, such as downloading data from a host, configuring a bar code label format, enabling and disabling printer options, or starting a print job and returning print job status.

Each command is entered as a line. A Fingerprint program can consist of a single line, or of many lines that include conditional branching and subroutines. Programs can be stored in the printer memory, loaded from USB mass storage device, or sent to the printer from a host PC.

For example, a simple Fingerprint program can look like:

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

`10...50`	Specify line numbers for the program.
`PRPOS`	Specifies the insertion point for a printed object using coordinates.
`DIR`	Specifies the print direction, where 3 indicates that printing follows the same direction as the print feed.
`ALIGN`	Specifies which anchor point of a printed object is at the insertion point. The value 5 represents the center anchor point.

`PRIMAGE`	Adds an image named "Globe.1" to the print buffer.
`PRINTFEED`	Prints one label.
`RUN`	Runs the program.

For more information on command syntax, see **"Understanding Fingerprint Syntax" on page 5**.

Fingerprint Operating Modes

Fingerprint has two operating modes:

- Immediate Mode. In this mode, Fingerprint commands are processed when the printer receives a carriage return. Generally, commands sent in Immediate Mode cannot be saved after they are processed.

  Use Immediate Mode when you want to test the effects of commands without saving the commands, such as when you are editing label formats. For more information, see **"About Immediate Mode" on page 12**.

- Programming Mode. In this mode, you can save one or more Fingerprint commands as programs. You can edit, copy, load, list, or merge programs with other programs. For more information, see **"About Programming Mode" on page 13**.

Sending Fingerprint Commands to the Printer

You can send commands to your printer using a serial port and a terminal emulation program. To send Fingerprint commands to an Intermec printer using a serial connection, you need:

- a computer with a screen and keyboard.
- a serial connection to the printer.
- a terminal emulation program, such as HyperTerminal or PuTTY, that can transmit and receive ASCII characters.

For a complete list of methods to send commands to your printer, see your printer user's manual.

Connect to the printer using a communications program

1. Connect the printer to the serial port (COM1) on your desktop PC. For more information, see the user manual for your printer.
2. Turn on the printer.
3. On your desktop PC, start the communications program.
4. Create a connection to the printer using the TCP Port 9100, or the serial port (COM1) and these parameters:

Baud rate	115200
Data bits	8
Parity	None

Stop bits	1
Flow control	None

These serial connection parameters are the default for Fingerprint printers. If you have changed the communication settings on your printer, use those settings instead.

5 In the communications program, type:

```
SETUP WRITE "uart1:"
```

6 Press **Enter**. The printer returns its current setup parameters.

```
EMULATION,MODE,DISABLED
EMULATION,ADJUST,BASE (mmX10),88
EMULATION,ADJUST,STOP (mmX10),55
FEEDADJ,STARTADJ,0
FEEDADJ,STOPADJ,0
MEDIA,MEDIA SIZE,XSTART,0
MEDIA,MEDIA SIZE,WIDTH,832
MEDIA,MEDIA SIZE,LENGTH,1200
MEDIA,MEDIA TYPE,LABEL (w GAPS)
MEDIA,PAPER TYPE,TRANSFER
MEDIA,PAPER TYPE,DIRECT THERMAL,LABEL CONSTANT,85
MEDIA,PAPER TYPE,DIRECT THERMAL,LABEL FACTOR,40
MEDIA,PAPER TYPE,TRANSFER,RIBBON CONSTANT,90
MEDIA,PAPER TYPE,TRANSFER,RIBBON FACTOR,25
MEDIA,PAPER TYPE,TRANSFER,LABEL OFFSET,0
MEDIA,CONTRAST,+0%
# MEDIA,TESTFEED,0 0 0
MEDIA,TESTFEED MODE,FAST
MEDIA,LEN (SLOW MODE),0
PRINT DEFS,PRINT SPEED,100
PRINT DEFS,CLIP DEFAULT,OFF

Ok
```

SETUP WRITE Command Results

2

Understanding Fingerprint Syntax

This chapter explains the basics of Fingerprint command syntax and includes these sections:

Learning Fingerprint Syntax

Fingerprint syntax consists of a variety of keywords, parameters, and operators. For specific command syntax, see the ***Fingerprint Command Reference Manual.***

About Keywords, Statements, and Lines

A Fingerprint command begins with a keyword. Keywords indicate an action, a printer setting to change, or other related information.

Keyword Examples

Keyword	Description
`BARSET`	Specifies a bar code.
`COPY`	Copies a file.
`FORMAT DATE$`	Specifies the format to be used for dates (such as YYMMDD).
`GOTO`	Branches unconditionally to a specified line.
`STORE IMAGE`	Sets up parameters for storing an image in printer memory.

In some cases, a space character is a required part of the keyword, as in LINE␣INPUT, where ␣ indicates a required space character.

Some keywords can be used in an abbreviated form (for example, PT instead of PRTXT). For more information, see the ***Fingerprint Command Reference Manual.***

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. The next table lists examples of statements.

Statement Examples

Keyword and Statement	Description
`PRTXT "HELLO"`	Keyword PRTXT indicates that the following data ("HELLO") is to be placed in a text field.
`ON BREAK 1 GOSUB 1000`	ON BREAK 1 GOSUB indicates that on the first break interrupt instruction, the program must branch to a subroutine at line 1000.
`FILES "tmp:", A`	Indicates that all files (A) in the "tmp:" directory should be listed to the printer OUT channel.

A line in a Fingerprint program may contain up to 32,767 characters and must always be terminated by a carriage return character (ASCII 13 decimal).

In Programming mode, lines are always numbered, although if you allow Fingerprint to number the lines automatically, the numbers are not visible until the program is listed. In Immediate mode or Direct Protocol, numbering is not required.

Note: By default, if you enter a carriage return on the host, the printer echoes back a Carriage Return + Line Feed (ASCII 13 + 10 decimal). With the setup option "New Line", you can restrict the printer to only echo back either a Carriage Return (ASCII 13 decimal) or a Line Feed (ASCII 10 decimal).

If you type the line numbers manually, start with number 10 and increment line numbers up by 10s (10, 20, 30, 40, etc.). That makes it easier to insert additional lines (for example 11,12,13...etc.) later.

After typing the line number, use a space character to separate it from the keyword and statement that follows, as in this example:

```
100 FONT "Univers"
```

To send multiple Fingerprint commands on the same line, add a colon (:) between each command:

```
100 FONT "Univers":PRTXT "HELLO"
```

Note: In Immediate Mode and in Direct Protocol, you can send a complete set of instructions on one line:

```
PP100,250:FT"Univers":PT"Text 1":PF ?
```

You cannot change a line after you send it to the printer. However, you can send a new line that uses the same line number to replace the existing line, or delete the line using a DELETE statement.

About Functions

A function is a statement which returns a value. A function consists of a keyword combined with values, flags, and/or operators enclosed by parentheses. The next table lists function examples.

Function Examples

Keyword and Function	Description
`CHR$(65)`	Return the readable character for ASCII code 65.
`TIME$("F")`	Return the current time based on the currently specified format.
`ABS(20*5)`	Return the absolute value of 20*5.
`IF(PRSTAT AND 1)...`	If the current position of the insertion point +1...

You can insert a function inside a statement, or on a line containing other instructions. They are often used in connection with conditional statements, as in this example:

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

You can add spaces to separate a function from other instructions on the same line, or to separate a keyword from the rest of the statement.

About Constants, Variables, and Expressions

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

- String constants are sequences of text. Numbers and other characters are considered part of the sequence and are not processed.

  String constants must always be enclosed by double quotation marks (ASCII 34 decimal); for example, "TEST.PRG". If the string constant is the last part of a line, the closing quotation mark is optional.

- Numeric constants are fixed values. Only decimal integers are allowed (1, 2, 3, and so on). Values are positive unless preceded by a minus sign (-). Optionally, you can indicate a positive value using a leading plus sign (+).

Variables also hold data, but their contents can change. You can specify the contents of a variable or use it as a container for data from Fingerprint operations. There are two types of variables:

- String variables store sequences of text. The maximum size of a string is 64 Kb (65,535 characters). String variables are indicated by a trailing $ sign, as in these examples:

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

- Numeric variables store only numbers. The maximum value of a numeric variable is 2,147,483,647. Numeric variables are indicated by a trailing % sign, as in these examples:

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

A variable name can include letters, numbers, and decimal points. The first character must always be a letter, and the complete name must not be identical to any keywords or keyword abbreviations. If part of the variable name is identical to a keyword or keyword abbreviation, other characters must precede and follow that part of the variable name or errors will result. The next table lists some examples.

Variable Name Examples

Variable Name	Description
LOC$	LOC is a keyword. This will cause an error.
LOCK$	LOC is not preceded by other characters. This causes an error.
CLOC$	LOC is not followed by other characters. This causes an error.
CLOCK$	LOC is preceded by C and followed by K. This variable name is valid.

Note: Intermec suggests that all variables and line labels start with a q.

For a list of reserved keywords, see **"Reserved Keywords and Symbols" on page 137**.

An expression can be either a constant or a variable. There are two types of expressions:

- String expressions (sometimes expressed as <*sexp*>) are carriers of alphanumeric text (string constants and string variables).
- Numeric expressions (sometimes expressed as <*nexp*>) contain numeric values, numeric variables, and operators (numeric constants and numeric variables).

About Operators

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

Using Arithmetic Operators

These operators perform calculations as described in the next table.

Arithmetic Operators

Operator	Description	Example
+	Addition	2+2=4
-	Subtraction	4-1=3
*	Multiplication	2*3=6
\	Integer division	6\2=3
MOD	Modulo arithmetic. Results in an integer value equaling the remainder of an interger division.	5MOD2=1
^	Exponent	5^2=25
( )	Specifies the order of calculation.	7+5^2\8 = 10 (7+5^2)\8 = 4

Using Relational Operators

These operators check the difference between numeric values as described in the next table.

Relational Operators

Operator	Description
<	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.

Using Logical Operators

Logical operators combine simple logical expressions to form more complicated logical expressions.

Logical Operators

Operator	Description
AND	Conjunction
OR	Disjunction
XOR	Exclusive OR

The logical operators operate bitwise on the arguments as in this example:

```
1 AND 2 = 0
```

Logical operators can be used to connect relational operators:

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

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

Examples of Logical Operator AND

A	B	A AND B
T	T	T
T	F	F
F	T	F
F	F	F

Examples of Logical OperatorXOR

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F

Examples of Logical Operator OR

A	B	A OR B
T	T	T
T	F	T
F	T	T
F	F	F

About Devices

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

Note: Use the DEVICES command to see the list of devices your printer supports.

You may need to specify a device in relation to a Fingerprint command. The next table lists available device names.

Communication Devices

Name	Refers To	Can Be OPENed For
console:	Printer display and/or keyboard	Input/Output
uart1:	Serial communication port	Input/Output
uart2:	Serial communication port (optional)	Input/Output
uart3:	Serial communication port (optional)	Input/Output
centronics:	Parallel communication	Input
net1:	EasyLAN communication (optional)	Input/Output
usb1:	USB communication port	Input/Output
finisher:	Printer finisher interface	Input/Output

Memory Devices

Name	Refers To	Can Be OPENed For
/rom	Printer firmware (Kernel) and read-only memory. Also called "rom:".	Input (files only)
/c	Main printer memory. Also called "c:" or "ram:"	Input/Output/Random
tmp:	Printer temporary memory.	Input/Output/Append/ Random (files only)

Note: Device names must be lowercase characters only and enclosed by quotation marks (for example, "/c"). Some devices must have a trailing colon (:), as shown.

Devices are referred to by name with directory commands, such as SAVE, KILL, or FORMAT, and with OPEN statements.

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:”

About Immediate Mode

In Immediate mode, Fingerprint commands are executed when a carriage return is received. Most commands can be used in Immediate mode, but cannot be saved after execution.

Immediate mode is primarily used to:

- Send commands to print a single label that is not reused.
- Send command strings which have been edited and saved as a file on the host computer. This method resembles “Escape sequences” used in other types of label printers.
- Send commands that can be used in either Immediate or Programming mode, such as DELETE, LOAD, MERGE, NEW, REBOOT, or RUN.

Any command line that does not begin with a number, but ends in a carriage return, is treated as an Immediate mode command.

Sending Command Strings in Immediate Mode

You can send command strings in Immediate Mode to print label formats. Command strings can be sent in a single line:

```
PRPOS 160,250:DIR 3:ALIGN 4:FONT “Univers”:PRTXT
“Hello”:PRINTFEED ?
```

Or, with each command on a separate line:

```
PRPOS 160,250
DIR 3
AN 4
FT “Univers”
PT “Hello”
PF
```

Note: The last example uses command abbreviations, such as PF for PRINTFEED. Not all Fingerprint commands can be abbreviated. For more information, see the ***Fingerprint Command Reference Manual.***

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 returns either an error message or "Ok" to the host.

Note: If you need more flexibility than Immediate mode provides, use Intermec Direct Protocol, since it allows variable input data to be combined with predefined layouts, handles counters, and includes a flexible error-handler. For more information, see the ***Intermec Direct Protocol Programmer's Reference Manual***.

About Programming Mode

Use Programming mode to create programs consisting of one or more program lines. The complete program can be saved in printer memory and used anytime. The program is executed when you issue a RUN statement.

Fingerprint assumes input for Programming mode:

- when a line starts with a number.
- after you disable Immediate mode by sending an IMMEDIATE OFF command.

One or several lines make up a program, which can be executed as many times as you wish. A program can be saved, copied, loaded, listed, merged, and killed. For more information, see **"Commands for Creating and Managing Program Files" on page 34**.

All program lines include line numbers that are either manually entered as the program is edited, or provided automatically and invisibly by Fingerprint after an IMMEDIATE ON statement has been executed.

The program is executed in ascending line number order when a RUN statement is entered on a line, followed by a carriage return. 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 files that start up automatically when the printer is switched on, and keep running indefinitely.

Using Line Numbers

You can manually enter line numbers as you write program lines. Intermec recommends 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 you do not use a space, Fingerprint automatically inserts a space character when the program is listed.

The next example shows a short program with line numbers:

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

The last line has no line number, and contains the RUN command plus a carriage return. This orders the printer to execute all preceding lines in consecutive ascending order according to their line numbers.

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 may be both easier and quicker as described in the next section.

Programming Without Line Numbers

To write program lines without manually entering line numbers, send the IMMEDIATE OFF command first. 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.

To make the execution branch to a certain line, such as a GOTO statement, the line 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 keywords reserved by Fingerprint. To branch to a line marked with a line label, just enter the line label (without the colon).

Finish the program by sending an IMMEDIATE ON command before you RUN it. The lines will automatically be numbered 10-20-30-40-50, and so on, but the line numbers are not visible until you LIST the program. Line labels are not replaced by line numbers.

The next example shows how line labels are used in a simple program:

```
IMMEDIATE OFF
GOSUB Q123
END
Q123:SOUND 440,50
RETURN
IMMEDIATE ON
RUN
```

If you next send the LIST command, Fingerprint automatically adds the line numbers:

```
10          GOSUB Q123
20          END
30          Q123: SOUND 440,50
40          RETURN
```

Sending Programs to the Printer

Each time a command line or program line is sent to the printer, the line is checked for possible syntax errors.

Note: If verbosity is on, the printer returns either “Ok” or an error message.

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

- One line at a time. If you have a “non-intelligent” terminal that can only transmit and receive ASCII characters, you must write and send each line separately. All lines must include line numbers. To correct a mistake, you must rewrite the complete line using the same line number.

- Copying and pasting lines from a file. If the host computer has both a communications program, such as HyperTerminal, and a text editor, you can write the program in the text editor and then copy and paste it into the communications program.
- Sending a text file to the printer. If the host computer has both a communication program and a text editor, you can write the program in the text editor and send the whole program as a text file to the printer using the communications program.

For more information, see **"Sending Programs to the Printer" on page 14**.

Commands for Editing Code

This section describes Fingerprint commands you use while editing programs in either Immediate Mode or Programming Mode:

- NEW

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

  Programs already in the working memory are deleted by a NEW statement. To keep a program you have been using, use a SAVE statement before you send the NEW statement.

- IMMEDIATE OFF | IMMEDIATE ON

  To write a program without entering line numbers, issue this statement to enter Programming mode. For more information, see **"Programming Without Line Numbers" on page 14**.

  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 using a RUN statement to start the program.

- REM

  Any characters preceded by REM are not regarded as part of the program and are not executed. Use REM to add comments to your program. REM statements can also be used at the end of lines if they are preceded by a colon (:).

- END

  Because subroutines are typically entered on lines with higher numbers than the main program, always finish the main program with an END statement to separate it from the subroutines. When an END statement is encountered, the execution is terminated and all OPENed files and devices are CLOSEd.

- LIST

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

- DELETE

  Remove program lines using the DELETE statement in Immediate mode. Both single lines and ranges of lines in consecutive order can be deleted.

- RENUM

  Program lines can be renumbered 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 are renumbered accordingly.

For debugging the program, use STOP, DBBREAK, DBBREAK OFF, DBSTDIO, DBSTEP, DBEND, or CONT commands. For more information, see **"Breaking Program Execution" on page 26**.

Using Conditional Instructions

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

Using an IF...THEN...[ELSE] Instruction

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
```

Using an IF...THEN...[ELSE]...END IF Instruction

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

This results in (for example):

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

About Branching

Both conditional and unconditional branching is possible in Fingerprint.

- For information on conditional branching, see the next section.
- For information on unconditional branching, see **"Unconditional Branching Using a GOTO Statement" on page 20**.
- For information on branching to subroutines, see the next section.

Branching to Subroutines

A subroutine is a range of program lines intended to perform a specific task separately from the main program execution. For example, branching to subroutines can occur 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.

You can also branch to a subroutine from different places in the same program. You only need to write the routine once, making the program more compact.

The instruction for unconditional branching to subroutines is the GOSUB statement. After branching, the subroutine is executed line by line until a RETURN statement is encountered.

The same subroutine can be branched to as often as needed 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.

Subroutines should be placed on lines with higher numbers than the main program. Append the main program with an END statement to avoid unintentional execution of subroutines.

The next example illustrates 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
```

Instructions for Conditional Branching

Conditional branching means that the program execution branches to a certain line or subroutine when a specified condition is met. The following instructions are used for conditional branching:

Using an IF...THEN GOTO...ELSE Instruction

If a specified condition is TRUE, the program branches to a certain line, but if the condition is FALSE, something else is done as shown in the next 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
```

Using an ON...GOSUB Instruction

Depending on the value of a numeric expression, the execution branches to one of several subroutines. If the value is 1, the program branches to the first subroutine in the instruction, if the value is 2 it branches to the second subroutine, and so on. The next example includes such an instruction:

```
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
```

Using an ON...GOTO Instruction

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

ON...GOTO is shown in this 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
```

Using an ON BREAK...GOSUB Instruction

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

In the next example, the program is interrupted when the **Shift** and **Pause** keys on the printer keyboard are pressed. 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.

```
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
```

Using an ON COMSET...GOSUB Instruction

When one of several specified conditions interrupts the background communication on a certain communication channel, the program branches to a subroutine, such as reading the buffer. The interrupt conditions (end character, attention string, or maximum number of characters) are specified by a COMSET statement as in this 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
```

Using an ON KEY...GOSUB Instruction

To use the printer keypad, each key can be enabled individually using a KEY ON statement and assigned to a subroutine using an ON KEY GOSUB statement. The subroutine should contain the instructions you want 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. For more information, see **"Using the Printer Keypad" on page 109**.

Note that ON KEY...GOSUB excludes data input from the printer keypad.

This example shows how the two unshifted keys **F1** (ID 10) and **F2** (ID 11) are used to change the printer contrast:

```
10          PRPOS 100,500
20          PRLINE 100,100
30          FONT "Univers"
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
```

Unconditional Branching Using a GOTO Statement

The simplest type of unconditional branching is the waiting loop, which means that a program line branches the execution to itself and waits for something to happen, such as a keypress.

This example shows how the program waits for the **F1** key to be pressed (line 30). When the key is pressed, the printer beeps:

```
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, as in this 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 lines (100 or 200), which print different messages. In either case, 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. For more information, see **"Instructions for Conditional Branching" on page 18**.

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

Branching to an Error-Handling Subroutine

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

Using an ON ERROR GOTO Instruction

ON ERROR GOTO branches the execution to a specified line when an error occurs, ignoring the standard error-trapping routine. If the line number is specified as 0, the standard error-trapping routine is used.

Resuming Execution After Error Handling

Use a RESUME statement to resume execution after an error-handling subroutine has been executed. RESUME is only used in connection with ON ERROR GOTO statements and can be used as follows:

- RESUME or RESUME 0 - Execution is resumed at the statement where the error occurred.
- RESUME NEXT - Execution is resumed at the statement after 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 is resumed.

```
10          ON ERROR GOTO 1000
20          PRTXT "HELLO"
30          PRINTFEED
40          END
1000        IF ERR=1019 THEN FONT "OCR-A" 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
```

About Loops

One type of loop has already been described in connection with the GOTO statement, where GOTO referred to the same line or a previous line. There are two instructions for using more advanced loops:

Using a FOR...NEXT Instruction

These statements create loops in which a counter is incremented or decremented until a specified value is reached. The counter is defined by a FOR statement as follows:

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

All program lines following the FOR statement are 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 is executed again. This is 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. Each loop must have a unique counter designation in the form of a numeric variable. The NEXT statement makes the execution loop back to the most recent FOR statement. 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 host keyboard can be printed with an even spacing:

```
10          FONT "Univers"
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
```

The next example includes 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
```

This results in:

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

This example shows how to create an incremental counter:

```
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 "Univers",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
```

Using a WHILE...WEND Instruction

This instruction creates loops in which a 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 are 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
```

Structuring Your Program

Use the structure below as a guideline for building your Fingerprint programs.

1. Program Information
   - Use REM to comment out items such as program type, version, release date, and byline.
2. Initiation

   Determines how the printer works and branches to subroutines as needed.
   - References to subroutines: ON BREAK GOSUB, ON COMSET GOSUB, ON ERROR GOSUB, ON KEY GOSUB, or other commands as necessary.
   - Printer setup: SETUP, OPTIMIZE ON/OFF, LTS& ON/OFF, CUT ON/OFF, FORMAT DATE$, FORMAT TIME$, NAME DATE$, NAME WEEKDAY$, SYSVAR, or other commands as necessary.
   - 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.
3. Main Loop

   Executes the program and keeps it running in a loop.
   - Reception of input data: INPUT, INPUT#, INPUT$, LINE INPUT#.
   - Printing routine: FORMFEED, PRINTFEED, CUT.
   - Looping instructions: GOTO.
4. Subroutines
   - Break subroutines: BREAK ON/OFF, BREAK.
   - Background communication subroutines: COM ERROR ON, COM ERROR OFF, COMSET, COMSET ON, COMSET OFF, COMBUF$, COMSTAT.
   - Subroutines for key-initiated actions: ON KEY.
   - Subroutines for display messages: PRINT#.
   - Error handling subroutines: ERR, ERL, PRSTAT.
   - Label layout subroutines: PRPOS, DIR, ALIGN, FONT, BARSET, PRTXT, PRBAR, PRIMAGE, PRBOX, PRLINE, and so on.

Executing the Program

To start the execution of the program currently residing in the printer working memory, issue a RUN statement.

Note: Do not issue a RUN statement on a numbered line, or on a line without a number in Programming Mode, or a "RUN statement in program" error occurs.

By default, program execution starts at the line with the lowest number and continues in ascending line number order, with the exception of possible loops and branches. Optionally, you can start execution at a specified line (for example, RUN 40 starts at line 40).

Use an EXECUTE statement to execute a program that is not currently loaded, or to execute Fingerprint programs from within another Fingerprint program.

When you are connected to the printer through a serial connection, the first error that stops the execution causes an error message to be returned to the host screen.

In case of program errors, the number of the line where the error occurred is 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 an error-handling routine is included in the program.

Writing, Executing, and Listing a Short Program

Follow the next procedure to write a short Fingerprint program, execute the program, and list it.

To write, execute, and list a short program

1. Connect the printer to a host PC and start a communications program on the host PC. For help, see **"Sending Fingerprint Commands to the Printer" on page 3**.
2. In a communications program, type `NEW` and press **Enter**. The printer returns "`Ok`".
3. Type `IMMEDIATE OFF` and press **Enter**. The printer returns "`Ok`".
4. Type the following text and press **Enter** at the end of each line:

```
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
```

   The printer returns "`Ok`".

5 Type `RUN` and press **Enter**. The printer executes the program and prints the text to the communications program window.

6 Type `LIST` and press **Enter**. The program is listed with line numbers.

Breaking Program Execution

You may write some programs that start automatically when the printer is turned on. Because there is no default break facility from the host via any communication channel, you should always include some break facilities in auto-start programs.

Note: If the startup program is stored on an external device, you can disconnect the device and restart the printer.

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 temporary memory and is not removed until the printer is restarted, unless you specifically delete it using a BREAK...OFF statement for the device.

In all break-related instructions, the serial communication channels and the keyboard are referred to by numbers:

0 = “console:” (the printer 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).

Using a BREAK Statement

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 keypad.

The default interrupt character for all serial channels is ASCII 03 dec. (ETX), or ASCII 158 dec. from the printer keypad (by pressing the **Shift** and **Pause** keys simultaneously).

Using a BREAK...ON or BREAK...OFF Statement

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

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

Using an ON BREAK ...GOSUB...Statement

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, when a break occurs the printer could branch the execution to another part of the program, show a message in the display, emit a warning signal, or ask for a password.

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

Note: A break interrupt character is saved in the printer 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.

This example shows how a break interrupt occurs 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 display.

```
10          BREAK 1,88
20          ON BREAK 1 GOSUB 1000
30          GOTO 50
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
```

Saving the Program

Use the SAVE command to save the current program. Programs can be saved in the printer permanent memory ("/c") or to a USB storage device ("usb1:"). You can also save a program in temporary memory ("tmp:"), but the program will be lost at power off or a power failure. Use the LIST command to list the program back to the host in order to make a backup copy.

For more information on printer memory, see **"About Printer Memory" on page 119**.

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 the file extension.

The filename can be specified in either uppercase or lowercase characters, but lowercase characters are automatically converted to uppercase when the program is saved.

If you omit the extension, Fingerprint automatically adds the extension ".PRG". If you plan to transfer the program file to a host platform, you need to consider conventions and restrictions imposed by the host operating system when you name the program.

The automatic case conversion and adding of the extension can be disabled using SYSVAR(43). For help, see **"Using the SYSVAR System Variable" on page 116**.

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.

```
SAVE "usb1:PROGRAM1.TXT"
```

saves the program as PROGRAM1.TXT to a USB storage device connected to the printer.

Protecting the Program

When a program is saved, you have the option to also protect it, meaning 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, so you should make an non-protected backup copy to use if you need to make any changes later.

The next example saves and protects the current program as PROGRAM1.PRG in the current directory:

```
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 the risk of interfering line numbers. Both programs should make use of line labels for referring to other lines, such as loops and branching instructions.

The next example saves the current program as PROGRAM1.PRG without line numbers in the current directory:

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

Making Changes

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

The next example changes the value of a variable in line 50, and replaces the original version with the new version:

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

Making Copies of Programs

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

The next example copies a program from the permanent memory to a 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 or in another directory, you will create a copy of the original program.

The next 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 (or any other file), 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"
```

Creating a Startup Program

The MKAUTO.PRG program is used to create autoexec.bat-files, which are programs that are loaded and run automatically as soon as the power is switched on and the printer is 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 printers memory. If there are more startup files, only one will be selected based on 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 permanent memory (device "/c").

3 The PUP.BAT file (Intermec Shell) in the read-only part of the printer permanent memory (device "/rom").

The MKAUTO.PRG program is included in the systems part of the printer 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
```

Follow the next procedure to create a startup program from an ordinary program:

To create a startup program

1 Connect the printer to a host PC and start a communications program on the host PC. For help, see **"Sending Fingerprint Commands to the Printer" on page 3**

2 Write and test your program.

3 Type `SAVE "MyFileName"` and then press **Enter**. The printer returns "`Ok`".

4 Type `RUN "/rom/MKAUTO"` and then press **Enter**. The printer returns:

```
Startup file name:
```

5 Type the name of the program you just saved (with or without the extension .PRG) and then press **Enter**. The printer returns "`Ok`".

Your program has been saved as a startup program in the current directory. When you restart the printer, the new startup program will start running, provided there is no other startup program with higher priority as described earlier.

6 (Optional) To undo the operation, type `KILL "AUTOEXEC.BAT"` and press **Enter**.

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".

3

Managing Files

This chapter describes a Fingerprint printer file system and how to manage files, including these sections:

Using Directories in the Printer File System

The read-only memory (/rom) and the read/write permanent storage memory (/c) in the printer support the use of directories. However, directories cannot be used in any other parts of the memory.

Use a slash character (/) to separate directories and files, as in the path "/c/DIR1/DIR2/FILE". The maximum length of a path is 255 characters. Use the slash only to indicate directories in /c or /rom, as other memory partitions (such as "net1:") do not support directories.

Note: For backward compatibility, "c:" is equivalent to "/c", and "rom:" is equivalent to "/rom". New applications should always use "/rom" or "/c".

These Fingerprint commands are used when working with directories:

- MKDIR creates a new directory in the printer permanent memory.
- CURDIR$ returns the current directory as the printer stores it.
- DIRNAME$ returns the directory names in a specified part of printer memory.

The FILES command gives a size of 0 for directories to minimize impact on applications that parse the output.

The FILENAME$ function only reports files to minimize impact on applications that use FILENAME$ to get file listings.

Note: For more information on commands, see the ***Fingerprint Command Reference Manual***.

Using Path Shortcuts

As a shortcut, each directory (including the root directories) contains a "parent directory". Use two periods (..) to change to the parent directory. Send the FILES,A command to list the files in the parent directory.

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 current directory.
`COPY "../DIR3/FILE", "FILE"`	Copies /c/DIR1/DIR3/FILE to /c/DIR1/DIR2/FILE.
`CHDIR ".."`	Goes up to "/c/DIR1".
`CHDIR "../"`	Goes up to /c. Note that a trailing slash can be used.

Note: While a file or directory name can contain all printable characters except ":" (colon) and "/" (slash), only "/c" supports using directories.

About File Types

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

- Program files
- Data files
- Image files. For more information, see **"Understanding Images and Image Files" on page 70**.
- Font files. For more information, see **"Managing Fonts" on page 66**.

Commands for Listing Files

The files stored in the printer memory can be listed using a FILES statement or a FILENAME$ function, as in these 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.

`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:"
```

Listing a File With the FILELIST Program

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

To list a file

1. On your terminal, enter:

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

   The printer prompts you to enter the name of the file to be listed:

   ```
   Filename?
   ```

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

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

3. Press **Enter**. The file is listed.

Commands for Creating and Managing Program Files

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. For more information, see **"About Programming Mode" on page 13**.

A startup file (also called an autoexec-file) is a program file that automatically runs when the printer is switched on. For more information, see **"Creating a Startup Program" on page 29**.

Use these Fingerprint commands for creating and handling program files:

- LOAD copies a specified program file to the printer working memory.
- LIST lists the program file in the working memory to the standard OUT channel, usually the screen of the host.
- MERGE adds a copy of a specified program file to the program file currently residing in the printer working memory.
- RUN executes the instruction in the program file. Must be issued in 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 in that directory, it is 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.

Commands for Creating and Managing Data Files

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

- Sequential input files
- Sequential output files
- Append files
- Random access files

Use these Fingerprint commands for creating and handling 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.
- 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.

Commands for Transferring Text and Binary Files

You can use these methods to transfer files:

- PrintSet
- the printer web page
- a USB storage device
- FTP
- SmartSystems

Text files (such as program files and data files in ASCII format) can be sent to the printer using a communication program. Text files can also be transferred back to the host, such as for backup purposes, by loading the file and using the LIST command to send its contents to a communication program.

For binary files, you can also use the TRANSFER KERMIT and ZMODEM commands.

Using the TRANSFER KERMIT Statement

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.

In this example, the printer is set up to receive a file on the standard IN channel:

```
TRANSFER KERMIT "R"
```

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

Using the ZMODEM Protocol

Files can be sent from host to printer (or vice versa) with the ZMODEM protocol. For more information, see the ***Fingerprint Command Reference Manual.***

Using a TRANSFER STATUS Statement

After a file has been transferred using a TRANSFER KERMIT or TRANSFER ZMODEM statement, the transfer can be checked using the TRANSFER STATUS statement. The statement places 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
```

Commands for 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.

Checking Transferred Files With CHECKSUM

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. For more information, see **"Commands for Working With Arrays" on page 36**.

In this example, the checksum in the lines 10 to 90,000 of a program is calculated:

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

Commands for Working With 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 (maximum 10.) Each array variable consists of a name and a number of subscripts, separated by commas, and enclosed by parentheses, as in this example:

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

The first time an array is referred to, the number of subscripts in an array variable decides its number of dimensions. The number of elements in each dimension is restricted to four (numbered 0 to 3) by default.

There are four commands that are particularly relevant for working 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.

Specifying Array Dimensions Using DIM

If more than four elements are needed, or to limit the size of the array, use a DIM statement to specify the number of dimensions as well as the number of elements in each dimension.

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 Color (0-4): ", B%
190         INPUT"Select Size (0-4): ", C%
200         PRINT TYPE$(A%)+ ", "+COLOUR$(B%)+", "+SIZE$(C%)
RUN
```

Sorting Arrays

The SORT statement sorts a one-dimensional array in ascending or descending order according to the ASCII values for the character 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 is 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
```

The printer returns:

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

Splitting String Expressions

The SPLIT function splits 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$"
30          DIM ARRAY$(5)
40          B%=SPLIT(A$,X$,47)
50          FOR C%=0 TO (B%-1)
60          PRINT ARRAY$(C%)
70          NEXT
RUN
```

The printer returns:

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

Calculating String Array Checksums

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. For help, see **"Checking Transferred Files With CHECKSUM" on page 36**.

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
```

The printer returns:

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

4

Managing Input and Output

This chapter explains how to manage input and output data for Fingerprint applications, and includes these topics:

Preprocessing Input Data

All input data comes to the printer in binary form. Text files are transmitted in ASCII format and preprocessed by the printer firmware. These Fingerprint commands can be used to provide file compatibility between the printer and the host:

- MAP
- NASC

A character received by the printer on a communication channel is first processed as directed by any included MAP statements. Then the character is checked for any COMSET or ON KEY... GOSUB conditions. When a character is to be printed, it is processed according to the character set selected using a NASC statement.

Modifying Character Sets Using a MAP Statement

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, select the set that comes closest and modify it using MAP statements.

Note: Do not map any characters to ASCII values occupied by characters used in Fingerprint instructions (such as keywords, operators, %, $, #, and certain punctuation marks). Mapped characters are reset to normal at power-up or reboot.

For a list of character sets and the corresponding reference numbers, see ***Fingerprint Command Reference Manual.***

For example, you may want to use the German character set (49) and 7 bit 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 "Univers"
40          PRPOS 100,100
50          INPUT "Enter character";A$
60          PRTXT A$
70          PRINTFEED
RUN
```

The printer returns:

```
Enter character?
```

Note: When using 7 bit communications, the printer cannot echo back the correct character to the host if its ASCII value exceeds 127. Although semicolon characters appear onscreen, the desired "£" characters are printed on the label.

Choosing a Character Set with a NASC Statement

The NASC statement is used to select a single- or double-byte character set, making it possible to adapt the printer to various national standards. By default, characters are printed according to the Roman 8 character set.

Fingerprint supports right-to-left and bidirectional text, as well as cursive glyphs, character shaping, and connecting headstrokes. You must specify a valid font and character set for your current language when printing complex scripts. You can use any TrueType® font (or TrueType-formatted OpenType® font) with your printer, and you can purchase additional fonts from Monotype at **www.fonts.com**.

While most alphanumeric characters and punctuation marks are the same from set to set, many international characters and symbols are different. For example, the ASCII 124 character is "|" according to the Roman 8 character set, "ù" according to the French character set, and "ñ" according to the Spanish set.

If no character set matches your requirements exactly, select the one that comes closest. Then, you can make final corrections using MAP statements as described in the previous section.

Using a NASC statement has the following consequences:

- Text is printed according to the selected character set. However, instructions concerning the printable label image that have already been processed before the NASC statement are not affected. You can use this to print multilingual labels.
- New messages on the display are affected by the most recent NASC statement. However, a message that is already displayed is not be updated automatically. The display can show most printable Latin characters. In Setup mode, all characters are mapped according to the US-ASCII standard.
- Data transmitted from the printer via any of the communication channels is not affected, since the data is defined by ASCII values and not as alphanumeric characters. The active character set of the receiving unit determines the graphic presentation of the input data (for example, on the screen of the host).
- For 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 stored in the print buffer is not affected.

This example selects the Italian character set:

```
NASC 39
```

Converting Input Data

These Fingerprint commands are used to convert data in numeric or string expressions:

- ABS returns the absolute value of a numeric expression.
- ASC returns the ASCII value of the first character in a string expression.
- CHR$ returns the readable character of a specified ASCII value. This is useful when a printer keyboard cannot produce a particular character.
- FLOATCALC$ calculates float numbers using arithmetic operators.
- FORMAT$ formats a number represented by a string and is typically used with FLOATCALC$.
- INSTR searches a string for a specific character or string of characters and returns its position if found.
- LEFT$ returns a specified number of characters from the beginning (left end) of a string.
- LEN returns the total number of characters and spaces in a string expression.
- MID$ returns a portion of a string expression. You can specify the start position and the number of characters to return.
- RIGHT$ returns a specified number of characters from the end (right side) of a string.
- SGN returns the sign of a numeric expression.
- SPACE$ returns a specified number of space characters. This command is useful for creating tables with monospace characters.
- STR$ returns the string representation of a numeric expression.
- STRING$ returns a specified number of a single character specified by its ASCII value.
- VAL$ returns the numeric representation of a string expression. This is typically used with random files, which only accept strings.

Note: Commands ending in $ typically return a string.

Generating Random Numbers

The Fingerprint commands RANDOM and RANDOMIZE are used to generate random numbers for test programs or other applications.

Calling the RANDOM Function

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
```

Using a RANDOMIZE Statement

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
```

The printer returns:

```
Random Number Seed (0 to 99999999) ?              (prompt)
```

For a higher degree of randomization, you can reseed the random integer generator with another random integer provided by a function such as TICKS:

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

The printer returns:

```
42
```

Setting the Standard IN and OUT Channels

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

You can configure any of the following communication channels as standard IN and/or standard OUT channel using the SETSTDIO statement. The next table lists valid values for SETSTDIO.

SETSTDIO Values

Value	Standard IN Channel	Standard OUT Channel
0	"console:"	"console:"
1	"uart1:"	"uart1:"
2	"uart2:"	"uart2:"
3	"uart3:"	"uart3:"
4	"centronics:"	Not applicable.
5	"net1:"	"net1:"
6	"usb1:"	"usb1:"
100	100 = "auto" (default)	100 = "auto" (default)

Note: Do not choose "console:" for both the standard IN and OUT channels, which makes only characters entered on the printer keypad appear in the display.

Input From a Host

The following Fingerprint commands can receive input from any communication channel:

- OPEN
- INPUT#
- INPUT$
- LINE INPUT#
- CLOSE

The standard IN channel is used for sending instructions and data from the host to the printer to perform a variety of tasks, such as controlling the printer in Immediate Mode, creating programs in Programming Mode, downloading program files, or transmitting input data.

The following Fingerprint commands receive data only on the standard IN channel:

- INKEY$
- INPUT
- LINE INPUT

Input From Sequential Files

To read from a sequential file (or a communication channel other than the std IN channel), the file 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.

In this 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, use these Fingerprint commands to read the data stored in the file or device:

- INPUT#
- INPUT$
- LINE INPUT#
- CLOSE

Reading Data to a Variable With INPUT#

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.

This example reads six records in a file and places the data in 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
```

The printer returns:

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

Reading a Specific Data Length With INPUT$

INPUT$ reads a specified number of characters from the specified sequential file or channel. By default, if no file or channel is specified, the data on the standard IN channel is 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 resumes 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 has been read, they cannot be read again until the file is closed and opened again. Subsequent INPUT$ statements 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
```

The printer returns:

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

Reading a Line to a Variable With LINE INPUT#

This command reads an entire line (including all punctuation) to a string variable. Commas inside a string are treated as punctuation marks and do not divide the string into records.

This example reads a complete line in a file and places the data in 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
```

The printer returns:

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

Close a File

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.

Verify the End of a File With EOF

The EOF function is used in connection with INPUT#, LINE INPUT#, or INPUT$ statements to avoid the "Input past end" error condition.

When the EOF function encounters the end of a file, it returns the value -1 (TRUE.) If not, it returns the value 0 (FALSE).

The next example shows how to use EOF:

```
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
```

Counting Data Blocks with LOC

LOC 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 number 100 has been read from the file:

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

Determining File Length with LOF

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

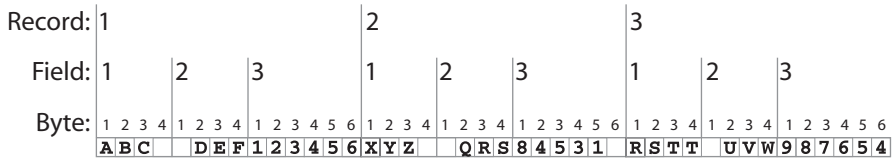
The example illustrates how the length of the file "PRICELIST" is returned:

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

Input From a Random File

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

This section uses the random file created in **"Generating Random Numbers" on page 43**, which can be illustrated as follows:




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

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. For more information, see **"Converting Input Data" on page 42**.

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

- FIELD
- GET
- CLOSE
- LOC
- LOF

Creating a Buffer with FIELD

A FIELD statement is used to create a single-record buffer for a random file, and divides the buffer into fields. Each field can be assigned a string variable.

Using the random file example, the following code chooses Field #1, and assigns the first 4 bytes to the string variable F1$, the second 4 bytes to F2$, and the remaining 6 bytes to F3$:

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

Copying a Specific Field with GET

A GET statement copies the desired record from the file. You can select any record (as opposed to sequential files, where records are read one after the other). For this example, we will copy the first record:

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

You can copy data from other records in the same file by issuing additional GET statements.

Now you can use the variables assigned to the fields using the FIELD statement to handle the data. Numeric expressions converted to string format before being put into the record can now be converted back to numeric format using VAL functions. In this example, the following code displays the data on the host screen:

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

Closing a File

Finally, close the file and execute:

```
50 CLOSE #1
RUN
```

The printer returns:

```
ABC DEF 123456
```

Finding the Last Field Read with LOC

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

This example closes the file "ADDRESSES" when record number 100 has been read from the file:

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

Determining File Length with LOF

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)
.....
.....
```

Input From the Printer Keypad

The input that can be provided from the printer keypad depends on your printer model and options.

Note: This section does not apply to keypad input for ON KEY...GOSUB statements and vice versa.

The following Fingerprint commands are used in connection with input from the printer keyboard:

- OPEN (opens the device "console:" for sequential INPUT)
- INPUT#
- INPUT$
- LINE INPUT#
- CLOSE

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

In case of INPUT# and LINE INPUT#, the input is not accepted until a carriage return is issued.

This example demonstrates how the printable character and decimal ASCII value of various keys on the printer keyboard can be printed to the screen of the host.

```
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
```

Controlling Communication

The following Fingerprint commands are used to control the communication between the printer and the host (or other connected devices):

- BUSY
- READY
- ON | OFF LINE
- VERBON | VERBOFF
- SYSVAR(18)

Using BUSY or READY Statements

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:

- For serial communication, the type of busy/ready signal is decided in the Setup Mode (Ser-Com; Flowcontrol):
  - 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).

  For more information, see the printer user's guide.
- 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, as in this example:

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

Note: Issuing a READY statement is no guarantee that the printer will receive data. There may be other conditions that prevent the printer from receiving data, such as a full receive buffer.

Using an ON LINE | OFF LINE Statement

These two statements are only used for the parallel Centronics communication channel and control the SELECT signal (pin 13 on the parallel interface board).

Controlling Printer Response with VERBON | VERBOFF

These commands control the printer verbosity, which refers to the printer response (on the standard OUT channel) to instructions received on the standard IN channel:

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

All responses are suppressed when a VERBOFF statement is issued. However, VERBOFF does not suppress question marks or other prompts displayed as a result of another command, such as an INPUT statement. Instructions like DEVICES, FILES, FONTS, IMAGES, LIST and PRINT also work normally.

When the printer receives a character, such as from the host keyboard, the same character is echoed back on the standard OUT channel by default. When an instruction has been checked for syntax errors and accepted, the printer returns "Ok". Otherwise 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
```

Managing Background Communication

Background communication means that the printer receives data on an IN channel while the program runs in a loop. The data is stored in a buffer that can be emptied at an appropriate moment by the running program, which then uses the data.

Background communication buffers are not the same as the receive buffers. Any input received is first stored in the channel receive buffer, awaiting processing. After processing, you can store the data in the background communication buffer.

The following Fingerprint commands are used in connection with background communication:

- COMSET sets the background reception parameters, including:
  - communication channel.
  - start and end character(s) of message string.
  - characters to be ignored.
  - attention string that interrupts 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 background 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.

Background Communication Example

This example uses the various Fingerprint commands to set up background communication. For specifics on each command, see the ***Fingerprint Command Reference Manual.***

To set up the printer for background communication

1. Enable the error handling for the desired background communication channel using a COM ERROR ON statement. For specifics, see COM ERROR ON in the ***Fingerprint Command Reference Manual.***

   It may be useful to create a few messages to indicate what caused the interruption. In this 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"
   ```

2. Continue with a COMSET statement specifying:

   - the communication channel to be used.
   - the character or string of characters used to tell the printer to start receiving data and to stop receiving data.
   - the character or characters to be ignored (filtered out from the received data).
   - the character or string of characters to use as an attention string that interrupts reception.

     Note: 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 and ETX can be specified using a CHR$ function. To specify no character, use an empty string.

   - the number of characters 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).

In this example, the background reception is set to channel "uart1:", the Start character is A, the End character is CHR$ (90) (the character Z), the character to be ignored is #, the attention string is BREAK, and the maximum number of characters in the buffer is 20:

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

3 Use an ON COMSET GOSUB statement to specify a subroutine to branch to when reception is interrupted. Interruption occurs when any of the following conditions are fulfilled:

- an end character is received.
- an attention string is received.
- the maximum number of characters have been received.

In this example, when the reception of data on communication channel 1 ("uart1:") is interrupted, the execution branches to a subroutine starting on line number 1000.

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

4 Turn on the COMSET:

```
80 COMSET 1 ON
```

Note: The COMSET interrupt must be turned on after it has occurred and been resolved.

5 When reception is interrupted, check the buffer contents. 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 reasons of interruption as specified by COMSET:

- Max. number of characters received (2).
- End character received (4).
- Attention string received (8).
- Communication error (32).

Different messages to be printed to the standard OUT channel, depending on what interrupted communication. By assigning the COMSTAT value to a numeric variable, execution is faster than checking the COMSTAT value several times for different values, as seen in this example:

```
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$
```

To temporarily turn off background reception during some part of the program execution, issue a COMSET OFF statement, and then turn reception on again using a new COMSET ON statement.

Note: Because COMSET ON/OFF statements empty the buffer, use COMBUF$ to read the buffer contents first.

6 Add a few lines to print the content of the buffer (line 1060) and create a loop that waits from input from the host (line 90). The entire example looks like this:

```
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
```

7 You can test the example by pressing **Enter** on the host keyboard. Then enter various characters and see what happens, starting with the start character, stop character, ignore character, attention string, and maximum number of characters parameters in the COMSET statement.

Retrieving Buffer Status With LOC or LOF

LOC and LOF return the status of the receive or transmitter buffers in an OPENed communication channel.

If the channel is OPENed for INPUT:

- LOC returns the remaining number of characters to be read from the receive buffer.
- LOF returns the remaining free space (in bytes) in the receive buffer.

If the channel is OPENed for OUTPUT:

- LOC returns the remaining free space (bytes) in the transmitter buffer.
- LOF returns the remaining number of characters to be transmitted from the transmitter buffer.

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%
...
...
```

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
```

Setting Up RS-422 Communication

Some Intermec printers can be fitted with an optional interface board that provides RS-422 connectivity (isolated or non-isolated) on "uart2:" or "uart3:".

Neither of these 4-line protocols provide the hardware handshake (RTS/CTS) feature, but XON/XOFF or ENQ/ACK can be used if so desired. Two lines transmit data and the other two receive data.

After you install the interface board in the printer, use the next procedure to set up RS-422 communication.

To set up the printer for RS-422 communication

1. Set the printer flow control as follows:

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

2. Use the SETSTDIO statement to set "uart2:" or "uart3:" as the standard I/O channel.

Output to the Standard OUT Channel

The standard OUT channel returns the printer responses to instructions received from the host. For simplicity, the same device is usually selected for both standard IN and OUT channels.

For every instruction received on the standard IN channel, the printer returns "Ok" or an error message (such as "Feature not implemented" or "Syntax Error") on the standard OUT channel. If the standard OUT channel is connected to the host computer, the message appears onscreen.

Use VERBOFF/VERBON statements to turn the verbosity off or on. The verbosity level can be selected by SYSVAR(18), and the type of error message can be selected by SYSVAR(19).

These Fingerprint commands return data only on the standard OUT channel:

- PRINT
- PRINTONE
- DEVICES
- FILES
- FONTS
- IMAGES
- LIST

Printing Expressions With PRINT

PRINT prints a line on the standard OUT channel (typically, to the screen of the host). The PRINT statement can be followed by one or more string or numeric expressions.

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

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

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

  The printer returns:

  ```
  Price      $10
  ```

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

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

  The printer returns:

  ```
  Price_$10
  ```

- A plus sign (+) places the following string expression immediately adjacent to the preceding string expression. Example:

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

  The printer returns:

  ```
  Price_$10
  ```

Each line is terminated by a carriage return to make the PRINT statement start on a new line. However, if a PRINT statement is appended by a semicolon, the carriage return is suppressed and the next PRINT statement is printed on the same line as the preceding one:

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

The printer returns:

```
Price_$10_per_dozen
```

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

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

The printer returns:

```
50
A
```

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

Printing Characters by ASCII Values With PRINTONE

PRINTONE prints the alphanumeric representation of one or more characters, specified by their respective ASCII values, to the standard OUT channel. The PRINTONE statement is useful 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
```

The printer returns:

```
Price $ 10
```

Note: The returned ASCII value depends on the currently selected character set, and on the current keypad mapping.

Redirecting Output to a File

Some Fingerprint commands return data on the standard OUT channel by default. However, it is possible to redirect such output to a file using a REDIRECT OUT statement.

When a REDIRECT OUT statement is issued with an appending string expression (REDIRECT OUT <*sexp*>), the expression specifies the name of a sequential file in which the output is stored. In this case no data is echoed back to the host.

When no file name appends the statement, the output is directed back to the standard OUT channel.

In the following example, the output is redirected to the file "IMAGES.DAT". The images in the printer memory are 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
```

Output to Sequential Files

This section describes the commands you use in connection with output to sequential files.

Using an OPEN Statement

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 is replaced.
- APPEND means that new data is 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 is 128 bytes). Up to 10 files and devices can be open at the same time.

In this example, the file "ADDRESSES" is opened for output and given the reference number 1:

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

In this example, the file "PRICELIST" is opened for appended data and is given the reference number 5:

```
OPEN "PRICELIST" FOR APPEND AS #5
```

Printing Expressions to a Sequential File With PRINT#

PRINT# prints data entered as string or numeric expressions to a sequential file. For more information, see **"Printing Expressions With PRINT" on page 56**.

There are two ways to divide the file into records:

- Each PRINT# statement creates a new record as seen in lines 20-40 in the example.
- Commas inside a string divide the string into records, as seen in line 50 in the example.

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"
```

Printing Characters by ASCII Values With PRINTONE#

The PRINTONE# statement prints characters entered as decimal ASCII values according to the selected character set to the selected file or device. For more information, see **"Printing Characters by ASCII Values With PRINTONE" on page 57**.

This 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
```

Using a CLOSE Statement

After writing data to the file, close it using the same reference number as when it was opened, as in this 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
```

Counting Data Blocks and Determining File Length With LOC and LOF

Use LOC to return the number of 128-byte blocks that have been written since the file was opened. For an example, see **"Counting Data Blocks with LOC" on page 47**.

LOF returns the length (in bytes) of a file that has been opened. For an example, see **"Determining File Length with LOF" on page 47**.

Output to Random Files

These Fingerprint commands are used in connection with output to random files:

- OPEN
- FIELD
- LSET/RSET
- PUT
- CLOSE
- LOC
- LOF

Opening a File for Random Input or Output With 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, as in this example:

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

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

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

Creating a Buffer With FIELD

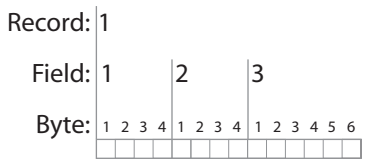
Next, create a buffer using a FIELD statement. The buffer is given a reference number and divided into a number of fields, each with a specified number of characters. A string variable is assigned to each field.

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$
```

The record produced looks like this:



The file can consist of many records, all with the same format. To produce 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 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. You can convert numbers to strings using the STR$ function.

Left or Right Justifying Data With LSET and RSET

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

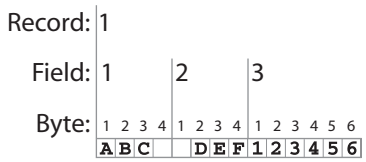
- LSET places the data left-justified.
- RSET places the data right-justified.

In other words, if the input data consists of fewer bytes than the field into which it is placed, it is placed either to the left (LSET) or to the right (RSET).

If the length of the input data exceeds the size of the field, the data is truncated from the end (LSET), or from the start (RSET).

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

This set of instructions produce the following record:



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, then digits 1 and 2 would have been omitted instead.

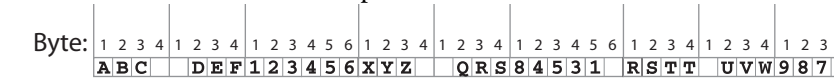
Transferring Data to the File with PUT

The next step is to transfer the record to the file using 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
```

You can place data into other records using additional sets of LSET, RSET and PUT statements. Below is an example of a three-record file:



Using a CLOSE Statement

When you are finished, close the file:

```
100          CLOSE #1
```

Nothing actually happens before you execute the program using a RUN statement. Then the data is placed into the fields and records as specified by the program, as in this 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
```

Finding the Last Field Read and Determining File Length With LOC and LOF

Use LOC to return the number of the last record read by the use of GET statement. For an example, see **"Finding the Last Field Read with LOC" on page 49**.

LOF returns the length (in bytes) of a file that has been opened. For an example, see **"Determining File Length with LOF" on page 47**.

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. For more information, see **"Output to Sequential Files" on page 58**.

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

These Fingerprint commands are used in connection with output to a communication channel:

- OPEN
- PRINT#
- PRINTONE#
- CLOSE
- LOC
- LOF
- COPY

In this example, “Record 1” and “Record 2” are printed 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
```

In this example, the file “datafile” (stored on a USB storage device) is printed to the serial communication channel “uart2:”:

```
COPY "usb1:datafile","uart2:"
```

Output to the Printer Display

The only device other than the serial communication channels that can be opened to receive output from a Fingerprint program is the printer display (“console:”). For more information, see your printer user manual, or the ***Fingerprint Command Reference Manual.***

5

Managing Fonts, Bar Codes, and Images

This chapter explains how to manage fonts, bar code printing, and images, and includes these topics:

Managing Fonts

Fingerprint includes a variety of commands you can use to manage fonts and font printing. For more information about the fonts included by default with your printer, see the ***Fingerprint Command Reference Manual.***

About Font Types

Fingerprint supports scalable fonts in TrueType® (.ttf) format, and TrueType-based OpenType® format. You can also purchase additional fonts from Monotype at **www.fonts.com**.

Single-byte fonts are mapped in the range of ASCII 0-127 dec (7-bit communication) or ASCII 0-255 dec (8-bit communication). Some examples are Latin, Greek, Cyrillic, Arabic, and Hebrew fonts.

Double-byte fonts are fonts that are mapped in the area of ASCII 0-65,536 dec. (8-bit communication only). Any glyph (such as characters, punctation marks, symbols, or digits) in the Unicode World Wide Character Standard can be specified. Example of languages that typically require double-byte fonts are Chinese, Japanese, and Korean.

Selecting Fonts

Use the FONT and BARFONT statements to select a font. Use the NASC statement to choose the corresponding character set. You can use these commands with both single- and double-byte fonts.

For illustrations of the available character sets, see **"Character Sets and Keywords" on page 133**.

All fonts stored in the printer memory can be listed to the standard OUT channel by a FONTS statement. This statement does not list font aliases. Another method of listing fonts is to use the FONTNAME$ function. You can also list fonts to the standard OUT channel using the FILES statement.

This example shows how to list all fonts:

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

Controlling Font Direction, Size, Slant, and Width

Fonts can be rotated in four directions using a DIR statement. Use the FONT and BARFONT commands to specify 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.

Adding and Removing Fonts

Use PrintSet, the printer web page, a USB storage device, FTP, or SmartSystems to copy font files to your printer. Font files stored in read/write devices ("/c", "tmp:", and "usb1:") can be deleted using KILL statements.

Note: The names of the font files can differ from the names of the corresponding fonts. Make sure you specify the correct font file name in the KILL statement.

Creating and Using Font Aliases

Font aliasing provides flexibility when moving from a legacy or competitive printer installation to an Intermec printer installation. When you use font aliasing, you do not need to change font names in a datastream, and you can adjust the width, height, and slant of a font. An alias provides a link from the font name in a datastream to a currently installed font. For example, you could create a font alias that points an Arial font to use Univers instead.

To create several font aliases automatically, create a Batch Alias file and copy it to /home/user/fonts/.fontalias.

The format of the file should be:

"alias_1_name", "reference_font", size, slant, width
"alias_2_name", "reference_font", size, slant, width

You can create an unlimited number of font aliases. A font alias can be used like any other font, but its size, slant, and width can not be changed.

Examples:

```
"BODYTEXT","Century Schoolbook",10,80
"HEADLINE","Univers Bold",18,110
"WARNING","Univers",12,10,95
```

About Bar Code Symbologies

Fingerprint supports the following bar code symbologies. When using Fingerprint commands to work with bar codes, use the name for the bar code as shown:

Standard Bar Codes

Bar Code Type	Use This Name
Aztec	"AZTEC"
Codabar	"CODABAR"
Code 11	"CODE11"
Code 16K	"CODE16K"
Code 39	"CODE39"
Code 39 full ASCII	"CODE39A"
Code 39 with checksum	"CODE39C"
Code 49	"CODE49"
Code 93	"CODE93"
Code 128	"CODE128"

Standard Bar Codes (continued)

Bar Code Type	Use This Name
Code 128 subset A	"CODE128A"
Code 128 subset B	"CODE128B"
Code 128 subset C	"CODE128C"
Datamatrix	"DATAMATRIX"
Dotcode	"DOTCODE"
DUN-14/16	"DUN"
EAN-8	"EAN8"
EAN-8 Composite with CC-A or CC-B	"EAN8_CC"
EAN-13	"EAN13"
EAN-13 Composite with CC-A or CC-B	"EAN13_CC"
EAN-128	"EAN128"
EAN-128 subset A	"EAN128A"
EAN-128 subset B	"EAN128B"
EAN-128 subset C	"EAN128C"
EAN.UCC 128 Composite with CC-A or CC-B	"EAN128_CCCAB"
EAN.UCC Composite with CC-CC	"EAN128_CCC"
Five-Character Supplemental Code	"ADDON5"
Gridmatrix	"GRIDMATRIX"
Industrial 2 of 5	"C2OF5IND"
Industrial 2 of 5 with checksum	"C2OF5INDC"
Interleaved 2 of 5	"INT2OF5"
Interleaved 2 of 5 with checksum	"I2OF5C"
Interleaved 2 of 5 A	"I2OF5A"
Matrix 2 of 5	"C2OF5MAT"
MaxiCode	"MAXICODE"
MicroPDF417	"MICROPDF417"
MSI (modified Plessey)	"MSI"
PDF 417	"PDF417"
Planet	"PLANET"
Plessey	"PLESSEY"
Postnet	"POSTNET"
QR Code	"QRCODE"
RSS-14	"RSS14"
RSS-14 Expanded	"RSS14E"
RSS-14 Expanded Stacked	"RSS14ES"
RSS-14 Limited	"RSS14L"
RSS-14 Stacked	"RSS14S"
RSS-14 Stacked Omnidirectional	"RSS14SO"
RSS-14 Truncated	"RSS14T"
Straight 2 of 5	"C2OF5"

Standard Bar Codes (continued)

Bar Code Type	Use This Name
Two-Character Supplemental Code	“ADDON2”
UCC-128 Serial Shipping Container Code	“UCC128”
UPC-5 digits Add-On Code	“SCCADDON”
UPC-A	“UPCA”
UPC-A Composite with CC-A or CC-B	“UPCA_CC”
UPC-D1	“UPCD1”
UPC-D2	“UPCD2”
UPC-D3	“UPCD3”
UPC-D4	“UPCD4”
UPC-D5	“UPCD5”
UPC-E	“UPCE”
UPC-E Composite with CC-A or CC-B	“UPCE_CC”
UPC Shipping Container Code	“UPCSCC”
USPS 4-State	"USPS4CB"

General Rules for Bar Code Printing

The printer contains a number of bar code generators, which can produce highly readable bar codes in four different directions.

Generally, it is more difficult to print a bar code with the bars across the media path (ladder style) than along the media path (picket fence style.) Therefore, to ensure a highly readable printout, Intermec recommends that you use narrow bars at least 3 dots wide when printing ladder-style bar codes.

Print speed also affects the printout quality of bar codes. Generally, a lower print speed gives a better quality, especially for ladder style bar codes and at low ambient temperatures.

Print speed should be only as high as necessary, considering the overall print cycle time. In some instances, a lower print speed may actually give better overall performance.

Intermec recommends that you do your own tests with your unique applications to find the best compromise between printout quality, performance, and media.

For more information on specific bar code parameters and settings, see the ***Fingerprint Command Reference Manual.***

Commands for Working With Bar Codes

Use these Fingerprint commands when working with bar codes:

- BARADJUST - Adjusts position of bar code to avoid faulty printhead dots.
- BARCODENAME$ - Lists available bar code fonts.
- BARFONT - Selects a human-readable font for bar code interpretive printing.
- BARFONT ON|OFF - Enables bar code interpretive printing.

- BARHEIGHT - Bar code height.
- BARMAG - Specifies a magnification for the width of bars in a bar code.
- BARRATIO - Sets the ratio between wide and narrow bars in a bar code.
- BARSET - Specifies a bar code type and sets additional parameters for complex bar codes.
- BARTYPE - Specifies a bar code type.
- PRBAR - Provides input data for a bar code.

For more information, see the ***Fingerprint Command Reference Manual.***

Understanding Images and Image Files

When discussing Fingerprint programming, there is a distinction between images and image files:

- "Image" is a generic term for all kinds of printable pictures, such as symbols or logotypes, in the internal bitmap format of Intermec Fingerprint.
- "Image Files" are files in various bitmap formats that can be converted to Fingerprint "images." Image files can be stored in memory, but must be converted to "images" before printing.

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 the Fingerprint internal bitmap format. This also includes print images saved to the image buffer using the PRBUF statement.

Note: Image files to be used in conjunction with the DISPLAY IMAGE, DISPLAY KEY, or DISPLAY STATE commands are not the same as image files to be used in label layouts as described in this section. For more information, see **"Customizing the Printer Display" on page 112**.

Standard Images

As a standard, the systems part ("Kernel") of the printer 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

Downloading Image Files

Use PrintSet, the printer web page, a USB mass storage device, FTP, or SmartSystems to copy image files to your printer.

Image files in .PCX, .PNG, .GIF, and .BMP format can also be downloaded, automatically converted to images, and installed using the IMAGE LOAD statement. You can save the current print buffer as a file and automatically convert it to an image using IMAGE BUFFER SAVE.

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
```

Note: You must use one-bit image file.

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 involves print images complying with the PRBUF protocol. These are not normal pictures or logotypes, but binary graphics data including printable objects which have been designed in some application program or printer driver in the host. Using the PRBUF statement, these print images can be downloaded directly to the printer image buffer and printed, but cannot be saved in the printer.

Listing Images

The names of all images stored in printer memory can be listed to the standard OUT channel using an IMAGES statement, or to a program using the IMAGENAME$ function.

Image files can be listed to the standard OUT channel using a FILES statement.

This example lists all standard images in the printer memory.

```
IMAGES
```

This results in:

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

```
3568692 bytes free      1717812 bytes used
Ok
```

Removing Images and Image Files

Images can be removed from the read/write devices ("/c", "tmp:", and "usb1:") using REMOVE IMAGE statements.

Image files can be removed from the read/write devices ("/c", "tmp:", and "usb1:") using KILL statements.

6

Designing Bar Code Labels

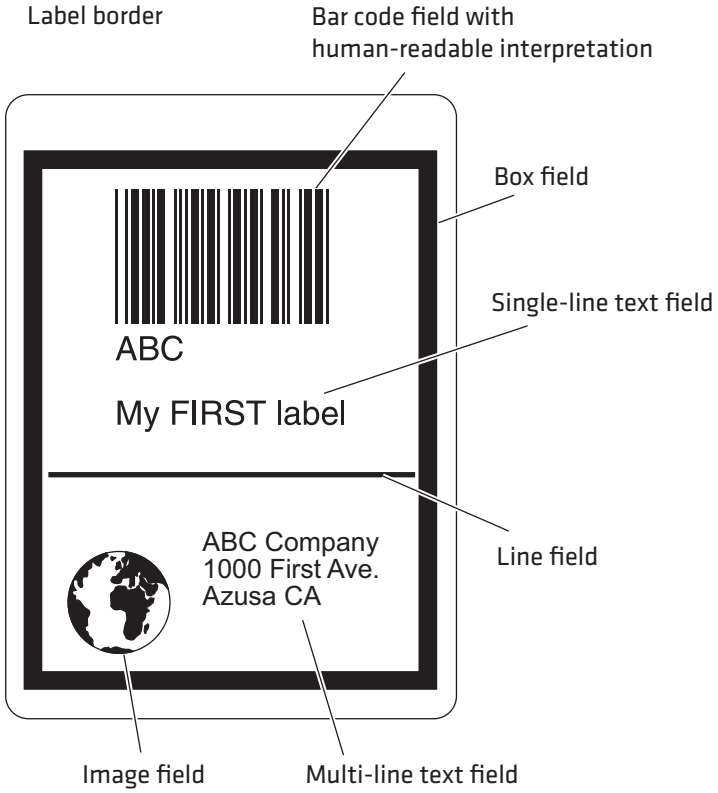
This chapter describes how to design and print a bar code label layout, and includes these topics:

Creating a Layout With Fields

A bar code label layout is made up of a number of fields. There are six different types of fields:

- Single-line text fields. For more information, see **"Creating Single-Line and Multi-Line Text Fields" on page 79**.
- Multi-line text fields. For more information, see **"Creating Single-Line and Multi-Line Text Fields" on page 79**.
- Bar code fields. For more information, see **"Creating Bar Code Fields" on page 81**.
- Image fields. For more information, see **"Creating Image Fields" on page 83**.
- Box fields. For more information, see **"Creating Boxes" on page 85**.
- Line fields. For more information, see **"Creating Lines" on page 85**.

The next illustration shows how these fields appear in a sample label.



Example of Fields in a Label Layout

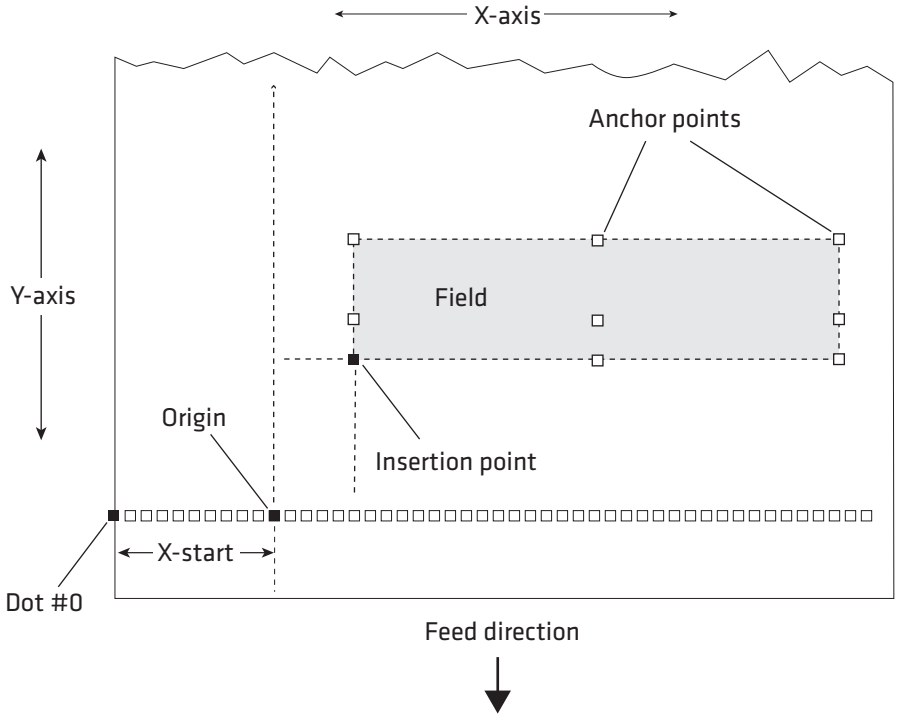
Positioning Fields in the Layout

All fields are positioned relative to the "origin," the point on the media that corresponds to the innermost active dot on the printhead at the time the PRINTFEED statement is executed.

The location of the origin is affected by:

- the printer X-start value. This value can be set by using a SETUP command, or by manually placing the printer in Setup mode and changing the value.
- the current Feed setting on the printer, and any FORMFEED statements executed before the current PRINTFEED statement or after the preceding PRINTFEED statement. This determines where the origin is relative to the front or rear edge of the label.

Starting from the origin, 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.



Field Positioning Settings

About Units of Measure

The unit of measure is always “dots”, which means that all measures depend on the density of the printhead.

For 300 dots/inch printheads, a dot = 0.00333 inches or 3.33 mils.

For 203 dots/inch printheads, a dot is = 0.00492 inches or 4.92 mils.

Because fonts are specified in points instead of dots, all fonts should print the same size regardless of the printhead density.

Dots are the same size along both the X-axis and the Y-axis.

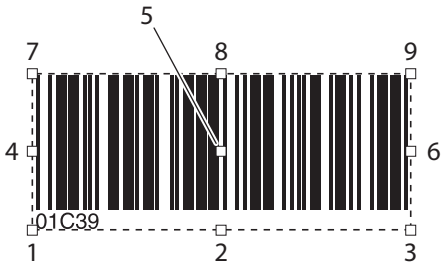
About Insertion and Anchor Points

The insertion point of any field is specified using a PRPOS<x-pos>,<y-pos> statement. For example, the statement PRPOS 100, 200 means that the object is inserted at a position 100 dots to the right of the origin and 200 dots further back along the media path.

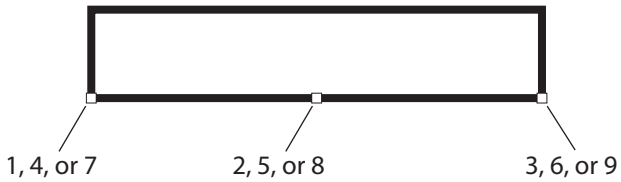
Each field has up to 9 anchor points. Use the ALIGN command to choose the anchor point that is positioned at the insertion point. For example, specifying ALIGN 1 places the lower left corner of a text field at the insertion point..

Note: For detailed information on the anchor points of bar codes where the interpretation is an integrated part of the bar code pattern (such as for EAN and UPC codes), see ALIGN in the ***Fingerprint Command Reference Manual.***

The next illustrations show the anchor point locations for the different fields.



Bar Code Field Anchor Points



Box Field Anchor Points

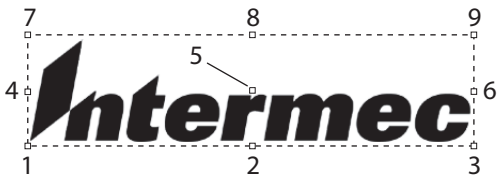
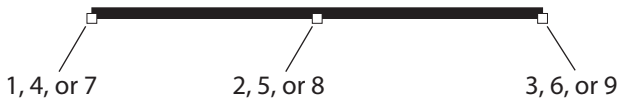
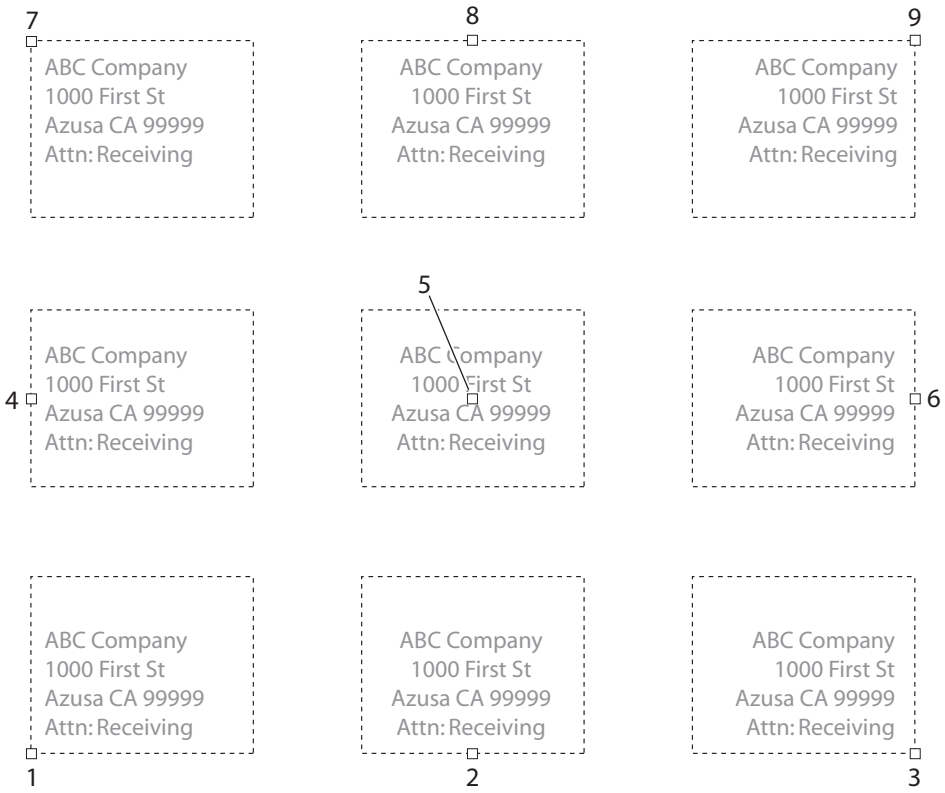


Image Field Anchor Points

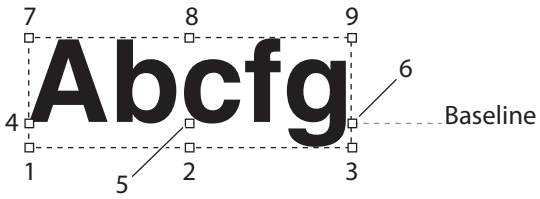


Line Field Anchor Points



Multi-Line Text Field Anchor Points

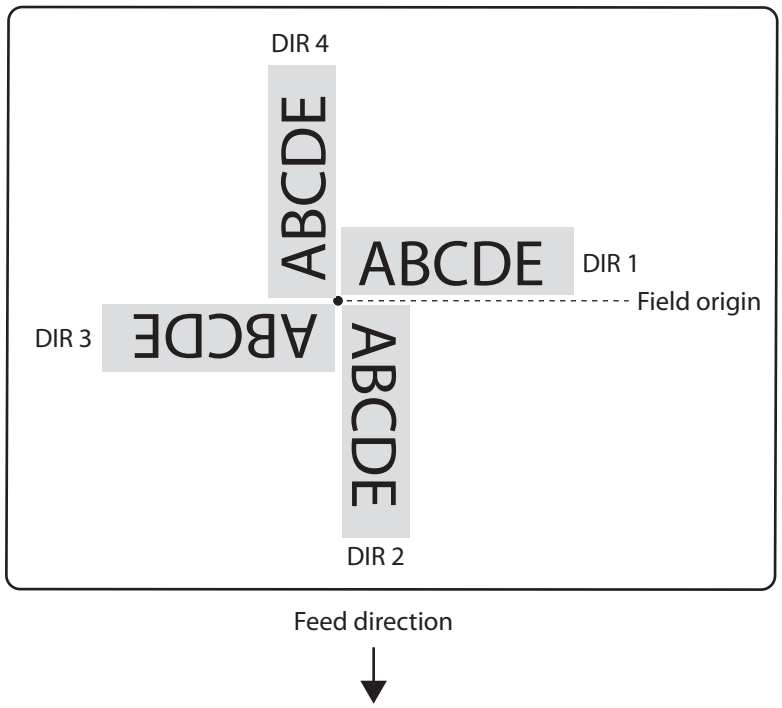
Note: For a multi-line text field, ALIGN sets the anchor points for both the text inside the box and the box surrounding the field. The box can be made visible or invisible.



Single-Line Text Field Anchor Points

About Print Directions

By default, all fields run across the media from left to right. Using a DIR command, you can rotate the field clockwise around the anchor point/insertion point in 90° increments (0°, 90°, 180°, or 270°), as seen in the next illustration.



Print Directions in Fingerprint: *In this example, the DIR command rotates the text field around anchor point 1.*

Checking the Current Position

As you position and specify fields in the label design, you may need to determine the position of the insertion point after a field is printed. Use the PRSTAT function to determine the current position of the insertion point. For example, after creating a single-line text field, you can use PRSTAT to return the exact location of the insertion point. By default, the next new object is placed at the insertion point 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, and 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, and PRSTAT (1) returns the X-value of the last executed PRPOS statement.

In the next example, an unknown number of logo images are printed with 10-dot spacing across the media path. The size of the logo is not known. To avoid an "field out of label" error, the PRSTAT command is used to check the width of the printed fields. If the printed width exceeds 550 dots, the printer continues printing on the next label:

```
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
```

Checking the Size and Position of a Field

When printing a label, Fingerprint instructions are processed into a bitmap pattern that can be sent to the printhead. This process is called rendering.

Use the RENDER OFF command to process instructions without printing anything. By combining RENDER OFF with various PRSTAT variables, you can determine the insertion point location, and thus the size and position of the field, without actually printing the field on a label. Use RENDER ON to restore printing as usual.

Creating Single-Line and Multi-Line Text Fields

A single-line text field consists of one or more alphanumeric characters on the same line. A multi-line text field consists of up to 20 lines of text with up to 300 single-byte characters per line.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a single-line or multi-line text field can contain the following commands:

- FONT - Specifies the font to be used for the field.
- NORIMAGE and INVIMAGE - Sets regular or inverted printing.
- PRTXT - Specifies input data for the text field.
- PRBOX - Specifies the size of the box in which a text field is printed.

Specifying a Typeface with FONT

Specifies the single- or double-byte font to use for text. The default font is Univers at 12-point size, and no slant. The specified font is used for all text until a new FONT statement is executed.

Inverting Black and White Printing with NORIMAGE or INVIMAGE

Normally, text is printed in black on the white background of the print media. Use the INVIMAGE command to print text in white on a black background. The size of the background is decided by the character cell. A NORIMAGE statement is only needed when changing back from INVIMAGE printing.

Specifying Text for Printing with PRTXT

Text for a single- or multi-line text area can be entered in the form of numeric expressions and/or string expressions. Two or more expressions 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.

Defining Borders With PRBOX

Single- or 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 is invisible (thickness = 0) or has 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 anchor point choice affects the positioning of the text inside the box. For information, see **"About Insertion and Anchor Points" on page 76**.

When a text line reaches the border of the box, it wraps to a new line according to the hyphenation settings.

For more information, see the ***Fingerprint Command Reference Manual.***

Summary for Text Fields

To print a single- or multi-line text field, the following information must be specified. If no value is specified, Fingerprint uses the default values.

Required Information for Single-Line Text Fields

Purpose	Command	Default	Remarks
X/Y Position	PRPOS	0/0	Number of dots
Alignment	ALIGN	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Typeface	FONT	Univers,12,0,100	
Style	INVIMAGE	no	White on black print
	NORIMAGE	yes	Black on white print
Text	PRTXT	–	Field input data
Print a label	PRINTFEED	–	Resets parameters to default

Example of a single-line text field:

```
10          PRPOS 100,200
20          ALIGN 7
30          DIR 2
40          FONT "Univers Bold,20,15,80"
50          INVIMAGE
60          PRTXT "HELLO"
```

```
70          PRINTFEED
RUN
```

Example of a multi-line text field:

```
10          DIR 1
20          ALIGN 8
30          R$="Hyphen&Sated words will be divid&Sed
            into sylla&Sbles."
40          NL$="NEWLINE"
50          S$="&S&Special Cases and EXTRAORDINARILY long
            words."
60          T$=R$+NL$+S$
70          PRPOS 300,300
80          PRBOX 700,500,20,T$,25,1,NL$,"&S - +"
90          PRINTFEED
RUN
```

Creating Bar Code Fields

Fingerprint supports many common bar code symbologies, including 2D bar codes and dot codes like PDF417 and MaxiCode. A single bar code, including its optional 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 commands:

- BARSET - Specifies the bar code type and printing.
- BARFONT ON|OFF - Specifies the font to be used to print the bar code.
- PRBAR - Input data for the bar code.

Specifying a Bar Code Symbology With BARSET

This statement specifies the type of bar code and how it is printed. BARSET can also replace separate BARHEIGHT, BARRATIO, BARTYPE, and BARMAG instructions.

BARSET contains optional parameters for specifying complex 2D bar or dot codes such as PDF417. For more information, see the ***Fingerprint Command Reference Manual.***

For common one-dimensional bar codes, include the following parameters in the BARSET statement:

- Bar code type. Name must be enclosed by quotation marks. Default is "INT2OF5".
- Ratio (wide bars). Default is 3.
- Ratio (narrow bars). Default is 1.
- Enlargement. Affects the bar pattern but not the interpretation, unless the bar font is an integrated part of the code (such as for EAN/UPC). Default is 2.
- Height of the bars in dots. Default is 100.

Choosing the Human-Readable Font with BARFONT

Specifies the single-byte font to be used for the human-readable interpretation. In some bar codes, such as EAN/UPC, the interpretation is an integrated part of the code.

You can change these font settings:

- Default font (Univers).
- Size in points. Default is 12.
- Slant in degrees. Default is 0.
- Vertical offset. Specifies the distance in dots between the bottom of the bar pattern and the top of the interpretation characters. Default is 6.
- Height magnification. Default is 1.
- Width magnification. Default is 1.
- Width enlargement in percent. Default is 100.

To disable bar code interpretation printing, use BARFONT OFF.

Specifying Input Data with PRBAR

Depending on the type of bar code, input data for the bar code can be entered in the form of numeric and/or string expressions. String constants must be enclosed by quotation marks. Variables are useful for printing the time, date, or various counters, or for when the same information is to appear in several places, such as in plain text and as bar code input data.

Summary for Bar Code Fields

To print a bar code field, the following information and instructions must be specified. If no value is given, defaults are substituted.

Required Information for Bar Code Fields

Purpose	Command	Default	Remarks
X/Y Position	PRPOS	0/0	Number of dots
Alignment	ALIGN	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	–	Input data to bar code field
Print a label	PRINTFEED	–	Resets parameters to default

This example shows a typical bar code field instruction:

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

Creating Image Fields

An image field is a field containing an image in the internal bitmap format of Fingerprint.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, an image field can contain the following instructions:

- MAG - Magnification value.
- NORIMAGE or INVIMAGE - Specifies inverse printing.
- PRIMAGE - Specifies the image file to use for the field.

Magnifying Images with MAG

Use a MAG statement to specify a magnification for the image. Images can be magnified up to 400% (1 to 4 times). Height and width are specified separately.

Inverting Black and White Printing with NORIMAGE or INVIMAGE

Use an INVIMAGE statement to reverse the black and non-printed background colors. The size of the background is determined by the size of the image. A NORIMAGE statement is only needed when changing back from INVIMAGE printing.

Specifying Images by Filename with PRIMAGE

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

The image must be in the default directory.

Summary for Image Fields

To print an image field, the following instructions must be specified. If no value is specified, default values are substituted.

Required Information for Image Fields

Purpose	Command	Default	Remarks
X/Y Position	PRPOS	0/0	Number of dots
Alignment	ALIGN	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Magnification	MAG	1,1	
Style	INVIMAGE	no	White-on-black
	NORIMAGE	yes	Black-on-white
Image	PRIMAGE	–	.1 or .2 depending on direction
Print a label	PRINTFEED	–	Resets parameters to default

This example shows a typical image field instruction:

```
10          PRPOS 50,50
20          ALIGN 9
30          DIR 3
40          MAG 2,2
50          INVIMAGE
60          PRIMAGE "GLOBE.1"
70          PRINTFEED
RUN
```

Creating Boxes

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 appears 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 PRBOX command, which specifies the height and width of the box, and the line weight (thickness) in dots of the optional border.

Summary for Boxes

To print a box, the following information and instructions must be specified. If no value is specified, default values are substituted.

Required Information for Box Fields

Purpose	Command	Default	Remarks
X/Y Position	PRPOS	0/0	Number of dots
Alignment	ALIGN	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Box specifications	PRBOX	–	Height, width, and line weight in dots
Print a label	PRINTFEED	–	Resets parameters to default

This example shows a typical box field instruction:

```
10          PRPOS 250,250
20          ALIGN 1
30          DIR 3
40          PRBOX 200,200,10
50          PRINTFEED
RUN
```

Creating Lines

You can print a line field in right angles to or across the media path.

In addition to the standard positioning statements PRPOS, ALIGN and DIR, a line field can only contain the PRLINE command, which specifies the length and weight (in dots) of the line. You can also use the PRDIAGONAL command to create a diagonal line.

Summary for Lines

To print a line field, the following information and instructions must be specified. If no value is specified, default values are substituted.

Required Information for Line Fields

Purpose	Command	Default	Remarks
X/Y Position	PRPOS	0/0	Number of dots
Alignment	ALIGN	1	Select ALIGN 1-9
Direction	DIR	1	Select DIR 1-4
Line specifications	PRLINE	–	Length and line weight in dots
Print a label	PRINTFEED	–	Resets parameters to default

Example:

```
10          PRPOS 100,100
20          ALIGN 1
30          DIR 4
40          PRLINE 200,10
50          PRINTFEED
RUN
```

Additional Printing Instructions

Fingerprint includes other commands you can use to further refine your bar code label designs.

Printing Partial Fields With the CLIP ON Command

Ordinarily, bar code labels are designed to fit inside a "print window," the size of which is determined by the combination of the printer X-Start, Width, and Length settings. Any field extending outside the print window causes a "Field out of label" error condition (Error 1003)

Using the CLIP ON command, you can make the program accept fields extending outside the print window. The CLIP ON command prints only the parts of the fields within the borders of the print window.

If you get a "Field out of label" error, you can use CLIP OFF to enable printing of fields that lie outside the printable area. This lets you see how much of the field is missing so you can adjust the layout accordingly.

The clipping of bar codes requires further specification of the CLIP statement. For more information, see the ***Fingerprint Command Reference Manual.***

Use CLIP OFF to return printing to its default state.

Inverting Intersection Printing With XORMODE

By default, the intersection of any crossed lines is printed black. Use XORMODE ON to print those intersections as white, and use XORMODE OFF to restore defaults and print intersections as black.

Using the LAYOUT Command

Many applications require the label layout, variable input data, and logotypes to be sent to the printer as files or arrays. This requires less programming in the printer and less data transfer between printer and host, but some kind of overhead program in the host to handle data input and file transfer is typically of great help.

The program instruction is a statement called LAYOUT. Before using this statement, a number of 52-byte files or arrays must be created:

- A layout file that specifies the type of field to be printed, along with any field-specific information such as position, direction, or font type. Use a layout file to set up single-line text fields, line and box fields, bar code fields, bar code extended fields, or bar code interpretation fields.
- A logotype file that specifies a graphic field to be printed, along with the name of the image file to be printed in the field. This file is required even if your label does not need or use a graphic field in this way.
- A data file (or data array) that specifies the data to be printed in the fields.
- An error file (or error array) that sets up error handling for the layout.

Each file 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 record type, and may specify direction, position, or some other parameter. Each such instruction corresponds to a Fingerprint instruction (for example, direction corresponds to DIR, alignment to ALIGN, and x- and y-positions to PRPOS). Note that there are only 10 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.

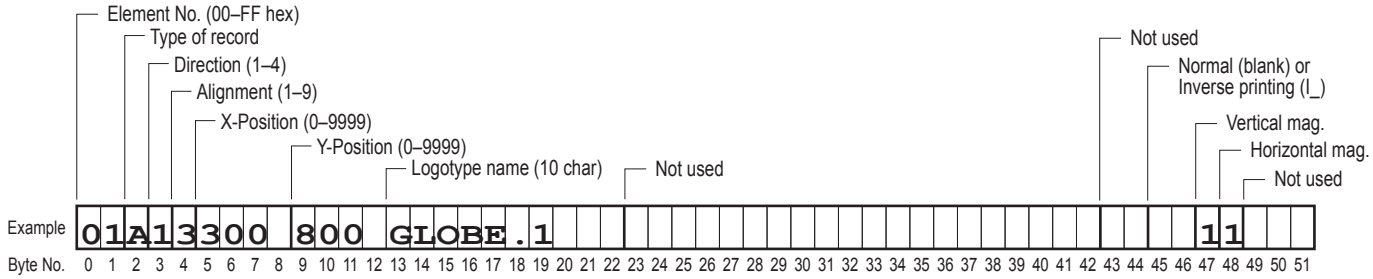
The LAYOUT statement does not support multi-line text fields.

About Layout Requirements

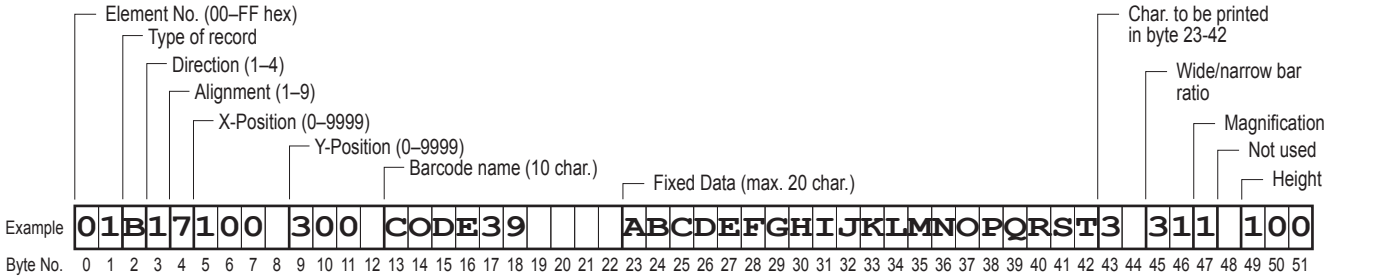
You must follow these rules when you create a layout:

- Each record must be exactly 52 bytes long, and the last character must be 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 inverted 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.

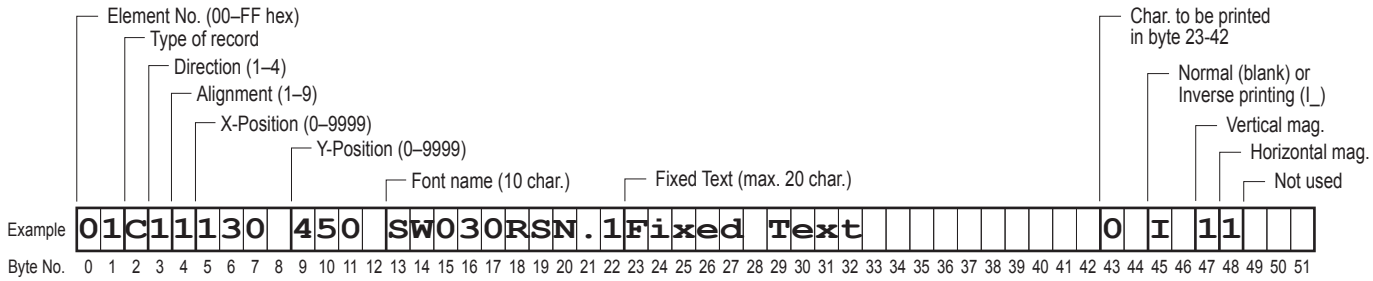
LOGOTYPE RECORD (by name):



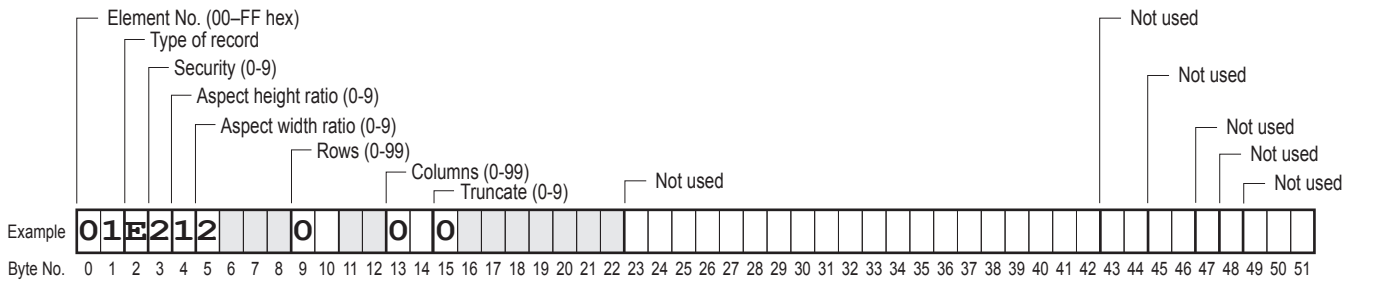
BAR CODE RECORD:



TEXT RECORD:

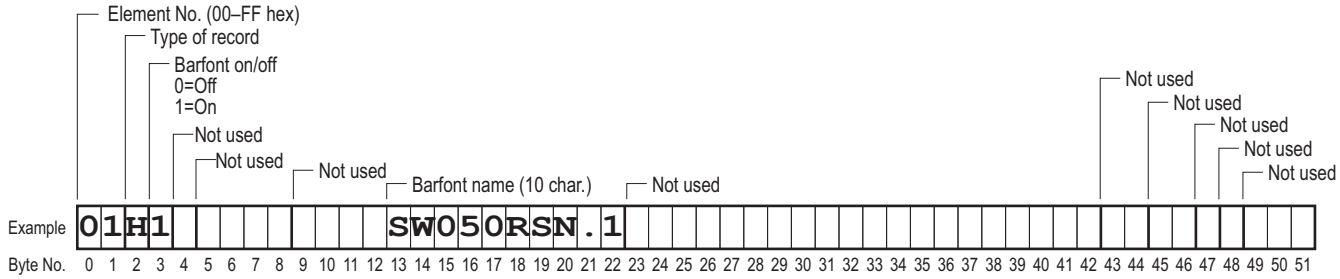


BAR CODE EXTENDED FIELD RECORD:

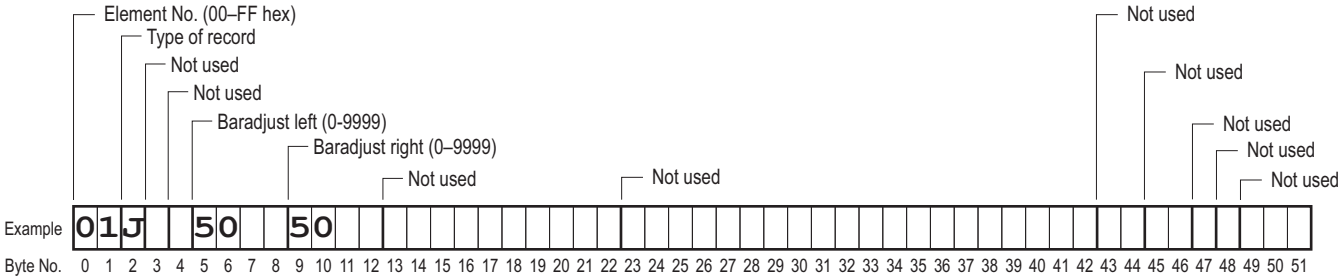


Syntax of Layout File Records (1)

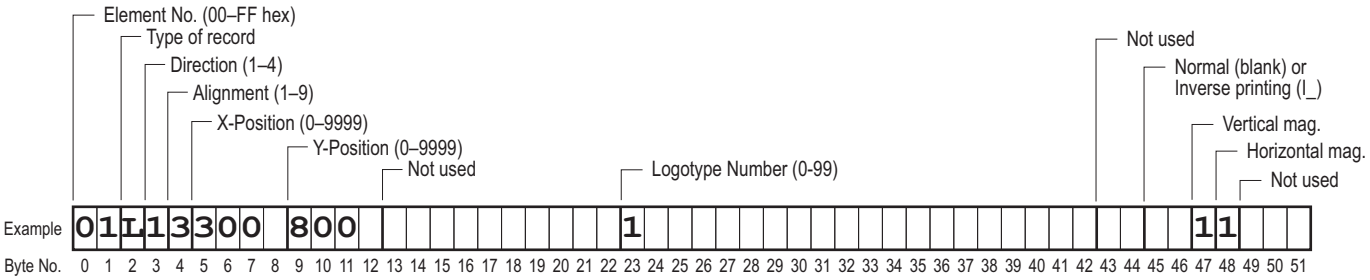
BAR CODE INTERPRETATION RECORD:



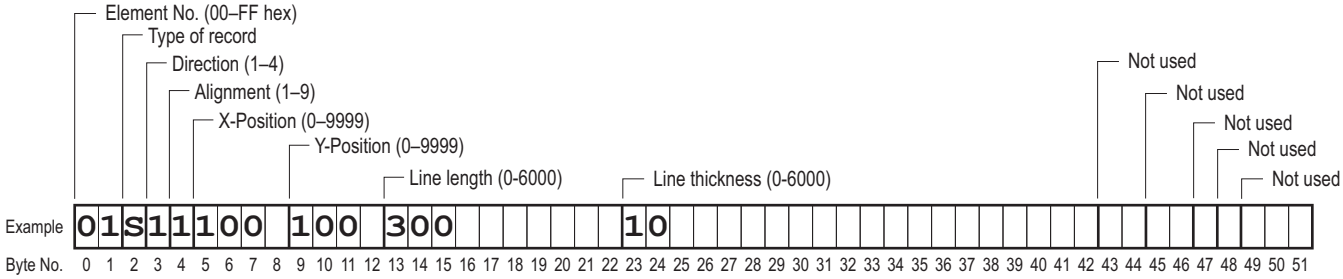
BARADJUST RECORD:



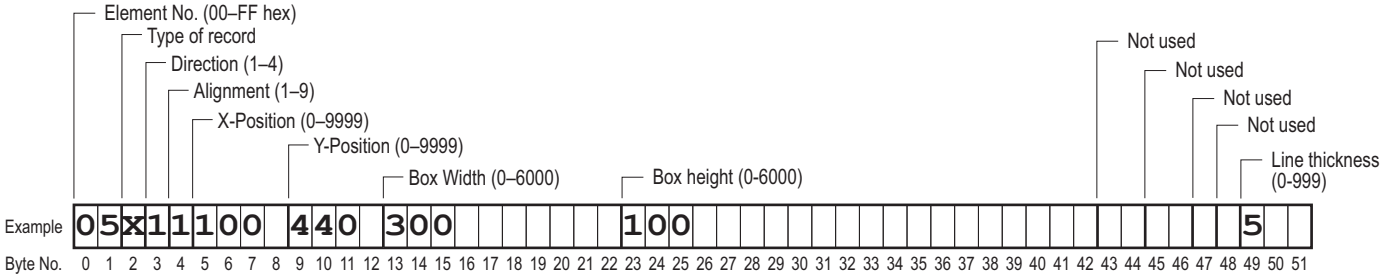
LOGOTYPE RECORD (by number):



LINE RECORD:



BOX RECORD:



Syntax of Layout File Records (2)

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                  111I  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
```

Creating a Logotype Name File

The next step is to create a logotype name file. A logotype name file is required even if you are not using a logotype in your layout, in which 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.

Note: The last record in a sequential file must be appended by a semicolon (;).

If you specify logotype-by-name (record type A), the printer memory is 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. "
30          PRINT#1, "GLOBE.2 "
40          PRINT#1, "DIAMONDS.1"
50          PRINT#1, "DIAMONDS.2";
60          CLOSE 1
```

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
```

Creating a Data File or Array

You 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 is placed in the position specified by the layout. Each data record starts with a hexadecimal element number (00-FF hex) that associates 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.

Note: The last record in a sequential file must be appended by a semicolon (;).

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
```

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.

Creating an Error Array

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
```

Creating an Error File

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:

For files:

```
LAYOUT F, <layout file>, <logotype file>,<data file>,<error
file>
```

For 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. For example, you must create an empty logotype file if your design does not use a logotype field.

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     "01H1                FONT1                                                                                ";
    #2,
110 PRINT     "02C11100  650 FONT1       Fixed Text                 11I  22     ";
    #2,
120 PRINT     "02C11130  450 FONT1       Fixed Text                 0    11     ";
    #2,
130 PRINT     "03B17100  300 CODE39      ABC                        3  311  100";
    #2,
140 PRINT     "04A12300  800 GLOBE.1                                      11     ";
    #2,
150 PRINT     "05X11100  440 300         100                              5    ";
    #2,
160 PRINT     "06S11100  100 300         10                                    ";
    #2,
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
```

Creating a Simple Label

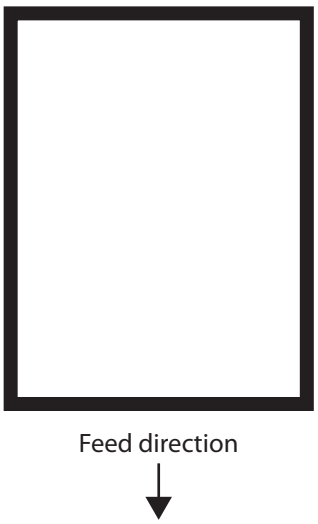
This tutorial walks you through creating a short Fingerprint program that prints a simple label. For more information on Fingerprint commands and syntax, see the *Fingerprint Command Reference Manual.*

To design and print a simple label

1. Connect the printer to a host PC. For help, see **"Sending Fingerprint Commands to the Printer" on page 3**.
2. In HyperTerminal, enter the following text. Press **Enter** at the end of each line:

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

   This code specifies a box 430 dots high and 340 dots wide, with a line thickness of 15 dots, and inserted at position X=10, Y=10.
3. Type `RUN` and press **Enter**. The printer prints this label:

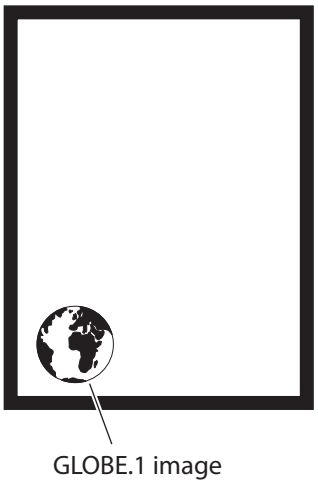


4. Enter the following text:

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

   This code specifies an image field at position X=30, Y=30, using the image named "GLOBE.1" in printer memory.

5 Type `RUN` and press **Enter**. The printer prints this label:



6 Enter the following text:

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

This code specifies a bar code field at location X=75, Y=270, using Code 39, with the data "ABC".

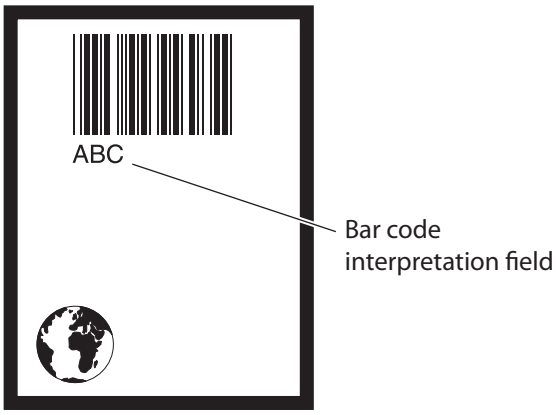
7 Type `RUN` and press **Enter**. The printer prints this label:

8 Enter the following text:

```
1 BARFONT ON
2 BARFONT "Univers", 6
```

This code specifies a bar code interpretation field using 6-point Univers font.

9 Type `RUN` and press **Enter**. The printer prints this label:



10 Enter the following text:

```
80 PRPOS 25,220
90 FONT "Univers", 6
100 PRTXT "My FIRST label"
```

This code specifies a single-line text field at location X=25, Y=220, using 6-point Univers font, with the text "My FIRST Label".

11 Type `RUN` and press **Enter**. The printer prints this label:

12 Type `LIST` and press **Enter**. The program lines are listed in ascending order:

```
Ok
100 prtxt "My FIRST Label"
run

Ok
LIST

1 BARFONT ON
2 BARFONT "Univers",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 "Univers",6
100 PRTXT "My FIRST Label"
200 PRINTFEED
300 END

Ok
_
```

13 To change any program line, you can rewrite the line using the same line number.

Enter the following text, which repositions the text field so the left side of the field is aligned with the left side of the bar code field:

```
80 PRPOS 75,220
```

14 Type `RUN` and press **Enter**. The printer prints this label:

15 (Optional) To save your program, enter the following text and then press **Enter**:

```
SAVE "LABEL1"
```

Your program is saved in the printer memory with the filename "LABEL1.PRG".

Handling Errors With ERRHAND.PRG

The program you just created is unlikely to have caused any errors. However, when writing more complex programs you may find use for an error handler. For this purpose, Intermec includes ERRHAND.PRG in the firmware.

Note: To use ERRHAND.PRG you must merge it with your program. For information, see "Merging Programs" in this section.

ERRHAND.PRG contains subroutines that displays the type of error on the printer 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.

Renumbering Lines When Merging Files

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 "Univers",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 "Univers",6
160         PRTXT "My FIRST label"
170         PRINTFEED
180         END
Ok
```

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"
```

Using the Print Key

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

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

Try pressing different keys on the printer keyboard. Only keys that have been assigned a function in ERRHAND.PRG (**Pause**, **Print**, **Setup**, and **Feed**) will work.

You can break the program by simultaneously pressing the **Shift** and **Pause** keys. Save the program again under the same name:

```
SAVE "LABEL1"
```

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

ERRHAND.PRG can easily be modified to fit into more complex programs and Intermec recommends that you include it in your programs until you are skilled enough to create your own routines for error handling. For more information, see **"Error Handling" on page 123**.

7

Controlling the Printer

This chapter describes how to control various functions in the printer and includes these topics:

Using Fingerprint to Control the Printer

Fingerprint includes many commands you can use to control printer operation, including handling media, settings for individual print jobs, conditions after printing, and setting the date and time for the printer real-time clock.

For specific printer information, or for details on media types and compatibility with your printer, see the printer user's guide.

Controlling Media Feed

Use these Fingerprint commands to control the media feed without printing any labels:

- CLEANFEED - Runs the printer media feed mechanism in order to facilitate cleaning of the platen roller.
- TESTFEED - Adjusts the label stop/black mark sensor while feeding out a number of blank forms or a specified lengh of media.
- FORMFEED - Feeds out a blank label (or similar) or optionally feeds out or pulls back a specified amount of media without printing.
- LBLCOND - Overrides the media feed setup.
- ACTLEN - Returns the length of the most reccently executed TESTFEED, FORMFEED, or PRINTFEED statement.

When a FORMFEED, TESTFEED, or PRINTFEED statement is executed and the media is fed out, the label stop sensor (LSS) on the printer detects the front edge of each new label, the rear edge of each detection gap, or the front edge of each black mark.

The printer Start Adjust and Stop Adjust settings determine how much of the media is fed out or pulled back before and after a FORMFEED, TESTFEED, or PRINTFEED statement is executed. For more information, see the printer user's guide.

Adjusting Media Feed Distance With TESTFEED

After loading a new supply of media into the printer, send a TESTFEED command to adjust the media feed distance. The LSS determines label length by measuring the distance between the forward edges of two consecutive labels, and adjusts feed distance accordingly. The same principle applies to tickets or tags with detection gaps and tickets with black marks.

There are several ways to let the program control media feed without changing the setup.

Feeding Media With FORMFEED

A FORMFEED statement causes the printer to feed out a complete blank label. You can also specify a positive or negative distance (measured in printhead dots) to move the media.

However, if there is an error condition such as "printhead lifted" or "out of media," FORMFEED has no effect.

Although you can use a FORMFEED statement to feed the media in small increments during program execution, for best results Intermec recommends that you make adjustments to the printer Start Adjust and Stop Adjust values instead. You can also use a LBLCOND statement to override the printer Start Adjust and Stop Adjust values when necessary, as described in the next section.

Overriding Start and Stop Adjust Values With LBLCOND

Use a LBLCOND statement to override the current Start Adjust and Stop Adjust values. LBLCOND can also disable the LSS or black mark sensor (BMS) for a specified length of media feed, such as when using irregularly shape labels, or when text or pictures on the backside of a ticket are being detected as black marks.

LBLCOND also allows you to choose between three modes for controlling the printing of very short labels. For information, see the ***Intermec Fingerprint Command Reference Manual***.

Rotating the Platen Roller With CLEANFEED

CLEANFEED rotates the platen roller forward or backward as specified. CLEANFEED is equivalent to FORMFEED, but will work regardless of any error conditions such as "printhead lifted" or "out of media."

Note: Intermec recommends that you use a CLEANFEED statement to remove labels that may be stuck on the platen roller. Manually pulling stuck media may damage the print mechanism.

Checking Media Feed Distance With ACTLEN

This function returns the approximate distance (in dots) of most recently executed media feed operation. For example, use the ACTLEN command to determine the length of the labels before printing a list, so the list can be divided into portions that fit the labels:

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

Controlling Printing

This section describes Fingerprint commands used in connection with printing bar code labels.

Enabling the Automatic Paper Cutter With CUT ON

A CUT statement activates the optional paper cutter independently from a PRINTFEED statement. The relation between the media and the cutting edge when a CUT statement is executed determines where the media is cut off. Because the distance from the printhead to the cutting edge is longer than the printhead-to-tear-bar distance, media feed may need to be adjusted by changing the Start Adjust and Stop Adjust values on the printer.

A CUT ON or CUT OFF statement enables and disables automatic cut-off that has been initiated by a PRINTFEED statement, and can also specify a distance to feed the media before and after cutting.

Enabling the Label Taken Sensor With LTS& ON

This statements enables or disables the optional label-taken sensor (LTS). When the LTS is enabled, the print job stops when a label is detected in the printer outfeed slot, and resumes printing after the label has been removed.

Repeating the Last Printing Operation With PRINTFEED

At execution of a PRINTFEED statement, the printer runs the last program that included printing instructions such as field content or positioning, and prints one copy. You can specify more than one copy, or the number of copies to be reprinted after an interruption. The printer does not adjust its media feed before printing.

After a PRINTFEED statement, these settings are reset to default values:

Settings after a Printfeed

Setting	Default
ALIGN	1
BARFONT	“Univers”, 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	“Univers”, 12, 0, 100
INVIMAGE	NORIMAGE
MAG	1, 1
PRPOS	0, 0

This affects new statements executed after the PRINTFEED statement, but not statements already executed.

In this example, five identical labels are printed:

```
10          PRPOS 100,100
20          FONT "Univers Bold",14,10,80
30          PRTXT "TEST LABEL"
40          PRINTFEED 5
RUN
```

This example prints five copies of the same label, numbered consecutively:

```
10          FOR A%=1 TO 5
20          PRPOS 100, 100
30          FONT "Univers Bold",14,10,80
40          PRTXT "LABEL ";A%
50          PRINTFEED
60          NEXT A%
RUN
```

Enabling Manual Printing With PRINT KEY ON

A PRINT KEY ON statement enables a single PRINTFEED operation when the **Print** key on the printer keypad is pressed manually. The default is PRINT KEY OFF. These commands can only be issued in the Immediate Mode and in the Intermec Direct Protocol.

Checking the Transfer Ribbon and Printhead With SYSVAR

A number of parameters in the system variable SYSVAR can be used to check the transfer ribbon or printhead:

- 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.
- 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.

For more information, see **"Using the SYSVAR System Variable" on page 116**.

Handling Faulty Dots With HEAD, SET FAULTY DOT, and BARADJUST

A faulty dot is a dot on the printhead that does not print properly, which can result in white lines across a printed bar code label. The HEAD function identifies possible faulty dots.

Note: Some printhead faults, such as cracked or dirty dots, will not be detected by this function because only the resistance is measured.

Use SET FAULTY DOT to mark specified dots on the printhead as faulty. You can also revoke all previous SET FAULTY DOT statements by marking all dots as correct.

BARADJUST enables the firmware to track all faulty dots, and relocates the bar code so the spaces between the bars are lined up with the faulty dots. This way, you can print bar code labels immediately without loss of quality, although the printhead should still be replaced.

Note: The BARADJUST statement cannot be used for ladder bar codes, stacked bar codes such as Code 16K, bar codes with horizontal lines such as DUN-14, EAN/UPC bar codes, or two-dimensional bar codes such as PDF417.

The next example lists a program 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
```

Checking Printhead Status With FUNCTEST or FUNCTEST$

The FUNCTEST statement checks the number of dots in the printhead and whether or not the printhead is lifted, and places the result in a string variable.

The next example shows how to use FUNCTEST on a PF4i:

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

The printer returns (for example):

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

The FUNCTEST$ function is similar to the FUNCTEST statement and is used for the same purpose. The next example shows how to use FUNCTEST$ on a PF4i:

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

The printer returns (for example):

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

Reprinting Labels After Interruptions

If an error occurs during batch printing, or if printing is otherwise interrupted, there are two ways to reprint lost or only partially printed labels without losing variable data, such as counter values.

Instead of specifying the number of copies in a batch in a PRINTFEED statement, you can specify how many copies of the last printed label in a batch should be reprinted. The syntax is:

```
PRINTFEED -1,<number of copies to reprint>
```

For example, if a 100-label batch print job is interrupted by an out-of-ribbon condition during the printing of label #70, and you 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 most recently printed label.

The PRSTAT function can detect printing progress 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.

About Batch Printing

Batch printing is the printing of many labels without stopping the media feed motor between the labels. The labels may be exact copies, or the labels may differ more or less in appearance.

When a PRINTFEED is executed, the printer renders the program instructions into a bitmap pattern and stores the pattern in one of the two image buffers. The buffer compensates for differences between processing time and printing time.

As the printer prints the label, it empties the image buffer. High print speeds empty the image buffer more quickly. After the buffer is emptied, the printer processes the next bitmap pattern and stores it in the second image buffer.

These Fingerprint commands can 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|OFF

If there are only small differences between the labels, write your program to use the CLL and FIELDNO instructions to manage the buffers first, and process variable data last.

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. For more information on adjusting the print speed, see the printer user's guide.

Clearing the Print Buffer With CLL and FIELDNO

The image buffer stores the bitmap pattern of the label layout between processing and printing. The image buffer can be cleared completely by a CLL statement, or partially by using a CLL statement and the FIELDNO function:

- Complete clearing is obtained by a CLL statement without any reference to a field, 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 is cleared from the specified field to the end of the program.

In this example, the text "Month" is kept in the image buffer, and the names of the months are cleared from the image buffer as soon as they are printed:

```
10          FONT "Univers Bold",18
20          PRPOS 100,300
30          PRTXT "MONTH:"
40          PRPOS 100,200
50          A%=FIELDNO
60          PRTXT "JANUARY":PRINTFEED
70          CLL A%
80          FONT "Univers Bold",18
90          PRPOS 100,200
100         PRTXT "FEBRUARY":PRINTFEED
110         CLL A%
120         FONT "Univers Bold",18
130         PRPOS 100,200
140         PRTXT "MARCH":PRINTFEED
150         CLL A%
RUN
```

Maintaining Print Speed With OPTIMIZE "BATCH" ON

Normally, after the first image buffer is emptied and printing is completed, the printer processes the next bitmap pattern and stores it in the second image buffer. Use an OPTIMIZE "BATCH" ON statement to enable processing and storage of the next label image while the first label is still being printed. Thus, by switching between the two image buffers, the printer can maintain a high print speed.

The default setting is OPTIMIZE "BATCH" OFF. However, OPTIMIZE "BATCH" ON is automatically invoked if:

- a value >1 is entered for the PRINTFEED statement.**"Using Conditional Instructions" on page 16**.
- the label taken sensor is disabled (LTS& OFF). This is the default.
- the paper cutter is disabled (CUT OFF). This is the 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 **Print** or **Pause** on the printer keypad. Printing can be resumed by pressing either of those keys again.

To prevent unauthorized use, each of these keys 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 or disabled. For more information, see **"Enabling Manual Printing With PRINT KEY ON" on page 105**.

Using the Printer Keypad

If your printer has a keypad, you can use it to:

- Control the printer in Setup Mode and Immediate Mode. In Setup Mode, keys can enter input data if "console:" is OPENed for INPUT.
- Enter input data in the form of ASCII characters.
- Make the program branch to subroutines according to ON KEY...GOSUB statements.

To prevent unauthorized or accidental use, keys can be mapped to unneeded or unnecessary ASCII values using MAP or KEYBMAP$ commands.

This section describes the Fingerprint commands you use to manage keypad input. For more information, see the ***Fingerprint Command Reference Manual.***

Branching to Subroutines With KEY...ON and ON KEY...GOSUB

To make the program branch to a subroutine when a specific key is pressed, first you need to enable the key using KEY...ON. Then you use ON KEY...GOSUB to specify the subroutine to be branched to. For more information, see

For the KEY...ON command, keys are specified by identification (id.) numbers. Each key has two id. numbers, one for its unshifted position and another for its shifted position. 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.

If a key is remapped, its id. number follows the key to its new position.

In the following example, 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
```

Defining Audio Beeps With KEY BEEP

Each time a key is pressed, the printer beeps (a 1200 Hz tone 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 turns off the beep for all keys.

Entering ASCII Characters With INPUT#, INPUT$, or LINE INPUT#

Provided "console:" is OPENed for sequential INPUT, the keys can be used to enter ASCII characters to the program. For more information and an example, see **"Input From a Random File" on page 48**.

Remapping the Keypad With KEYBMAP$

Printer keypads are fully remappable (exception for the **Shift** key), using the KEYBMAP$ command. Each key can produce two ASCII characters (shifted and unshifted). Mapping also decides the ID numbers for the keys.

The basis of the remapping process is the position number of each key. For more information, see **"Character Sets and Keywords" on page 133**.

Note: KEYBMAP$ instructions do not affect the printer in Setup Mode.

The current keyboard mapping can be read to a string variable using the KEYBMAP$ command. This example reads the unshifted characters on the keyboard of a 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, with the following syntax:

```
KEYBMAP$(n) = <string>
```

where:

n = 0 maps the unshifted characters in ascending position number order.

n = 1 maps the shifted characters in ascending position number order.

Note: Position numbers and ID numbers are not the same thing.

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.

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

Non-existing key positions are mapped as NUL (CHR$(0)).

Using the Keypad in Immediate Mode

When a printer has been placed in Immediate Mode (by sending an IMMEDIATE ON statement), these keys on the printer keypad work as follows:

- The **Print** key or button produces a FORMFEED operation. If the printhead is lifted, pressing **Print** produces a CLEANFEED operation, which runs the platen roller several times to facilitate cleaning. During batch printing, pressing **Print** interrupts or resumes printing.

Note: In Immediate Mode, **Print** can also start a print job if the key has been enabled using a PRINT KEY ON statement. For more information, see **"Enabling Manual Printing With PRINT KEY ON" on page 105**.

- **Pause** interrupts or resumes batch printing.
- **Feed** works the same way as the **Print** key.
- Pressing **Shift** + **Feed** at the same time produces a TESTFEED operation.
- **Setup** places the printer in Setup Mode.
- Pressing **i** displays information on the communication channels. The left and right arrow keys toggle between communcations channels when channel information is displayed.

Using the Printer Display

If your printer has an LCD screen, you can use Fingerprint commands to control the printer screen. These commands have no effect if you are using an icon printer.

When the printer is placed in Setup mode, the character set is automatically switched to US-ASCII. When the printer exits Setup mode, the previous mapping is restored.

Customizing the Printer Display

Several different images can appear on your printer display to notify you of an event, alert you to errors, or give feedback when you press a key. You can use these default images in your programs, or add custom images. You can also override some of the default images to show custom images instead.

- Use a DISPLAY STATE statement to show an icon in the notification area.
- Use a DISPLAY KEY statement to show an image when a key is pressed.
- Use a DISPLAY IMAGE statement to show an image when an error occurs.

Each of these commands has predefined values that are associated with default images stored on the printer. For more information on how to use custom images with these commands, see the ***Fingerprint Command Reference Manual.***

Controlling the LEDs and Beeper

If your printer has status LEDs instead of a display, you can control these LEDs using a Fingerprint program.

Using an LED ON|OFF|BLINK Statement

Use the LED ON|OFF|BLINK statement to control the LEDs. This statement can turn the Ready or Error LEDs on and off, or it can blink the LEDs (at 0.4 sec intervals) with or without incoming data.

In this example, the Ready LED (0) is lit until an error occurs, at which time the Error LED (1) is lit and 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 "Univers Bold",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
```

Using a BEEP or SOUND Statement

In addition to the visual signals from the display and LEDs, you can use a BEEP or SOUND statement for audible notification if your printer has a beeper.

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. The SOUND statement even allows you to compose your own melodies.

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 "Univers", 36
40          PRTXT "OK!"
50          PRINTFEED : BEEP
60          END
1000        SOUND 880,25 : SOUND 988,25 : SOUND 30000,10
1010        RESUME
RUN
```

Setting the Date and Time

Some Intermec printers are equipped with a real-time clock (RTC) with battery backup. If an RTC is installed, the internal clock is updated from the RTC at each startup.

If no RTC is installed, you need to manually set the clock using either a DATE$ or a TIME$ variable, or an error occurs when trying to read the date or time. 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, 081232 is corrected to 090101).

In addition to the standard formats (YYMMDD and HHMMSS), other formats for date and time can be specified by these Fingerprint commands:

- FORMAT DATE$
- FORMAT TIME$
- NAME DATE$
- NAME WEEKDAY$

Reading the Clock and Calendar

These Fingerprint commands are used to read the clock and calendar:

- <svar>=DATE$
- <svar>=DATE$("F") - Returns the current date in the format specified by FORMAT DATE$ to a string variable.
- <svar>=TIME$

- <svar>=TIME$("F") - Returns the current time in the format specified by FORMAT TIME$ to a string variable.
- DATEADD$
- TIMEADD$
- DATEDIFF
- TIMEDIFF
- WEEKDAY
- 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
- TICKS

In most cases, you can specify the current date or time using DATE$ or TIME$ respectively, as in this example:

```
WEEKDAY$ (DATE$)
TIMEDIFF (TIME$, "120000")
```

The next 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
```

The printer returns (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
```

Using Setup Mode Programmatically

To change a printer setup parameter as a part of the program execution, you can use a SETUP statement.

A SETUP statement can:

- place the printer in Setup Mode for manual configuration (pressing keys on the printer keypad). When the printer is in Setup Mode, it does not respond to Fingerprint commands sent from a communications application.
- create a copy of the current setup and save it as a file, or return the current setup to a specified communication channel.
- change some or all setup parameters according to a setup file.
- change a single parameter.

Reading the Current Setup

To read the current printer setup, use a SETUP WRITE statement to return the setup to the standard OUT channel, as in this example:

```
SETUP WRITE "uart1:"
```

Creating a Setup File

1. Open a file for sequential output.
2. Use a PRINT# statement to enter each parameter you want to change. See the SETUP command in the Fingerprint command reference manual for specific syntax information.
3. Close the file.

Changing the Setup Using a Setup File

To change the setup based on a setup file, use a SETUP<*filename*> statement. If the setup file is stored in another part of the printer memory than the current directory, the file name should start with a reference to the correct path for the file.

In the following example, save the current setup under a new file name and then create a setup file that changes the size of the transmit buffer on "uart1:". Finally, use the setup file to change the printer 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

To change a single setup parameter without creating a file, use a SETUP statement with a string with the same syntax as the corresponding parameter. Do not use a leading PRINT# statement.

This example changes the "uart1:" settings:

```
SETUP "SER-COM,UART1,TRANS BUF,2000"
```

Saving the Setup

You can decide whether a change in the printer setup should be permanent or temporary using SYSVAR(35):

- If SYSVAR(35) = 0 (default), the setup is saved as a file and remains effective after a reboot or power down.
- If SYSVAR(35)=1, the setup is not saved, and the last saved setup values are effective after a reboot or power down.

For more information, see the next section.

Using the SYSVAR System Variable

Some sensors and other conditions can be read or set using the SYSVAR system variable. You can use SYSVAR to return the current conditions of a variable, when can then affect your running program. For specific information on using SYSVAR, see the ***Fingerprint Command Reference Manual.***

The next table lists SYSVAR values and the information returned.

SYSVAR Values and Descriptions

Value	Description
14	Returns the number of errors since last power on.
15	Returns the number of errors since the previously executed SYSVAR(15) instruction.
16	Returns the number of bytes received at the execution of a STORE INPUT statement.
17	Returns the number of frames received at the execution of a STORE INPUT statement.
18	Returns or sets the verbosity level.
19	Returns or sets the type of error messages transmitted by the printer.
20	Returns 0 if the printer is set up for direct thermal or 1 if set up for thermal transfer printing.
21	Returns the printhead density in dots/mm.

SYSVAR Values and Descriptions (continued)

Value	Description
22	Returns the number of dots in the printhead.
23	Returns 1 if a transfer ribbon is detected, 0 otherwise.
24	Returns 1 if a power-up has been performed since last SYSVAR(24), or 0 otherwise.
26	Returns 1 if the ribbon sensor detects that the diameter of the ribbon supply roll is equal to or less than the diameter specified in Setup Mode, or 0 otherwise.
27	Sets condition for label reprinting at out-of-ribbon error.
28	Decides if the information on the position of the media against the printhead should be cleared or not when the printhead is lifted.
29	Returns Data Send Ready (DSR) condition on “uart2:”.
30	Returns Data Send Ready (DSR) condition on “uart3:”.
31	Returns last control character sent from the MUSE protocol (special applications).
32	Returns the length of media (in 10-meter increments) that have been fed past the printhead.
33	Returns Data Send Ready (DSR) condition on “uart1:”.
34	Sets or returns TrueType® character positioning mode.
35	Sets or returns permanent or volatile setup saving.
36	Sets or returns whether changes of program mode should be printed to the Debug Std Out port in connection with debugging.
37	Sets or returns minimum gap length.
39	Enables or disables slack compensation.
41	Sets or returns conditions for overriding error detection at predefined feed length.
42	Sets or returns conditions for aligning the gaps in the media with the tear bar.
43	Sets or returns file name conversion enabled or disabled.
44	Sets or returns current state of filtering of NUL characters during background communication.
45	Returns the printhead resolution in dots per inch.
46	Returns 1 if the paper low sensor detects that the diameter of the media supply roll is equal to or less than the diameter specified in Setup Mode, or 0 otherwise.
47	Sets or returns current state of using Start Adjust and Stop Adjust values together with FORMFEED values.
48	Sets or returns use of bidirectional Direct Protocol.
49	Temporarily sets a lower print speed after a negative Start Adjust value.
50	Sets a lower print speed after lowering the printhead.
51	Sets the enabled limit for SYSVAR 49 and 50.
53	Sets or returns the highest allowed diamter (in mm) of the ribbon supply.
54	Modifies the DNS timeout value. Default is 5 (150 seconds). Each increment or decrement equals ±30 sec.

Checking Hardware and Firmware Versions

The VERSION$ function returns one of three characteristics of the printer:

- VERSION$(0) returns the firmware version (for example, “P10.03.006424”).
- VERSION$(1) returns the printer family (for example, “PM43”).
- VERSION$(2) returns the CPU board (for example, “Platform version 1.0”).

VERSION$ allows you to create programs that 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.

The next 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
...
1000        SETUP "SETUP_PF2i.SYS"
1010        GOTO 50
2000        SETUP "SETUP_PF4i.SYS"
2010        GOTO 50
3000        SETUP "SETUP_PM4i.SYS"
3010        GOTO 50
```

Checking Immediate Mode and STDIO Status

Use the IMMEDIATE statement to check the current Immediate Mode status or the status of the standard IN and OUT channels.

IMMEDIATE MODE prints a line to the standard OUT port that shows the status (on or off) of the following modes:

- Execution - On indicates that a Fingerprint application is running.
- Immediate - On indicates the printer is in Immediate Mode.
- Input - On indicates that Direct Protocol is enabled.
- Layout input - On indicates that a layout is being recorded in Direct Protocol.
- Debug STDIO (DBSTDIO) - On indicates that the debug standard IO is active.

IMMEDIATE STDIO prints two lines to the standard OUT port with information on the current communication settings for the STDIN and STDOUT channels.

Restarting the Printer

Restarting the printer has the same consequences as switching the power off and then on. Use a REBOOT statement to restart the printer as part of the program execution. When the printer is restarted, a number of things happen:

- The printer temporary memory ("tmp:") is erased, which means that programs not saved to "/c" or "usb1:" are lost, all buffers are emptied, all files are closed, all date- and time-related formats are lost, all arrays are lost, and all variables are set to zero. Fonts and images stored in the temporary memory are erased.
- All parameters in the Fingerprint instructions are reset to default.
- The printer performs a number of self-diagnostic tests, such as printhead resistance check and memory checksum calculations.
- The printer checks for possible optional devices such as an interface board or cutter.
- The printer memory is searched for possible startup programs. The first startup program found is executed.
- The printer internal clock is reset to default, or updated from the real-time clock if one is installed.

Restarting does not affect the printer setup, unless the printer hardware configuration has changed during the power-off period (for example, if the printhead has been replaced or an interface board has been installed or removed).

About Printer Memory

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 printer memory consists of a number of parts, some with directories.

Permanent Memory

The permanent memory, "c:" or "/c" (also called /ram or "ram:" in some printer manuals) resides in a flash memory SIMM. Additional flash SIMMs are also included in the device "/c".

You can also use a USB storage device ("usb1:") as permanent memory.

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 "/c" file system uses 1K blocks. Files smaller than 1K use 1K of space. File space always rounds up, so a 4.5K file uses 5K of file space. 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 is automatically reorganized to save space. This process takes some time and can make the flash memory comparatively slow.

Temporary Memory

Temporary memory has no battery backup and is completely erased at power-off. However, the following Fingerprint commands can be used to prevent 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 printer.
- 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 does not need to be saved after power-off.
- To temporarily store data before it is 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, avoid using the permanent memory to save data that will be of no use after power off.

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, the remaining memory will be shared between the various tasks.

Other Memory Devices

The “storage:” device is a memory device that is used for special applications and should not be used for normal Fingerprint programming.

Changing the Current Directory

"Current directory" is the directory which Fingerprint uses unless you specify another directory. By default, the current directory is set to "/c".

Use a CHDIR statement to change the current directory. To return the current directory, use the CURDIR$ function.

The next example shows how to change the current directory from the default ("/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 Immediate Mode.

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

Example:

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

Results in (for example)

```
2382384
```

Providing More Free Memory

In order to free up memory space in temporary memory, you can use a CLEAR statement to empty all strings, set all variables to zero, and reset all arrays to their default values. 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 images stored in "/c" and or "tmp:". If the printer is not equipped with the maximum amount of memory, you have the option to install additional or larger Flash or SDRAM SIMM packages.

Note: Make backup copies on the host before you replace memory units or install additional memory in the printer.

Formatting the Permanent Memory

The printer permanent memory ("/c") can be formatted either partially or completely by using the FORMAT command as follows:

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

The A parameter indicates that you want to erase all files in the device (hard formatting). However, this does not erase your printer configuration.

```
FORMAT "/c"
```

erases all files, except those starting with a period (.) character (soft formatting). Many Fingerprint system filenames begin with a period character.

Using the Industrial Interface

The optional Serial/Industrial Interface Board includes 8 digital IN ports, 8 digital OUT ports, and 4 OUT ports with relays.

When used with the optional Serial/Industrial Interface Board, Fingerprint can control external equipment such as conveyor belts, gates, turnstiles, and control lamps, in addition to the printer. Conversely, the status of various external devices can be used to control both the printer and other equipment. Thus, a Fingerprint program can independently control workstations without an online connection to a host computer.

There are several Fingerprint commands used in connection with the Serial/ Industrial Interface Board:

- PORTOUT ON/OFF: Sets one of the OUT ports (digital or relay) to either on or off.
- PORTOUT.DATAREADY ON: Asserts the Dataready signal, which can be used to control a print applicator.
- PORTIN: This function returns the status of a specified port, or checks the current state of a specified input or output signal.

8

Error Handling

This chapter describes how Fingerprint handles errors and includes these topics:

Standard Error Handling

In most application programs, it is useful to include some kind of error handler. The complexity of the error handler depends on the application and how independently the printer works. For flexibility, Intermec Fingerprint includes a number of tools for designing custom error handling routines.

Fingerprint includes hardware-based error handling, such as reporting "out of media" errors when the **Print** or **Feed** keys are pressed during a print job, or if batch printing is interrupted by pressing the **Print** or **Pause** keys.

Additionally, during program execution, Fingerprint performs these checks:

- Syntax Check. Each program line or instruction received on the standard IN channel is checked for possible syntax errors before it is accepted. If two-way communication is established, error messages (for example, "Feature not implemented" or "Font not found") are transmitted to the host on the standard OUT channel.
- Execution Check. Any program or hardware error that stops the execution is reported on the standard OUT channel. In case of program errors, the number of the line where the error occurred is also 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 the program includes a routine for dealing with the error condition.

Note: For two-way communications, these conditions must be fulfilled:

- Serial communications are established with the host
- Std IN channel = Std OUT channel
- Verbosity is enabled

Choosing an Error Message Format

Using the system variable SYSVAR(19), you can choose between four types of error messages. This is illustrated by the following examples using error #19:

- "Invalid font in line 10" (default)
- "Error 19 in line 10: Invalid font"
- "E19"
- "Error 19 in line 10"

For more information, see the SYSVAR information in the ***Fingerprint Command Reference Manual.***

Checking for Programming Errors

Use these Fingerprint commands to check for possible programming errors:

- TRON|TROFF - Trace the execution line by line during execution.
- STOP or CONT - Temporarily stop execution, and resume after being stopped.
- DBBREAK - Create a breakpoint.
- DBSTEP - Create a breakpoint after a specified number of lines are executed.

Using a TRON|TROFF Statement

If the program does not work as expected, there may be a programming error that prevents the program from being executed in the intended order. Use a TRON (Trace On) statement to trace the execution. When the program is run, each line number is returned on the standard OUT channel in the order of execution. TROFF (Trace Off) disables TRON.

Using STOP and CONT Statements

Use a STOP statement to temporarily stop execution when examining or changing variables. Program execution can be resumed 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 been edited during the break.

Specifying Breakpoints

To make it easier to debug a program step by step, you can specify breakpoints in the program. Use a DBBREAK statement to create or delete a breakpoint. Alternatively, you can use the DBSTEP statement to specify how many lines should be executed before next break.

At a break, the message "break in line *nnn*" is sent 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 debugger.

Commands for Error-Handling Routines

This section describes Fingerprint commands you use to create error-handling routines.

Branching to Subroutines With ON ERROR GOTO...

Use ON ERROR GOTO... to branch execution to a subroutine if any kind of error occurs when a program is run. The error can be identified and managed, and program execution can be resumed at an appropriate program line. For more information, see **"Instructions for Conditional Branching" on page 18**.

Checking Error Codes with ERR and ERL

ERR returns the reference number of an error that has occurred. For more information, see the Error Codes topic in the ***Fingerprint Command Reference Manual.***

ERL returns the number of the line on which an error has occurred.

Resuming Execution After Errors

This statement resumes execution after the error has been handled in a subroutine. 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. For more information, see **"Instructions for Conditional Branching" on page 18**.

Returning Print Job and Printhead Status with PRSTAT

In addition to returning insertion point and field information, PRSTAT can return print job and printhead status, including multiple error conditions.

Calling PRSTAT without parameters returns a numeric value. If 0 returns, the printer is OK. Any other value indicates a print job or printhead error condition, or some combination of error conditions as shown in the next table.

PRSTAT Error Values

Value	Description
0	Printer is OK
1	Printhead lifted
2	Label not removed (valid if Label Taken Sensor is installed, and returns 0 if printer has no LTS)
4	Label Stop Sensor does not detect a label
8	Printer out of thermal transfer ribbon, or the printer is set for direct thermal printing, a ribbon is installed
16	Printhead voltage too high
32	Printer is feeding
128	Printer out of media

Multiple errors are indicated by the sum of the values. For example, if the printhead is lifted (1), and the printer is out of media (128) and ribbon (8), then PRSTAT returns 137.

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
...
```

For more information, see PRSTAT in the ***Fingerprint Command Reference Manual.***

Error Handling Example

In this example one error condition (Error 1019, "Invalid Font") is managed. The same principles can be used for more errors. 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 "Univers",24 : INVIMAGE
2040      RESUME
3000      PRINT #1 : PRINT #1
3010      PRINT #1, "Undefined error"
3020      PRINT #1, "Program Stops!";
3030      RESUME NEXT
RUN
```

Using the ERRHAND.PRG Utility Program

For simple error handling, Fingerprint includes ERRHAND.PRG. This utility program contains basic routines for handling errors, managing the keyboard and display, and printing. Merge ERRHAND.PRG with your program to use ERRHAND subroutines for error handling.

Note: 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.

The approximate size of ERRHAND.PRG is 4 KB. To use ERRHAND.PRG with more than one application stored in printer memory, you can save valuable memory space by merging ERRHAND.PRG with the current program directly after loading.

To merge ERRHAND.PRG with your program, your code should look like this:

```
NEW
LOAD "MY PROGRAM.PRG"
MERGE "/rom/ERRHAND.PRG"
RUN
```

Modifying ERRHAND Variables and Subroutines

There are two sets of variables in ERRHAND.PRG that you can use or modify:

- NORDIS1$ and NORDIS2$ at line 10 contain the main display texts. You can replace them with your own text.
- DISP1$ and DISP2$ contain the actual text that appears in the printer display on lines 1 and 2 respectively.

The next table lists subroutines you can use or modify.

ERRHAND.PRG Subroutines

At Line	Description
160000	Errors which normally may occur during printing are handled: • 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.
200000	Includes error-handling routines that can be called from routines where errors may occur. See lines 200000 through 200080.
400000	The FEED-routine executes a FORMFEED with error-checking. Called by the statement GOSUB 400000.
500000	The PRINT-routine executes a PRINTFEED with error-checking. Called by the statement GOSUB 500000.
600000	Clears the printer display and makes the display texts stored in the variables DISP1$ and DISP2$ appear on the first and second line in the display. Called by the statement GOSUB 600000.
700000	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.
1500000	**Pause** key (key No. 15) interrupts the program until pressed a second time. Called by the statement GOSUB 1500000.
1700000	Routine for **Print** key (key No. 17) that calls subroutine 500000. Called by the statement GOSUB 1700000.
1800000	Routine for **Setup** key (key No. 18). Places the printer in Setup Mode. Called by the statement GOSUB 1800000.
1900000	Routine for **Feed** key (key No. 19), that calls subroutine 400000. Called by the statement GOSUB 1900000.

For more information, see the next section.

Complete Listing of ERRHAND.PRG

```
10 PROGNO$="Ver. 1.21 2005-11-25"
15 NORDIS1$="FP-APPLICATION" : NORDIS2$= "VERSION 1.21"
20 GOSUB 700000 : 'Initiate
100000 'Error routine
100010 EFLAG%=ERR
100050 'PRINT EFLAG%:'Activate for debug
100060 LASTERROR%=EFLAG%
100200 RESUME NEXT
160000 'PRINT "Last error = ";LASTERROR%:'Activate for debug
160050 'IF LASTERROR%<>0 THEN PRINT "At line ";ERL
160100 LASTERROR%=0
160200 RETURN
200000 'Error handling routine
200010 IF EFLAG%=1006 THEN GOTO 200040 : ' Formfeed instead of print
200020 LED 1 ON : LED 0 OFF : BUSY
200030 SOUND 400,10
200040 IF EFLAG%=1031 THEN GOSUB 300000
200050 IF EFLAG%=1005 THEN GOSUB 310000
200060 IF EFLAG%=1006 THEN GOSUB 320000
200070 IF EFLAG%=1022 THEN GOSUB 330000
200080 IF EFLAG%=1027 THEN GOSUB 340000
200090 DISP1$=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$(ERR)
340030 GOSUB 600000
340040 IF PRSTAT AND 8 THEN GOTO 340040
340050 'GOSUB 1500000
340051 GOSUB 1501000
340200 IF PCOMMAND% THEN GOSUB 500000
349000 RETURN
400000 'Feed routine
400010 EFLAG%=0
400020 FORMFEED
400200 IF EFLAG%<>0 THEN GOSUB 200000
400300 RETURN
500000 'Print routine
500010 EFLAG%=0
500020 PCOMMAND%=1
500030 PRINTFEED
500040 IF EFLAG%<>0 THEN GOSUB 200000
500100 PCOMMAND%=0
500300 RETURN
600000 'Display handler
600010 PRINT #10
600020 PRINT #10
600030 PRINT #10, DISP1$
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
1501000 'PD41 Pause function
1501005 ON KEY 17 GOSUB 1501000 : 'Temporarily hijack the PRINT key.
1501010 KEY 17 ON
```

```
1501020 PAUSE%=PAUSE% XOR 1
1501030 BUSY : LED 0 OFF
1501040 DISP1$="Press <PRINT>" : DISP2$="to continue"
1501050 GOSUB 600000
1501060 IF PAUSE%=0 THEN GOTO 1501100
1501070 SOUND 131,2
1501080 SOUND 30000,20
1501090 IF PAUSE% THEN GOTO 1501070
1501100 READY : LED 0 ON
1501110 DISP1$=NORDIS1$ : DISP2$=NORDIS2$
1501120 GOSUB 600000
1501130 ON KEY 17 GOSUB 1700000
1502000 RETURN
1503000 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 'Feedkey
1900010 KEY 19 OFF
1900020 GOSUB 400000
1900030 KEY 19 ON
1900200 RETURN
```

Extensions to ERRHAND.PRG

The following subroutines can be added manually to stop new input via the printer keyboard while a subroutine is executed.

To enable all keys after completing a subroutine:

```
800000    'Turn all keys on
800010    FOR I% = 0 TO 21
800020    KEY (I%) ON
800030    NEXT I%
800040    RETURN
```

To disable all keys before entering a subroutine:

```
900000    'Turn all keys off
900010    FOR I% = 0 TO 21
900020    KEY (I%) OFF
900030    NEXT I%
900040    RETURN
```

Standard Error Codes

When a problem occurs, your printer may display an error code. For a list of standard error codes and the explanations for the error, see the ***Fingerprint Command Reference Manual.***

A

Character Sets and Keywords

This chapter includes an introduction to character sets and a list of Fingerprint keywords reserved for use by commands.

Introduction to Character Sets

The following information applies to all single-byte character sets:

- Characters between ASCII 00 decimal and ASCII 31 decimal are unprintable control characters as listed below.
- Characters between ASCII 32 decimal and ASCII 127 decimal can always be printed, regardless of 7-bit or 8-bit communication protocol, provided that the selected font contains those characters.
- Characters between ASCII 128 decimal and ASCII 255 decimal can only be printed if the selected font contains those characters and an 8-bit communication protocol is used. If you use 7-bit communication, select another national character set with the NASC command, or use a MAP statement to remap a character set.
- If a character which does not exist in the selected font is used, an error condition occurs.

Non-Printable Control Characters (ASCII 00-31 dec)

ASCII	Character	Meaning	ASCII	Character	Meaning
00	NUL	Null	16	DLE	Data link escape
01	SOH	Start of heading	17	DC1	Device control one
02	STX	Start of text	18	DC2	Device control two
03	ETX	End of text	19	DC3	Device control three
04	EOT	End of transmission	20	DC4	Device control four
05	ENQ	Enquiry	21	NAK	Negative acknowledge
06	ACK	Acknowledge	22	SYN	Synchronous idle
07	BEL	Bell	23	ETB	End of transmission block
08	BS	Backspace	24	CAN	Cancel
09	HT	Horizontal tabulation	25	EM	End of medium
10	LF	Line feed	26	SUB	Substitute
11	VT	Vertical tabulation	27	ESC	Escape
12	FF	Form feed	28	FS	File separator
13	CR	Carriage return	29	GS	Group separator
14	SO	Shift out	30	RS	Record separator
15	SI	Shift in	31	US	Unit separator

For the full list of character sets supported by Fingerprint, see the ***Fingerprint Command Reference Manual.***

About the UTF-8 Character Set

The UTF-8 character set was created to encode all Unicode characters while maintaining compatibility with the US-ASCII (0 to 127 dec.) range of characters. Data is encoded with 1, 2, 3 or 4 bytes, depending on the character number range. The table below shows the UTF-8 binary sequences corresponding to the Unicode character number.

Unicode character number range		**UTF-8 Byte sequence**
Hex	Binary	Binary
0000-007F	$x_7x_6x_5x_4x_3x_2x_1$	One byte: $\mathbf{0}x_7x_6x_5x_4x_3x_2x_1$
0080-07FF	$y_5y_4y_3y_2y_1x_6x_5x_4x_3x_2x_1$	Two bytes: $\mathbf{110}y_5y_4y_3y_2y_1\ \mathbf{10}x_6x_5x_4x_3x_2x_1$
0800-FFFF	$z_4z_3z_2z_1y_6y_5y_4y_3y_2y_1x_6x_5x_4x_3x_2x_1$	Three bytes: $\mathbf{1110}z_4z_3z_2z_1\ \mathbf{10}y_6y_5y_4y_3y_2y_1\mathbf{10}\ x_6x_5x_4x_3x_2x_1$
010000-10FFFF		Four bytes: Not currently supported.

Follow the next procedure to convert a Unicode character code in hex format to the UTF-8 byte decimal value necessary to print the characters.

To convert a hex format Unicode character code to a decimal value

1. Determine the Unicode hex value for the character. For example, the hex value for the Cyrillic capital letter ZHE (Ж) is 0416.
2. Based on the hex value, determine the number of bytes required for UTF-8 encoding:

Hex value of character	**Number of bytes required**
0000 to 007F	One
0080 to 07FF	Two
0800 to FFFF	Three

   Using the same example, a hex value of 0416 requires two bytes for UTF-8 encoding.

3. Convert the hex value to binary. Using the same example, a hex value of 0416 equals the binary value 10000010110.
4. Identify *x*, *y*, and *z* bits as applicable. Start with the least significant digits to the right and pad with zeros to the left if necessary.

   In this example, the first five digits of the binary value 10000010110 correspond to the *y* bits, and the remaining six digits correspond to the *x* bits. No padding zeros are necessary.

   The first byte is 11010000.

   The second byte is 10010110.

5 Convert the bytes to decimal format. Using this example, the byte value 11010000 equals a decimal value of 208, and the byte value 10010110 equals a decimal value of 150.

Now that you have determined the decimal value for the Unicode character, you can use the values in a print command:

```
prtxt chr$(208)+chr$(150)
```

When selecting UTF-8 with the NASC command, the font must be selected with the FONT command. Disable UTF-8 encoding by choosing a different character set with the NASC command.

When using UTF-8, it is important that the font contains the desired characters. The default font, Univers, contains the largest number of glyphs of the pre-installed fonts. Unicode character numbers can be found at the web site of the Unicode organization (www.unicode.org). It is not recommended to have UTF-8 enabled when printing bar codes since bar code data will use the UTF-8 byte sequence as input, while the human readable uses the UTF-8 mapped character number.

Note: FONT and FONTD commands are reset to their defaults after a PRINTFEED (or CLL) command. NASC and NASCD commands are not reset to default after a PRINTFEED (or CLL) command.

Example

This example prints the Hiragana Letter Small A character (Unicode hex 3041), corresponding to the UTF-8 sequence 227 dec. + 129 dec. + 129 dec., in the Song font. This is followed by the Cyrillic Capital Letter ZHE (Unicode hex 0416) in the Univers font.

```
10        NASC "UTF-8"
20        FONT "Song"
30        PRTXT CHR$(227)+CHR$(129)+CHR$(129)
40        PRTXT " = Hiragana Letter Small A"
50        PRPOS 0,35
60        FONT "Univers"
70        PRTXT CHR$(208)+CHR$(150)
80        PRTXT " = Cyrillic Capital Letter ZHE"
90        PRINTFEED
```

Reserved Keywords and Symbols

This list includes keywords and symbols reserved for use by Fingerprint commands. Do not create variable or line label names that start with these keywords or errors will result.

List of Reserved Keywords

#
'
(
)
*
+
,
-
/
\
:
;
<
<=
<>
=
=<
=>
>
><
>=
?
“
^
ABS
ACTLEN
ALIGN
AN
AND
AS
ASC
BARADJUST
BARCODENAME$
BARFONT
BARFONTD
BARFONTDSIZE
BARFONTDSLANT
BARFONTSIZE
BARFONTSLANT
BARHEIGHT
BARMAG
BARRATIO
BARSET
BARTYPE
BEEP
BF
BH
BLINK
BM
BR
BREAK
BT
BUFFER
BUSY
CHDIR
CHECKSUM
CHR$
CLEANFEED
CLEAR
CLIP
CLL
CLOSE
COM ERROR
COMBUF$
COMSET
COMSTAT
CONT
COPY
COUNT&
CSUM
CURDIR$
CUT
DATA
DATAIN
DATE$
DATEADD$
DATEDIFF
DBBREAK
DBEND
DBSTDIO
DBSTEP
DELETE
DELETEPFSVAR
DEVICES
DIM
DIR
DIRNAME$
ELSE
END
EOF
ERL
ERR
ERR$
ERROR
ETUPLE
EXECUTE
FF
FIELD
FIELDNO
FILE&
FILENAME$
FILES
FIX
FLOTCALC$
FONT
FONTD
FONTDSIZE
FONTDSLANT
FONTNAME$
FONTS
FONTSIZE
FONTSLANT
FOR
FOR APPEND AS
FOR INPUT AS
FOR OUTPUT AS
FORMAT
FORMAT$
FORMFEED
FRE
FT
FUNCTEST
FUNCTEST$
GET
GETASSOC$
GETASSOCNAME$
GETPFSVAR
GOSUB
GOTO
HEAD
HEX$
HOLIDAY$
IF
II
IMAGE
IMAGENAME$
IMAGES
IMMEDIATE
INKEY$
INPUT
INPUT$
INSTR
INT
INVIMAGE
IP
KEY
KEYBMAP$
KILL
LAYOUT
LBLCOND
LED
LEFT$
LEN
LET
LINE INPUT
LIST
LISTPFSVAR
LOAD
LOC
LOF
LSET
LTS&
MAG
MAKEASSOC
MAP
MERGE
MID$
MKDIR
MOD
MODE
NAME
NASC
NASCD
NEW
NEXT
NI
NORIMAGE
NOT
OFF
OFF LINE
ON
ON BREAK
ON COMSET
ON ERROR GOTO
ON HTTP GOTO
ON KEY
ON LINE
OPEN
OPTIMIZE
OR
PB
PEC2DATA
PEC2LAY
PECTAB
PF

List of Reserved Keywords (continued)

PRINT USING	REDIRECT OUT	SAVE	STR$	TRON
PRINTFEED	REM	SET FAULTY DOT	STRING$	VAL
PRINTONE	REMOVE	SETASSOC	SYSTEM	VERBOFF
PRLINE	RENDER	SETPFSVAR	SYSVAR	VERBON
PRPOS	RENUM	SETSTDIO	TESTFEED	VERSION$
PRSTAT	RESTORE	SETUP	THEN	WEEKDAY
PRTXT	RESUME	SGN	TICKS	WEEKDAY$
PT	RESUME HTTP	SORT	TIME$	WEEKNUMBER
PUT	RESUME NEXT	SOUND	TIMEADD$	WEND
PX	RETURN	SPACE$	TIMEDIFF	WHILE
RANDOM	RIBBON	SPLIT	TO	WRITE
RANDOMIZE	RIGHT$	STDIO	TRANSFER	XOR
READ	RND	STEP	TRANSFER$	XORMODE
READY	RSET	STOP	TRANSFERSET	XYZZY
REBOOT	RUN	STORE	TROFF	

Index

Intermec

Worldwide Headquarters
6001 36th Avenue West
Everett, Washington 98203
U.S.A.

tel 425.348.2600

fax 425.355.9551

www.intermec.com

Fingerprint Developer's Guide

P/N 934-067-001