Honeywell

Fingerprint

Command Reference

Disclaimer

Honeywell International Inc. (“HII”) reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult HII to determine whether any such changes have been made. The information in this publication does not represent a commitment on the part of HII.

HII shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Android is a trademark of Google Inc.

Other product names or marks mentioned in this document may be trademarks or registered trademarks of other companies and are the property of their respective owners.

Web Address: www.honeywellaidc.com

Other product names or marks mentioned in this document may be trademarks or registered trademarks of other companies and are the property of their respective owners.

Patents

For patent information, please refer to www.hsmpats.com.

TABLE OF CONTENTS

Customer Support

Technical Assistance

To search our knowledge base for a solution or to log in to the Technical Support portal and report a problem, go to support.honeywellaidc.com.

For our latest contact information, see www.honeywellaidc.com/locations.

Product Service and Repair

Honeywell International Inc. provides service for all of its products through service centers throughout the world. To obtain warranty or non-warranty service, return your product to Honeywell (postage paid) with a copy of the dated purchase record. To learn more, go to www.honeywellaidc.com and select **Service & Repair** at the bottom of the page.

Limited Warranty

For warranty information, go to www.honeywellaidc.com and click **Get Resources > Product Warranty**.

CHAPTER

1

FINGERPRINT COMMAND REFERENCE

Fingerprint is a BASIC-inspired, printer-resident programming language you use to write custom printer application software.

This manual includes Fingerprint command descriptions and syntax for the PM23c, PM42, PM43, PM43c, PC23d, PC43d, PC43t, PD43, PD43c, PX4ie, PX6ie, and PX940 printers; PC42d and PC42t (Direct Protocol only)

Click a link below to see:

- Fingerprint Commands listed by name.
- SETUP command information for network parameters.
- supported bar code symbology information.
- supported RFID tag format information.

About Direct Protocol

Direct Protocol is a subset of Fingerprint, which you can use to download layouts and variable data from a host and combine them into labels, tickets, and tags with minimum programming. Direct Protocol also includes a versatile error handler and a flexible counter function.

Some Fingerprint commands are supported only when Direct Protocol is enabled. To enable or disable Direct Protocol, use the INPUT ON/OFF command.

CHAPTER

2

COMMANDS LISTED BY NAME

Command Name	Purpose
ABS	Returns the absolute value of a numeric expression.
ACTLEN	Returns the length of the most recently executed PRINTFEED, FORMFEED, or TESTFEED statement.
ALIGN (AN)	Specifies which part (anchor point) of a text field, bar code field, image field, line, or box will be positioned at the insertion point.
ASC	Returns the decimal ASCII value of the first character in a string expression.
BARADJUST	Enables or disables automatic adjustment of bar code position in order to avoid faulty printhead dots.
BARCODENAME$	Returns the names of the bar code symbologies stored in the printer.
BARFONT (BF)	Specifies fonts for the printing of bar code interpretation.
BARFONT (BF) ON/OFF	Enables or disables the printing of bar code interpretation.
BARHEIGHT (BH)	Specifies the height in dots of a bar code.
BARMAG (BM)	Specifies a magnification for the width of the bars in a bar code.
BARRATIO (BR)	Specifies the ratio between the wide and the narrow bars in a bar code.
BARSET	Specifies a bar code and sets additional parameters for complex bar codes.
BARTYPE (BT)	Specifies the type of bar code.
BATTERY$	Returns the voltage level, status, and charging status of the battery.
BEEP	Orders the printer to emit a beep.
BREAK	Specifies a break interrupt character separately for the keyboard and each serial communication channel.
BREAK ON/OFF	Enables or disables break interrupt character separately for the keyboard and each serial communication channel.
BUSY	Orders a busy signals for example XOFF, CTS/RTS, or PE, to be transmitted from the printer on the specified communication channel.
CHDIR	Changes to the specified directory.
CHECKSUM	Calculates the checksum of a range of program lines in connection with the transfer of programs.

Command Name	Purpose
CHR$	Returns the character represented by a decimal ASCII code.
CLEANFEED	Runs the printer's feed mechanism.
CLEAR	Clears strings, variables, and arrays in order to free memory space.
CLIP	Enables or disables the printing of partial fields.
CLL	Partial or complete clearing of the print image buffer.
CLOSE	Closes one or several files and/or devices for input/output.
COM ERROR ON/OFF	Enables or disables error handling on the specified communication channel.
COMBUF$	Reads the data in the buffer of the communication channel specified by a COMSET statement.
COMSET	Sets the parameters for background reception of data to the buffer of a specified communication channel (see COMBUF$).
COMSET OFF	Turns off background data reception and empties the buffer of the specified communication channel.
COMSET ON	Empties the buffer and turns on background data reception on the specified communication channel.
COMSTAT	Reads the status of a communication channel buffer.
CONT	Resumes execution of a program that has been interrupted by means of a STOP, BREAK, or DBBREAK statement.
COPY	Copies files.
COUNT&	Creates a counter (Direct Protocol only).
CSUM	Calculates the checksum of an array of strings.
CURDIR$	Returns the current directory as the printer stores it.
CUT	Activates an optional cutter.
CUT ON/OFF	Enables or disables automatic cutting after PRINTFEED execution. Optionally, CUT ON/OFF can adjust the media feed before and after the cutting.
DATE$	Variable for setting or returning the current date.
DATEADD$	Returns a new date after a number of days have been added to, or subtracted from, the current date or a specified date.
DATEDIFF	Returns the difference between two dates as a number of days.
DBBREAK	Adds or deletes a breakpoint for the Fingerprint Debugger.
DBBREAK OFF	Deletes all breakpoints for the Fingerprint Debugger.
DBEND	Terminates the Fingerprint Debugger.
DBSTDIO	Selects the standard IN/OUT channel for the Fingerprint Debugger.

Command Name	Purpose
DBSTEP	Specifies the interval between breaks for the Fingerprint Debugger and executes the program accordingly.
DELETE	Deletes one or several consecutive program lines from the printer's working memory.
DELETEPFSVAR	Deletes variables saved at power failure.
DEVICES	Returns the names of all devices on the standard OUT channel.
DIM	Specifies the dimensions of an array.
DIR	Specifies the print direction.
DIRNAME$	Returns the names of the directories stored in the specified part of the printer's memory.
DISPLAY IMAGE	Specifies a custom image to be shown on the display when a specific error occurs.
DISPLAY KEY	Specifies a custom image to be shown on the display.
DISPLAY STATE	Specifies a custom image to be shown on the display when the printer runs your application.
END	Ends the execution of the current program or subroutine and closes all open files and devices.
EOF	Checks for an end-of-file condition.
ERL	Returns the number of the line on which an error condition has occurred.
ERR	Returns the code number of an error.
ERR$	Returns the explanation of an error code in plain text.
ERROR	Defines error messages and enables error handling for specified error conditions (Direct Protocol only).
EXECUTE	Executes a Fingerprint program line or a file with Fingerprint program lines from within another Fingerprint program.
External Command; ZMODEM	External commands for receiving and sending data using the ZMODEM protocol.
FIELD	Creates a single-record buffer for a random file and divides the buffer into fields to which string variables are assigned.
FIELDNO	Gets the current field number for partial clearing of the print buffer by a CLL statement.
FILE& LOAD	Receives and stores binary files in the printer's memory.
FILENAME$	Returns the names of the files stored in the specified part of the printer's memory.
FILES	Lists the files stored in one of the printer's directories to the standard OUT channel.
FLOATCALC$	Calculates with floating point numbers.

Command Name	Purpose
FONT (FT)	Selects a scalable TrueType, OpenType, or bitmap font for printing subsequent PRTXT statements.
FONTD	Obsolete but retained for compatibility. See FONT.
FONTNAME$	Returns the names of the fonts stored in the printer's memory.
FONTS	Returns the names of all fonts stored in the printer's memory to the standard OUT channel.
FOR...TO...NEXT	Creates a loop in the program execution, where a counter is incremented or decremented until a specified value is reached.
FORMAT	Formats either the printer's permanent memory or a USB storage device.
FORMAT DATE$	Specifies the format of the string returned by DATE$("F") and DATEADD$(..... , "F") functions.
FORMAT INPUT	Specifies separators for the LAYOUT RUN statement used in the Direct Protocol.
FORMAT TIME$	Specifies the format of the string returned by TIME$("F") and TIMEADD$("F") functions.
FORMAT$	Formats a number represented by a string.
FORMFEED (FF)	Feeds out or pulls back a specified length of media.
FRE	Returns the number of free bytes in a specified part of the printer memory.
FUNCTEST	Performs various hardware tests.
FUNCTEST$	Returns the result of various hardware tests.
GET	Reads a record from a random file to a random buffer.
GETASSOC$	Gets a value from a string association.
GETASSOCNAME$	Traverses the tuples of a string association.
GETPFSVAR	Recovers saved variables.
GOSUB	Branches to a subroutine.
GOTO	Branches unconditionally to a specified line number or line label.
HEAD	Returns the result of a thermal printhead check.
IF...THEN...(ELSE)	Specifies conditional execution controlled by the result of a numeric expression.
IMAGE BUFFER MIRROR	Mirrors the print image around the Y-axis.
IMAGE BUFFER SAVE	Saves the content of the image buffer as a file.
IMAGE LOAD	Receives, converts, and installs image and font files.

Command Name	Purpose
IMAGENAME$	Returns the names of the images stored in the printer memory.
IMAGES	Returns the names of all images stored in the printer memory to the standard OUT channel.
IMMEDIATE	Enables or disables Immediate mode in connection with program editing without line numbers, for reading the current mode, or for reading the current standard IN and OUT channels.
INKEY$	Reads the first character in the receive buffer of the standard IN channel.
INPUT (IP)	Receives input data via the standard IN channel during the execution of a program.
INPUT ON/OFF	Enables or disables the Direct Protocol.
INPUT#	Reads a string of data from an open device or sequential file.
INPUT$	Returns a string of data, limited in regard of number of characters, from the standard IN channel, or optionally from an open file or device.
INSTR	Searches a specified string for a certain character, or sequence of characters, and returns its position in relation to the start of the string.
INVIMAGE (II)	Inverts the printing of text and images from "black-on-white" to "white-on-black."
KEY BEEP	Resets the frequency and duration of the sound produced by the beeper when any of the keys on the printer keyboard are pressed.
KEY ON/OFF	Enables or disables a specified key on the printer's front panel to be used in connection with an ON KEY...GOSUB statement.
KEYBMAP$	Returns or sets the keyboard map table.
KILL	Deletes a file, directory, or complete directory sub-trees from a printer-accessible file system.
LAYOUT	Handles layout files.
LAYOUT END	Stops the recording of a layout description and saves the layout (Direct Protocol only).
LAYOUT INPUT	Starts the recording of a layout description (Direct Protocol only).
LAYOUT RUN	Provides variable input data to a predefined layout (Direct Protocol only).
LBLCOND	Overrides the media feed setup.
LED ON/OFF	Controls the dual-color "Status" indicator on icon printers.
LEFT$	Returns a specified number of characters from the start (the extreme left side) of a given string.
LEN	Returns the number of character positions in a string.
LET	Assigns the value of an expression to a variable.

Command Name	Purpose
LINE INPUT	Assigns an entire line, including punctuation marks, from the standard IN channel to a single string variable.
LINE INPUT#	Assigns an entire line, including punctuation marks, from a sequential file or a device to a single string variable.
LIST	Lists the current program completely or partially, or lists all variables, to the standard OUT channel.
LISTPFSVAR	Lists variables saved at power failure.
LOAD	Loads a copy of a program residing in the current directory or in another specified directory into the printer working memory.
LOC	Returns the current position in an open file, or returns the status of the buffers in an open communication channel.
LOF	Returns the length in bytes of an open sequential or random file, or returns the status of the buffers in an open communication channel.
LSET	Places data left-justified into a field in a random file buffer.
LTS& ON/OFF	Enables or disables the label taken sensor.
MAG	Magnifies a font, barfont, or image with regard to height and width.
MAKEASSOC	Creates an association.
MAP	Changes the ASCII value of a character when received on the standard IN channel, or optionally on another specified communication channel.
MERGE	Merges a program in the printer's current directory (or in another specified directory) with the program currently residing in the printer's working memory.
MIBVAR&	Reads or writes MIB variables for the Simple Network Management Protocol (SNMP).
MID$	Returns a specified part of a string.
MKDIR	Creates a directory.
NAME DATE$	Formats the month parameter in return strings of DATE$("F") and DATEADD$(...,"F").
NAME WEEKDAY$	Formats the day parameter in return strings of WEEKDAY$.
NASC	Selects a single-byte character set, alternatively the multi-byte character set UTF-8.
NASCD	Selects a double-byte character set, typically the multi-byte character set UTF-8.
NEW	Clears the printer's working memory in order to allow a new program to be created.
NORIMAGE (NI)	Returns the print mode to normal printing after an INVIMAGE statement has been issued.
ON BREAK GOSUB	Branches to a subroutine when a break interrupt instruction is received.

Command Name	Purpose
ON COMSET GOSUB	Branches to a subroutine when the background reception of data on the specified communication channel is interrupted.
ON ERROR GOTO	Branches to an error-handling subroutine when an error occurs.
ON GOSUB	Branches conditionally to one of several subroutines.
ON GOTO	Branches conditionally to one of several lines.
ON HTTP GOTO	Branches to a subroutine when a request for an application CGI is received.
ON KEY GOSUB	Branches to a subroutine when a specified key on the printer's front panel is activated.
ON MIBVAR& GOSUB	Branches to a subroutine when a specified SNMP MIB variable is changed.
ON/OFF LINE	Controls the Select signal on the "centronics:" communication channel.
OPEN	Opens a file or device (for example, a network connection), or creates a new file—for input, output, or append, allocating a buffer, and specifies the access mode.
OPTIMIZE BATCH ON/OFF	Enables or disables optimization for batch printing.
PORTIN	Reads the status of a port on a Serial/Industrial Interface Board.
PORTOUT ON/OFF	Sets relay ports or optical ports on a Serial/Industrial Interface Board to either on or off.
POWER	Places the portable printer in standby or deep sleep mode.
PRBAR (PB)	Provides input data to a bar code.
PRBOX (PX)	Creates a box containing a single text line or a frame of multiple hyphenated text lines.
PRBUF	Receives and prints bitmap image data using the PRBUF protocol.
PRDIAGONAL	Creates a diagonal line.
PRELLIPSE	Creates an ellipse.
PRIMAGE (PM)	Renders an image stored in the printer's memory to the label buffer.
PRINT	Outputs data to the standard OUT channel.
PRINT KEY ON/OFF	Enables or disables printing of a label by pressing the Print key.
PRINT#	Outputs data to a specified opened device or sequential file.
PRINTFEED (PF)	Prints and feeds out one or a specified number of labels, tickets, tags, or portions of strip, depending on the printer setup.
PRINTONE	Outputs characters (specified by their ASCII values) to the standard OUT channel.

Command Name	Purpose
PRINTONE#	Outputs characters specified by their ASCII values to a device or sequential file.
PRLINE (PL)	Creates a line.
PRPOS (PP)	Specifies the insertion point for a line of text, a bar code, an image, a box, or a line.
PRSTAT	Returns the current printer status or the current position of the insertion point.
PRTXT (PT)	Provides the input data for a text field.
PUT	Writes a given record from the random buffer to a given random file.
RANDOM	Generates a random integer within a specified interval.
RANDOMIZE	Reseeds the random number generator, optionally with a specified value.
READY	Orders a ready signal, for example XON, CTS/RTS or PE, to be transmitted from the printer on the specified communication channel.
REBOOT	Restarts the printer.
REDIRECT OUT	Redirects the output data to a created file.
REM	Adds headlines and comments to the program without including them in the execution.
REMOVE IMAGE	Removes a specified image from the printer memory.
RENDER ON/OFF	Enables or disables the rendering of text, bar code, image, box, and line fields.
RENUM	Renumbers the lines of the program currently residing in the printer's working memory.
REPRINT ON/OFF	Enables or disables the reprinting of a label in Direct Protocol.
RESUME	Resumes program execution after an error-handling subroutine has been executed.
RETURN	Returns to the main program after having branched to a subroutine due to a GOSUB statement.
REWINDCONTROL	Controls the internal rewind motor in PM-series printers.
REWINDVOID	Rewinds VOID labels with the liner in PM-series printers with RFID enabled.
RIGHT$	Returns a specified number of characters from a given string, starting from the end (extreme right end) of the string.
RSET	Places data right-justified into a field in a random file buffer.
RUN	Starts the execution of a program.
SAVE	Saves a file in the printer's memory or in a USB storage device.

Command Name	Purpose
SET FAULTY DOT	Marks one or several dots on the printhead as faulty, or marks all faulty dots as correct.
SETASSOC	Sets a value for a tuple in a string association.
SETPFSVAR	Registers variables to be saved at power failure.
SETSTDIO	Selects standard IN and OUT communication channel.
SETUP	Enters Setup mode or changes the setup.
SETUP GET	Gets the current setting for a single setup object.
SETUP KEY	Enables or disables the access to Setup mode from the printer keypad.
SETUP WRITE	Creates a file containing the current printer setup, or returns the setup file on a specified communication channel.
SGN	Returns the sign (positive, zero, or negative) of a specified numeric expression.
SORT	Sorts a one-dimensional array.
SOUND	Makes the printer beeper produce a sound specified in regard of frequency and duration.
SPACE$	Returns a specified number of space characters.
SPLIT	Splits a string into an array according to the position of a specified separator character and returns the number of elements in the array.
STOP	Terminates execution of a program and returns the printer to Immediate mode.
STORE IMAGE	Sets up parameters for storing an image in the printer memory.
STORE INPUT	Receives and stores protocol frames of image data in the printer memory.
STORE OFF	Terminates the storing of an image and resets the storing parameters.
STR$	Returns the string representation of a numeric expression.
STRING$	Repeatedly returns the character of a specified ASCII value, or the first character in a specified string.
SYSHEALTH	Sets or retrieves the system health and controls the Ready-to-Work indicator on the printer front panel.
SYSHEALTH$	Returns the error causing the current system health status.
SYSVAR	System array for reading or setting system variables.
TAGFIELD	Defines a field in the RFID tag memory area available for RFID operations.
TAGFORMAT	Specifies the format of the data to be read from or written to an RFID tag.
TAGPROTECT	Protects tag data from being overwritten.

Command Name	Purpose
TAGREAD	Reads an RFID tag field.
TAGWRITE	Writes to an RFID tag field.
TESTFEED	Adjusts the label stop, ribbon end/low and paper low sensors, and RFID module while running the media and ribbon feed mechanisms.
TICKS	Returns the elapsed time since the last power up in the printer, expressed in number of "TICKS" (1 TICK = 0.01 sec).
TIME$	Sets or returns the current time.
TIMEADD$	Returns a new time after a number of seconds have been added to or subtracted from the current time or from a specified time.
TIMEDIFF	Returns the difference between two specified times in number of seconds.
TRANSFER KERMIT	Transfers data files using the KERMIT communication protocol.
TRANSFER NET	Transfers files to and from the printer using FTP.
TRANSFER STATUS	Checks the last TRANSFER KERMIT or TRANSFER ZMODEM operation.
TRANSFER ZMODEM	Transfers data files using the ZMODEM communication protocol.
TRANSFER$	Executes a transfer from source to destination as specified by a TRANSFERSET statement.
TRANSFERSET	Enters setup for the TRANSFER$ function.
TRON/TROFF	Enables or disables tracing of the program execution.
VAL	Returns the numeric representation of a string expression.
VERBON/VERBOFF	Specifies the verbosity level of the communication from the printer on the standard OUT channel (serial communication only).
VERSION$	Returns the firmware version, printer family, or type of CPU board.
WEEKDAY	Returns the weekday of a specified date.
WEEKDAY$	Returns the name of the weekday from a specified date.
WEEKNUMBER	Returns the number of the week for a specified date.
WHILE...WEND	Executes a series of statements in a loop providing a given condition is true.
XORMODE ON/OFF	Enables or disables the xor/flip mode of Fingerprint in connection with graphical operations.
VERIFIER RESULT	Retrieve or print out a summary of the label verification results.

ABS

Purpose

Returns the absolute value of a numeric expression.

Syntax

ABS(<*nexp*>)

Parameters

nexp
Numeric expression from which the absolute value will be returned.

Notes

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

Examples

Input	Results in
PRINT ABS(20-25)	5
PRINT ABS(25-20)	5
PRINT ABS(5-5)	0
PRINT ABS(20*-5)	100

ACTLEN

Purpose

Returns the length in dots of the most recently executed PRINTFEED, FORMFEED, or TESTFEED statement.

Syntax

ACTLEN

Notes

Due to technical reasons concerning the stepper motor control and label gap detection, a small deviation from the expected result may occur.

Example

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

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

This results in:

```
1121
```

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

ALIGN

Purpose

Specifies which anchor point of a text field, bar code field, image field, line, or box will be positioned at the insertion point. This command can be abbreviated as AN.

Syntax

ALIGN<*nexp*>

or

AN<*nexp*>

Parameters

<nexp>
Anchor point of the object. Range is 1 (default) to 9. Reset to default by PRINTFEED execution.

Notes

Each text, bar code, or image field has nine possible anchor points, whereas lines and boxes have three. The selected anchor point determines the position of the object relative to the insertion point. The field will be rotated around the anchor point according to the nearest preceding DIR statement.

The nine anchor points of a text, bar code, or image field are oriented this way:

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

Text Field Anchor Points

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

Bar Code Field Anchor Points

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

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

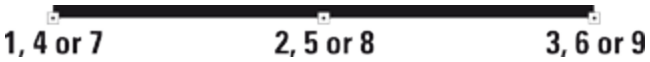
For composite bar codes, the human readable bar code interpretation for the 2-dimensional element is outside the imaginary box.

Image Field Anchor Points

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

Line Anchor Points

Lines have only three anchor points:



The ALIGN command does not apply to diagonal lines (PRDIAGONAL).

Box Anchor Points

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

A special case is multi-line text fields in a box. The fields can be aligned in nine positions in relation to the box, while the box itself only has three anchor points, as described above. For more information on alignment of multi-line text fields, see PRBOX (PX).

Example

This example prints one line of text aligned left on the baseline:

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

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

ASC

Purpose

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

Syntax

ASC(*<sexp>*)

Parameters

<sexp>
String expression from which the ASCII decimal value of the first character is returned.

Notes

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

Example 1

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

This results in:

```
71
```

Example 2

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

This results in:

```
49
```

BARADJUST

Purpose

Enables or disables automatic adjustment of bar code position in order to avoid faulty printhead dots.

Syntax

BARADJUST<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
Maximum left offset in dots. Default is 0.

<*nexp2*>
Maximum right offset in dots. Default is 0.

Notes

Occasionally a printer may have to run for some time with a faulty printhead before a replacement printhead can be installed.

Single faulty dots will produce very thin "white" lines along the media. Faulty dots may make horizontal (picket fence) bar codes unreadable.

However, if the bar code is moved slightly to the left or right, the trace of a faulty dot may come between the printed bars of the bar code and the symptom is remedied temporarily. BARADJUST attempts to adjust the bar code even if several faulty dots have been specified, but returns error 1052 if it is unable to adjust the bar code successfully.

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

The BARADJUST statement does not work with:

- Vertically printed bar codes (ladder style)
- Stacked bar codes (such as Code 16K)
- Bar codes with horizontal lines (such as DUN-14/16)
- EAN/UPC-codes (interpretation not repositioned)

Example

This example enables BARADJUST within 10 dots to the left and 5 dots to the right of the original position for a specific bar code, then disables BARADJUST:

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

BARCODENAME$

Purpose

Returns the names of the bar codes stored in the printer.

Syntax

BARCODENAME$(<*nexp*>)

Parameters

<*nexp*>
Set to true (non-zero value) or false (0). False indicates first bar code. True indicates next bar code.

Notes

BARCODENAME$(0) produces the first bar code name in alphabetical order.
BARCODENAME$(≠0) produces the next name. If no bar codes remain, an empty string is returned. This statement can be repeated as long as there are bar code names remaining.

Example

Use a program like this to list the names of all bar codes installed on the printer:

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

A typical result could be as follows:

```
ADDON2
ADDON5
C2OF5
C2OF5IND
C2OF5INDC
C2OF5MAT
CODABAR
CODE11
CODE128
CODE128A
CODE128B
CODE128C
CODE16K
CODE39
CODE39A
CODE39C
CODE49
CODE93
DATAMATRIX
```

DUN
EAN128
EAN128A
EAN128B
...

BARFONT

Purpose

Specifies fonts for the printing of bar code interpretation. This command can be abbreviated as BF.

Syntax

BARFONT[#<*ncon*>,]<*sexp1*>[,<*nexp1*>[,<*nexp2*>[,<*nexp3*>[,<*nexp4*> [,<*nexp5*>[,<*nexp6*> [,<*nexp7*>[,<*nexp8*>[,<*nexp9*>]]]]]]]]][ON]

or

BF[#<*ncon*>,]<*sexp1*>[,<*nexp1*>[,<*nexp2*>[,<*nexp3*>[,<*nexp4*> [,<*nexp5*>[,<*nexp6*>[,<*nexp7*> [,<*nexp8*>[,<*nexp9*>]]]]]]]]][ON]

Parameters

#<*ncon*>
(Optional) Specifies which parameter in the syntax is the first parameter in the statement (possibly bypassing some of the initial parameters). Default is #1.

<*sexp1*>
Name of the font selected for bar code interpretations. Corresponds to the FONT statement, but only affects bar code interpretation.

<*nexp1*>
Font height in points. Default is 12. Corresponds to the FONT statement, but only affects bar code interpretation.

<*nexp2*>
Clockwise slant in degrees. Range is 0° to 90°. Values greater than 65° to 70° will be unreadable. Slanting increases clockwise. Default is 0. Corresponds to the FONT statement, but only affects bar code interpretation.

<*nexp3*>
Distance in dots between the bottom of the bar code pattern and the top of the character cell. Default is 6.

<*nexp4*>
Magnification in regard to height. Range is 1 (default) to 4. Corresponds to the MAG statement.

<*nexp5*>
Magnification in regard to width. Range is 1 (default) to 4. Corresponds to the MAG statement.

<*nexp6*>
Width enlargement in percent relative to height. Range is 1 to 1000 (for example, 200 doubles the width). Default is 100. Not valid for bitmap fonts. When using this parameter, all parameters in the syntax must be included in the statement (name, height, slant, and width).

<*nexp7*>
Insertion point for the bar code interpretation. A value between 1 and 9 can be set,

corresponding to positions in the figure. The values 4, 5, and 6 are interpreted as 7, 8, and 9 respectively. Default is 0 (disabled). This function overrides the ALIGN command.

<*nexp8*>
Horizontal offset in dots from the insertion point for the bar code interpretation. Default is 0. The offset is set with respect to the bar code direction, not necessarily the paper feed direction.

<*nexp9*>
Vertical offset in dots from the insertion point for the bar code interpretation. Default is 0. The offset is set with respect to the bar code direction, not necessarily the paper feed direction.

Notes

Reset to default by executing a PRINTFEED. If a MAG statement is executed after a BARFONT statement, the size of the barfont is affected by the MAG statement.

The printing of bar code interpretation can be enabled by a trailing ON, which corresponds to a BARFONT ON statement. Exceptions to this condition are as follows:

- In all EAN and UPC bar codes, the interpretation is an integrated part of the code. Such an interpretation is not affected by a BARFONT statement, but will be printed in according to specification, provided that interpretation printing has been enabled by a BARFONT ON statement.
- Certain bar codes, like Code 16K, cannot contain any interpretation at all. In this case, the selected barfont will be ignored.

Example

This example prints a Code 39 bar code, selects the same barfont for all directions, and enables printing of the bar code interpretation:

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

BARFONT ON/OFF

Purpose

Enables or disables the printing of bar code interpretation. This command can be abbreviated as BF ON|OFF.

Syntax

BARFONT ON|OFF[,<*nexp*>]

or

BF ON|OFF[,<*nexp*>]

Parameters

<*nexp*>
Numerical expression that disables or enables guard bar printing for EAN/UPC bar codes. Default is enabled. Reset to default by executing a PRINTFEED.

Notes

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

BARFONT ON can be replaced by a BARFONT statement appended by a trailing ON. When printing EAN/UPC bar codes, the trailing parameter 0 can be specified after the BARFONT OFF command to disable the printing of the guard bars, which by default will print.

BARFONT OFF,0 disables printing of the guard bars. BARFONT OFF,1 is the default (guard bars enabled) and is equivalent to BARFONT OFF.

Examples

This example sets up a Code 39 bar code, selects a barfont for each direction, and enables the printing of the bar code interpretation. Compare with the example for BARFONT statement:

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

This example sets up an EAN8 bar code and disables the guard bars:

```
10 BARTYPE "EAN8"
20 BARFONT OFF,0
```

```
30 PRBAR "1234567"
40 PRINTFEED
```

BARHEIGHT

Purpose

Specifies the height of a bar code. This command can be abbreviated as BH.

This command is not applicable to QR Code.

Syntax

BARHEIGHT<*nexp*>

or

BH<*nexp*>

Parameters

<*nexp*>
Height of the bars in the bar code expressed in number of dots. Default is 100. Reset to default by executing a PRINTFEED.

Notes

In bar codes consisting of several elements on top of each other (for example, Code 16K) the barheight specifies the height of one element. The height is not affected by BARMAG statements. BARHEIGHT can be replaced by a parameter in the BARSET statement.

Example

This example sets up a Code 39 bar code, selects a barfont for all directions and sets the bar code height to 120 dots:

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

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

BARMAG

Purpose

Specifies a magnification for the width of the bars in a bar code. This command can be abbreviated as BM.

Syntax

BARMAG<*nexp*>

or

BM<*nexp*>

Parameters

<*nexp*>
Magnification for the width of the bars in the bar code. Range depends on type of bar code. Default is 2. Reset to default by executing a PRINTFEED.

Notes

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

The magnification also affects the interpretation in EAN/UPC bar codes, since the interpretation is an integrated part of the EAN/UPC code. BARMAG can be replaced by a parameter in the BARSET statement.

Examples

This example sets up a Code 39 bar code, selects a barfont for all directions and sets the magnification to 3:

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

A more compact method is illustrated by the BARSET example.

BARRATIO

Purpose

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

This command can be abbreviated as BR.

Syntax

BARRATIO<*nexp1*>,<*nexp2*>

or

BR<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
Thickness of the wide bars relative to the narrow bars.

<*nexp2*>
Thickness of the narrow bars relative to the wide bars.

Default ratio is 3:1. Reset to default by executing a PRINTFEED.

Notes

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

For example:

The default BARRATIO is 3:1 and the default BARMAG is 2.

(3:1) × 2 = 6:2

That is, the wide bars are 6 dots wide and the narrow bars are 2 dots wide.

Note that certain bar codes, such as EAN/UPC codes, have a fixed ratio. In those cases, BARRATIO is ignored. BARRATIO can be replaced by two parameters in the BARSET statement.

Example

This example sets up a Code 39 bar code, selects a barfont for all directions and enables printing of the bar code interpretation:

```
10 PRPOS 30,400
20 DIR 1
30 ALIGN 7
40 BARTYPE "CODE39"
50 BARRATIO 2,1
60 BARHEIGHT 120
70 BARMAG 3
80 BARFONT "Univers"ON
```

```
90 PRBAR "ABC"
100 PRINTFEED
```

A more compact method is illustrated by the BARSET example.

BARSET

Purpose

Specifies a bar code and sets additional parameters for complex bar codes.

Syntax

BARSET[#<*ncon*>,][<*sexp*>[,<*nexp1*>[,<*nexp2*>[,<*nexp3*>[,<*nexp4*>[,<*nexp5*>[,<*nexp6*>[,<*nexp7*>[,<*nexp8*>[,<*nexp9*>[,<*nexp10*>[,<*nexp11*>]]]]]]]]]]]

Parameters

Each bar code symbology uses BARSET parameters differently. For more information on how your selected symbology affects the syntax of this command, or for programming examples, see Supported Symbologies.

#<*ncon*>
Start parameter which specifies which parameter in the syntax is the first optional parameter (#1–#12). This way you may bypass some of the initial parameters (for example, ratio and enlargement). Default is 1.

<*sexp*> (#1)
Bar code symbology. No default. Corresponds to BARTYPE.

<*nexp1*> (#2)
Thickness of the wide bars relative to the narrow bars. Default is 3. Corresponds to BARRATIO.

<*nexp2*> (#3)
Thickness of the narrow bars relative to the wide bars. Default is 1. Corresponds to BARRATIO.

<*nexp3*> (#4)
Enlargement. Default is 2. Corresponds to BARMAG.

<*nexp4*> (#5)
Height of the code in dots. Default is 100. Corresponds to BARHEIGHT.

<*nexp5*> (#6)
Bar code specific parameter. Default is 2.

<*nexp6*> (#7)
Bar code specific parameter. Default is 3.

<*nexp7*> (#8)
Bar code specific parameter. Default is 1.

<*nexp8*> (#9)
Bar code specific parameter. Default is 0.

<*nexp9*> (#10)
Bar code specific parameter. Default is 0.

<*nexp10*> (#11)
Bar code specific parameter. Default is 0.

<*nexp11*> (#12)
Bar code specific parameter. Default is 0.

Notes

This statement can replace the statements BARHEIGHT, BARRATIO, BARTYPE, and BARMAG. Although primarily intended for 2-dimensional and composite bar codes, BARSET can be used for any type of bar code if irrelevant parameters are left out (for example, <*nexp5*> to <*nexp11*>).

Example

This example shows how to use a BARSET statement to print a Datamatrix bar code:

```
10 BARSET "DATAMATRIX",3,1,4,1,1
20 PB "This is a long string in Datamatrix format"
30 PF
```

This example shows how to use a BARSET statement to specify a Code 39 bar code:

```
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
```

BARTYPE

Purpose

Specifies the bar code symbology to use. This command can be abbreviated as BT.

Syntax

BARTYPE<*sexp*>

or

BT<*sexp*>

Parameters

<*sexp*>
Specifies the bar code symbology to use. Must be a valid bar code name. There is no default value. For a list of valid bar code names, see Supported Symbologies.

Notes

The selected bar code type must exist in the printer memory and must be entered as a string expression. BARTYPE can be replaced by a parameter in the BARSET statement.

Example

This example sets up a Code 39 bar code, selects a barfont for all directions, and enables printing of the bar code interpretation:

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

A more compact method is illustrated by the BARSET example.

BATTERY$

This command is supported only by printers with a battery.

Purpose

Returns the voltage level, status, and charging status of the battery.

Syntax

BATTERY$(*<nexp>)*

Parameters

<nexp>
Specifies the parameter to return. Must be one of the following:

Value	Description
0	(Default) Returns the current battery voltage.
1	Returns the current battery status: "battery depleted", "battery low", "battery good", or "battery full". For definitions of these battery parameters, see the documentation for your printer or battery.
2	Returns the current battery charging status: "not charging", "charging complete", "charging in progress" or "charging error".

Notes

A hardware limitation causes the printer to return "battery not installed" if BATTERY$(1) is issued while the printer is printing. This message is returned even if the printer is equipped with a battery and connected to an external power supply.

Example

This example sets the low battery threshold value to 12.5 V and returns the current battery status:

```
10 SETUP "BATTERY,LOW BATTERY,12.5"
20 PRINT BATTERY$(1)
 RUN
```

BEEP

Purpose

Orders the printer to emit a beep.

Syntax

BEEP

Notes

This statement makes the printer emit a beep of ~800 Hz for 1/4 of a second. If a different frequency and/or duration is desired, use a SOUND statement instead.

Example

In this example, the printer beeps when an error occurs:

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

BREAK

Purpose

Specifies a break interrupt character separately for the keyboard and serial communication channels.

Syntax

BREAK<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
0: "console:"
1: "uart1:"
6: "usb1:" (Supported when the virtual COM port is enabled)

<*nexp2*>
Decimal ASCII value for the break interrupt character. The default value for PM series printers is ASCII 136 (Shift+C). The default value for PC series printers is ASCII 8 (Back key).

Notes

The execution of a program can be interrupted using a method specified by the BREAK statement.

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

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

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

o prevent an erroneous program from running in an unbreakable loop, Honeywell strongly recommends that you include some facility for issuing a break interrupt from the host computer in startup (autoexec) files.

Examples

In this example, the ASCII character 13 decimal (Enter key) is selected and enabled as BREAK character on the console:

```
10 BREAK 0,13
20 BREAK 0 ON
30 GOTO 30
```

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

BREAK ON/OFF

Purpose

Enables or disables break interrupt separately for the keyboard and serial communication channels.

Syntax

BREAK<*nexp*>ON|OFF.

Parameters

<nexp>
0: "console:"
1: "uart1:"
6: "usb1:" (Supported when the virtual COM port is enabled)

ON|OFF
Enable or disable the break function.

Notes

The use of the break interrupt specified by a BREAK statement can be enabled or disabled separately for each serial communication channel, or for the printer keyboard by BREAK ON or BREAK OFF statements.

By default, break interrupt is enabled from the keyboard and disabled from all communication channels. BREAK OFF deletes any existing break interrupt character stored in the printer temporary memory for the specified device.

Example

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

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

BUSY

Purpose

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

Syntax

BUSY[*<nexp>*]

Parameters

<nexp>
(Optional) Specifies the channel as:
1: "uart1:"
2: "uart2:"
3: "uart3:"
4: "centronics:"
6: "usb1:" (Supported when virtual COM port is enabled)
7: "uart4:"
8: "uart5:"

Notes

A communication protocol usually contains some "busy" signal, which tells the host computer that the printer is unable to receive data.

Use the BUSY statement to order a busy signal to be transmitted on the specified communication channel. If no channel is specified, the signal is transmitted on the standard OUT communication channel. For more information, see SETSTDIO.

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

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

Example

This example prevents the printer from receiving more data on "uart1:" during the process of printing a label:

```
10 FONT "Univers"
20 PRTXT "HELLO!"
30 BUSY1
40 PRINTFEED
50 READY1
RUN
```

CHDIR

Purpose

Changes to the specified directory.

Syntax

CHDIR*<scon>*

Parameters

<scon>
Specifies the current directory (see DEVICES). Default is "/c".

Notes

Directory names are case sensitive. You must first set SYSVAR(43) to 1 before running this command in order for the printer to recognize the directory you specify.

By default, the printer permanent memory ("/c") is the current directory. The current directory is used if the Fingerprint command does not specify a directory (such as FILES). This implies that to access other directories (such as "d:", "tmp:", or "/rom"), you must include such a reference in your instructions (such as FILES "/rom".)

Example

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

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

CHECKSUM

Purpose

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

Syntax

CHECKSUM(<*nexp1*>,<*nexp2*>)

Parameters

<*nexp1*>
Number of the first line in a range of program lines.

<*nexp2*>
Number of the last line in a range of program lines.

Notes

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

Example

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

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

This results in:

```
60095
```

CHR$

Purpose

Returns the character represented by a decimal ASCII code.

Syntax

CHR$(*<nexp>*)

Parameters

<nexp>
Decimal ASCII code to be converted to a readable character.

Notes

This function is useful for entering characters that cannot be produced from the host keyboard, such as non-printable characters ASCII 0 to ASCII 31 decimal. Only integers between 0 and 255 are allowed. Values less than 0 or larger than 255 result in an error 41, "Parameter out of range."

Example

In this example, the decimal ASCII code for "A" is 65 and for "B" is 66.

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

This results in:

```
A
B
```

CLEANFEED

Purpose

Runs the printer feed mechanism.

Syntax

CLEANFEED*<nexp>*.

Parameters

<nexp>
Feed length expressed as a positive or negative number of dots.

Notes

The CLEANFEED statement activates the stepper motor that drives the printer's platen roller (the rubber roller beneath the printhead). For thermal transfer printers, it may also drive the ribbon mechanism. The motor runs regardless of possible error conditions, such as if the printhead is lifted, or if there is no ribbon or media supply left. Thus, the CLEANFEED statement is suitable for cleaning and for the loading of transfer ribbon.

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

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

Executing a CLEANFEED, as opposed to TESTFEED, does not affect the adjustment of the label stop sensor or black mark sensor, regardless of what type of media or other object passes the sensor.

Note that CLEANFEED, as opposed to FORMFEED, must always specify the feed length.

Example

This example pulls a cleaning card back and forth under the printhead three times. To set this up, three 1200-dot positive CLEANFEEDs and three 1200-dot negative CLEANFEEDs are performed:

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

CLEAR

Purpose

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

Syntax

CLEAR

Notes

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

Example

In this example, the CLEAR statement empties the strings to free up memory:

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

This results in:

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

CLIP

Purpose

Enables or disables the printing of partial fields.

Syntax

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

Parameters

BARCODE
Toggles between partial bar code fields enable/disable.

HEIGHT
Clips the height of the bar so the bar code will fit inside the print window. A one-dimensional bar code may still be readable. Only used with the BARCODE parameter.

INFORMATION
Clips the bar code lengthwise, so some bars will be missing, making the bar code unreadable. Only used with the BARCODE parameter.

X
Clips the part of the bar code that is outside the X-dimension of the print window. Only used with the BARCODE parameter.

Y
Clips the part of the bar code that is outside the Y-dimension of the print window. Only used with the BARCODE parameter.

TEXTBOX|X
Toggles between partial text fields enable/disable.

ON|OFF
Enables or disables use of partial text, image, line, and box fields.

Notes

"Partial fields" means that the firmware accepts and prints text, bar code, image, lines, and box fields even if they extend outside the print window as specified by the printer setup with regard to X-Start, Width, and Length.

Negative PRPOS values are allowed. However, all parts of the fields outside the print window are excluded from the printout and the field is clipped at the borders of the print window.

There are two main cases:

- CLIP BARCODE [HEIGHT|INFORMATION|X|Y] is used for bar code fields only. Note that some bar codes, like Maxicode, consist of images and should in this context be regarded as image fields.
- CLIP TEXTBOX ON|OFF or CLIP X ON|OFF is used only for the PRBOX command.

- CLIP ON|OFF is only used for text, image, line, and box fields. When the use of partial fields is disabled, error 1003 ("Field out of label") results if any field extends outside the print window. Note the difference between the physical size of the label and the size of the print window specified by the printer setup, which determines where the fields will be clipped.

Example

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

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

CLL

Purpose

Partially or completely clears the print image buffer.

Syntax

CLL [<*nexp*>]

Parameters

<*nexp*>
Specifies the field from which the print image buffer should be cleared. If no value for <*nexp*> is specified, the entire print image buffer is cleared.

Notes

The print image buffer stores the printable image after processing while awaiting the printing to be executed. The buffer can be cleared, partially or completely, by the use of a CLL statement. Note that there must be no changes in the layout between the PRINTFEED and the CLL statements, or the layout will be lost. Also note that partial clearing always starts from the end, which means that the fields which are executed last are cleared first.

- CLL<*nexp*> partially clears the buffer from a specified field to the end of the program. The field is specified by a FIELDNO function. Partial clearing is useful in connection with print repetition. To avoid superfluous reprocessing, one or several fields can be erased from the buffer and be replaced by other information, while the remaining parts of the label are kept in the buffer.
- CLL (without any field number) clears the buffer completely. When certain error conditions have occurred, it is useful to be able to clear the print image buffer without having to print a faulty label. Should the error be attended to, without the image buffer being cleared, there is a risk that the correct image will be printed on top of the erroneous one on the same label. When you are working with more complicated programs in which all implications may be difficult to grasp, Honeywell recommends that you include a CLL statement in your error-handling subroutines.

Examples

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

```
10 PRPOS 100,300
20 FONT "Univers"
30 PRTXT "HAPPY"
40 A%=FIELDNO
50 PRPOS 100,250
60 PRTXT "NEW YEAR!"
70 PRINTFEED
80 CLL A%
```

```
90 PRPOS 100,250
100 PRTXT "BIRTHDAY!"
110 PRINTFEED
RUN
```

This example demonstrates complete clearing. The print image buffer is completely cleared if error 1030 ("Character missing in chosen font") occurs.

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

CLOSE

Purpose

Closes one or several files and/or devices.

Syntax

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

Parameters

#
(Optional) Indicates that whatever follows is a number.

<*nexp*>
Number assigned to a file or device when it is opened using the OPEN command.

Notes

This statement is the opposite of OPEN. Only files or devices which already have been open using the OPEN command, they can be closed using the CLOSE command.

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

You must CLOSE the file or it will not be saved if the printer is turned off.

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

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

Examples

This statement closes all open files and devices:

200 CLOSE

A number of files or devices (1 through 4 in this example) can be closed simultaneously using any of the following types of statement:

200 CLOSE 1,2,3,4

or

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

or

200 CLOSE 1,2,#3,4

COM ERROR ON/OFF

Purpose

Enables or disables error handling on the specified communication channel.

Syntax

COM ERROR<*nexp*>ON|OFF

Parameters

<*nexp*>
One of the following communication channels:

1: "uart1:"
2: "uart2:"
3: "uart3:"
4: "centronics:"
7: "uart4:"
8: "uart5:"

Default is COM ERROR OFF on all channels.

Notes

This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COMSTAT, and COMBUF$. Each character received is checked for the following errors:

- Received break
- Framing error
- Parity error
- Overrun error

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

Example

In this example, a message appears on the printer's screen if reception is interrupted by any of the four specified COMSET conditions:

```
10 COM ERROR 1 ON
20 A$="Max. number of char. received"
30 B$="End char. received"
40 C$="Attn. string received"
50 D$="Communication error"
60 COMSET 1, "A",CHR$(90),"#","BREAK",20
70 ON COMSET 1 GOSUB 1000
80 COMSET 1 ON
```

```
90 IF QDATA$="" THEN GOTO 90
100 END
1000 QDATA$=COMBUF$(1)
1010 IF COMSTAT(1) AND 2 THEN PRINT A$
1020 IF COMSTAT(1) AND 4 THEN PRINT B$
1030 IF COMSTAT(1) AND 8 THEN PRINT C$
1040 IF COMSTAT(1) AND 32 THEN PRINT D$
1050 PRINT QDATA$:RETURN
```

COMBUF$

Purpose

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

Syntax

COMBUF$(<*nexp*>)

Parameters

<*nexp*>
Describes one of the following communication channels:

1: "uart1:"

2: "uart2:"

3: "uart3:"

4: "centronics:

5: "net1:"

6: "usb1:"

7: "uart4:"

8: "uart5:"

9: "usbhost:"

10: "bluetooth:"

11: "ftp1:"

12: "http1:"

13: "lpr1:"

Notes

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

With COMBUF$, the buffer can be read and its content used in your program. When the communication has been interrupted by "end character", "attention string", or "max. no. of char." (see COMSET), you may use an ON COMSET GOSUB subroutine and assign the data from the buffer to a variable as illustrated in the example below.

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

Example

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

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

COMSET

Purpose

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

Syntax

COMSET<*nexp1*>,<*sexp1*>,<*sexp2*>,<*sexp3*>,<*sexp4*>,<*nexp2*>

Parameters

<*nexp1*>
Describes one of the following communication channels:

1: "uart1:"

2: "uart2:"

3: "uart3:"

4: "centronics:"

5: "net1:"

6: "usb1:"

7: "uart4:"

8: "uart5:"

9: "usbhost:"

10: "bluetooth:"

11: "ftp1:"

12: "http1:"

13: "lpr1:"

<*sexp1*>
Specifies the start of the message string. Maximum 12 characters.

<*sexp2*>
Specifies the end of the message string. Maximum 12 characters.

<*sexp3*>
Specifies characters to be ignored. Maximum 42 characters.

<*sexp4*>
Specifies the attention string. Maximum 12 characters.

<nexp2>
Specifies the maximum number of characters to be received. Enter a value > or = 1. If *<nexp2>* = 0, the first character is lost.

Notes

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

Related instructions are COMSTAT, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. The communication channels are explained in connection with the DEVICES statement.

The start and end strings are character sequences which tell the printer when to start or stop receiving data. Each string can be up to 12 characters and may be " " (ASCII character 32).

It is possible to make the printer ignore certain characters. Such characters are specified in a string of up to 42 characters, where the order of the individual characters does not matter, and may be " ". The attention string (maximum 12 characters) interrupts reception and may be " ".

The length of the COMSET strings are checked before they are copied into the internal structure. If any of these strings are too long, error 26 ("Parameter too large") occurs.

When the printer receives the specified maximum number of characters without previously encountering any end string or attention string, the transmission is interrupted. The maximum number of characters also decides how much of the memory is allocated to the buffer.

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

- If an end string is encountered.
- If an attention string is encountered.
- If the maximum number of characters is received.
- If error-handling is enabled for the communication channel in question (see COM ERROR ON/OFF) and a communication error occurs.

Note: This condition can be checked by a COMSTAT function. Any interruption will have a similar effect as a COMSET OFF statement (interrupting reception), but the buffer will not be emptied and can still be read by a COMBUF$ function. After reception is interrupted, an ON COMSET GOSUB statement can be issued to control what happens next. COMSET does not support auto-hunting (see SETSTDIO).

Example

This example shows how to open "uart1:" for background communication.

Any record starting with the character # and ending with the character & will be received. The characters X, Y and Z are ignored. The character = stops reception. A maximum of 50 characters are allowed.

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

COMSET OFF

Purpose

Turns off background data reception and empties the buffer of the specified communication channel.

Syntax

COMSET<*nexp*>OFF

Parameters

<*nexp*>
Describes one of the following communication channels:

1: "uart1:"

2: "uart2:"

3: "uart3:"

4: "centronics:"

5: "net1:"

6: "usb1:"

7: "uart4:"

8: "uart5:"

9: "usbhost:"

10: "bluetooth:"

11: "ftp1:"

12: "http1:"

13: "lpr1:"

Notes

This statement is closely related to COMSET, ON COMSET GOSUB, COMSTAT, COMSET ON, COM ERROR ON/OFF, and COMBUF$. The COMSET OFF statement closes the reception and empties the buffer of the specified communication channel.

Example

In this example, the COMSET OFF statement closes "uart1:" for background reception and empties the buffer:

```
1 REM Exit program with #STOP&

10 COMSET1,"#","&","ZYX","=",50

20 ON COMSET 1 GOSUB 2000
```

```
30 COMSET 1 ON
40 IF A$ <> "STOP" THEN GOTO 40
50 COMSET 1 OFF
.....
.....
1000 END
2000 A$= COMBUF$(1)
2010 PRINT A$
2020 COMSET 1 ON
2030 RETURN
```

COMSET ON

Purpose

Empties the buffer and turns on background data reception on the specified communication channel.

Syntax

COMSET<*nexp*>ON

Parameters

<nexp>
Describes one of the following communication channels:

1: "uart1:"

2: "uart2:"

3: "uart3:"

4: "centronics:"

5: "net1:"

6: "usb1:"

7: "uart4:"

8: "uart5:"

9: "usbhost:"

10: "bluetooth:"

11: "ftp1:"

12: "http1:"

13: "lpr1:"

Notes

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

Use COMSET ON to open any of the communication channels for background data reception with an empty buffer, provided the communication parameter for the channel has already been set up by a COMSET statement.

When reception is interrupted by the reception of an end character, an attention string or the maximum number of characters, issue a new COMSET ON to empty the buffer and reopen reception.

Example

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

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

COMSTAT

Purpose

Reads the status of a communication channel buffer.

Syntax

COMSTAT(*<nexp>*)

Parameters

<nexp>
Describes one of the following communication channels:

1: "uart1:"

2: "uart2:"

3: "uart3:"

4: "centronics:"

5: "net1:"

6: "usb1:" (Hardware handshake bit supported when virtual COM port is enabled)

7: "uart4:"

8: "uart5:"

9: "usbhost:"

10: "bluetooth:"

11: "ftp1:"

12: "http1:"

13: "lpr1:"

Notes

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

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

Value	Condition
0 or 1	Copy of hardware handshake bit. Not valid on "net1:" or "usbhost:".
2	Interruption: Maximum characters received.
4	Interruption: End character received.
8	Interruption: Attention string received.
32	Interruption: Communication error. Not valid on "net1:" or "usbhost:".

Example

In this example, a message appears on the screen when the reception is interrupted by any of four specified COMSET conditions:

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

CONT

Purpose

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

Syntax

CONT

Notes

When you use a CONT statement to resume program execution after a STOP, BREAK, or DBBREAK, execution continues at the point where the break occurred with the STDIO settings restored.

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

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

Example

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

PRINT A%
100
Ok

PRINT B%
50
Ok
B%=100
Ok

CONT
Equal
Ok
```

COPY

Purpose

Copies files.

Syntax

COPY<*sexp1*>[,<*sexp2*>]

Parameters

<*sexp1*>
Name and optional directory of the original file.

<*sexp2*>
(Optional) New name and/or directory for the copy.

Notes

Directory names are case sensitive. You must first set SYSVAR(43) to 1 before running this command in order for the printer to recognize the directory you specify.

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

If no directory is specified for the original and/or copy, the current directory is used by default. By default, the current directory is "/c", which is the printer permanent memory. If the file is to be copied from or to another directory than the current one, the file name must contain a directory reference. A file cannot be copied to the same name in the same directory.

In addition to copying files to the printer permanent or temporary memory (or a USB storage device), a file can also be copied to an output device such as the printer's screen or a serial communication channel.

Copying a program to the standard OUT channel has the same effect as LOADing and LISTing it.

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

In the next examples, "/c" is the current directory.

Example 1

Copy a file from "d:" to the current directory without changing the file name:

```
COPY "d:LABEL1.PRG"
```

Example 2

Copy a file from "d:" to the current directory and changing the file name:

COPY "d:FILELIST.PRG","COPYTEST.PRG"

Example 3

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

COPY "/c/FILELIST.PRG","d:FILELIST.PRG"

Example 4

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

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

COUNT&

Purpose

Creates a counter (Direct Protocol only).

Syntax

COUNT& <*sexp1*>,<*nexp1*>,<*sexp2*>

Parameters

<*sexp1*>
Specifies the type of counter parameter to be set:

START
Start value. Decides the first value to be printed:

If a single letter is entered (A to Z), the counter becomes an alpha counter. Default is A for alpha counters.

If one or several digits are entered the counter is numeric. Numeric values can be positive or negative. Negative values are indicated by a leading minus sign. Default is 1 for numeric counters.

WIDTH
Minimum number of digits to be printed. Used only in numeric counters. If the counter value contains a lesser number of digits, leading zero (0) characters are added until the specified number of digits is obtained. If the number of digits in the counter value is equal to or larger than specified in the width parameter, the value is printed in its entirety. Default is 1 (no leading zeros).

COPY
Determines how many copies (of labels, graphics, or other components) will be printed before the counter is updated according to the INC parameter. Default is 1.

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

STOP
Sets the value after which the counter should begin again at the value specified by RESTART.

If a single letter is entered (A to Z), the counter becomes an alpha counter, and if one or several digits are entered the counter is numeric. When a counter is decremented, a stop value less than the start value must be given, since the default stop value will never be reached otherwise. Default is 2,147,483,647 (numeric) or Z (alpha).

RESTART
Sets the value at which the counter begins again after exceeding the STOP value. If a single letter is entered (A to Z), the counter becomes an alpha counter, and if one or several digits are entered the counter is numeric. Default is 1 (numeric) or A (alpha).

<*nexp1*>
Specifies the counter reference number (integers only).

<sexp2>
Parameter value.

Notes

This instruction can only be used in the Direct Protocol.

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

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

The value of the start, stop, and restart parameters decide the type of counter (alpha or numeric). If different types of counters are specified in these parameters, the last entered parameter decides the type. Alpha counters count A to Z. Numeric counters use numbers with no practical limit.

Counters are not saved in the printer memory, but are recreated after each power up. Therefore, you may need to save the COUNT& statements as a file in the host.

Example

This example creates a counter that starts at 100 and is updated by a value of 50 after every second label until the value 1000 is reached. Then the counter starts again at the value 200. All values are expressed as 4-digit numbers with leading zeros.

```
COUNT& "START",1,"100"

COUNT& "WIDTH",1,"4"

COUNT& "COPY",1,"2"

COUNT& "INC",1,"50"

COUNT& "STOP",1,"1000"

COUNT& "RESTART",1,"200"
```

CSUM

Purpose

Calculates the checksum of an array of strings.

Syntax

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

Parameters

<*ncon*>
Specifies the type of checksum calculation:

1: Longitudinal Redundancy Check (LRC). XOR in each character in each string array [0][0] xor array[0][1] ... array[n][n]
2: Diagonal Redundancy Check (DRC). Right rotation, then XOR on each character in each string rot(array[0][0] xor array[0][1]
3: Longitudinal Redundancy Check (LRC). Strip string of DLE (0x10) before doing the LRC

<*svar*>
If <*ncon*> = 1 or 2: The array of strings of which the checksum is to be calculated.

If <*ncon*> = 3: Checksum string.

<*nvar*>
The variable in which the result is presented.

Notes

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

Example

This example calculates the DRC checksum of an array of strings:

```
10 ARRAY$(0)="ALPHA"

20 ARRAY$(1)="BETA"

30 ARRAY$(2)="GAMMA"

40 ARRAY$(3)="DELTA"

50 CSUM 2,ARRAY$,B%

60 PRINT B% :REM DRC CHECKSUM

RUN
```

This results in:

```
252
```

CURDIR$

Purpose

Returns the current directory as the printer stores it.

Syntax

CURDIR$

Notes

Directory names are case sensitive. You must first set SYSVAR(43) to 1 before running this command in order for the printer to recognize the directory you inquire about.

Example

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

This results in:

/c/DIR1/DIR2

CUT

Purpose

Activates the label cutter if a cutter is installed.

You must first set the CUT ON/OFF command to OFF before you can use this command. Sending the CUT OFF command allows you to cut labels manually. If your printer is not set to cut manually, labels are cut immediately after they are printed.

Syntax

CUT

Notes

This statement only works with printers fitted with a cutter, which cuts non-adhesive paper strip or through the liner between self-adhesive labels. You must use a PRINTFEED command before using the CUT statement.

Example

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

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

CUT ON/OFF

Purpose

Enables or disables automatic cutting after PRINTFEED execution. Optionally, CUT ON/OFF can adjust the media feed before and after the cutting.

Syntax

CUT [<*nexp*>] ON|CUT OFF

Parameters

<*nexp*>
(Optional) Sets the length of media to be fed out before cutting and pulled back after cutting. Default is CUT OFF.

Notes

This statement makes it possible to enable or disable automatic execution of a CUT operation directly after the execution of each PRINTFEED statement. This statement only works with printers fitted with a cutter, which cuts non-adhesive paper strip or through the liner between self-adhesive labels.

If any extra media feed in connection with the cutting operation is required, use start adjust and stop adjust setup, or specify the desired length of media to be fed out before the cutting is performed and pulled back afterwards in the CUT ON statement.

The amount of media feed (<*nexp*>) is not automatically reset to 0 by a CUT OFF statement. You must manually specify 0 to reset the media feed amount to 0 (for example, CUT 0 ON:CUT OFF). However, restarting the printer resets this media feed distance to 0.

Example

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

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

DATE$

Purpose

Variable for setting or returning the current date.

Syntax 1

Setting the date:

DATE$=<*sexp*>

Parameters 1

<*sexp*>
Sets the current date by a 6-digit number specifying the year, month, and day in the format YYMMDD.

Syntax 2

Returning the date:

<*svar*>=DATE$[(<*sexp*>)]

Parameters 2

<*svar*>
Returns the current date according to the printer's calendar.

<*sexp*>
(Optional) Flag "F", indicating that the date should be returned as specified by FORMAT DATE$.

Notes

This variable works best if a real-time clock circuit (RTC) is fitted on the printer CPU board. Honeywell mobile printers do not have an RTC. The RTC has a battery backup circuit and keeps record of the time if the power is turned off or lost.

If no RTC is installed, the internal clock is used. After startup, an error occurs when trying to read the date or time before the internal clock has been manually set by means of either a DATE$ or a TIME$ variable. If only the date is set, the internal clock starts at 00:00:00 and if only the time is set, the internal clock starts at Jan 01, 1980.

After setting the internal clock, you can use the DATE$ and TIME$ variables the same way as when an RTC is included, until a power off or REBOOT causes the date and time values to be lost.

By default, date is always entered and returned in the format YYMMDD, where:

- YY = Last two digits of the year (for example, 2007 = 07).
- MM = Two digits representing the month. Range is 01 to 12.
- DD = Two digits representing the day. Range is 01 to 31.

Example: December 1, 2007 is entered as "071201".

The built-in calendar corrects incorrect values for the years 1980-2048 (for example, the incorrect date 031232 is corrected to 040101). The format for how the printer returns dates can be changed by means of FORMAT DATE$ and returned by DATE$("F").

Example

This example sets and returns the date in two different formats:

```
10 DATE$ = "071201" : 'sets date

20 FORMAT DATE$ "DD/MM/YYYY" : 'sets date format

30 PRINT DATE$ : 'returns  unformatted date

40 PRINT DATE$("F") : 'returns formatted date

RUN
```

This results in:

```
071201

01/12/2007
```

DATEADD$

Purpose

Returns a new date after a number of days have been added to or subtracted from the current date or a specified date.

Syntax

DATEADD$([<*sexp1*>,]<*nexp*>[,<*sexp2*>])

Parameters

<*sexp1*>
Any date given according to the DATE$ format.

<*nexp*>
Number of days to be added to or subtracted from the current date, (or, optionally, the date specified by <*sexp1*>).

<*sexp2*>
(Optional) One or two flags, "F" or "M":

"F" specifies that the date will be returned according to the format specified by FORMAT DATE$.
"M" specifies the value given in <*nexp*> indicates months instead of days.

Notes

This variable works best if a real-time clock circuit (RTC) is fitted on the printer's CPU board. Honeywell mobile printers do not have an RTC.

The original date (<*sexp1*>) should be entered according to the syntax for the DATE$ variable, that is in the order YYMMDD, where:

- YY = Last two digits of the year (for example, 2007 = 07).
- MM = Two digits representing the month. Range is 01 to 12.
- DD = Two digits representing the day. Range is 01 to 31.

For example, December 1, 2007 is entered as "071201".

The built-in calendar corrects incorrect values for the years 1980-2048 (for example, the incorrect date 031232 is corrected to 040101).

Specify the number of days to be added or subtracted as a positive or negative numeric expression.

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

"F" and "M" flags can be combined in the same instruction without any separating character, that is, "FM" or "MF". No other characters are accepted.

Example 1

```
10 DATE$ = "071201"
20 A%=15
30 B%=-10
40 FORMAT DATE$ "DD/MM/YY"
50 PRINT DATEADD$("071201",A%)
60 PRINT DATEADD$("071201",A%,"F")
70 PRINT DATEADD$("071201",B%,"F")
RUN
```

This results in:

```
071216
16/12/07
21/11/07
```

Example 2

```
10 DATE$="080131"
20 FORMAT DATE$ "YYYY/MM/DD"
30 ? DATEADD$(1,"F"),DATEADD$(1,"M"),DATEADD$(1,"FM")
RUN
```

This results in:

```
2008/02/01 080229 2008/02/29
```

Example 3

```
? DATEADD$("080131",1,"F"), DATEADD$("080131",2,"M"),DATEADD$("080131",3,"FM")
```

This results in:

```
2008/02/01 2008/03/31 2008/04/30
```

DATEDIFF

Purpose

Returns the difference between two dates as a number of days.

Syntax

DATEDIFF(<*sexp1*>,<*sexp2*>)

Parameters

<*sexp1*>
Specifies one of two dates (date 1).

<*sexp2*>
Specifies the other of two dates (date 2).

Notes

This variable works best if a real-time clock circuit (RTC) is fitted on the printer's CPU board. Honeywell mobile printers do not have an RTC.

To get the result as a positive numeric value, the two dates should be entered with the earlier of the dates (date 1) first and the later of the dates (date 2) last. See the first example below.

If the later date (date 2) is entered first, the resulting value will be negative, as seen in the second example.

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

- YY = Last two digits of the year (for example, 2007 = 07).
- MM = Two digits representing the month. Range is 01 to 12.
- DD = Two digits representing the day. Range is 01 to 31.

For example, December 1, 2007 is entered as "071201".

The built-in calendar corrects incorrect values for the years 1980-2048 (for example, the incorrect date 071232 is corrected to 080101).

Example 1

This example calculates the difference in days between October 1, 2007 and November 30, 2007:

```
10 A%=DATEDIFF("071001","071130")
20 PRINT A%
RUN
```

This results in:

```
60
```

Example 2

This example demonstrates what happens if the later date is entered first:

```
10 A%=DATEDIFF("071130","071001")
20 PRINT A%
RUN
```

This results in:

```
-60
```

DBBREAK

Purpose

Adds or deletes a breakpoint for the Fingerprint debugger.

Syntax

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

Parameters

<*nexp*>
Line number where the debugger breaks, and serves as the name of the breakpoint.

<*sexp*>
Line label where the debugger breaks, and serves as the name of the breakpoint.

ON adds the specified breakpoint (default).
OFF deletes the specified breakpoint.

Notes

Program execution breaks at each program line specified as a breakpoint, and the message "break in line *nnn*" is transmitted on the Debug STDOUT port. If a CONT statement is issued, the execution continues at the next line. If RUN is issued, the execution starts again from the first program line.

The line number or line label does not to have to exist when a breakpoint is added, but if a nonexistent breakpoint is deleted an error 39 or 70 occurs.

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

When a NEW statement is issued, all breakpoints are deleted. If a breakpoint is set on a line with a call to a FOR or WHILE loop, there is only one break on that line the first time it is executed.

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

Example

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

This results in:

```
A
Break in line 20
```

DBBREAK OFF

Purpose

Deletes all breakpoints for the Fingerprint debugger.

Syntax

DBBREAK OFF

Notes

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

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

DBEND

Purpose

Terminates the Fingerprint debugger.

Syntax

DBEND

Notes

This statement terminates the Fingerprint debugger immediately and restores the previous STDIO settings.

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

DBSTDIO

Purpose

Selects the standard IN/OUT channel for the Fingerprint debugger.

Syntax

DBSTDIO *<nexp1>*,*<nexp2>*[,*<sexp1>*,*<sexp2>*]
DBSTDIO [*<nexp1>*,*<nexp2>*,]*<sexp1>*,*<sexp2>*

Parameters

<nexp1>
Desired Debug STDIN channel:

0: "console:"
1: "uart1:" (default)
2: "uart2:"
3: "uart3:"
4: "centronics:"
5: "net1:"
6: "usb1:"
7: "uart4:"
8: "uart5:"
9: "usbhost:"
10: "bluetooth:"

<nexp2>
Desired Debug STDOUT channel:

0: "console:"
1: "uart1:" (default)
2: "uart2:"
3: "uart3:"
5: "net1:"
6: "usb1:"
7: "uart4:"
8: "uart5:"
9: "usbhost:"

<sexp1>
Preamble (default: empty string)

<sexp2>
Postamble (default: empty string)

Notes

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

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

Example

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

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

DBSTEP

Purpose

Specifies the interval between breaks for the Fingerprint debugger and executes the program accordingly.

Syntax

DBSTEP<*ncon*>

Parameters

<*ncon*>
Number of lines to be executed before break. Default is 1 line.

Notes

If <*ncon*> is omitted, one line is executed, but if <*ncon*> = 0, nothing happens.

DBSTEP cannot be used in execution mode, which results in error 78. When DBSTEP is used on the last line in a program, the line is executed but there is no break.

If DBSTEP is used in a program with a FOR or WHILE loop, there is only one break on the line that calls for the FOR or WHILE loop the first time it is executed.

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

Example

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

DELETE

Purpose

Deletes one or several consecutive program lines from the printer working memory.

Syntax

DELETE<*ncon1*>[-<*ncon2*>]

Parameters

<*ncon1*>
Line, or the first line in a range of lines, to be deleted.

<*ncon2*>
(Optional) Last line in a range of program lines to be deleted.

Notes

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

Examples

DELETE 50 deletes line 50 from the program.

DELETE 50–100 deletes line 50 thru 100 from the program.

DELETE 50– deletes all lines from line 50 to the end of the program.

DELETE –50 deletes all lines from the start of the program to line 50.

DELETEPFSVAR

Purpose

Deletes variables saved at power failure.

Syntax

DELETEPFSVAR<*sexp*>

Parameters

<*sexp*>
Name of the variable to be deleted.

Notes

Related instructions are SETPFSVAR, GETPFSVAR, and LISTPFSVAR.

Examples

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

DEVICES

Purpose

Returns the names of all devices on the standard OUT channel.

Syntax

DEVICES

Notes

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

Device	Explanation	Can be opened for
c: (= "/c/")	The printer permanent read/write memory (Flash SIMMs). It supports file systems with directories, and retains its content when the power is switched off. For backwards compatibility, "ram:" is also accepted.	Input/Output/Random
d: (= "/d/")	Depending on the device, d: can be an external memory unit such as a USB storage device or a partition of the permanent read/write memory. It supports file systems with directories, and retains its content when the power is switched off.	Input/Output/Random
centronics:	The Centronics parallel port.	Input
console:	Printer screen and/or keyboard. The keyboard can be used for input only and the screen for output only. External keyboards are not supported.	Input/Output
dll:	Special applications only.	-
finisher:	The device controlling the finisher interface, where a device such as a cutter can be connected.	Input/Output
ftp1:	FTP printing port.	Input
http1:	HTTP printing port.	Input/Output

Device	Explanation	Can be opened for
lpr1:	LPR printing port.	Input
net1:	Communication channel for the EasyLAN interface board.	Input/Output
par:	Special applications only.	-
rom: (="/rom/")	An alias for tmp:.	Input
tmp:	The printer temporary read/write memory (SDRAM SIMMs). It will lose its content when the power is turned off or at a power failure. Valuable data that cannot be recreated should be copied to "/c/". One advantage of using "tmp:" instead of "/c" is that data can be written to SDRAM faster than to the flash memory. To speed up operation, the Honeywell Fingerprint firmware (except program modules with dynamic downloading) is copied from "/rom/" to "tmp:" at startup and used from "tmp:".	Input/Output/Random
uart1:	Standard RS-232 port for serial communication.	Input/Output
uart2:	Additional serial port on an optional interface board.	Input/Output
uart3:	Additional serial port on an optional interface board.	Input/Output
uart4:	Additional serial port on an optional interface board.	Input/Output
uart5:	Additional serial port on an optional interface board.	Input/Output
usb1:	USB device port.	Input/Output
usbhost:	USB host port.	Input/Output

DIM

Purpose

Specifies the dimensions of an array.

Syntax

DIM<<*nvar*>|<*svar*>>(<*nexp1*>[,<*nexp2*>...])....
[,<<*nvar*>|<*svar*>>(<*nexp1*>[,<*nexp2*>...])]

Parameters

<nvar>|<svar>
Name of the array.

<nexp1>
Maximum subscript value for the first dimension.

<nexp2-10>
(Optional) Maximum subscript values for the following dimensions (2 through 10).

Notes

An array is created by entering a variable followed by a number of subscripts (maximum of 10) separated by commas. All subscripts are enclosed by parentheses and each subscript represents a dimension. The first time an array is referred to, the number of subscripts in the array variable determines its number of dimensions regardless of line number. By default, the number of elements in each dimension is restricted to four (numbered 0 to 3).

Note that 0 = 1st element, 1 = 2nd element, and so on. If more than 4 elements in any dimension are desired, a DIM statement must be issued. For example, ARRAY$(1,2,3) creates a three-dimensional array, where the dimensions each contain 4 elements (0 to 3) respectively. This corresponds to the statement DIM ARRAY$(3,3,3).

It is not possible to change the number of dimensions of an array that already has been created during runtime, and error 57 ("Subscript out of range") occurs in this case.

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

Example 1

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

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

Example 2

In this example, two one-dimensional arrays are created on the same program line:

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

This results in:

```
PRINTER $1995
```

DIR

Purpose

Specifies the print direction.

Syntax

DIR*<nexp>*

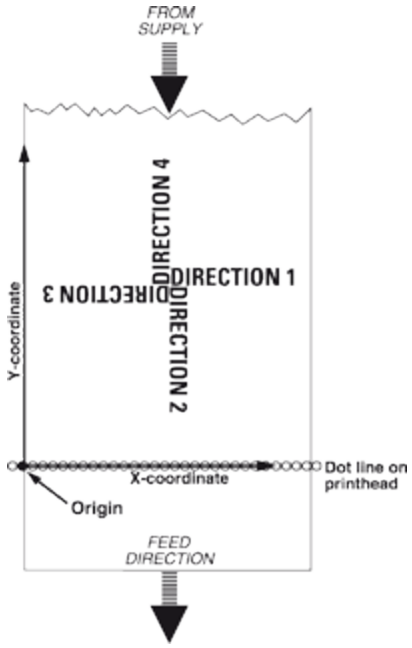
Parameters

<nexp>
Print direction (1, 2, 3, or 4). Default is 1.

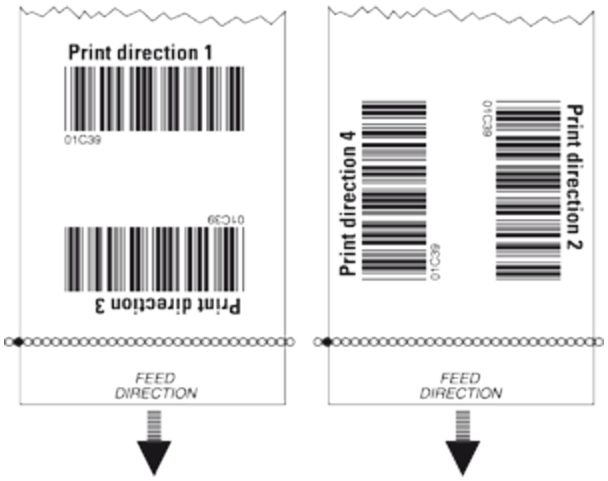
Notes

A change of print direction affects all printing statements (PRTXT, PRBAR, PRIMAGE, PRBOX, PRDIAGONAL, or PRLINE) that are executed later in the program until a new DIR statement or a PRINTFEED statement is executed. Use PRINTFEED to reset to default values.

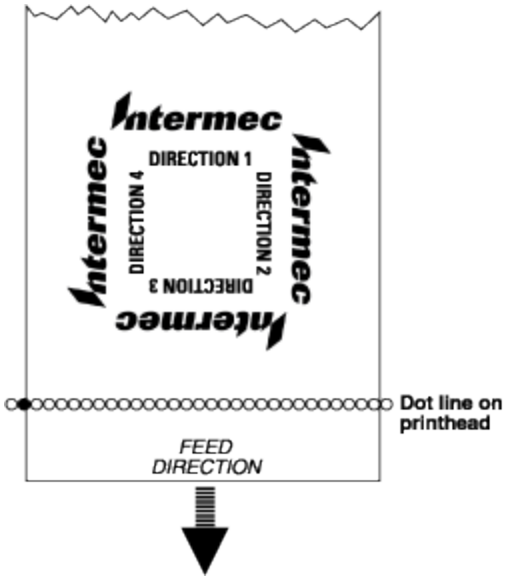
Print Direction for Text Fields



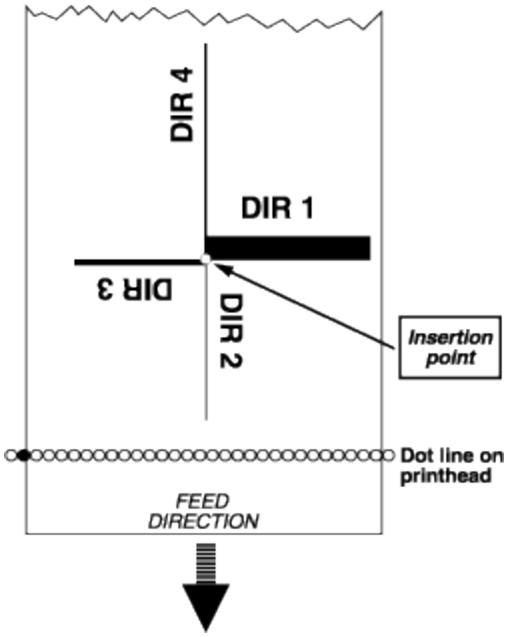
Print Direction for Bar Code Fields



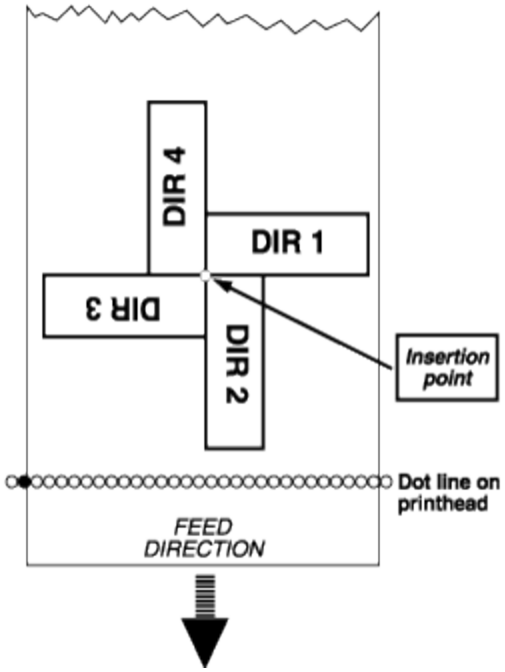
Print Direction for Images



Print Direction for Lines



Print Direction for Boxes and Diagonal Lines



Example 1

This example prints a label with one line of text and draws a line beneath the text:

```
10 PRPOS 30,300
20 DIR 1
30 ALIGN 4
40 FONT "Univers",18
```

```
50 PRTXT "TEXT PRINTING"
60 PRPOS 30,280
70 PRLINE 555,10
80 PRINTFEED
RUN
```

Example 2

This example is similar but prints the same information vertically, requiring new positioning to avoid Error 1003, "Field out of label."

```
10 PRPOS 300,30
20 DIR 4
30 ALIGN 4
40 FONT "Univers",18
50 PRTXT "TEXT PRINTING"
60 PRPOS 320,30
70 PRLINE 555,10
80 PRINTFEED
RUN
```

DIRNAME$

Purpose

Returns the names of the directories stored in the specified part of the printer memory.

Syntax

DIRNAME$[(<*sexp*>)]

Parameters

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

Notes

In <*sexp*>, parts of directory names and wildcards (*) are allowed. If <*sexp*> is omitted, the next directory name in the same memory device is listed.

DIRNAME$ can be repeated. When there are no directories left to list, the output string is empty. For more information, see FILENAME$.

Example

```
10 PRINT "Subdirectories in /c"
20 qTemp$ = DIRNAME$("/c/*")
30 WHILE LEN (qTemp$) > 0
40 PRINT qTemp$
50 qTemp$ = DIRNAME$
60 WEND
```

DISPLAY IMAGE

Purpose

Specifies a custom image that appears on the printer's screen when a specific error occurs.

Syntax

DISPLAY IMAGE <*nexp*>

Parameters

<*nexp*>
Image number, which corresponds to an image file in the printer memory. For more information, see the Notes. These numbers are preassigned to existing images:

0: Remove the graphic on the display.
1: Generic error.
2: Field out of label.
3: Out of paper.
4: Head lifted.
5: Out of transfer ribbon.
6: Next label not found.
7: Printhead too hot.
8: Testfeed not done.
9: Cutter device not found, or cutter not responding.
39: Ignored unless user downloaded (due to legacy).
40: Maintenance.
41: IP link error.
42: Press feed not done.
43: Label not taken.

If the image number specified for DISPLAY IMAGE is not in the correct location, an "Image not found" error is raised. For more information on the image and image number, see the Notes.

Notes

You can override the default error images by using your own custom images and specifying a default image number. For compatibility with future versions of Fingerprint, Honeywell recommends that you use image numbers starting with 100. Image numbers used in legacy versions of Fingerprint are supported, but those images are no longer installed by default. Recommended values for n are from 100 to 1024.

The graphic for your custom image must be in PNG format and stored in the /home/user/display/ directory. The filename must match the image number you want to use. The file name must also be in the form of "image_x.png" where x is the number used as <*nexp*>. For example if you had a file named display image_124.png, you could send the command DISPLAY IMAGE 124.

If the number you choose is the same as an existing image number, Fingerprint uses your image instead of the default image.

DISPLAY KEY

Purpose

Specifies a custom key image to be shown on the printer screen. These images are used for the printer soft keys.

Syntax

DISPLAY KEY <*nexp1*>, <*nexp2*>

Parameters

<*nexp1*>
Soft key pictogram to be changed. Range is 1 to 5, where 1 corresponds to the F1 (leftmost) key on the printer display.

<*nexp2*>
Pictogram number, which must correspond to a file in the DISPLAY directory or to one of the following preassigned pictogram numbers:

0: Clear the pictogram area for the key specified by <*nexp1*>.
1: F1 key
2: F2 key
3: F3 key
4: F4 key
5: F5 key
6-10: Ignored unless user downloaded (due to legacy)
11: Left arrow
12: Up arrow
13: Right arrow
14: Down arrow
15: Enter key
16: Left arrow (pressed state)
17: Up arrow (pressed state)
18: Right arrow (pressed state)
19: Down arrow (pressed state)
20: Enter key (pressed state)

For more information on the pictogram and pictogram number, see the Notes.

Notes

You cannot override the preassigned values 0 through 5, but other values can be overridden. For compatibility with future versions of Fingerprint, Honeywell recommends that you use image numbers starting with 100. Image numbers used in legacy versions of Fingerprint are supported, but those images are no longer installed by default. Recommended values for n are from 100 to 1024.

The pictogram specified by <*nexp2*> appears until another key is pressed or another DISPLAY KEY command is executed.

The graphic for your custom image must be in PNG format and stored in the /home/user/display/ directory. The file name must match the image number you want

to use. The file name must also be in the form "funckey_x.png" where x is the number used as <*nexp2*>. For example if you had a file named funckey_121.png, you could send the command DISPLAY KEY 1, 121.

If the number you choose is the same as an existing image number, Fingerprint uses your image instead of the default image.

For PC Series printers, pictograms are automatically resized to be 60 pixels wide by 80 pixels tall. For PM43, PM43c, and PM23c printers, pictograms are autosized to be 42 pixels wide by 41 pixels tall.

DISPLAY STATE

Purpose

Specifies a custom background image that appears on the printer's screen when the printer is running your application.

The PC-Series printer does not support DISPLAY STATE. There is insufficient room on the display between the two-line text display and the function key areas for a meaningful graphic to display. This command will not cause an error, but it will be ignored.

Syntax

DISPLAY STATE <*nexp*>

Parameters

<*nexp*>
State image number corresponding to an image file in the printer memory that will appear as the background.

0: Remove background (default background).
1: Setup mode.
2: Pause mode.
3: Information mode.
4-6: Ignored unless user downloaded (due to legacy).
7: Upgrading mode.
8: Idle mode.
9: Printing mode.

Notes

You can override the default images by using your own custom images and specifying an image number. For compatibility with future versions of Fingerprint, Honeywell recommends that you use image numbers starting with 100. Image numbers used in legacy versions of Fingerprint are supported, but those images are no longer installed by default. Recommended values for n are from 100 to 1024.

The graphic for your custom image must be in PNG format and stored in the /home/user/display/ directory. The filename must match the image number you want to use: The file name must also be in the form of "background_x.png" where x is the number used as <*nexp1*>. For example if you had a file named background_124.png, you could send the command DISPLAY STATE 124.

If the number you choose is the same as an existing image number, Fingerprint uses your image instead of the default image.

END

Purpose

Ends the execution of the current program or subroutine and closes all files devices opened using the OPEN command.

Syntax

END

Notes

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

Example

A part of a program which produces fixed line-spacing might look like this:

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

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

EOF

Purpose

Checks for an end-of-file condition.

Syntax

EOF(*<nexp>*)

Parameters

<nexp>
Number assigned to the file when it is opened using the OPEN command.

Notes

The EOF function can be used with files opened for sequential input (in connection with the statements INPUT#, LINE INPUT#, and INPUT$) to avoid the error condition "Input past end" which has no error message.

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

Example

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

This results in:

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

ERL

Purpose

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

Syntax

ERL

Notes

Also useful in connection with an ON ERROR GOTO statement.

Example 1

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

PRINT ERL

A typical result might look like the following:

40

Example 2

The line number of the line where an error has occurred determines the action to be taken. In this case the font size is too large for the label width:

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

This results in:

PRINT ERROR

Example 3

You can use ERL in programs without line numbers also, since these programs automatically generate hidden line numbers that are revealed when the program is LISTed. This is the same program as above but without line numbers:

```
NEW
IMMEDIATE OFF
ON ERROR GOTO QAAA
FONT "Univers",100
PRTXT "HELLO EVERYBODY"
PRINTFEED
END
QAAA: IF ERL=40 THEN PRINT "PRINT ERROR"
```

```
RESUME NEXT
IMMEDIATE ON
RUN
```

This results in:

```
PRINT ERROR
```

ERR

Purpose

Returns the code number of an error.

Syntax

ERR

Notes

Fingerprint can detect a number of error conditions, represented by code numbers (for help, see Table of Error Codes). The ERR function enables the program to read the coded error number. You can design your program to take proper action depending on which type of error has occurred.

Example 1

In this example, the error code determines the action to be taken:

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

Example 2

This example shows the number of the last error since power up:

```
PRINT ERR
```

This results in:

```
1022
```

ERR$

Purpose

Returns the explanation of an error code in plain text.

Syntax

ERR$(<*nexp*>)

Parameters

<*nexp*>
Error code number

Notes

The explanation of the error is returned in English. For a complete list of error codes, see Table of Error Codes.

Example

PRINT ERR$(1003)

This results in:

Field out of label

ERROR

Purpose

Defines error messages and enables error handling for specified error conditions (Direct Protocol only).

Syntax

ERROR *<nexp1>*[<[,*<sexp>*[,*<nexp2>*]]>|,*<nexp3>*]

Parameters

<nexp1>
Number of the error condition.

<sexp>
Desired error message.

<nexp2>
Number of the image to be shown on the printer's screen (along with the specified error message) when the error occurs. If the specified image does not exist, image number 1 is shown. For more information on error images, see DISPLAY IMAGE.

<nexp3>
Number of the image to be shown on the printer display (without an error message) when the error occurs. If the specified image does not exist, image number 1 is shown. For more information on error images, see DISPLAY IMAGE.

Notes

The ERROR statement can only be used in the Direct Protocol for the purpose of enabling error-handling and creating customized error messages, as described below. The built-in error-handler of the Direct Protocol will always handle these standard errors:

Number	Error Condition	Message displayed on printer
15	Font not found	Font not found
18	Disk full	Disk full
37	Printer cannot find cutter	Cutter device not found
43	Memory is overflowing	Memory overflow
1003	Attempting to print to field which extends out of label	Field out of label
1005	Printer is out of paper	Out of paper

Number	Error Condition	Message displayed on printer
1006	No field to print	No field[s]
1022	Printhead lifted	Head lifted
1027	Out of thermal transfer ribbon	Out of ribbon
1031	Next label not found	Label not found
1058	Transfer ribbon is installed	Ribbon installed
1606	Testfeed not done	Testfeed not done

Other errors are not handled unless they have been specified by an ERROR statement.

The ERROR statement also allows you to edit a suitable message in any language. This message appears in the printer display if the error occurs. The error message is truncated to 33 characters. Characters 1 through 16 appear on the upper line and characters 18 through 33 appear on the lower line. Character 17 is always ignored.

An empty string removes any previously entered message for the error in question. Likewise, an existing message can be replaced by a new one. When a standard error or an error defined by an ERROR statement is detected, the printer sets its standard IN port to BUSY, sets the Data transfer LED to on, and displays the error messages. When you press the Print key, the error message is cleared, the LED is turned off, and the standard IN port is set to READY. In some cases, the error itself must be cleared (for example, by loading more labels).

Error messages are not saved in printer memory, but new ERROR statements must be downloaded after each power up. Honeywell recommends that you save a set of ERROR statements as a file in the host computer.

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

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

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

Example

In these examples, a few errors are specified.

Note: Note the blank spaces for character position 17 in each message (space characters are indicated by double headed arrows):

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

EXECUTE

Purpose

Executes a Fingerprint program line or a file with Fingerprint program lines from within another Fingerprint program.

Syntax

EXECUTE*<sexp>*

Parameters

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

Notes

\Directory names are case sensitive. You must first set SYSVAR(43) to 1 before running this command in order for the printer to recognize the directory you specify.

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

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

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

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

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

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

Example

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

```
IMMEDIATE OFF
DIR 1
ALIGN 7
BARSET "CODE39",2,1,3,120
BARFONT "Univers",10,8,5,1,1
BARFONT ON
IMMEDIATE ON
SAVE "tmp:BARCODE.PRG",L
NEW
10 PRPOS 30,400
```

```
20 EXECUTE "tmp:BARCODE.PRG"
30 PRBAR "ABC"
40 PRINTFEED
RUN
```

External Command: ZMODEM

Purpose

External commands for receiving and sending data using the ZMODEM protocol.

Syntax 1

RUN "rz [<*switches*>] [<*filename*>]" (receive data)

Parameters

<*switches*>

-c	Forces no crash recovery, even if sender requests ZCRESUM (resume interrupted file transfer).
-e	Prints last error to standard OUT channel.
-l	[<logfile>] Sends verbose output to logfile. Default logfile name is "tmp:. zmodemlog".
-r	If ZMCLOB is not set and the file already exists, replaces file if the transfer is successful.
-v	[<level>] Sets verbosity level. Level is a decimal number. Default level is 1.
-u	Translates file name to uppercase. If a filename is given as parameter, no translation is done.

<*filename*>
(Optional) The name to which the file will be saved.

Syntax 2

RUN "sz [<*switches*>] [<*filename*>]" (send data)

Parameters

<switches>

-l	[<logfile>] Sends verbose output to logfile. Default logfile name is "tmp:.z-modemlog".
-v	[<level>] Sets verbosity level. Level is a decimal number. Default level is 1.

<filename>
The name of the file.

Notes

Note that rz and sz must be entered in lowercase characters.

If a file name is given in the rz statement, this name overrides the name given by the transmitting unit.

For more information on the ZMODEM protocol, please refer to http://www.omen.com. Related instruction is TRANSFER ZMODEM.

FIELD

Purpose

Creates a single-record buffer for a random file and divides the buffer into fields to which string variables are assigned.

Syntax

FIELD[*#*]<*nexp1*>,<*nexp2*>AS<*svar1*>[,<*nexp3*>AS<*svar2*>...]

Parameters

#
(Optional) Indicates that whatever follows is a number.

<*nexp1*>
Number assigned to the file when it is opened using the OPEN command.

<*nexp2-n*>
Number of bytes to be reserved for the string variable that follows (Null not allowed).

<*svar1-n*>
Designation of the string variable, for which space has been reserved.

Notes

The buffer is divided into fields, each of which is given an individual length in bytes. A string variable is assigned to each field. This statement only creates and formats the buffer, allowing you to place the data using LSET and RSET statements.

Before using this statement, consider the maximum number of characters (including space characters) needed for each variable. Make sure the total does not exceed the record size given when the file was opened (default is 128 bytes). When a file is CLOSEd, all its FIELD definitions are lost.

Example

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

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

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

FIELDNO

Purpose

Gets the current field number for partial clearing of the print buffer by CLL.

Syntax

FIELDNO

Remarks

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

Example

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

prints three labels:

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

FILE & LOAD

Purpose

Receives and stores binary files in the printer memory.

Syntax

FILE& LOAD[<*nexp1*>,]<*sexp*>,<*nexp2*>[,<*nexp3*>]

Parameters

<*nexp1*>
(Optional) Number of bytes to skip before starting to read the file data.

<*sexp*>
Name for the file when stored in the printer memory. Maximum length is 30 characters including any extension.

<*nexp2*>
Size of the file in bytes.

<*nexp3*>
(Optional) Specifies a communication channel for opened for INPUT by the number assigned to the device. Default is standard IN channel.

Notes

Directory names are case sensitive. You must first set SYSVAR(43) to 1 before running this command in order for the printer to recognize the directory you specify.

This statement prepares the printer to receive a binary file on the standard IN channel (see SETSTDIO statement) or on another communication channel for INPUT using the OPEN command. Image files and font files can also be downloaded using the IMAGE LOAD statement.

Unlike the IMAGE LOAD statement, FILE& LOAD does not immediately install the fonts, but the font files remain in the printer memory until next power-up. The optional first parameter makes it possible to use this statement in MSDOS (CR/LF problem).

The name of the file, when stored in the printer memory, may consist of up to 30 characters, including possible extension. The size of the original file should be given in bytes according to its size in the host.

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

Example

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

FILENAME$

Purpose

Returns the names of the files stored in the specified part of the printer memory.

Syntax

FILENAME$[(<*sexp*>)]

Parameters

<*sexp*>
Name of the memory device from which the first file name (in alphabetical order) will be listed. Parts of file names and wildcards (*) are allowed. Maximum size is 30 characters.

If <*sexp*> is omitted, the next file name in the same device is listed. This function can be repeated. When there are no files left to list, the output string is empty.

Notes

Directory names are case sensitive. You must first set SYSVAR(43) to 1 before running this command in order for the printer to recognize the directory you specify.

Specified memory device must be mounted. The file name is case-sensitive and must correspond to the name of the file stored in the memory device. Wildcards (* = ASCII 42 decimal) can be used. The list may include all types of files. Even system file names preceded by a period character (for example, .FONTALIAS) may be listed.

No directories are listed and the order of listing is not specified. To list directories, see DIRNAME$

Example

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

```
10 PRINT "Files in /c"
20 qTemp$ = FILENAME$("/c")
30 WHILE LEN(qTemp$) > 0
40 PRINT qTemp$
50 qTemp$ = FILENAME$
60 WEND
RUN
```

FILES

Purpose

Lists the contents of a printer directory to the standard OUT channel.

Syntax

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

Parameters

<*sexp*>
(Optional) Specifies the directory. For more information, see DEVICES.

R
Lists directories recursively.

A
Lists all files including system files (files with a name starting with a period (.) character).

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

If no directory is specified, files in the current directory (default is "/c") are listed. To change the current directory, see CHDIR

By including a reference to a memory device (such as "/c", "d:", or "tmp:"), the files of the specified directory are returned without changing the current directory.

If the "A" flag is omitted, all files, except system files, are listed. The flags A and R can be entered in any order, but R is always processed first. The number of bytes for each file and the total number of free and used bytes in the specified directory will also be included in the list.

Examples

The presentation may look like this on the host screen:

FILES "/c",

Files on /c
STDIO 2 FILE2 4
DIR1/ 0

Files on /c/DIR1/
FILE1 4 DIR2/ 0
STDIO 2
No files on /c/DIR1/DIR2
4121600 bytes free 12 bytes used

FILES,R,A

```
Files on /c
./ 0 ../ 0
DIR1/ 0 FILE2 4
STDIO 2 .setup.saved 239
Files on /c/DIR1/
./ 0 ../ 0
DIR2/ 0 STDIO 2

FILES

Files on /c/DIR1/DIR2/
./ 0 ../ 0
4121600 bytes free 251 bytes used
FILES "/c/DIR1"
Files on /c/DIR1
FILE1 4 DIR2/ 0
STDIO 2
4121600 bytes free 6 bytes used
```

FLOATCALC$

Purpose

Calculates with floating point numbers.

Syntax

FLOATCALC$(*<sexp1>*,*<sexp2>*,*<sexp3>*[,*<nexp1>*])

Parameters

<sexp1>
First operand.

<sexp2>
The operator (+, -, *, or /).

<sexp3>
Second operand.

<nexp1>
(Optional) Precision in decimals (default 10).

Notes

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

Note the mathematical rules:

- - yields +

- + yields -

+ - yields -

+ + yields +

The following arithmetic operators are allowed:

+ (addition)	ASCII 043 decimal
- (subtraction)	ASCII 045 decimal
* (multiplication)	ASCII 042 decimal
/ (division)	ASCII 047 decimal

Other operators or characters generate an error.

The optional precision parameter specifies the number of decimals in the result of the calculation. The result is truncated accordingly. For example, if the number of decimals is specified as 5, the result 5.76123999 is presented as 5.76123.

The result of a FLOATCALC$ function can be formatted using a FORMAT$ function.

Examples

Addition:

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

This results in:

```
1235.90000
```

Subtraction:

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

This results in:

```
134.88
```

FONT

Purpose

Selects a scalable TrueType, OpenType, or bitmap font for printing subsequent PRTXT statements.

This command can be abbreviated as FT.

Syntax

FONT<*sexp1*>[,<*nexp1*>[,<*nexp2*>[,<*nexp3*>]]]

or

FT<*sexp1*>[,<*nexp1*>[,<*nexp2*>[,<*nexp3*>]]]

Parameters

<*sexp1*>
Font name. Default is Univers.

<*nexp1*>
(Optional) Font height in points. Default is 12. Use MAG to enlarge bitmap fonts.

<*nexp2*>
Clockwise slant in degrees. Range is 0 (default) to 90. Does not work with bitmap fonts.

<*nexp3*>
Width enlargement in percent relative to height. Range is 1 to 1000. Default is 100. Does not work with bitmap fonts.

Reset to default by executing a PRINTFEED.

Font Support

Fingerprint supports scalable fonts that comply with the Unicode standard. Fingerprint also 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. For more information, see Complex Scripts.

Use a FONTS statement to list the names of all fonts installed in your own printer to the standard OUT channel. For a list of fonts and font aliases, use a RUN command.

To maintain compatibility with programs created in earlier versions of Fingerprint, you can also specify bitmap font names, such as "SW030RSN" or "MS060BMN.2". For a standard bitmap font name, the firmware selects the corresponding scalable font in the printer's memory and sets font parameters so its direction, appearance, and size come as close to the specified bitmap font as possible. The standard complement of outline fonts must already be in the printer memory. For more information on legacy bitmap font support, see Default Fonts.

Nonstandard bitmap fonts can also be used. They will retain the bitmap format but will not produce any outline fonts. Any extension to the bitmap font name is ignored.

Slant Effects

Slanting means that you can create the same effect as in ITALIC characters. Higher values increase the amount of clockwise slant. Slanting cannot be used with bitmap fonts.

Examples:

Font Sizes

The height of the font is given in points (1 point = 1/72 inch, approximately 0.35 mm), which means that text is printed in the same size regardless of the printhead density of the printer. Sizes less than 4 points will be unreadable.

In case of bitmap fonts, use MAG to enlarge the font instead of specifying a font height. Any font may be magnified up to 4 times separately in regard of height and width using a MAG statement. Bitmap fonts will get somewhat jagged edges when magnified, whereas outline fonts will remain smooth. For more information, see MAG.

A scalable font can enlarged in terms of width relative to height. The value is given as a percentage (range is 1 to 1000). This means that if the value is 100, there is no change in the appearance of the characters. A value of 50 reduces the font width by half, and a value of 200 doubles the width.

Examples

This example prints one line of 12p text with the default direction and alignment:

```
10 FONT "Univers"
20 PRTXT "HELLO"
30 PRINTFEED
RUN
```

This example prints the same text but at 24p, with 20° slant, and with 75% width:

```
10 FONT "Univers",24,20,75
20 PRTXT "HELLO"
30 PRINTFEED
RUN
```

This example prints the complex script using UTF-8:
(The MheiC-Medium-Big5HKSCS font is not installed by default. Install the font before using this example, or use a different font that includes Chinese characters.)

```
10 FONT "MheiC-Medium-Big5HKSCS"
20 NASC "UTF-8"
30 PRTXT
CHR$(230);CHR$(136);CHR$(145);CHR$(232);CHR$(131);CHR$(189);CHR$(229);CHR
$(144);CHR$(158);CHR$(228);CHR$(184);CHR$(139)
40 PRINTFEED
```

FONTD

Purpose

Obsolete but retained for compatibility. See FONT (FT).

FONTNAME$

Purpose

Returns the names of the fonts stored in the printer memory, not including aliases.

Syntax

FONTNAME$(*<nexp>*)

Parameters

<nexp>
Specify 0 to return the first font name in memory, or specify non-zero value to return the next font name in memory. Can be repeated as long as there are font names left.

If there are no more font names in memory, this returns an empty string.

Example

This example lists all font names:

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

FONTS

Purpose

Returns the names of all fonts stored in the printer memory to the standard OUT channel.

Syntax

FONTS

Notes

This command does not list font aliases. To list all of the fonts and font aliases installed on the printer, use the following RUN command:

RUN "ls -l /printer/fonts"

FOR...TO...NEXT

Purpose

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

Syntax

FOR<*nvar*>=<*nexp1*>TO<*nexp2*>[STEP<*nexp3*>]NEXT[<*nvar*>]

Parameters

<*nvar*>
Variable to be used as a counter.

<*nexp1*>
Initial value of the counter.

<*nexp2*>
Final value of the counter.

<*nexp3*>
Value of the increment or decrement.

Notes

The FOR statement is always used with a NEXT statement.

The counter <*nvar*> is given an initial value by the numeric expression <*nexp1*>. If no increment value (STEP <*nexp3*>) is given, the value 1 is assumed. A negative increment value produces a decremental loop.

Each time the statement NEXT is encountered, the loop is executed again until the final value (specified by <*nexp2*>), is reached. Then the execution proceeds from the first line after the NEXT statement.

If the optional variable is omitted in the NEXT statement, the program execution loops back to the most recently encountered FOR statement.

If the NEXT statement includes a variable, the execution loops back to the FOR statement specified by the same variable.

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

Example 1

In this example, the counter A% is incremented from 10 to 50 in steps of 20:

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

This results in:

```
10
30
50
```

Example 2

In this example, the counter B% is decremented from 50 to 10 in steps of 20:

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

This results in:

```
50
30
10
```

Example 3

In this example, the counters A% and B% are used in a nested loop.

```
10 FOR A%=0 TO 3 STEP 1
20 FOR B%=0 TO 2 STEP 1
30 PRINT A%,B%
40 NEXT B%
50 NEXT A%
RUN
```

This results in:

```
0       0
0       1
0       2
1       0
1       1
1       2
2       0
2       1
2       2
3       0
3       1
3       2
```

FORMAT

Purpose

Formats the printer permanent memory.

Syntax

FORMAT<*sexp*>[,A]

Parameters

<sexp>
Specifies the device to be formatted.

A
Removes hidden and system files.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

Be careful when using FORMAT "/c" to format the printer permanent memory. System files use a leading period character (for example, .setup.saved). To format the permanent memory without erasing the system files (soft formatting), do not include an "A" flag in the statement. Your user permissions determine which files and directories are erased.

If an "A" flag is included in the statement, all files are removed, but the printer is not reset to factory default settings. To reset the printer to factory default settings, use the printer web page.

You can use the FORMAT command to erase all of the files on a USB storage device attached to d:, but you cannot change its file system parameters.

Examples

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

FILES "/c",A

results in:

Files on /c			
./	0	../	0
APPLICATION	1	boot/	0
ADMIN/	0	.setup.saved	222
STDIO	4		
2222080 bytes free		227 bytes used	
Ok			
FORMAT "/c"			
Ok			

FILES "/c",A

results in:

Files on /c			
./	0	../	0
boot/	1	ADMIN	0
.setup.saved	222		
222412 bytes free		222 bytes used	

FORMAT DATE$

Purpose

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

Syntax

FORMAT DATE$*<sexp>*

Parameters

<sexp>
String representing the year, month and date plus possible separating characters. Default format is YYMMDD, where:

YY = Last two digits of the year (for example, 2007 = 07).
MM = Two digits representing the month. Range is 01 to 12.
DD = Two digits representing the day. Range is 01 to 31.

Example: December 1, 2007 is entered as "071201".

Notes

Reset to default by specifying an empty string (" ")

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

In the FORMAT DATE$ statement, each Y, M or D character generates one digit from the number of the year, month, or day respectively. If the number of Ys exceeds 4, or the number of Ms or Ds exceeds 2, the extra characters generate leading space characters.

The Y, M, and D characters return information from right to left.

Examples for the year 2007:

Y generates 7
YY generates 07
YYY generates 007
YYYY generates 2007
YYYYY generates .2007 (. represents a space)

Characters other than Y, M, or D are treated as separators and are returned as entered.

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

Examples

This example changes the date format to British standard:

FORMAT DATE$ "DD/MM/YY"

This changes the date format back to default (YYMMDD):

```
FORMAT DATE$ ""
```

This changes the date format to Swedish standard:

```
FORMAT DATE$ "YY-MM-DD"
```

FORMAT INPUT

Purpose

Specifies delimiting strings used by the LAYOUT RUN statement.

Syntax

FORMAT INPUT<*sexp1*>[,<*sexp2*>[,<*sexp3*>[,<*sexp4*>]]]

Parameters

<*sexp1*>
Start-of-record string (default is STX, ASCII 02 decimal).

<*sexp2*>
End-of-record string (default is EOT, ASCII 04 decimal).

<*sexp3*>
End-of-field string (default is CR, ASCII 13 decimal).

<*sexp4*>
String of characters to be filtered out.

Notes

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

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

Optionally, it is possible to specify a string (up to 10 characters) to be filtered out. By default, the string is empty and is reset to default if a new FORMAT INPUT with less than four arguments is issued.

There is a timeout if the end-of-field string is not found within 60 seconds after STX has been received.

Always execute FORMAT INPUT in the Immediate mode.

If you are using the Direct Protocol, exit it using an INPUT OFF statement before changing the delimiting strings using a FORMAT INPUT statement. Then you can enter the Direct Protocol again using an INPUT ON statement.

An error occurs if you issue a FORMAT INPUT statement where one, two or three delimiting strings are identical to those already in effect without leaving the Direct Protocol.

If a certain separating character cannot be produced by the keyboard of the host, use a

CHR$ function to specify the character by its ASCII value. The delimiting strings are stored in the temporary memory and must be transmitted to the printer after each power-up.

Example

This example changes the start-of-text string to #, the end-of-text separator to LF (linefeed), and the end-of-field string to @ after temporarily switching to Immediate mode.

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

FORMAT TIME$

Purpose

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

Syntax

FORMAT TIME$<*sexp*>

Parameters

<*sexp*>
String format for reporting time, where:

H represents hours in a 24 hour cycle (one digit per H).
h represents hours in a 12 hour cycle (one digit per h).
M represents minutes (one digit per M).
S represents seconds (one digit per S).
P represents AM/PM in connection with a 12 hour cycle.
p represents am/pm in connection with a 12 hour cycle.

All other characters produce separator characters. Default string is HHMMSS. Reset to default by sending an empty string.

Notes

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

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

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

h generates 8 P generates A
hh generates 08 PP generates AM
hhh generates ␣08 p generates a
pp generates am

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

Separating characters are returned as entered in the string. Any character other than H, h, M, S, P, or p is regarded as a separator character.

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

Examples

Changing the time format according to Swedish standard:

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

Changing the date format to British standard:

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

FORMAT$

Purpose

Formats a number represented by a string.

Syntax 1

FORMAT$("<*sexp1*>","<*nexp1*>b"|"d")

Parameters 1

<sexp1>
String of integers or ASCII characters.

<nexp1>
Number of bytes to output in integer to ASCII conversion.

b
Specifies conversion from integer to ASCII format.

d
Specifies conversion from ASCII format to integer.

Syntax 2

FORMAT$(*<sexp1>*,*<sexp2>*)

Parameters 2

<sexp1>
String of digits which may contain a period (.) to separate integers from decimals. A leading plus (+) or minus (-) is allowed.

<sexp2>
Specifies the format of the string.

Notes

When used as in Syntax 1, FORMAT$ converts strings from ASCII to integers and vice versa.

In the command FORMAT$("<*sexp1*>","<*nexp*>b"), b signifies that the integer string *<sexp1>* is converted to the corresponding characters in ASCII format. *<nexp1>* specifies the number of bytes to output. For example, FORMAT$("1380337988","4b") results in "RFID".

In the command FORMAT$("<*sexp1*>","d"), the d signifies that the string *<sexp1>* is converted from an ASCII string to the corresponding number in integer format. For example, FORMAT$("A","d") results in "65".

With Syntax 2, FORMAT$ can also be used to convert *<sexp1>* to a specific display format. The format is specified by the string *<sexp2>* and can contain any characters, but some have special significance as described in the next table:

Character	Description
0	Digit placeholder. Displays a digit or zero. • If the input number has fewer digits than there are zeros (on either side of the decimal separator) in the format string, leading or trailing zeros are displayed. • If the number has more digits to the left side of the decimal separator than there are zeros to the left side of the separator in the format string, the digits are displayed. • If the number has more digits to the right of the decimal separator than there are zeros to the right of the separator in the format string, the decimal is truncated to as many decimal places as there are zeros.
#	Digit placeholder. Displays a digit or nothing. • If there is a digit in the position where the # appears in the format string, display the digit or otherwise display nothing in that position. • If the number has more digits to the left side of the decimal separator than there are # to the left side of the separator in the format string, the digits are displayed.
.	Decimal separator between the integer and decimal digits.
,	Decimal separator between the integer and decimal digits.
\	Display the next character in the format string. The backslash itself is not displayed. Use two backslashes (\\) to print a single backslash. Only the space character is displayed in the formatted string without a backslash.
Space	Space, displayed as a literal character wherever it is in the expression format.

Other rules to remember when using FORMAT$:

- An empty format string is equivalent to "0.##########".
- 0 and # cannot be mixed in every way. Before the decimal separator, use # first and then 0. After the decimal separator, use 0 first and then #. For example: ####00.000### is OK but #00##0.##0#00 is not.
- A period (.) or a comma (,) separate integers and decimals. The decimal separator used in the format is the returned separator type. Independent of the separator type in the number, the format type controls the return type. Default type is a period.
- You can use space characters as separators between thousands or from a unit such as dollars ($), as in this example: "$ ### ### 000.00".
- The attached number string is truncated to the decimal quantity of the format.
- Characters are not displayed on the left side of the decimal separator if there is a # on the left side of the characters, and the string to be formatted does not have a digit

in the same position as the #. On the right side of the decimal separator, characters are not displayed if there is a # on the right side of the characters, and the string to be formatted does not have a digit in the same position as the #.

General Examples

This table shows the results of different format strings when applied to various input strings, using Syntax 2.

Input String (<*sexp1*>)	Format String (<*sexp2*>)	FORMAT$(<*sexp1*>,<*sexp2*>)
1.1 55 0.33 55.33	"\$#\t\e\x\t0.0\t\e\x\t#\$"	1.1 $5text5.0$ $0.3-text3$ $5text5.3text3$
"5" "-5" "0.5" "55555" "0.666666666666"	" "	5 -5 0.5 55555 0.6666666666
"5" "-5" "0.5" "55555" "0.666666666666"	"0"	5 -5 0 55555 0
"5" "-5" "0.5" "55555" "0.666666666666"	"0.00"	5.00 -5.00 0.50 55555.00 0.66
"5" "-5" "0.5" "55555" "0.666666666666"	"\$0,0"	$5,0 $-5,0 $0.5 $55555,0 $0,6
"5" "-5" "0.5" "55555" "0.666666666666"	"0.0##"	5.0 -5.0 0.5 55555.0 0.666
"5" "-5" "0.5" "55555" "0.666666666666"	"###\,000.0"	005.0 -005.0 000.5 55,555.0 000.6
"5" "-5" "0.5" "55555" "0.666666666666"	"# 0 0.0"	0 5.0 -0 5.0 0 0.5 555 5 5.0 0 0.6

Addition Example

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

This results in:

"1235.900000000000000"

Subtraction Examples

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

This results in:

"134.88"

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

This results in:

"$ 134,880"

If a higher precision is used in FLOATCALC$, A$ yields "$134,887".

Multiplication Example

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

This results in:

"300.0"

```
C$="0 0 0,00###"
```

PRINT FORMAT$(A$,C$)

This results in:

"3 0 0,00"

Division Example

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

This results in:

"0.3333333333"

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

This results in:

"$ 000.33333"

FORMFEED

Purpose

Feeds out or pulls back a specified length of media.

This command can be abbreviated as FF.

Syntax

FORMFEED[<*nexp*>]

or

FF[<*nexp*>]

Parameters

<*nexp*>
(optional) Feed length expressed as a positive or negative number of dots.

Notes

If <*nexp*> is not specified, the printer feeds out one single label, ticket, tag, or portion of continuous stock according to the printer setup. For more information, see the printer user's guide.

If <*nexp*> is specified, the media is fed out or pulled back by the corresponding number of dots:

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

It is important whether FORMFEED is issued before or after a PRINTFEED statement:

- FORMFEED issued before PRINTFEED affects the position of the origin on the first copy to be printed.
- FORMFEED issued after PRINTFEED does not affect the position of the origin on the first copy, but the next copy will be affected. Do not use FORMFEED as a replacement for start- and stop adjustments in the Setup mode or in connection with batch printing.

Example 1

This example prints a line of text and feeds out an extra 60 dots of media after printing:

```
10 FONT "Univers"
20 PRPOS 30,200
30 PRTXT "HELLO"
40 PRINTFEED
```

```
50 FORMFEED 60
RUN
```

Example 2

This example pulls back the media 20 dots before printing:

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

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

FRE

Purpose

Returns the number of free bytes in a specified part of the printer memory.

Syntax

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

Parameters

<sexp>
Designation of the part of the printer memory from which the number of free bytes should be returned, for example "/c", "d:", or "tmp:".

<nexp>
Dummy argument that returns the number of free bytes in the printer temporary memory ("tmp:").

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

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

If the name of a device that is not a part of the printer memory (for example "console:"), is entered as an argument, FRE returns 0. See DEVICES for more information on memory and non-memory devices.

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

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

Examples

Input	Results in
PRINT FRE("tmp:")	2382384
PRINT FRE("con-sole:")	0
PRINT FRE(1)	2382384

FUNCTEST

Purpose

Performs various hardware tests.

Syntax

FUNCTEST<*sexp*>,<*svar*>

Parameters

<*sexp*>
Type of test to be performed:

<sexp>	Response	Description
"HEAD"	HEAD OK, SIZE:n DOTS	Test was successful and *n* is the number of printhead dots.
	HEAD LIFTED	Printhead is lifted and must be lowered before test can be performed.
	FAULTY PRINTHEAD	One or more dots on the printhead are not working. Printhead voltage is not checked. Use HEAD for additional tests.

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

Notes

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

Example

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

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

This results in:

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

FUNCTEST$

Purpose

Returns the result of various hardware tests.

Syntax

FUNCTEST$(*<sexp>*)

Parameters

<sexp>
Type of test to be performed:

<sexp>	Response	Description
"HEAD"	HEAD OK, SIZE:n DOTS	Test was successful and *n* is the number of printhead dots.
	HEAD LIFTED	Printhead is lifted and must be lowered before test can be performed.
	FAULTY PRINTHEAD	One or more dots on the printhead are not working. Printhead voltage is not checked. Use HEAD for additional tests.

Notes

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

Example

This example shows a test program using FUNCTEST$ (compare with the FUNCTEST example):

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

This results in:

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

GET

Purpose

Reads a record from a random file to a random buffer.

Syntax

GET[#]<*nexp1*>,<*nexp2*>

Parameters

#
(Optional) Indicates that whatever follows is a number.

<*nexp1*>
Number assigned to the file when it is opened using the OPEN command.

<*nexp2*>
Number of the record. Must be a positive integer (1 or greater).

Notes

The GET statement is used to read a specified record in a random file to a buffer, where the record is assigned to variables based on the FIELD statement given for the buffer. After the GET statement is executed, you can use references to the variables defined by the FIELD statement to read the characters in the random buffer.

Numeric expressions converted to string expressions (by STR$ functions before being put into the buffer) can be converted back to numeric expressions using VAL functions.

Example

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

This results in:

SMITH — — — JOHN — — — — — — 12345630

GETASSOC$

Purpose

Gets a value from a string association.

Syntax

GETASSOC$ (*<sexp1>*, *<sexp2>*)

Parameters

<sexp1>
Name of the association (case-sensitive).

<sexp2>
Name of a tuple in the association.

Notes

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

Example

This example defines a string, including three string names associated with three start values, and changes one of them (time):

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

This results in:

```
time=UNKNOWN LABEL=321 DESCRIPTION=DEF
time=153355
```

GETASSOCNAME$

Purpose

Traverses the tuples of a string association.

Syntax

GETASSOCNAME$(<*sexp*>,<*nexp*>)

Parameters

<*sexp*>
Association to be traversed (case-sensitive).

<*nexp*>
Specifies the tuple in the association:

<*nvar*> = 0 (zero) specifies the first tuple.
<*nvar*> = non-zero specifies the next tuple. This number can be a positive or negative integer.

Notes

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

Example

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

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

This results in:

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

GETPFSVAR

Purpose

Recovers saved variables.

Syntax

GETPFSVAR(*<sexp>*[,D])

Parameters

<sexp>
Name of the variable (uppercase characters only).

D
(Optional) Specifies that the variable is to be deleted after recovery.

Notes

Use GETPFSVAR to recover variables registered to be saved at power failure by means of a SETPFSVAR statement. The function returns -1 on success or 0 at failure.

If a D flag is included, the variable is deleted after it has been recovered. Use this to make sure that the variable is up-to-date. The variable name is limited to 20 characters.

Related instructions are SETPFSVAR, DELETEPFSVAR, and LISTPFSVAR.

Example

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

GOSUB

Purpose

Branches to a subroutine.

Syntax

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

Parameters

<*ncon*>
Number of the first line in the desired subroutine.

<*line label*>
Label of the first line in the desired subroutine.

Notes

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

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

Subroutines may be nested, which means that a subroutine may contain a GOSUB statement for branching to a secondary subroutine and so on. Subroutines are normally placed on program lines with higher numbers than the main program. The main program should be appended by an END statement to avoid unintentional execution of subroutines.

Example 1

This example makes use of line numbers:

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

This results in:

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

Example 2

This example has the same result as Example 1, but instead of line numbers, the program uses line labels to make the program branch to subroutines:

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

GOTO

Purpose

Branches unconditionally to a specified line number or line label.

Syntax

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

Parameters

<ncon>
Number of the line to be branched to.

<line label>
Label of the line to be branched to.

Notes

If the specified line contains an executable statement, both that statement and all that follows are executed. If the specified line does not exist, an error condition occurs.

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

Example

In this example the first bar of the tune "Colonel Bogey" is played only if the title is entered correctly. The message "Try again" is displayed until you type the right name.

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

This results in:

TITLE

The way GOTO is used in line 50 to create a loop, which makes the printer await the condition specified in line 40 before the execution is resumed. Instead of line numbers,

line labels can be used following the same principles as illustrated in the second example for the GOSUB statement.

HEAD

Purpose

Returns the result of a printhead resistance check.

Syntax

HEAD(<*nexp1*>)

Or

<*nexp2*> = HEAD(<*sexp*>)

Parameters

<*nexp1*>
> 0: Specifies the number of a dot for which the resistance (in ohms) is returned. A dot resistance value that deviates considerably from the mean resistance value of the printhead indicates that the dot may be faulty. The dot numbering starts at 0 (zero), so for example in an 832-dot printhead the dots are numbered 0 to 831.

-1: Printhead check. Returns -1 (true) if no dot is more than ±15% from the mean resistance value, or 0 (false) otherwise.

-7: Returns mean printhead resistance in ohms. PD4X printers return the nominal resistance value of the dots.

<*nexp2*>
Returns the total number of faulty dots.

<*sexp*>
Returns the dot number and resistance for each faulty dot.

Notes

There is no guarantee that all defect "dots" will be detected by HEAD, since only the resistance is checked. For example, dirty or cracked dots can only be detected visually.

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

Although this command may indicate a faulty printhead, the printhead may not actually be defective even if the resistance measurement is not ±15% of the mean resistance value.

Example 1

To read the resistance value of dot No. 5:

PRINT HEAD(5)

This results in (for example):

603

Example 2

To perform a printhead check:

PRINT HEAD(-1)

If the head condition is within normal ranges, this results in:

-1

Example 3

To read the mean resistance value of the printhead:

PRINT HEAD(-7)

In this example, the mean resistance value returned is:

613

Example 4

To check the printhead for faulty dots and their respective resistance values:

A%=HEAD(B$)

In this example, there are 5 potentially faulty dots. Note that the printer returns an Ok message before the primary output:

Ok

PRINT A%
5
Ok

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

IF...THEN...(ELSE)

Purpose

Specifies conditional execution controlled by the result of a numeric expression.

Syntax

(Periods are used here to indicate concatenation.)

IF<*nexp*>[,]THEN<*stmt1*>[ELSE<*stmt2*>]

IF<*nexp*>[,]THEN .

<*stmt1*> .

[...<*stmt1+n*>] .

[ELSE .

<*stmt2*> .

[...<*stmt2+n*>]] .

ENDIF

Parameters

<*nexp*>
Numeric expression to be evaluated.

<*stmt1*>
Statement or list of statements telling the program what to do should the IF-condition be true.

<*stmt2*>
Optional statement or list of statements specifying what happens should the IF-condition be false.

Notes

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

Example 1

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

This results in:

A LARGER THAN B

Example 2

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

This results in:

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

Example 3

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

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

This results in:

```
NOT EQUAL
```

Example 4

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

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

This results in:

```
NOT EQUAL
```

IMAGE BUFFER MIRROR

Purpose

Mirrors the print image around the Y-axis.

Syntax

IMAGE BUFFER MIRROR

Notes

This statement mirrors the current defined image buffer around the Y-axis (media feed direction). Fields defined after execution of IMAGE BUFFER MIRROR are rendered normally. The image buffer width is always 8-bit aligned, even when the X-start parameter in the setup is not. Because of this, Honeywell recommends that you test that the mirrored image is printed correctly where intended. In some cases, a small correction using the PRPOS statement or the X-start parameter may be necessary.

Example

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

This results in:

IMAGE BUFFER SAVE

Purpose

Saves image buffer content as a file.

Syntax

IMAGE BUFFER SAVE*<sexp>*

Parameters

<sexp>
Desired file name, with an optional reference to the device where the file should be saved.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

This statement saves the current content of the print buffer as an image file in RLL format. After saving the file, it is automatically installed as an image which can be printed using a PRIMAGE statement in DIR 1 and DIR 3. You can now create label templates and easily add variable data.

The size of the print buffer image depends on the size of the print image at the moment the buffer is saved. The width is determined by the Media, Media Size, Width setup value, with the first pixel according to the Media, Media Size, Xstart setup value. The height is determined by the actual height in the y-dimension of the print image. Note that space characters or invisible "white" parts of an image are included in the height of the print image, even if they are not visible on the printed label.

Example

IMAGE BUFFER SAVE "TEMPLATE7"

IMAGE LOAD

Purpose

Receives, converts, and installs image and font files.

Syntax

IMAGE LOAD[<*nexp1*>,]<*sexp1*>,<*nexp2*>[,<*sexp2*>[,<*nexp3*>]]

Parameters

<*nexp1*>
(Optional) Number of bytes to skip before starting to read the data.

<*sexp1*>
Desired name of the image or font to be created.

<*nexp2*>
Size of the original file in number of bytes.

<*sexp2*>
Optional flag: "S" specifies that the image or font will be saved in the printer's permanent memory ("/c"). This option may result in poor performance. An empty string ("") specifies that the image or font will be stored in the printer's temporary memory ("tmp:").

<*nexp3*>
(Optional) Specifies a communication channel opened for INPUT by the number assigned to the device. Default is standard IN channel.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

IMAGE LOAD prepares the printer to receive an image file or a font file on the standard IN channel or on another communication channel opened for INPUT. When the file is received, it is automatically converted to an image in Fingerprint bitmap format (if necessary) or to a scalable font. Fingerprint supports black and white (no grayscale) images in PNG, PCX, GIF, and BMP formats.

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

The name of an image may consist of a maximum of 30 characters including possible extensions. The image will have the same direction as the original image file and can only be rotated 180° using DIR. Honeywell recommends that you include the extension .1 or .2 to indicate the intended print direction. Font file names are restricted to only 30 characters. The size of the original file should be given in bytes according to its size in the host.

Before IMAGE LOAD can be used on a serial channel, the setup must be changed to 8 characters, CTS/RTS handshake. When IMAGE LOAD is executed, the execution stops and waits to receive the number of bytes specified. During the transfer of image file data to the printer, there is a 25-second timeout between characters.

If a new character is not received within the timeout limit, Error 80 "Download timeout" occurs. When the specified number of characters is received, the execution resumes. If the download was successful, the downloaded image or font is installed automatically and can be used without restarting the printer.

Example

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

IMAGENAME$

Purpose

Returns the names of the images stored in the printer memory.

Syntax

IMAGENAME$(<*nexp*>)

Parameters

<*nexp*>
Specify 0 to return the name of the first image stored in memory. Specify a non-zero number to return the name of the next image stored in memory.

Notes

This function can be used to produce a list of all images. You can also use the IMAGES statement.

Image files downloaded by means of a TRANSFER KERMIT statement are not returned, since the software regards them as files rather than images.

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

IMAGENAME$(_,0) produces the next name, and can be repeated as long there are any image names left. When no more image files are found, IMAGENAME$ returns an empty string.

Example

Use a program like this to list all image names:

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

A typical result looks like:

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

IMAGES

Purpose

Returns the names of all images stored in the printer memory to the standard OUT channel.

Syntax

IMAGES

Notes

IMAGES (or IMAGENAME$) can be used to list all image names. Image files downloaded by means of a TRANSFER KERMIT statement are not printed, since the firmware regards them as files instead of images.

Example

Send this command to see a list of images stored in the printer memory:

IMAGES

A typical result might be:

CHESS2X2.1 CHESS4X4.1
DIAMONDS.1 GLOBE.1
3568692 bytes free 1717812 bytes used
Ok

IMMEDIATE

Purpose

Enables or disables Immediate mode in connection with program editing without line numbers, for reading the current mode, or for reading the current standard IN and OUT channels.

Syntax

IMMEDIATE ON|OFF|MODE|STDIO

Parameters

ON
Enables Immediate mode.

OFF
Disables Immediate mode.

MODE
Prints a line to the STDOUT port with information on the current status of the following modes (ON or OFF):

Execution
Immediate
Input
Layout Input
Debug STDIO (dbstdio)

STDIO
Prints two lines to the STDOUT port with information on current settings for the STDIN and STDOUT channels.

Notes

Before you begin writing a program without line numbers, disable Immediate mode by sending IMMEDIATE OFF. If not, each line will be executed immediately. After an IMMEDIATE OFF statement, you can enter program lines without leading line numbers. References to lines are done using "line labels," which are called in GOTO or GOSUB and related statements.

A line label is a name followed by a colon (:). The label must not interfere with any standard Fingerprint keywords or start with a digit, and the line must start with the line label. When a line label is used as a reference to another line (for example, within a GOTO statement), omit the colon.

The program should be appended by a IMMEDIATE ON statement. At the execution of this statement, the program lines are numbered automatically in ten-step incremental order, starting with the first line (10- 20-30-40-50 and so on). These line numbers will not appear on the screen until the program is LISTed, LOADed, or MERGEd. Line labels are not converted to line numbers.

Do not issue RUN before sending IMMEDIATE ON, or an error occurs.

When you send IMMEDIATE MODE, the printer returns information on mode status as follows:

- Execution On/Off indicates if a Fingerprint program is running or not.
- Immediate On/Off indicates whether the Immediate mode is enabled or disabled as specified by IMMEDIATE ON/OFF.
- Input On/Off indicates whether Direct Protocol is enabled or disabled as specified by INPUT ON/OFF.
- Layout Input On/Off indicates whether or not a layout is being recorded in Direct Protocol as specified by LAYOUT INPUT and LAYOUT END.
- Dbstdio On/Off indicates whether the debug standard I/O is active or not.

The following conditions are not reported:

- Running a Fingerprint application.
- Execution of a TRANSFER KERMIT, FILE& LOAD, IMAGE LOAD, LOAD, and STORE INPUT instruction.
- Running external commands (ush), for example RUN "z....."
- Running the Setup mode or execution of a SETUP statement.

When you send IMMEDIATE STDIO, two lines are transmitted on the STDOUT port with connection information (port, baud rate, character length, parity, and stop bits) for the current STDIN and STDOUT channels.

Example 1

A program can be written without line numbers. First send the following:

IMMEDIATE OFF

results in:

Ok

Then the following program is written without line numbers. QQQ is used as a line label:

```
PRINT "LINE 1"
GOSUB QQQ
END
QQQ: PRINT "LINE 2"
RETURN
IMMEDIATE ON
Ok
RUN
```

This results in:

```
LINE 1
LINE 2
Ok
```

Example 2

This example shows how to check mode status:

IMMEDIATE MODE

This results in:

execution=OFF, immediate=ON, input=OFF, layout input = Off

Example 3

This example shows how to check STDIN and STDOUT channel status:

IMMEDIATE STDIO

This results in:

stdin=uart1:, 9600, 8, NONE, 1
stdout=uart1:, 9600, 8, NONE, 1

INKEY$

Purpose

Reads the first character in the receive buffer of the standard IN channel.

Syntax

INKEY$

Notes

For information on standard I/O channels, see SETSTDIO.

As opposed to INPUT, INKEY$ does not interrupt the program flow to wait for input data, unless a loop is created by means of a GOTO statement as seen in line 20 in the example.

INKEY$ is useful when the host computer is unable to end the input data with a "Carriage Return" (CR; ASCII 13 decimal), but must use some other character, for example "End of Text" (ETX; ASCII 3 decimal). Then a routine interpreting the substitute character as a carriage return can be created.

Example

In this example, none of the characters received on the standard IN channel are printed on the host screen until a # character (ASCII 35 decimal) is encountered.

```
10 A$ = INKEY$
20 IF A$ = "" GOTO 10
30 IF A$ = CHR$(35) THEN PRINT B$
40 IF A$ = CHR$(35) THEN END
50 B$ = B$ + A$
60 GOTO 10
RUN
```

Type a number of characters on the host keyboard. They are not printed on the host screen until you type #. Then all the characters appear simultaneously, except for the #-sign.

Note the loop between line 10 and 20, which makes the program wait for you to activate a key.

INPUT

Purpose

Receives input data via the standard IN channel during the execution of a program.

This command can be abbreviated as IP.

Syntax

INPUT[*<scon><;|,>*]*<<nvar>|<svar>*>[,*<<nvar>|<svar>*>...]

or

IP[*<scon><;|,>*]*<<nvar>|<svar>*>[,*<<nvar>|<svar>*>...]

Parameters

<scon><;|,>
(Optional) Prompt string, followed by a semicolon or comma.

<<nvar>|<svar>
Variables to which the input data will be assigned.

Notes

For information on standard I/O channel, see SETSTDIO.

During program execution, INPUT interrupts the execution. A question mark or a prompt appears on the host screen, indicating that the program is expecting additional data to be entered. The prompt can be used to tell the operator what type of data to enter.

The prompt is appended by a question mark if a semicolon (;) is entered after the prompt string. If a comma (,) is used in that position, the printing of the question mark is suppressed.

If a prompt is not used, the question mark is always displayed. Do not enter a comma or semicolon directly after the keyword, but only after the prompt, or to separate variables.

The input data should be assigned to one or several variables. Each item of data should be separated from the next item by a comma. The number of data items entered must correspond to the number of variables in the list, or an error condition occurs. The variables may be any mix of string variables and numeric variables, but the type of input data must agree with the type of the variable to which the data is assigned.

Input can also be done directly to the system variables TIME$, DATE$, and SYSVAR. The maximum number of characters that can be read using an INPUT statement is 32,767 characters.

Notes

INPUT filters out any incoming ASCII 00 decimal characters (NUL). INPUT does not support autohunting (see SETSTDIO).

Example 1

This example shows input to one numeric variable and one string variable:

```
10 INPUT "ADDRESS";A%,B$
20 PRINT A%;" ";B$
30 IF A% > 0 THEN GOTO 50
40 GOTO 10
50 END
RUN
```

This results in:

ADDRESS?

When the prompt "ADDRESS?" appears on the screen, you can type the input data on the host keyboard:

999, HILL STREET

Note the separating comma. If the input text data contains a comma to be printed, enclose the input data with quotation marks ("...."):

999, "HILL STREET, HILLSBOROUGH"

Numeric input data must not include any decimal points.

Example 2

This example shows how the date can be set directly from the host keyboard:

```
INPUT "Enter date: ",DATE$
```

This results in:

Enter date:

When the prompt "Enter date:" appears on the host screen, you can type the date as a six-digit combination of year, month and day (see DATE$ variable). Time can also be set using the same method.

INPUT ON/OFF

Purpose

Enables or disables the Direct Protocol (disabled by default).

Syntax

INPUT ON|OFF

Notes

INPUT ON enables the Direct Protocol, which does the following:

- Enables reception of input data to a stored layout
- Enables the error handler
- Disables verbosity (SYSVAR (18) = 0)
- Shows "Direct Protocol" in the printer Ready screen (if the printer is set up to show the command language)

INPUT OFF disables the Direct Protocol.

The following commands are supported only by Direct Protocol (after an INPUT ON statement has been executed): COUNT&, ERROR, FORMAT INPUT, INPUT OFF, LAYOUT END, and LAYOUT RUN.

Example

This example illustrates how to enable Direct Protocol, specify new separators, store a layout in printer memory, combine variable data with the layout, print a label, and disable Direct Protocol:

```
INPUT ON
FORMAT INPUT "#","@","&"
LAYOUT INPUT "tmp:LABEL1"
FT "Univers"
PP 100,250
PT VAR1$
PP 100,200
PT VAR2$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
#Line number 1&Line number 2&@
PF
INPUT OFF
```

INPUT#

Purpose

Reads a string of data from an OPEN device or sequential file.

Syntax

INPUT#*<nexp>*,*<<nvar>|<svar>>*[,*<<nvar>|<svar>>*...]

Parameters

<nexp>
The number assigned to the file or device when it is opened using the OPEN command.

<<nvar>|<svar>>
Variable to which the input data will be assigned.

Notes

This statement resembles INPUT, but allows the input to come from other files, or devices other than the standard IN channel. Like INPUT, commas can be used to assign different portions of the input to different variables. INPUT# does not allow prompts to be used.

When reading from a sequential file, the records can be read one after the other by the repeating INPUT# with the same file reference.

Once a file record has been read, it cannot be read again until the file is closed using the CLOSE command and then opened using the OPEN command.

INPUT # can access updated information in a file only if the file is OPENed, written to, and then CLOSEd.

The maximum number of characters that can be read using an INPUT# statement is 32,767. INPUT# filters out any incoming ASCII 00 decimal characters (NUL).

Example

This example assigns data from the first record in the sequential file "Addresses" to the three string variables A$, B$, and C$, and from the second record in the same file to the string variables D$ and E$:

```
. . . . .
. . . . .
100 OPEN "ADDRESSES" FOR INPUT AS #5
110 INPUT#5, A$, B$, C$
120 INPUT#5, D$, E$
. . . . .
. . . . .
```

INPUT$

Purpose

Returns a limited-length data string from the standard IN channel (or optionally from a file or device opened using the OPEN command).

Syntax

INPUT$(<*nexp1*>[,<*nexp2*>])

Parameters

<*nexp1*>
Number of characters to be read.

<*nexp2*>
(Optional) Specifies a file or device using the number assigned to it when it is opened using the OPEN command.

Notes

If no file or device is specified, the input comes from the standard IN channel. For more information, see SETSTDIO.

Otherwise, input comes from the specified file or device. The execution is held until the specified number of characters have been received from the keyboard console, file, or communication channel. If a file does not contain the specified number of characters, execution is resumed as soon as all available characters in the file have been received.

The maximum number of characters that can be returned using an INPUT$ statement is 65,536 characters.

Example 1

This example reads a sequence of 25 characters from the printer built-in keyboard and assigns them to a string variable named Z$:

```
. . . . .
. . . . .
1000 OPEN "CONSOLE:" FOR INPUT AS #1
1010 Z$=INPUT$(25,1)
. . . . .
. . . . .
```

Example 2

In this example, 10 characters are read from the standard IN channel and assigned to a variable:

```
10 A$=INPUT$(10)
```

INSTR

Purpose

Searches a specified string for a certain character or sequence of characters, and returns its position relative to the start of the string.

Syntax

INSTR([*<nexp>*,]*<sexp1>*,*<sexp2>*)

Parameters

<nexp>
(Optional) Position where the search will start.

<sexp1>
String to be searched.

<sexp2>
Character(s) for which the string is searched.

Notes

Optionally, it is possible to specify a certain position in the string from which the search will start. If no start position is specified, the search starts at the beginning of the string. The result is zero if:

- the start position value exceeds the length of the string.
- the string is empty.
- the searched combination of characters cannot be found.

Example 1

In this example, the string "INTERMEC_PRINTER_AB" is searched for the character combination "AB". No start position is specified.

```
10 A$="INTERMEC PRINTER AB"
20 B$="AB"
30 PRINT INSTR(A$,B$)
RUN
```

This results in:

```
18
```

Example 2

In this example, the string "INTERMEC_PRINTER_AB" is searched for the character "I" and the start position is specified as 4.

```
10 A$="INTERMEC PRINTER AB"
20 B$="I"
```

```
30 PRINT INSTR(4,A$,B$)
RUN
```

This results in:

```
12
```

INVIMAGE

Purpose

Inverts the printing of text and images from "black-on-white" to "white-on-black."

This command can be abbreviated as II.

Syntax

INVIMAGE

or

II

Notes

Reset to default by executing PRINTFEED.

INVIMAGE can only be used in connection with the printing of text and images (PRTXT and PRIMAGE). In the matrix of the font or image, all "white" dots will be black and all black dots will be "white." Not all fonts are suited for inverse printing. Thin lines, serifs, and ornaments may be difficult to distinguish. There may also be an imbalance between the ascending and descending black background.

The same principles apply to images. The normally invisible background may be larger than expected or be less favorably balanced. Small "white" details tend to be blurred out by the black background. Therefore, before using an inverse image, make a printout sample. INVIMAGE is revoked by a NORIMAGE statement.

Example

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

KEY BEEP

Purpose

Resets the frequency and duration of the sound produced by the printer when any of the keys on the printer keyboard are pressed.

Syntax

KEY BEEP<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
Frequency of the sound in Hz (maximum value 9999, a higher value will be ignored and give no sound).

<*nexp2*>
Duration of the sound in periods of 0.020 seconds each (no maximum limit).

Notes

This statement sets the response for all keys of the printer. To turn off the audible key response, set the frequency to a value higher than 9999. A value higher than 9999 (no sound) is the default behavior for KEY BEEP.

The table below illustrates the relation between frequencies and the musical scale (same as in the SOUND statement).

Frequency in Hz				
Pitch	**1st octave**	**2nd octave**	**3rd octave**	**4th octave**
C	131	262	523	1047
C#/Db	138	277	554	1109
D	147	294	587	1175
D#/Eb	155	311	622	1245
E	165	330	659	1319
F	175	349	699	1397
F#/Gb	185	370	740	1480
G	196	392	784	1568
G#/Ab	208	415	831	1662
A	220	440	880	1760

Frequency in Hz				
Pitch	1st octave	2nd octave	3rd octave	4th octave
A#/Bb	233	466	933	1865
B	247	494	988	1976

Note that C in the 2nd octave (262 Hz) corresponds to "middle C."

Example

In this example, the beeper produces a concert "A" for one second each time a key is pressed down.

```
10 KEY BEEP 440,50
. . . . .
. . . . .
```

KEY ON/OFF

Purpose

Enables or disables a specified key on the printer front panel, to be used in connection with an ON KEY...GOSUB statement.

Syntax

KEY(<*nexp*>)ON|OFF

Parameters

<*nexp*>
ID number of one of the keys on the printer front panel.

ON|OFF
Disables or enables the specified key.

Notes

Using an ON KEY... GOSUB statement, any key (except the **Shift** key) can be assigned to make the program branch to a subroutine when pressed. The keys are enabled or disabled individually and are specified by means of their respective ID. numbers in unshifted and shifted positions. To specify a shifted key, add 100 to the unshifted ID. number of the key.

ID numbers of the keys are not the same as the ASCII values they produce when pressed. For more information, see KEYBMAP$.

For more information about printer keypad layouts, see Printer Keypad Layouts.

Example

In this example, the left arrow key (F1, ID number 10) is first enabled and then used to branch to a subroutine, and finally disabled. Each time you press F1, a label is printed.

```
10 ON KEY (10) GOSUB 1000
20 KEY (10) ON
30 GOTO 30
.....
.....
.....
1000 FONT "Swiss 721 BT"
1010 PRPOS 30,100
1020 PRTXT "HELLO"
1030 PRINTFEED
1040 END
RUN
```

KEYBMAP$

Purpose

Returns or sets the mapping for a button on the printer front panel, if the printer has keyboard buttons.

Syntax

To read the mapping for a key button:

<*svar*> = KEYBMAP$(<*nexp*>)

To set the mapping for a key button:

KEYBMAP$(<*nexp*>) = <*sexp*>

Parameters

<*svar*>
Returns the keyboard mapping string.

<*nexp*>
Type of keys to be returned or remapped:

0: **Shift** key not activated (64 characters)
1: **Shift** key activated (64 characters)

<*sexp*>
String specifying the ASCII value for each key position in the selected type of string.

Notes

In principle, each physical key can produce two different ASCII values: one in unshifted position, and another in shifted position. One key is specified as **Shift**. When **Shift** is pressed at the same time as another key, the unshifted ASCII value of the second key is increased by 128.

You can use KEYBMAP$ to:

- Read the keyboard mapping based on whether **Shift** is activated. The printer returns a string of ASCII values in ascending key position number. Because many keys return non-printable ASCII values (ASCII 00 to ASCII 31 decimal), not all are returned to the host screen or printed on a label.
- Change the keyboard mapping. You can change the mapping of the keyboard so a key produces a different ASCII value than before. To do so, create a string which specifies the ASCII value for each of all 64 unshifted, shifted, or **Alt**-initiated key positions in ascending order. Regardless of what the keyboard looks like, there are always 64 theoretical key positions.

Characters that cannot be produced by the host keyboard can be substituted by CHR$ functions, where the character is specified by its ASCII decimal value according to the character set selected by NASC. Key positions which should be disabled or are not

included in the physical keyboard can be mapped as NUL, using the function CHR$(0). Note that the position of **Shift** cannot be remapped.

In Setup mode, the keys have fixed positions and are not affected by KEYBMAP$.

The KEYBMAP variable is reset each time the printer is restarted or the power is cycled.

Note: For more information about printer keypad layouts, see Printer Keypad Layouts.

Example

```
10 B$=CHR$(1)+STRING$(4,0)+CHR$(2)+ STRING$(4,0)+CHR$(3)
20 B$=B$+STRING$(4,0)+CHR$(4)+STRING$(4,0)+ CHR$(5)+STRING$(9,0)
30 B$=B$+CHR$(13)+CHR$(28)+CHR$(29)+CHR$(30)+ STRING$(6,0)
40 B$=B$+.147"+CHR$(0)+"0258"+CHR$(0)+CHR$(8)+"369"+CHR$(0)+(CHR$(31)
50 KEYBMAP$(0)=B$
RUN
```

KILL

Purpose

Deletes a file, directory, or complete directory sub-trees from the printer memory or from external memory (such as a USB storage device).

Syntax

KILL<*sexp*>[,R[,A]]

Parameters

<*sexp*>
File or directory to be deleted.

R
Recursively removes all non-system files in the specified sub-tree and then removes all empty directories in the same sub-tree.

A
(Optional) Removes all files including system files.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

The name of the file to be deleted must match the name given when the file was saved and must include the extension. If no extension was entered manually when the file was SAVEd, the extension ".PRG" is added automatically.

To KILL a file residing in a directory other than the current one, include a reference to the directory in question when you specify the file (for example, "d:<filename>.XYZ").

KILL cannot be used for files residing in "rom:". A directory cannot be removed if it contains files or directories unless the R flag is included in the KILL statement. Otherwise error 1073 ("Directory not empty") occurs.

A trailing slash character (/) may be added to directory names but is not necessary.

The A and R flags are only applicable when removing directories, or error 1034 ("Not a directory") occurs.

The current directory may be removed (for example KILL CURDIR$,R), but the root directory of a device cannot be removed. The current directory is not changed after such a command, but is invalid. A successful CHDIR statement is necessary to restore the current directory to one that exists (CHDIR ".." may not work).

Example

```
10 ON ERROR GOTO 1000
20 CHDIR("/c")
30 MKDIR "DIR1" : 'Create the directory DIR1
```

```
40 FILES
50 COPY "STDIO", "DIR1" : 'Copy STDIO into DIR1
60 FILES "DIR1" : 'List files in DIR1
70 KILL "DIR1" : 'Try to remove DIR1
80 KILL "DIR1",R : 'Remove the directory recursively
90 FILES
100 END
1000 PRINT "error number "; ERR;"in line ";ERL
1010 RESUME NEXT
RUN
```

Here are the results of this example:

```
Files on /c
Dir1/           0       APPLICATION  0
boot/           0       ADMIN/       0
STDIO           4

22210562 bytes free 4       bytes used

STDIO           4
2220032 bytes free  4       bytes used
Error number 1073 in line 70
Files on /c

APPLICATION     0       boot/        0
ADMIN/          0       STDIO        4

2222080 bytes free  4       bytes used
```

LAYOUT

Purpose

Handles layout files.

Syntax

LAYOUT[F,] *<sexp1>*,*<sexp2>*,*<svar>*|*<sexp3>*,*<nvar>*|*<sexp4>*

Parameters

F
(Optional) Allows use of data and error files instead of arrays.

<sexp1>
Layout file.

<sexp2>
Logotype name file.

<svar>|*<sexp3>*
Data array (*<svar>*) or data file (*<sexp3>*).

<nvar>|*<sexp4>*
Error array (*<nvar>*) or error file (*<sexp4>*).

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

The next table lists the format elements of a layout file in ascending order by bytes.

Input: H = hex digit, D = Numeric digit, C = Alpha character

Byte #	Parameter	Layout Type	Input	Notes
0 - 1	Element number		HH	
2	Layout type: A - Logotype by name B - Bar code C - Text E - Bar code extended field H - Barfont on/off J - Baradjust L - Logotype by number S - Line X - Box		C for all layout types	See Note 1
3	Direction Barfont on/off: 0: off, 1: on Security	A, B, C, L, S, X H E	D D D	
4	Alignment Aspect height ratio Barfont alignment	A, B, C, L, S, X E H	D D D	
5-8	X-position Aspect width ratio Baradjust left Horizontal offset	A, B, C, L, S, X E J H	DDDD D DDDD CDDD	
9-12	Y-position Rows in bar code Baradjust right Vertical offset	A, B, C, L, S, X E J H	DDDD D DDDD CDDD	

Byte #	Parameter	Layout Type	Input	Notes
13-22	Font name Logotype name Bar code name Barfont name Line length Box width Columns in bar code Truncate according to code specs	C A B H S X E E	C (up to 10 chars) C (up to 10 chars) C (up to 10 chars) C (up to 10 chars) DDDD DDDD DD D	See Note 2 Byte 13-14 Byte 15
23-42	Fixed text or alphanumeric data Fixed numeric data Logotype number Box height Line thickness Character set	B or C B L X S D	C (up to 20 chars) D (up to 20 chars) DD DDDD DDDD DDDDD or C (up to 20 characters)	
43-44	Number of chars to print from bytes 23-42	B or C	DD	
45-46	Image type (I = inverse image) Bar code ratio (wide:narrow bars)	A, C, L B	C DD	
47	Vertical magnification Bar code magnification	A, C, L B	D or C D or C	See Note 3 See Note 3
48	Horizontal magnification	A, C, L	D	
49-51	Bar code height Line thickness	B X	DDD DDD	

The bar code extended field record (I) corresponds to the six last parameters in the BARSET statement. Must have a lower element number than the corresponding bar code record (B), which specifies the other bar code parameters.

The horizontal and vertical offset of BARFONT can be a positive or negative value. For negative values, the '-' sign must be defined in Byte 5 or Byte 9, or the offset is treated as a positive value.

The font name length in the LAYOUT statement must be exactly 10 characters. Because most font names in Fingerprint are longer, use font name aliases that are exactly 10 characters long.

If a magnification of 0-9 is sufficient, enter a numeric digit. If a higher magnification than 9 is required, enter the character with the ASCII decimal number that corresponds to the desired magnification minus 48. For example, if magnification 10 is desired, enter the character: (colon, ASCII 58 decimal).

Logotype name file format #1

Record 1–n, 10 bytes each. No embedded spaces in name.

C1...C10 Name for logotype 1
. . .
. . .
C1...C10 Name for logotype n

Logotype name file format #2

Record 1...–n, 13 bytes each. Records sorted in ascending logotype number order.

DD Logotype number (2 digits)
C Always ":" (colon). Separator. Distinguishes format 2.
C1...C10 Name of logotype (10 characters)

Logotype name file formats #1 and #2 are alternative.

Data array/file format

One array position/One file line. Sorted in ascending order.

HH Element number
C1...Cn Data

If a data element cannot be used in the layout, an error occurs. The index of the unused element and error code -1 are placed in the error array/file.

Error array/file format

Sorted in ascending order.

Array position/File line 0: Record number for error 1
Array position/File line 1: Error number for error 1
. . .
. . .
Array position/File line 2*n*-2: Record number for error *n*
Array position/File line 2*n*-1: Error number for error *n*

To improve performance, Honeywell recommends that you create the layout and logotype name files in the printer temporary memory ("tmp:"). Once they have been created in "tmp:", they can be copied to the printer permanent memory to avoid losing them at power off.

Note: Do not confuse this statement with the statements LAYOUT INPUT, LAYOUT END, and LAYOUT RUN.

Example

Note that the 10 characters available to define a font in the LAYOUT statement in most cases cannot accommodate modern outline font names. Instead, use font aliases as seen in lines 90–120 and 150:

```
10 DIM QERR%(10)
20 LAYDATA$(0)="01DAY"
30 LAYDATA$(1)="04123456789012"
40 QERR%(0)=0
50 OPEN "tmp:LOGNAME.DAT" FOR OUTPUT AS 19
```

```
60 PRINT# 19,"DIAMONDS.1";
70 CLOSE 19
80 OPEN "tmp:LAYOUT.DAT" FOR OUTPUT AS 6
90 PRINT# 6,"01C11100 10 font alias 00I 11 ";
100 PRINT# 6,"01C11100 40 font alias 00 22 ";
110 PRINT# 6,"01C11100 100 font aliasWEDNES 06I 11 ";
120 PRINT# 6,"01C11100 130 font aliasSATURNUS 05I 11 ";
130 PRINT# 6,"02L11300 70 1 33 ";
140 PRINT# 6,"03S11100 210 300 3 ";
150 PRINT# 6,"04H1 font alias ";
160 PRINT# 6,"04B14100 300 EAN13 0 312 100";
170 CLOSE 6
180 LAYOUT "tmp:LAYOUT.DAT","tmp:LOGNAME.DAT",LAYDATA$,QERR%
190 IF QERR% (1) = 0 THEN GOTO 260
200 PRINT "-ERROR- LAYOUT 1"
210 I%=0
220 IF QERR%(I%)=0 THEN GOTO 260
230 PRINT " ERROR ";QERR%(I%+1);" in record ";QERR%(I%)
240 I%=I%+2
250 GOTO 220
260 PRINTFEED
```

LAYOUT END

Purpose

Stops the recording of a layout description and saves the layout. Supported only by Direct Protocol.

Syntax

LAYOUT END

Notes

LAYOUT END can only be used in the Direct Protocol, and only after a layout has been recorded by LAYOUT INPUT. After LAYOUT END, no more data is added to the layout.

By default, the layout is saved in the printer permanent memory ("/c"). To speed up execution the layout can be saved in the temporary memory (see LAYOUT INPUT statement). The layout can be copied and killed as with any program file.

Example

This example illustrates how to enable Direct Protocol, specify new separators, store a layout in the printer temporary memory, combine variable data with the layout, print a label, and disable Direct Protocol:

```
INPUT ON
FORMAT INPUT "#","@","&"
LAYOUT INPUT "tmp:LABEL1"
FT "Univers"
PP 100,250
PT VAR1$
PP 100,200
PT VAR2$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
#Line number 1&Line number 2&@
PF
INPUT OFF
```

LAYOUT INPUT

Purpose

Starts the recording of a layout description. Supported only by Direct Protocol.

Syntax

LAYOUT INPUT *<sexp>*

Parameters

<sexp>
Desired name of the layout (maximum 30 characters). Includes the name of the device where the layout is to be stored.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

LAYOUT INPUT can only be used in the Direct Protocol and starts the recording of a layout. After sending a LAYOUT INPUT command, all formatting instructions (such as PRPOS, MAG, FONT, BARFONT, BARSET, PRTXT, PRBAR, PRIMAGE, PRBOX, PRLINE, and so on) sent to the printer before a LAYOUT END statement are included in the layout.

Honeywell recommends that you create layouts in the printer temporary memory, or a USB storage device. Once a layout has been created in the temporary memory, it can be copied to permanent storage so it will not be lost after you restart the printer.

Variable input data to text, bar code, and image fields can be provided separately using LAYOUT RUN. Such variable data are indicated in the layout by string variables VAR*n*$ where *n* is the number of the field in the LAYOUT RUN string of data. For example, the statement PRTXT "Hello" in the layout results in a fixed text, whereas the statement PRTXT VAR1$ results in a variable text provided by the first field in a LAYOUT RUN string.

The layout must not contain any PRINTFEED statements and is not saved until a LAYOUT END statement is executed.

Example

This example illustrates how to enable Direct Protocol, specify new separators, store a layout in the printer temporary memory, combine variable data with the layout, print a label, and disable Direct Protocol:

```
INPUT ON
FORMAT INPUT "#","@","&"
LAYOUT INPUT "tmp:LABEL1"
FT "Univers"
PP 100,250
PT VAR1$
PP 100,200
```

```
PT VAR2$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
#Line number 1&Line number 2&@
PF
INPUT OFF
```

LAYOUT RUN

Purpose

Provides variable input data to a predefined layout. Supported only by Direct Protocol.

Syntax

LAYOUT RUN *<sexp>*

Parameters

<sexp>
Name of the layout (as specified by LAYOUT INPUT).

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

LAYOUT RUN can only be used in the Direct Protocol and selects a predefined layout in a specified part of the printer memory. This command can also provide input to string variables in the layout. Such variables are indicated by VAR*n*$, where *n* indicates a field in the string of data that should follow the LAYOUT RUN statement.

The string of input data is composed according to the following syntax, where <Srec> is the start-of-record string, <Efld> is the end-of-field string and <Erec> is the end-of-record string (as defined in FORMAT INPUT):

- <Srec><input to VAR1$><Efld><input to VAR2$><Efld>&ldots;<input to VARn$><Efld><Erec>
- Note that every field must end with the end-of-field string, including the last field.
- Before reverting to normal Fingerprint printing after using variable data (LAYOUT INPUT, LAYOUT END, and LAYOUT RUN), clear the data using LAYOUT RUN with an empty string (LAYOUT RUN " ").

Example

This example illustrates how to enable Direct Protocol, specify new delimiting strings, store a layout in the printer temporary memory, combine variable data with the layout, print a label, and disable Direct Protocol:

```
INPUT OFF
FORMAT INPUT "#","@","&"
INPUT ON
LAYOUT INPUT "tmp:LABEL1"
FT "Univers"
PP 100,250
PT VAR1$
PP 100,200
PT VAR2$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
```

```
#Line number 1&Line number 2&@
PF
INPUT OFF
```

LBLCOND

Purpose

Overrides the media feed setup.

Syntax

LBLCOND<*nexp1*>,<*nexp2*>|<*nexp3*>

Parameters

<*nexp1*>
Specifies the type of action:

0: Override the stop adjust.
1: Override the start adjust.
2: Turn off the Label Stop Sensor/Black Mark Sensor.
3: Select the mode specified by <*nexp3*>.

<*nexp2*>
Specifies <*nexp1*> = 0, 1, or 2 as a number of dots.

<*nexp3*>
Specifies one of the following modes:

0: Legacy mode
1: IPL mode
2: Gap Truncate mode

Default is LBLCOND 3,2.

Notes

LBLCOND allows you to override the printer's feed-adjust setup or to temporarily disable the label stop sensor or black mark sensor.

<*nexp1*> = 0 temporarily sets the stop adjust to the value specified by <*nexp2*>.

<nexp1> = 1 temporarily sets the start adjust to the value specified by <nexp2>.

<*nexp1*> = 2 makes the label stop sensor (LSS) or black mark sensor temporarily ignore any gaps or marks detected within the length of media feed specified by <*nexp2*>. However, the label length must be greater than the distance between the LSS and the tear bar (if not, use LBLCOND 3,xx). This allows the use of labels of such shapes that would make the LSS react prematurely, or tickets with preprint at the back of the media that would interfere with the detection of the black mark.

<*nexp1*> = 3 is useful as an alternative to LBLCOND 2,xx when the length of the label or ticket is shorter than the distance between the LSS and the tear bar, making it possible to select one of the modes specified by <*nexp3*>:

- Legacy mode (<*nexp3*> = 0): If the print image is longer than the physical length of the label or ticket, the print image will extend into the next label until the media feed

stops according to the stop adjust setup (for example, when the gap becomes aligned with the tear bar). This means that the print image may be truncated, the next label may have to be discarded, and some of the print image may coincide with a gap or slot. This mode was called "Default mode" in earlier versions of Fingerprint.

- IPL mode (<*nexp3*> = 1): If the print image is longer than the physical length of the label or ticket, the print image will extend into the following labels) until the entire print image has been printed. Then the media is fed out to the next gap or mark according to the stop adjust setup. This means that the print image will not be truncated but may extend into one or more consecutive labels, and some of the print image may coincide with gaps or slots.
- Gap Truncate mode (<*nexp3*> = 2): If the print image is longer than the physical length of the label or ticket, only the part of the print image that fits on the label or ticket will be printed and the remainder will be ignored. This means that some of the print image may not be printed at all, but the following labels will not be affected.

Verifying a start adjust or stop adjust value in the Setup mode by pressing key number 16 (normally labeled "Enter"), or by setting the value using a setup file or setup string, evokes any LBLCOND statement for the parameter in question.

The label stop sensor is returned to normal operation by the statement:

LBLCOND 2,0

All current LBLCOND statements are revoked at startup or after you run a REBOOT command. This means that the start and stop adjust are decided by the setup and the label stop sensor works normally.

Example

In this example, the start adjust value in the setup mode is overridden and the label stop sensor is set to ignore any gaps in the web within 20 mm (160 dots at 8 dots/mm; 240 dots at 12 dots/mm) of media feed:

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

LED ON/OFF

Purpose

This command specifies the behavior of the printer status LEDs. This command is applicable only for the PC23d, PC43d, PC43t, PM23c, PM43, and PM43c icon model printers.

Syntax

LED<*nexp*>ON|OFF|BLINK

Parameters

Where the values for <*nexp*> are as follows:

nexp	LED
0	Data transfer
1	Error
2	Warning
3	Paper error
4	Ribbon error
5	Printhead lifted
6	Overheat
7	Configuration error
8	Pause
9	Maintenance
nexp	LED
0	Data transfer
1	Error
2	Warning
3	Paper error
4	Ribbon error
5	Printhead lifted
6	Overheat
7	Configuration error
8	Pause
9	Maintenance

Notes

The LED starts blinking when data is received on the standard input channel, and stays lit when the channel has been silent for 0.8 seconds. The LED does not blink during data transfer on PM23c, PM43, or PM43c printers.

Under Direct Protocol, the LED can also be affected by the error handler. See ERROR for more information.

If the DATA IN flag is set with the BLINK mode, reception of data on the standard input channel controls whether the LED blinks or is switched off.

For more information about icon layouts for your printer, see Printer Keypad Layouts.

LEFT$

Purpose

Returns a specified number of characters from the start (the extreme left side) of a given string.

Syntax

LEFT$(<*sexp*>,<*nexp*>)

Parameters

<*sexp*>
String from which the characters will be returned.

<*nexp*>
Number of characters to be returned.

Notes

LEFT$ is the complementary function for RIGHT$. If the number of characters to be returned is greater than the number of characters in the string, then the entire string is returned. If the number of characters is set to zero, a null string is returned.

Example 1

```
10 PRINT LEFT$("THERMAL PRINTER",7)
RUN
```

This results in:

```
THERMAL
```

Example 2

```
10 A$="THERMAL PRINTER":B$="LABEL"
20 PRINT LEFT$(A$,8);LEFT$(B$,10);"S"
RUN
```

This results in:

```
THERMAL LABELS
```

LEN

Purpose

Returns the number of character positions in a string.

Syntax

LEN(*<sexp>*)

Parameters

<sexp>
String from which the number of characters will be returned.

Notes

The number of characters to be returned includes unprintable characters.

Example 1

In this example, lines 40 and 50 illustrate two ways of using the LEN function to add up the number of characters from several string expressions.

```
10 A$="INTERMEC" : '8 char.
20 B$="THERMAL" : '7 char.
30 C$="PRINTERS" : '8 char.
40 PRINT LEN(A$+B$+C$)
50 PRINT LEN(A$)+LEN(B$)+LEN(C$)
RUN
```

This results in:

```
23
23
```

Example 2

This example demonstrates that unprintable characters (such as space characters) are included in the value returned by the LEN function:

```
PRINT LEN("INTERMEC THERMAL PRINTERS")
```

This results in:

```
25
```

LET

Purpose

Assigns the value of an expression to a variable.

Syntax

[LET]<<*nvar*>=<*nexp*>>|<<*svar*>=<*sexp*>>

Parameters

<nvar>
Numeric variable to which a value will be assigned.

<nexp>
Numeric expression from which the value will be assigned to the numeric variable.

Or,

<svar>
String variable to which the content of the string expression will be assigned.

<sexp>
String expression from which content will be assigned to the string variable.

Notes

LET is retained for compatibility with previous versions of Fingerprint. The equal sign (=) is sufficient to make the assignment. Both the expression and the variable most be either string or numeric.

Example

```
10 LET A%=100 : 'numeric variable
20 B%=150 : 'numeric variable
30 LET C$="INTERMEC" : 'string variable
40 D$="THERMAL PRINTERS" : 'string variable
50 PRINT A%+B%,C$+" "+D$
RUN
```

This results in:

```
250 INTERMEC THERMAL PRINTERS
```

LINE INPUT

Purpose

Assigns an entire line, including punctuation marks, from the standard IN channel to a single string variable.

Syntax

LINE INPUT[*<scon>*;]*<svar>*

Parameters

<scon>;
(Optional) Prompt plus a semicolon.

<svar>
String variable to which the input line is assigned.

Notes

For information on the standard I/O channel, see SETSTDIO.

LINE INPUT differs from INPUT in that an entire line of up to 32,767 characters is read. Possible commas appear as punctuation marks in the string instead of dividing the line into portions.

During the execution of a program, LINE INPUT interrupts execution. You can create a prompt for the host screen that notifies the operator when a program is expecting additional data to be entered. The input is terminated and the program execution resumes when a carriage return character (ASCII 13 decimal) is received.

The carriage return character will not be included in the input line. Note that LINE INPUT filters out any incoming ASCII 00 decimal characters (NUL).

Example

Print your own visiting card like this:

```
10 LINE INPUT "ENTER NAME: ";A$
20 LINE INPUT "ENTER STREET: ";B$
30 LINE INPUT "ENTER CITY: ";C$
40 LINE INPUT "ENTER STATE + ZIPCODE: ";D$
50 LINE INPUT "ENTER PHONE NO: ";E$
60 FONT "Univers", 8
70 ALIGN 5
80 PRPOS 160,300:PRTXT A$
90 PRPOS 160,250:PRTXT B$
100 PRPOS 160,200:PRTXT C$
110 PRPOS 160,150:PRTXT D$
120 PRPOS 160,100:PRTXT "Phone: "+E$
130 PRINTFEED
RUN
```

LINE INPUT#

Purpose

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

Syntax

LINE INPUT#<*nexp*>,<*svar*>

Parameters

<*nexp*>
Number assigned to the file when it is opened using the OPEN command.

<*svar*>
String variable to which the input line is assigned.

Notes

This statement differs from the INPUT# statement in that an entire line of up to 32,767 characters will be read, and possible commas in the line are included in the string as punctuation marks.

The lines can be read one after the other by repeating the LINE INPUT# statement using the same file reference. Once a line has been read, it cannot be read again until the file is closed using the CLOSE command and then opened using the OPEN command.

LINE INPUT# is useful when the lines in a file have been broken into fields.

LINE INPUT# filters out any incoming ASCII 00 decimal characters (NUL).

Example

This example assigns data from the three first lines of the file "Addresses" to the string variables A$, B$, and C$ respectively:

```
. . . . .
. . . . .
100 OPEN "ADDRESSES" FOR INPUT AS #5
110 LINE INPUT# 5, A$
120 LINE INPUT# 5, B$
130 LINE INPUT# 5, C$
. . . . .
. . . . .
```

LIST

Purpose

Lists the current program completely or partially, or lists all variables, to the standard OUT channel.

Syntax

LIST[[<*ncon1*>[–<*ncon2*>]] | ,V | ,B]

Parameters

<*ncon1*>
Single line, or the first line number in a range of lines.

<*ncon2*>
Optionally the last line number in a range of lines.

,V
Lists all variables.

,B
Lists all breakpoints.

Notes

Protected applications cannot be listed. For more information on protecting an application, see SAVE.

LIST is useful after LOADing a program, if you have changed or renumbered lines, or added new lines and want to bring some order to the presentation on the screen of the host. LIST also removes unnecessary characters and adds assumed keywords. LIST is usually given in Immediate mode (on a line without a preceding line number).

The LIST statement can be used in several different ways:

- If no line number is entered after LIST, the entire current program is listed. If the program has been written without line numbers (see IMMEDIATE), the lines are automatically numbered in 10-step increments starting with 10 (10-20-30-40-50 and so on).
- If a single line number is entered after LIST, only the specified line is listed.
- If a line number followed by a hyphen (-) is entered after LIST, all lines from the specified line to the end of the program are listed.
- If a hyphen (-) followed by line number is entered after LIST, all lines from the start of the program through the specified line are listed.
- If two line numbers are entered after LIST, they specify the first and last lines in a range of lines to be listed.

- If LIST,V is entered, all integer variables, integer array variables, string variables, and string array variables in the printer memory are listed.
- If LIST,B is entered, all breakpoints of the Fingerprint Debugger (see DBBREAK) are printed in line number order. Line labels that have not been updated (which occurs at program execution) may be misplaced.

Examples

LIST : 'Lists all lines in the program.

LIST 100 : 'Lists line 100 only.

LIST 100- : 'Lists all lines from line 100 to the end of the program.

LIST -500 : 'Lists all lines from the start through line 500.

LIST,V : 'Lists all variables.

LIST,B : 'Lists all breakpoints.

LISTPFSVAR

Purpose

Lists variables saved at power failure.

Syntax

LISTPFSVAR

Notes

Related instructions are SETPFSVAR, GETPFSVAR, and DELETEPFSVAR.

Example

LISTPFSVAR

This results in:

QS$
QCPS%
A%

LOAD

Purpose

Loads a copy of a program into the printer working memory.

Syntax

LOAD*<scon>*

Parameters

<scon>
Program to be loaded into the working memory.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

If the program has the extension .PRG, the name of the program can be given with or without an extension. Otherwise, the extension must be included in the name. If the program resides in another directory than the current one (see CHDIR), the name must also contain a reference to the directory in question.

LOAD closes any open files and deletes all program lines and variables residing in the working memory before loading the specified program. If the previous program in the working memory has not been saved, it is lost and cannot be retrieved.

While the program is loaded, a syntax check is performed. If a syntax error is detected, loading is interrupted and an error message is transmitted on the standard OUT channel.

Example 1

This example loads the program "LABEL127.PRG" from the current directory:

LOAD "LABEL127"

or

LOAD "LABEL127.PRG"

When "Ok" appears on the screen, the LOAD is complete. Use LIST to display the program on the host screen.

Example 2

This example shows how to specify a directory other than the current one:

LOAD "/rom/MKAUTO.PRG"

or

LOAD "d:PROGRAM1.PRG"

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

LOC

Purpose

Returns the current position in a file opened with the OPEN command, or returns the status of the buffers in an OPEN communication channel.

Syntax

LOC(<*nexp*>)

Parameters

<*nexp*>
Number assigned to the file or communication channel when is opened using the OPEN command.

Notes

In a random file, LOC returns the number of the last record read by a GET statement or written by a PUT statement.

In a sequential file, LOC returns the number of 128-byte blocks that have been read or written since the file is opened using the OPEN command.

LOC also checks the receive or transmit buffer of the specified communication channel:

- If a channel is opened for INPUT using the OPEN command, the remaining number of characters (bytes) to be read from the receive buffer is returned.
- If a channel is opened for OUTPUT using the OPEN command, the remaining free space (bytes) in the transmit buffer is returned.

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

Example 1

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

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

Example 2

This example reads the number of bytes remaining to be received from the receive buffer of "uart2:":

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

LOF

Purpose

Returns the length in bytes of an OPEN sequential or random file, or returns the status of the buffers in an OPEN communication channel.

Syntax

LOF(<*nexp*>)

Parameters

<*nexp*>
Number assigned to the file or communication channel when it is opened using the OPEN command.

Notes

LOF also checks the receive or transmit buffer of the specified communication channel:

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

Example 1

This example illustrates how the length of the file "Pricelist" is returned:

```
10 OPEN "PRICELIST" AS #5
20 A%=LOF(5)
30 PRINT A%
. . . .
. . . .
```

Example 2

The second example shows how to calculate the number of free bytes in the receive buffer of communication channel "uart2:":

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

LSET

Purpose

Places data left-justified into a field in a random file buffer.

Syntax

LSET<*svar*>=<*sexp*>

Parameters:

<*svar*>
String variable assigned to the field by a FIELD statement.

<*sexp*>
Holds the input data.

Notes

After OPENing a file and formatting it using a FIELD statement, you can enter data into the random file buffer using LSET and RSET (RSET right-justifies the data).

The input data can only be stored in the buffer as string expressions. Therefore, a numeric expression must be converted to string format by the use of an STR$ function before LSET or RSET is executed.

If the length of the input data is less than the length of the field, the data is left-justified and the remaining number of bytes are printed as space characters.

If the length of the input data exceeds the length of the field, the input data is truncated on the right side.

Example

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

This results in:

SMITH — — — JOHN — — — — — — 12345630

LTS& ON/OFF

Purpose

Enables or disables the label taken sensor (default is disabled). Before using this command, you must first calibrate the label taken sensor (LTS) sensitivity. For more information on how to calibrate the LTS, see your printer user manual.

Syntax

LTS& ON|OFF

Notes

The label taken sensor (LTS) can be fitted at the printer label outfeed slot and detects if a printed label or ticket has been removed. Usually, a self-adhesive label is not fed out completely, but remains partly stuck to the liner so it does not fall off.

Using the LTS& ON statement, you can order the printer to stop the execution at the next PRINTFEED statement until the LTS no longer detects a label. Then the PRINTFEED is executed. This is most useful when printing batches of labels or tickets. As soon as a label is taken, the next one is printed and awaits handling.

LTS& OFF revokes LTS& ON.

Example

```
10 LTS& ON
20 FOR A%=1 TO 5
30 B$=STR$(A%)
40 FONT "Univers"
50 PRPOS 200,200
60 PRTXT B$
70 PRINTFEED
80 NEXT
RUN
```

MAG

Purpose

Magnifies a font, barfont, or image with regard to height or width.

Syntax

MAG*<nexp1>*,*<nexp2>*

Parameters

<nexp1>
Height magnification. Range is 1 (default) to 999.

<nexp2>
Width magnification. Range is 1 (default) to 999.

Notes

Reset to default by executing a PRINTFEED.

MAG makes the object grow in directions away from the selected anchor point. For more information, see ALIGN.

The implementation of scalable fonts makes using the MAG statement unnecessary. Although MAG works for such fonts, printout quality is improved by using a larger font size instead of magnifying a smaller one. MAG is retained for backwards compatibility with previous versions of Fingerprint.

MAG also works with images, but a larger version of an image gives better printout quality than using MAG to enlarge a small image.

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

Example

This example illustrates how the image "GLOBE.1" is printed both with its original size and magnified 4 times. Note the jagged edges of the curves in the enlarged image.

```
10 ALIGN 2
20 PRPOS 300,50
30 FONT "Univers"
40 PRTXT "Normal Size"
50 PRPOS 300,125
60 PRIMAGE "GLOBE.1"
70 PRPOS 300,300
80 PRTXT "Enlarged 4X"
90 PRPOS 300,375
100 MAG 4,4
110 PRIMAGE "GLOBE.1"
120 PRINTFEED
RUN
```

MAKEASSOC

Purpose

Creates an association.

Syntax

MAKEASSOC *<sexp1>*, *<sexp2>*, *<sexp3>*

Parameters

<sexp1>
Specifies the (case-sensitive) name of the association to be created.

<sexp2>
Contains an argument list of parameter tuples according to the convention in *<sexp3>*.

<sexp3>
Should always be "HTTP" (case sensitive).

Notes

HTTP implies that the argument list in *<sexp2>* is encoded in "x-www-urlencoding."

Example

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

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

This results in:

```
time=UNKNOWN LABEL=321 DESCRIPTION=DEF
time=153355
```

MAP

Purpose

Changes the ASCII value of a character when received on the standard IN channel (or optionally on another specified communication channel).

Syntax

MAP[*<nexp1>*,]*<nexp2>*,*<nexp3>*

Parameters

<nexp1>
(Optional) Specifies a communication channel:

0: "console:"
1: "uart1:"
2: "uart2:"
3: "uart3:"
4: "centronics:"
5: "net1:"
6: "usb1:"
7: "uart4:"
8: "uart5:"
9: "usbhost:"
10: "bluetooth:"
11: "ftp1:"
12: "http1:"
13: "lpr1:"

Default is the standard IN channel.

<nexp2>
Original ASCII decimal value.

<nexp3>
New ASCII decimal value after mapping.

Notes

This statement modifies a character set or filters out undesired characters. For example, to print a Q (ASCII 81 decimal) as the letter Z (ASCII 90 decimal) instead, enter the MAP statement as MAP 81,90. The mapping interprets any ASCII 81 decimal value received on the standard IN channel as ASCII 90 decimal, so that when you press Q on the keyboard of the host, the character Z is printed. However, pressing "Z" still produces a "Z" because that character has not been remapped.

To reset the mapping, map the character back to its original ASCII value (MAP 81,81).

When the printer receives a character, it is processed with regard to possible MAP statements before it "enters" the Fingerprint firmware. This allows you to filter undesired control characters by mapping them as NUL (ASCII 0 decimal).

After processing, the selected character set controls how characters are printed or displayed. If none of the character sets meets your demands completely, use MAP statements to modify the set that comes closest. Note that MAP statements are processed before any COMSET or ON KEY..GOSUB strings are checked.

NASC and NASCD statements are processed last.

Do not map any characters to ASCII values occupied by characters used in Fingerprint instructions, such as keywords, operators, %, $, #, and certain punctuation marks. Mapping is reset to normal after you restart the printer.

Examples

You can check what characters the host produces by using a simple program. Pressing different keys on the host should produce the corresponding characters both on the label and on the host screen. If not, try another character set (defined by NASC). This example assumes that the keyboard produces ASCII 81 decimal and ASCII 90 decimal when you press the Q and Z keys respectively. Should any unexpected characters be printed on the labels or the screen, check the manuals of the host for information on what ASCII values are produced by the various keys and how the screen presents various ASCII values received from the printer.

```
10 FONT "Univers"
20 PRPOS 30,100
30 INPUT "Enter character";A$
40 PRTXT A$
50 PRINTFEED
```

By adding a MAP statement in line 5, you can test what happens. In this case we remap the character Q to be printed as Z. After printing, we map the character Q back to its original position.

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

Assume that a device connected to "uart2:" produces strings that always start with the control character STX (ASCII 2 decimal). STX can be filtered out by mapping it as NUL (ASCII 0 decimal):

```
10 MAP 2,2,0
```

Should "uart2:" be appointed the standard IN channel (see SETSTDIO), the first parameter can be omitted from the example above:

```
10 MAP 2,0
```

MERGE

Honeywell recommends that you back up the current program before issuing MERGE.

Purpose

Merges a copy of a program with the program currently in printer working memory.

Syntax

MERGE<*scon*>

Parameters

<*scon*>
Name of the program to be merged with the program currently residing in the printer's working memory.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

MERGE creates a copy of a program and blends its lines into the program currently residing in the printer working memory.

If there are lines with the same numbers in both programs, the lines in the program currently residing in the working memory will be replaced by the corresponding lines in the MERGEd program. This also applies to programs written without line numbers, since they will automatically be assigned hidden line numbers at the execution of IMMEDIATE ON.

To avoid overwriting any lines, you can SAVE a program without line numbers using a SAVE <*scon*>, L statement. When MERGEd, it will be appended to the current program and assigned line numbers that start with the number of the last line of the current program plus 10.

MERGE makes it possible to store blocks of frequently used program instructions and include them in new programs. The printer ROM memory contains a number of useful programs which can be MERGEd into programs of your own creation.

Be careful not to include any MERGE statement as a part of a program, or else the execution stops after the MERGE statement has been executed.

The EXECUTE statement offers an alternative method for combining Fingerprint programs.

Examples

In this example, the program "XYZ.PRG" is merged with the current program. If there are identical line numbers in both programs, the lines from "XYZ. PRG"replace those in the current program.

MERGE "XYZ.PRG"	(from current directory)
MERGE "/c/XYZ.PRG"	(from permanent memory)
MERGE "tmp:XYZ.PRG"	(from temporary memory)
MERGE "d:XYZ.PRG"	(from USB storage device)

MIBVAR&

Purpose

Reads or writes MIB variables for the Simple Network Management Protocol (SNMP).

Syntax

MIBVAR&(<*nexp*>)

Parameters

<*nexp*>
Number of the variable to access. Range is 0 to 9.

Notes

SNMP MIB variables are normally used for network management tasks. These variables can be set from the outside through SNMP, and are available to Fingerprint programs through the MIBVAR& command. The SNMP variables must be of string type (maximum 128 characters) and are placed under the SNMP node "enterprises.1963.20.15.15.21.5.1.1., indexes 0-9." The public community is read-only and the pass community is read-write. There are no read or write restrictions from Fingerprint.

A related command is ON MIBVAR& GOSUB.

Example

```
MIBVAR&(1) = "ON"
A$ = MIBVAR&(1)
PRINT A$
```

This results in:

```
ON
```

MID$

Purpose

Returns a specified part of a string.

Syntax

MID$(<*sexp*>,<*nexp1*>[,<*nexp2*>])

Parameters

<*sexp*>
Original string from which a specified part is to be returned.

<*nexp1*>
Specifies which character position in the original string is to be the first character in the part to be returned.

If this value is less than or equal to zero, then Error 44 ("Parameter out of range") occurs. If this value exceeds the length of the original string, an empty string is returned, but no error condition occurs.

[,<*nexp2*>]
(Optional) Specifies the number of characters to be returned. If omitted, all characters from the start position specified by <*nexp1*> to the end of the string are returned. If this value is less than zero, then error 44 ("Parameter out of range") occurs.

Notes

If the value of <*nexp1*> does not exceed the length of the original string, but the sum of <*nexp1*> and <*nexp2*> exceeds the length of the original string, the remainder of the original string is returned.

Example 1

```
10 A$=MID$("INTERMEC PRINTERS",6,3)
20 PRINT A$
RUN
```

This results in:

```
MEC
```

Example 2

```
10 A$="INTERMEC PRINTERS"
20 B%=10
30 C%=7
40 D$=MID$(A$,B%,C%)
50 PRINT D$
RUN
```

This results in:

PRINTER

MKDIR

Purpose

Creates a directory.

Syntax

MKDIR<*sexp*>

Parameters

<*sexp*>
Specifies the directory to be created.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

<*sexp*> Can end with a slash (/) character, but / is not required.

Do not follow the directory name with a colon (:) character, which may cause unexpected behavior.

Example

```
NEW
MKDIR "DIR1"
SAVE "DIR1/PROGRAM.PRG
FILES "/c/DIR1"
```

This results in:

```
FILES on /c/DIR1
PROGRAM.PRG 2
2220032 bytes free 2 bytes used
```

NAME DATE$

Purpose

Formats the month parameter used in DATE$("F") and DATEADD$(...,"F").

Syntax

NAME DATE$ *<nexp>*, *<sexp>*

Parameters

<nexp>
Month number. Range is 1 to 12.

<sexp>
Desired name of the month.

Notes

NAME DATE$ allows you to assign names to the different months in any form and language you like. The names are returned instead of the corresponding numbers in connection with DATE$("F") and DATEADD$("F"), provided that FORMAT DATE$ has been executed.

The number of characters assigned to represent months in the FORMAT DATE$ statement decides how much of the names, as specified in the NAME DATE$ statement, will be returned.

The Y, M, and D characters return information from right to left.

Example:

```
FORMAT DATE$ "YY.MMM:DD"
NAME DATE$ 1,"JANUARY"
PRINT DATE$("F")
```

This results in:

```
03.ARY.06
```

Usually, it is best to restrict the month parameter in the FORMAT DATE$ statement to 2 or 3 characters (MM or MMM) and enter the names of the months in the NAME DATE$ statement accordingly.

Example

This example shows how to make the printer return dates in a U.S. format:

```
10 DATE$="090115"
20 NAME DATE$ 1, "JAN"
30 NAME DATE$ 2, "FEB"
40 NAME DATE$ 3, "MAR"
50 NAME DATE$ 4, "APR"
60 NAME DATE$ 5, "MAY"
70 NAME DATE$ 6, "JUN"
```

```
80 NAME DATE$ 7, "JUL"
. . . . .
140 FORMAT DATE$ "MMM DD, YYYY"
150 PRINT DATE$("F")
RUN
```

This results in:

```
JAN 15, 2009
```

NAME WEEKDAY$

Purpose

Formats the day parameter in return strings of WEEKDAY$.

Syntax

NAME WEEKDAY$ *<nexp>*, *<sexp>*

Parameters

<nexp>
Number of the weekday according to the WEEKDAY$ function syntax (Monday = 1... Sunday = 7).

<sexp>
Name of the weekday. Default is the full English name in lowercase characters (monday, tuesday, and so on).

Notes

NAME WEEKDAY$ allows you to assign names to the different weekdays in any form and language. The names are returned instead of the corresponding numbers in connection with WEEKDAY$ function.

Example

This example shows how to make the printer return the name of the weekday as an English 3-letter abbreviation:

```
10 FORMAT DATE$ ", MM/DD/YY"
20 DATE$="091201"
30 NAME WEEKDAY$ 1, "Mon"
40 NAME WEEKDAY$ 2, "Tue"
50 NAME WEEKDAY$ 3, "Wed"
60 NAME WEEKDAY$ 4, "Thu"
70 NAME WEEKDAY$ 5, "Fri"
80 NAME WEEKDAY$ 6, "Sat"
90 NAME WEEKDAY$ 7, "Sun"
100 PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN
```

This results in:

```
Fri, 12/01/09
```

NASC

Purpose

Selects a single- or double-byte character set.

Syntax

NASC<*nexp*>|<*sexp*>

Parameters

<*nexp*>
Reference number of a character set as described in the next table. Leading zeroes are ignored:

Character Set	Reference
Roman 8	1 (default)
UTF-8	8 or "UTF-8"
UTF-16 Little Endian*	"UTF-16LE"
UTF-16 Big Endian*	"UTF-16BE"
French	33
Spanish	34
Italian	39
Switzerland	41
English (UK)	44
Swedish	46
Norwegian	47
German	49
Japanese Latin (Romají)	81
Portuguese	351
PCMAP	-1
ANSI (same as 1252)	-2
MS-DOS Latin 1	850
MS-DOS Greek 1	851
MS-DOS Latin 2	852
MS-DOS Cyrillic	855

Character Set	Reference
MS-DOS Turkish	857
Thai	874
Shift-JIS (Japanese)	932
Simplified Chinese	936
Korean (Hangul)	949
Traditional Chinese (BIG-5)	950
Windows Latin 2 (Central Europe)	1250
Windows Cyrillic (Slavic)	1251
Windows Latin 1 (ANSI, same as -2)	1252
Windows Greek	1253
Windows Latin 5 (Turkish)	1254
Hebrew	1255
Arabic	1256
Windows Baltic Rim	1257
Vietnamese	1258
Johab/Wansung	1361
GB18030 Simplified Chinese	"GB18030"

* UTF-16 encodings only support double-byte Unicode characters.

<sexp> is either:

- "UTF-8" = UTF-8
- "file name" = NASC file character set.

Notes

For complete character set tables, see Character Sets.

By default, after processing any MAP statements, Fingerprint prints all characters according to the Roman 8 character set. However, Fingerprint includes a number of other character sets useful for displaying information in other languages or countries, or for adapting the printer to the host operating system. Thus, an ASCII code received by the printer may print or display a different character depending on the selected 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. For more information, see Complex Scripts.

If none of the character sets available contains the desired character(s), use a MAP statement to reMAP the character set that comes closest to your needs. Note that MAP statements are processed before NASC statements.

A NASC statement has these consequences:

- The font used is the one last specified by the FONT command. If none has been specified, the default (Univers) is used.
- Text on labels is printed according to the selected character set. However, parts of the label that have already been processed and stored in the print buffer (before the NASC statement is executed) are not affected. This allows for multiple character sets in a single label, which could be used to create multi-lingual labels.
- New messages in the printer's screen are affected by a NASC statement. However, a message that is already displayed is not updated automatically. The screen can show all printable characters. In the Setup mode, all characters are mapped according to US-ASCII standard.
- Data transmitted via any of the communication channels is not affected, since the data is defined as 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, the screen of the host).
- The pattern of printed bar codes 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 already processed and stored in the print buffer is not affected.

Example

This example selects the Italian character set, prints the character corresponding to 123 decimal in that set, changes the set to Swedish, and prints the character corresponding to 123 decimal:

```
10 NASC 39
20 PRTXT CHR$(123)
30 NASC 46
40 PRTXT CHR$(123)
50 PRINTFEED
```

This results in:

àä

NASCD

Purpose

Selects a single- or double-byte character set.

This command is identical to NASC, and is preserved for compatibility purposes.

NEW

Purpose

Clears the printer working memory in order to allow a new program to be created.

Syntax

NEW

Notes

The NEW statement deletes the program currently residing in the printer working memory, closes all files, and clears all variables and breakpoints. Note that clearing the printer working memory does not imply that the host screen is cleared too. The lines of the previous program remain on screen until replaced by new lines.

If the current program has not been saved (see SAVE statement), it is lost and cannot be restored.

In the Direct Protocol, all counters are removed when a NEW statement is executed.

NORIMAGE

Purpose

Returns the print mode to normal printing after an INVIMAGE statement has been issued. This command can be abbreviated as NI.

Syntax

NORIMAGE

or

NI

Notes

Normal image is the default type of printing. Text and images are printed in black-on-white.

Using an INVIMAGE statement, the printing of text and images can be inverted. Such inverse printing is discontinued for all PRTXT and PRIMAGE statements that follow a NORIMAGE statement.

Example

In this example, the first line is printed in inverted fashion and the second line in the normal fashion:

```
10 PRPOS 30,300
20 ALIGN 4
30 INVIMAGE
40 FONT "Univers"
50 PRTXT "INVERSE PRINTING"
60 PRPOS 30, 200
70 NORIMAGE
80 PRTXT "NORMAL PRINTING"
90 PRINTFEED
RUN
```

ON BREAK GOSUB

Purpose

Branches to a subroutine when a break interrupt instruction is received.

Syntax

ON BREAK<*nexp*>GOSUB<*ncon*>|<*line label*>

Parameters

<*nexp*>
0: "console:"
1: "uart1:"
6: "usb1:" (Supported when virtual COM port is enabled)

<*ncon*>|<*line label*>
Number or label of the program line to which the program will branch on BREAK.

Notes

ON BREAK GOSUB is closely related to BREAK and BREAK ON/OFF. When break interrupt is enabled (see BREAK ON) and the operator issues a break interrupt instruction (see BREAK), program execution is interrupted and branched to a specified line in a subroutine.

Example

In this example, the printer emits a special signal when a break interrupt is issued from the printer F1 key:

```
10 BREAK 0, 1 : 'FI key
20 BREAK 0 ON
30 ON BREAK 0 GOSUB 1000
40 GOTO 20
. . . . .
. . . . .
1000 FOR A%=1 TO 3
1010 SOUND 440,50
1020 SOUND 349,50
1030 NEXT A%
1040 END
```

ON COMSET GOSUB

Purpose

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

Syntax

ON COMSET<*nexp1*>GOSUB<*nexp2*>|<*line label*>

Parameters

<*nexp1*>
Communication channel:

0: "console:"
1: "uart1:"
2: "uart2:"
3: "uart3:"
4: "centronics:"
5: "net1:"
6: "usb1:"
7: "uart4:"
8: "uart5:"
9: "usbhost:"
10: "bluetooth:"
11: "ftp1:"
12: "http1:"
13: "lpr1:"

<*nexp2*>|<*linelabel*>
Number or label of the program line to be branched to.

Note

ON COMSET GOSUB is closely related to COMSET, COMSTAT, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. Use this statement to branch to a subroutine when:

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

These three parameters are set for the specified communication channel by a COMSET statement.

Examples

In this example, the program branches to a subroutine to read the buffer of the communication channel:

```
1 REM Exit program with #STOP&
10 COMSET1,"#","&","ZYX","=",50
```

```
20 ON COMSET 1 GOSUB 2000
30 COMSET 1 ON
40 IF A$ <> "STOP" THEN GOTO 40
50 COMSET 1 OFF
.....
.....
1000 END
2000 A$= COMBUF$(1)
2010 PRINT A$
2020 COMSET 1 ON
2030 RETURN
```

The same example written without line numbers looks like this:

```
IMMEDIATE OFF
REM Exit program with #STOP&
COMSET1,"#","&","ZYX","=",50
ON COMSET 1 GOSUB QQQ
COMSET 1 ON
WWW: IF A$ <> "STOP" THEN GOTO WWW
COMSET 1 OFF
.....
.....
END
QQQ: A$=COMBUF$(1)
PRINT A$
COMSET 1 ON
RETURN
IMMEDIATE ON
```

ON ERROR GOTO

Purpose

Branches to an error-handling subroutine when an error occurs.

Syntax

ON ERROR GOTO<*ncon*>|<*line label*>

Parameters

<*ncon*>|<*line label*>
Number or label of the line to which the program should branch when an error condition occurs.

If an error condition occurs after this statement has been encountered, the standard error-trapping routine is ignored and the program branches to the specified line, which should be the first line in an error-handling subroutine.

If the line number is 0, the standard error-trapping routine is enabled and no error-branching within the current program is executed.

Examples

If you try to run this example with the printhead raised (or if any other error occurs), a warning signal sounds and the error LED turns on.

```
10 LED 0 ON:LED 1 OFF
20 ON ERROR GOTO 1000
30 FONT "Univers"
40 PRTXT "HELLO"
50 PRINTFEED
60 END
. . . . .
1000 LED 0 OFF:LED 1 ON
1010 FOR A%=1 TO 3
1020 SOUND 440,50
1030 SOUND 359,50
1040 NEXT A%
1050 RESUME NEXT
```

The same example written without line numbers looks like this:

```
IMMEDIATE OFF
LED 0 ON:LED 1 OFF
ON ERROR GOTO QQQ
FONT "Univers"
PRTXT "HELLO"
PRINTFEED
END
. . . . .
QQQ: LED 0 OFF:LED 1 ON
```

```
FOR A%=1 TO 3
SOUND 440,50
SOUND 359,50
NEXT A%
RESUME NEXT
IMMEDIATE ON
```

ON GOSUB

Purpose

Branches conditionally to one of several subroutines, determined by input value.

Syntax

ON<*nexp*>GOSUB<*ncon*>|<*line label*>[,<*ncon*>|<*line label*>...]

Parameters

<nexp>
Numeric expression determining the index of the subroutine to branch to.

<ncon>|<line label>
Line label, number, or list to which the program should branch.

Notes

ON GOSUB is closely related to ON GOTO. The numeric expression may result in any positive value. The expression is truncated to an integer value before the statement is executed. If the resulting value is negative, 0, or larger than the number of subroutines, the statement is ignored.

The value of the numeric expression determines which of the subroutines the program should branch to. For example, if the value of the numeric expression is 2, the program will branch to the second subroutine in the list.

Examples

In this example, different text strings appear on the screen depending on which key (1, 2, or 3) you press on the host keyboard.

```
10 INPUT "PRESS KEY 1-3 ", A%
20 ON A% GOSUB 1000,2000,3000
30 END
1000 PRINT "You have pressed key 1"
1010 RETURN
2000 PRINT "You have pressed key 2"
2010 RETURN
3000 PRINT "You have pressed key 3"
3010 RETURN
```

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
INPUT "PRESS KEY 1-3 ", A%
ON A% GOSUB QQQ,WWW,ZZZ
END
QQQ: PRINT "You have pressed key 1"
RETURN
WWW: PRINT "You have pressed key 2"
RETURN
```

```
ZZZ: PRINT "You have pressed key 3"
RETURN
IMMEDIATE ON
```

ON GOTO

Purpose

Branches unconditionally to one of several subroutines, determined by input value.

Syntax

ON<*nexp*>GOTO<*ncon*>|<*line label*>[,<*ncon*>|<*line label*>...]

Parameters

<nexp>
Numeric expression determining the index of the subroutine to branch to.

<ncon>|<line label>
Line label, number, or list to which the program should branch.

Notes

This statement is closely related to the ON GOSUB statement. The numeric expression may result in any positive value. The expression is truncated to an integer value before the statement is executed. If the resulting value is negative, 0, or larger than the number of lines, the statement is ignored.

<nexp> determines which of the lines the program should branch to. For example, if *<nexp>* = 2, the program branches to the second line in the list.

Examples

In this example, different text is printed on the screen depending on which of the keys (1, 2, or 3) you press on the host keyboard.

```
10 INPUT "PRESS KEY 1-3 ", A%
20 ON A% GOTO 1000,2000,3000
30 END
1000 PRINT "You have pressed key 1"
1010 GOTO 30
2000 PRINT "You have pressed key 2"
2010 GOTO 30
3000 PRINT "You have pressed key 3"
3010 GOTO 30
```

The same example written without line numbers looks like this:

```
IMMEDIATE OFF
INPUT "PRESS KEY 1-3 ", A%
ON A% GOSUB QQQ,WWW,ZZZ
YYY: END
QQQ: PRINT "You have pressed key 1"
GOTO YYY
WWW: PRINT "You have pressed key 2"
GOTO YYY
ZZZ: PRINT "You have pressed key 3"
```

```
GOTO YYY
IMMEDIATE ON
```

ON HTTP GOTO

Purpose

Branches to a subroutine when a request for an application CGI is received.

Syntax

ON HTTP GOTO*<ncon>|<line label>*

Parameters

<ncon>|<line label>
Number or label of the line to which the program will branch when the CGI request is received.

Notes

This statement can be used by any printer with a Wi-Fi or Ethernet connection, and defines a Fingerprint subroutine that handles the CGI-request. Setting the line number or line label to 0 disables the handler.

When a request for an application CGI is received, the current execution point is pushed on to the stack and the execution commences in the handler with STDIN and STDOUT redirected to or from the Web browser.

A related instruction is RESUME HTTP.

ON KEY GOSUB

Purpose

Branches to a subroutine when a specified key on the printer front panel is activated.

Syntax

ON KEY(*<nexp>*)GOSUB*<ncon>*|*<line label>*

Parameters

<nexp>
ID number of one of the keys on the printer front panel.

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

Notes

Each of the keys on the printer keypad can be enabled individually using its ID number in a KEY ON statement. Then the key can be assigned, alone or in combination with the **Shift** key, to make the program branch to a subroutine using an ON KEY... GOSUB statement.

Shift adds 100 to the unshifted ID number of each key.

Note the difference between the ID numbers of the keys and the ASCII values they are able to produce (see BREAK for more information).

BREAK takes precedence over any ON KEY statement, provided that break interrupt is not disabled for the "console:" by a BREAK 0 OFF statement.

For more information about printer keypad layouts, see Printer Keypad Layouts.

Examples

This example illustrates how activating the **F1** key (ID = 10) makes the program branch to a subroutine containing the PRINTFEED statement. Note line 30 where the execution waits for the key to be pressed.

```
10 ON KEY (10) GOSUB 1000
20 KEY (10) ON
30 GOTO 30
.....
.....
.....
1000 FONT "Univers"
1010 PRPOS 30,100
1020 PRTXT "HELLO"
1030 PRINTFEED
1040 END
RUN
```

The same example can be written without line numbers this way:

```
IMMEDIATE OFF
ON KEY (10) GOSUB QQQ
KEY (10) ON
WWW: GOTO WWW
.....
.....
.....
QQQ: FONT "Univers"
PRPOS 30,100
PRTXT "HELLO"
PRINTFEED
END
IMMEDIATE ON
RUN
```

ON MIBVAR& GOSUB

Purpose

Branches to a subroutine when a specified SNMP MIB variable is changed.

Syntax

ON MIBVAR&(*<nexp>*)GOSUB*<ncon>*|*<line label>*

Parameters

<nexp>
Number of a MIBVAR& variable. Range is 0 to 9.

<ncon>|*<line label>*
Number or label of the line to which the program will branch when the specified MIB variable is modified.

Notes

This statement makes the program branch to a subroutine whenever the specified MIB variable is modified. This value is then maintained during the entire subroutine execution. External changes during the subroutine execution cause a new subroutine call at return. A related instruction is MIBVAR&.

Example

This example prints the value of MIBVAR&(1) on a label every time its value is changed. Note that the program is idle (on line 20) as long as the MIBVAR& variable remains unchanged.

```
10 ON MIBVAR&(1) GOSUB 1000
20 GOTO 20
.....
.....
.....
1000 FONT "Univers"
1010 PRPOS 30,100
1020 A$ = MIBVAR&(1)
1030 PRTXT A$
1040 PRINTFEED
1050 RETURN
RUN
```

ON/OFF LINE

Purpose

Controls the SELECT signal on the "centronics:" communication channel.

Syntax

ON|OFFLINE*<nexp>*

Parameters

<nexp>
Specifies the communication channel.

4: "centronics:"

Notes

Pin 13 in the Centronics/IEEE 1284 interface connector contains the SELECT signal:

- ON LINE 4 sets the SELECT signal high.
- OFF LINE 4 sets the SELECT signal low.

If no ON/OFF LINE statement is issued, the SELECT signal is high (the Centronics channel will be ON LINE). ON LINE/OFF LINE for the serial channel "usb1:" is implemented according to USB Device Class for Printing Devices v1.09, January 2000.

Example

In this example, the "centronics:" communication channel is disabled, a new setup is performed on the printer by means of a setup file, and the channel is enabled:

```
10 OFF LINE 4
20 SETUP "New Setup.SYS"
30 ON LINE 4
. . . . .
. . . . .
. . . . .
```

OPEN

Purpose

Opens a file or device (such as a network connection), or creates a new file for input, output, or append, allocating a buffer, and specifies the access mode.

Syntax

OPEN<*sexp*>[FOR<INPUT|OUTPUT|APPEND>]AS [#]<*nexp1*>[LEN=<*nexp2*>]

Parameters

<sexp>
File or device to be opened, or the file to be created. File names must not contain a colon character (:). A network connection can be specified.

#
(Optional) Indicates that whatever follows is a number.

<nexp1>
Designation number for the file or device opened using the OPEN command.

<nexp2>
(Optional) Length of the record in bytes. Default is 128.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

An OPEN statement must be executed before a file or device can be used for input, output, and/or append. A maximum of 25 files and/or devices can be open at the same time.

You must CLOSE the file or it will not be saved if the printer is turned off.

Sequential Access Mode

The access mode can optionally be specified as sequential INPUT, OUTPUT, or APPEND:

- INPUT: Sequential input from the file or device, replacing existing data. Existing files/devices only.
- OUTPUT: Sequential output to the file or device, replacing existing data.
- APPEND: Sequential output to the file or device, where new data will be appended without replacing existing data.

Random Access Mode

If no access mode is specified in the statement, the file or device is opened for both input and output (RANDOM access mode). FIELD, LSET, RSET, PUT, and GET can only

be used on records in files opened in the RANDOM access mode using the OPEN command.

See DEVICES for information on which devices can be opened for the different modes of access.

Use FILES to get lists of the files stored in the printer memory.

Network Connections

Special syntax as follows can be used for the <*sexp*> expression that opens the Net1 device:

OPEN "net1:<*host*>[:*port*][,<*timeout*>]" AS #<*nexp*>

where:

<*host*> is an IP address or DNS name.

[:*port*] is the port number. Range is 0 to 65535. Default is 9100.

<*timeout*> is the timeout value expressed in ticks (0.01 sec). Range is 10 to 6000. Default is 4500 (45 sec).

Only one Net1 connection can be opened at a time. Once the device is open, information can be sent or received according to the corresponding protocol.

Example 1

This example opens and writes to a file:

```
10 OPEN "TEST.TXT" FOR OUTPUT AS 1
20 PRINT#1,"AAAA"
30 PRINT#1,"!234"
40 CLOSE#1
```

Example 2

This example opens the file and prints it line by line:

```
10 OPEN "TEST.TXT" FOR INPUT AS #2
20 WHILE NOT EOF(2)
30 INPUT#2,qA$
40 PRINT qA$
50 WEND
60 CLOSE #2
RUN
```

Example 3

You can allow sequential output to the printer's screen using the OPEN statement this way. The method is the same as for files:

```
10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT#1:PRINT#1
30 PRINT#1, "GONE TO LUNCH"
40 PRINT#1, "BACK SOON";
RUN
```

The text appears as:

```
GONE TO LUNCH
 BACK SOON
```

Example 4

This example opens the file "PRICELIST" for random access with the reference number #8 and a record length of 254 bytes:

```
10 OPEN "PRICELIST" AS #8 LEN=254
```

Example 5

The following example opens the Net1 device to connect to the NIST Internet Time Service (ITS) web site. Row 10 specifies socket port 13. Rows 30-40 create a delay to allow the result to be received as A$ within the specified time of 500 ticks:

```
10 OPEN "net1:time.nist.gov:13" AS 1
20 PRINT #1, "time?"
30 QTICKS% = TICKS
40 WHILE TICKS-QTICKS% < 500
50 WEND
60 LINE INPUT #1, A$
70 PRINT A$
80 CLOSE 1
```

OPTIMIZE "BATCH" ON/OFF

Purpose

Enables or disables optimization for batch printing.

Syntax

OPTIMIZE "BATCH" ON|OFF

Parameters

ON|OFF
Enables or disables optimization. Default is disabled (OFF). In Direct Protocol, default is enabled (ON).

Notes

This facility is intended to speed up batch printing (uninterrupted printing of large numbers of identical or similar labels). The program execution does not wait for the printing of the label to be completed, but proceeds executing the next label image into the other of the two image buffers as soon as possible.

OPTIMIZE "BATCH" is automatically enabled (ON) during a batch (PRINTFEED more than one) if the following conditions are fulfilled:

1 LTS& OFF (default)

2 CUT OFF (default)

Examples

Run these two examples and note the differences in the printer performance:

```
10 OPTIMIZE "BATCH" ON
20 FOR I%=1 TO 10
30 PRTXT I%
40 PRINT "Before printfeed"
50 PRINTFEED
60 PRINT "After printfeed"
70 NEXT
RUN
```

```
10 OPTIMIZE "BATCH" OFF
20 FOR I%=1 TO 10
30 PRTXT I%
40 PRINT "Before printfeed"
50 PRINTFEED
60 PRINT "After printfeed"
70 NEXT
RUN
```

PORTIN

Purpose

Reads the status of a port on a Serial/Industrial Interface Board, or checks the current state of a specific signal (supported only when applicator port status is enabled).

This command is applicable only for the PM23c, PM43, and PM43c printers.

Syntax

PORTIN(*<nexp>*)|.*<sexp>*

Parameters

<nexp>
Number of the port to be read:

IN ports (optical): 101-108 (301-308)
OUT ports (relay): 201-204 (401-404)
OUT ports (optical): 221-228 (421-428)

.*<sexp>*
(With enabled applicator port only) Specific signal to check the state of. Returns -1 if the signal is currently asserted, or 0 if the signal is de-asserted.

Valid values for IN signals:
STARTPRINT
FEED
REPRINT
PAUSE
APPERR1
APPERR2
APPERR3
RTWIINEXT

Valid values for OUT signals:
RIBBONLOW
MEDIALOW
SERVICEREQ
ENDPRINT
MEDIAOUT
RIBBONOUT
RFIDTAGERROR
DATAREADY
RTWOUT

Notes

Values for .*<sexp>* are valid only if the applicator port on the printer is enabled.

This function works with the Serial/Industrial Interface Board and can read the status of 8 IN ports with optocouplers, 8 OUT ports with optocouplers, and 4 OUT ports with relays. For information on how to set the OUT ports, see the PORTOUT statement.

A current can be led through an optocoupler in each IN port:

- If the current is on, the PORTIN function returns the value -1 (true).
- If the current is off, the PORTIN function returns the value 0 (false).

This feature allows the execution of Fingerprint to be controlled by various types of external sensors or non-digital switches.

The status of the OUT ports, as set by PORTOUT statements, can also be read by PORTIN functions.

Some printers have more than one serial interface. In this case, the ports on the inner board (that is, the board closest to the CPU board) are specified by the low numbers (101-108, 201-204, and 221-228) while the ports on the outer board are specified by the high numbers (301-308, 401-404, and 421-428).

See the documentation of the Serial/Industrial Interface Board for more information.

Example

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

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

PORTOUT ON/OFF

Purpose

Sets relay ports or optical ports on a Serial/Industrial Interface Board to either on or off.

This command is applicable only for the PM23c, PM43, and PM43c printers.

Syntax

PORTOUT[(<*nexp*>) ON|OFF]

Parameters

<*nexp*>
Number of the port to be set:

OUT ports (relay): 201 to 204 (401 to 404)
OUT ports (optical): 221 to 228 (421 to 428)

See the Notes for more information on port numbers.

Notes

This statement works with the Serial/Industrial Interface Board and is able to control 8 IN ports with optocouplers, 8 OUT ports with optocouplers, and 4 OUT ports with relays. For information on how to read the status of the various ports, see the PORTIN function.

This feature allows Fingerprint to control various external devices like gates, lamps, or conveyor belts.

Some printers have more than one serial port. In this case, the ports on the inner board (that is, the board closest to the CPU board) are specified by the low numbers (201 to 204 and 221 to 228) while the ports on the outer board are specified by the high numbers (401 to 404 and 421 to 428).

See the documentation of the Serial/Industrial Interface Board for more information.

Example 1

The relay of OUT port 201 on a Serial/Industrial Interface Board is Opened and then Closed like this:

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

POWER

Purpose

Places the printer in either deep sleep or Standby mode, or wakes the printer when it is in Standby mode.

Syntax

POWER OFF|STANDBY

POWER OFF is only applicable for PC23d, PC43d, and PC43t printers.

Parameters

OFF sends the printer into deep sleep mode, saving power. If no applications are listed in the autoexec.bat file (to auto-start when the printer boots), the printer restarts in Fingerprint mode.

STANDBY alternately places and wakes the printer from Standby mode. If the printer has an LCD screen, the POWER STANDBY command turns the LCD screen off.

Notes

To turn off the Ready-to-Work™ indicator and the LCD backlight, the printer sleep timer must be set to ALWAYS OFF before issuing a POWER STANDBY command. When the printer wakes up after being in Standby mode, the application should reset the sleep timer to the usual operating value.

Example

This example can be used to test the power implementation for Fingerprint applications on the printer. The application performs different operations depending on button presses.

```
10 ON ERROR GOTO ZERROR
20 QSTANDBY%=0
30 KEY 16 ON
40 ON KEY 16 GOSUB ZKEYA
50 KEY 17 ON
60 ON KEY 17 GOSUB ZKEYB
70 KEY 19 ON
80 ON KEY 19 GOSUB ZKEYC
90 KEY 20 ON
100 ON KEY 20 GOSUB ZKEYD
110 PRINT "starting"
120 ZSTART:
130 GOTO ZSTART
140 ZKEYA:
150 PRINT "POWER OFF"
160 RUN "sleep 1"
170 POWER OFF
180 RETURN
190 ZKEYB:
200 PRINT "FORMFEED"
```

```
210 RUN "sleep 1"
220 FORMFEED
230 RETURN
240 ZKEYD:
250 PRINT "SETUP"
260 RUN "sleep 1"
270 SETUP
280 RETURN
290 ZKEYC:
300 PRINT "STANDBY"
310 RUN "sleep 1"
320 IF QSTANDBY%=1 THEN
330 SETUP "POWER,SLEEP TIMER,ALWAYS ON"
340 QSTANDBY%=0
350 ELSE
360 SETUP "POWER,SLEEP TIMER,ALWAYS OFF"
370 QSTANDBY%=1
380 END IF
390 POWER STANDBY
400 RETURN
410 ZERROR:
420 PRINT ERR
430 IF ERR=1022 THEN
440 END
450 END IF
460 RESUME NEXT
```

PRBAR

Purpose

Provides input data to a bar code. This command can be abbreviated as PB. You can use a PRBAR command to generate AddOn bar codes for EAN and UPC symbologies.

Syntax

PRBAR<<*sexp*>|<*nexp*>>

or

PB<<*sexp*>|<*nexp*>>

Parameters

<<*sexp*>|<*nexp*>>
Input data to the bar code generator.

Notes

The bar code must be defined by BARSET, BARTYPE, BARRATIO, BARHEIGHT, BARMAG, BARFONT, and/or BARFONT ON/OFF statements, or by the corresponding default values. Make sure that the type of input data (numeric or string) and the number of characters agree with the specification for the selected bar code type.

Example

Two different bar codes, one with numeric input data and one with string input data, can be generated this way. You can also enter the input data using a variable.

```
10 BARFONT "Univers", 8 ON
20 PRPOS 50,400
30 ALIGN 7
40 BARSET "INT2OF5",2,1,3,120
50 PRBAR 45673
60 PRPOS 50,200
70 BARSET "CODE39",3,1,2,100
80 pC39$ = "ABC"
90 PRBAR qC39$
100 PRINTFEED
RUN
```

PRBOX

Purpose

Creates a box containing a single text line or a frame of multiple hyphenated text lines. This command can be abbreviated as PX.

Syntax

PRBOX<*nexp1*>,<*nexp2*>,<*nexp3*>[,<*sexp1*>[,<*nexp4*>[,<*nexp5*>[,<*sexp2*>[,<*sexp3*>]]]]]

or

PX<*nexp1*>,<*nexp2*>,<*nexp3*>[,<*sexp1*>[,<*nexp4*>[,<*nexp5*>[,<*sexp2*>[,<*sexp3*>]]]]]

Parameters

<*nexp1*>
Box height in dots. Range is 1 to 6000.

<*nexp2*>
Box width in dots. Range is 1 to 6000.

<*nexp3*>
Line weight in dots. Range is 0 to 6000.

<*sexp1*>
Text to be written inside the box. Maximum of 20 lines. Single-byte fonts and code points only (for example, no UTF-16 or multi-byte UTF-8 code points).

<*nexp4*>
Horizontal distance between inner edge of the box line and the text frame. Range is -100 to 100 dots. Default is 0.

<*nexp5*>
Vertical distance between the inner edge of the box line and text frame, and between each line of text in the frame. Range is -100 to 100 dots. Default is the same value as <*nexp4*>.

<*sexp2*>
Line delimiter (maximum 9 characters) that replaces the default delimiter string CHR$(10) or CHR$(13). Each time this delimiter is encountered in the text string (<*sexp1*>), the remaining text wraps to the next line.

<*sexp3*>
is a control string for hyphen delimiter and replacement.

Notes

PRBOX creates a transparent, rectangular region surrounded by a line, or specifies a text frame that can contain up to 20 lines of hyphenated text. These two purposes can be combined so the text frame is surrounded by a black box.

PRBOX supports inline formatting adjustments, such as boldface or italicization. For more information, see Inline Text Formatting.

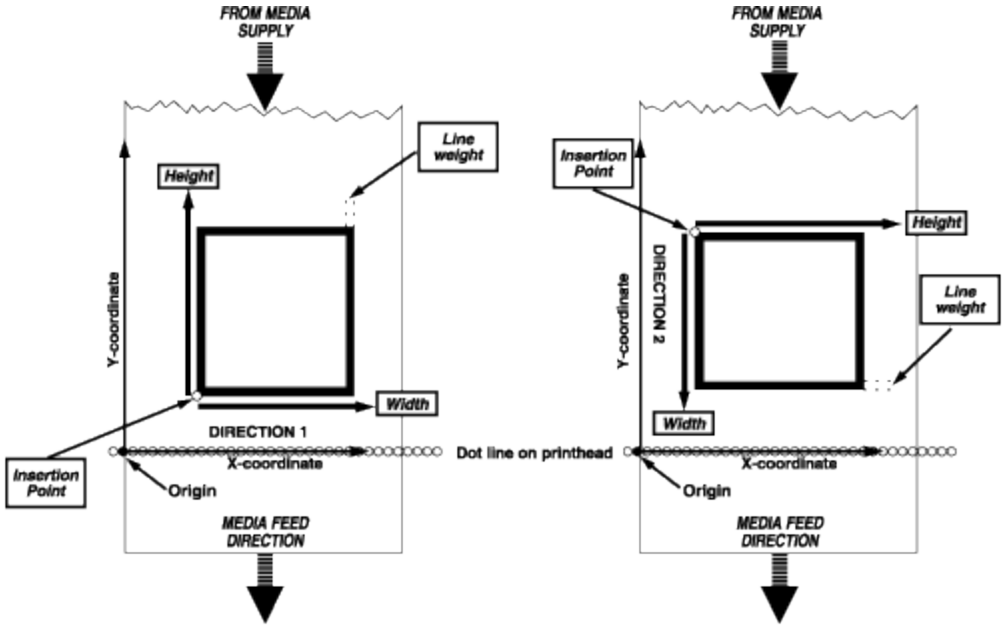
Creating a Simple Box

To create a simple box, specify the height, width, and line weight. The box is drawn with its anchor point (see ALIGN) at the insertion point, as specified by the nearest preceding PRPOS statement. A box can be aligned left, right, or center along its baseline.

The print direction specifies how the box is rotated in relation to its anchor point.

The line weight increases inward from the anchor point. The heavier the line, the less white area inside the box. Thus, it is possible to create a black area using a box with very heavy lines. For a simple box with no text field, the line weight must be greater than 0, and the white area inside a box can be used for printing. Boxes, lines, and text may cross (also see XORMODE ON/OFF).

The next illustration shows how the height and width of the box are defined for different print directions:



Creating a Multiline Text Field

The PRBOX statement can also create an area in which a field of wrapped and hyphenated text is printed. There is no need to specify each line of text separately as with the PRTXT statement.

The text field can be framed by the box (line weight > 0), or the box can be invisible (line weight = 0). The maximum number of characters on each line is 300 and the maximum number of lines is 20.

The position of the text frame inside the box is affected by the direction (set by DIR) and alignment (set by ALIGN), and by <*nexp4*> and <*nexp5*> in the PRBOX statement.

The direction rotates the box with its text field around the anchor point as specified by the alignment. The alignment specifies the anchor point of the box itself, and determines field alignment inside the box and text justification.

Horizontal and vertical mean different things depending on how the text is printed. For directions 2 and 4, horizontal and vertical have opposite meanings than in directions 1 and 3.

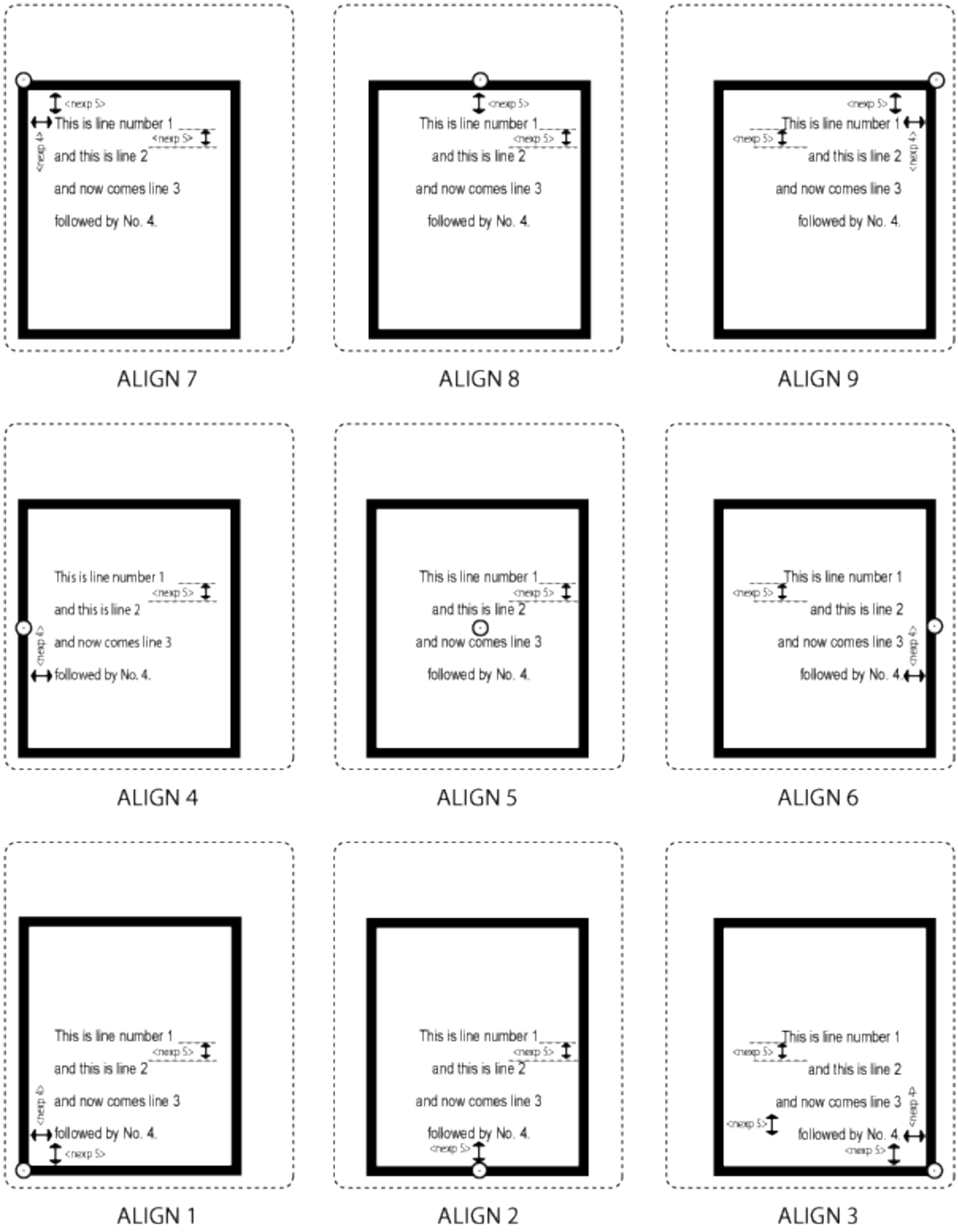
The horizontal distance between the inside of the box line and the borders of the text field is specified by <*nexp4*>:

- For ALIGN 1, 4, or 7, <*nexp4*> determines the distance between the inside of the left side box line and the left-hand edge of the text field.
- For ALIGN 3, 6, or 9, <*nexp4*> determines the distance between the inside of the right side box line and the right-hand edge of the text field.
- For ALIGN 2, 5, or 8, <*nexp4*> is not necessary.

The line spacing and the vertical distance between the inside of the box line and the borders of the text field is specified by <*nexp5*>:

- For ALIGN 1, 2, or 3, <*nexp5*> determines the distance between the inside of the bottom box line and the bottom edge of the text field, as well as line spacing.
- For ALIGN 7, 8, or 9, <*nexp5*> determines the distance between the inside of the right side box line and the right-hand edge of the text field, as well as line spacing.
- For ALIGN 4, 5, or 6, <*nexp5*> determines line spacing only.

The next illustration shows how alignment affects the location of multi-line text:



If the text in *<sexp1>* is entered as a continuous string of characters without any spaces, linefeeds, or carriage returns, the text wraps to the next line when there is no room left for any more characters on a line.

If any combination of a carriage return (CR = ASCII 13 decimal,) and a linefeed (LF = ASCII 10 decimal) is encountered, the remaining text wraps once to the next line.

Space characters (ASCII 32 decimal) also initiate a line wrap. If there is more than one space character, the wrapping is at the last space character that fits into the line in question.

You can replace the default line delimiters (CR, LF, and CR/LF) with another line delimiter specified in a string of up to 9 characters (<*sexp2*>). This delimiter is not printed, even if it is a printable character. Each time the delimiter is encountered, the text wraps to a new line.

Hyphenation Support

You can put "invisible" hyphen delimiters in the text string at suitable wrap-around positions. The default delimiter is a hyphen sign (ASCII 45 decimal). You can use a string of up to nine characters instead, but be careful so the string is not confused with the text.

If a wrap-around is performed, the corresponding hyphen delimiter is printed as a hyphen sign (ASCII 45 decimal) by default, whereas hyphen delimiters not used for wrap-around will not be printed.

To print characters other than hyphens, you can specify a string of hyphen replacement characters. It is possible to use a string of up to 9 characters, but the shorter the string the less likely that a line will wrap outside the box.

If you have a text string with long words and have not inserted all necessary line delimiters, a line-wrap may occur unexpectedly. You can optionally specify a hyphen delimiter for this case as well. Default is no delimiter.

Specify the parameter <*sexp3*> in PRBOX using the following syntax:

<*sexp3*>=<*sexp3a*>[space<*sexp3b*>[space<*sexp3c*>]]

where:

<*sexp3a*> is a soft hyphen delimiter. If the text does not have enough room on one line, the rest of the text is wrapped from the last space or from the position marked by the soft hyphen delimiter.

Exception: Two adjacent soft hyphen delimiters revoke each other. Default is a normal hyphen (-). Maximum length is 9 characters.

space is a string delimiter with the value CHR$(32).

<*sexp3b*> is one or more characters that are printed at the end of a line which has been hyphenated according to a hyphen delimiter (see <*sexp3a*>). Default is a normal hyphen (-). Maximum length is 9 characters, but fewer is preferred.

<*sexp3c*> is a string of hyphen extension characters, used on single words too long to be printed on one line and have no hyphen delimiter specified. The hyphen extension character is printed at the right end of the line and the remainder of the word is printed on the next line. Default is no character. Maximum length is 9 characters.

If <*sexp3*> is not specified, the rules for hyphen delimiter and replacement are the same as for printing hyphens in text. Two adjacent hyphens are printed as one.

Examples

This example draws a rectangle without any text:

```
10 PRPOS 50,50
20 PRBOX 200,400,5
30 PRINTFEED
RUN
```

This program illustrates a multi-line text field with line wrap, where "&S" is the soft hyphen delimiter:

```
10 FONT "Univers",12
20 DIR 1
30 ALIGN 7
40 T$="Hyphen-ated words are divid-ed into sylla-bles.|||-- New Line, Special Cases and
EXTRAORDINARY long words."
50 PRPOS 100,300
60 PRBOX 300,640,1,T$,5,1,"|||","- -"
70 PRINTFEED
RUN
```

PRBUF

Purpose

Receives and prints bitmap image data using the PRBUF protocol.

Syntax

PRBUF<*nexp1*>[,<*nexp2*]<*new line*><*image data*>

Parameters

<*nexp1*>
Number of bytes of the image in PRBUF protocol.

<*nexp2*>
(optional) Timeout between characters in TICKS (0.01 sec). Default is ~12.7 sec/character.

<*new line*>
Any combination of CR, CR/LF, or LF.

<*image data*>
Image according to the PRBUF protocol.

Notes

PRBUF is useful for receiving and printing bitmap images from a source such as a Windows print driver, since it is more effective and requires less memory than using a STORE IMAGE...PRIMAGE sequence. The bitmap image is printed directly and is not saved anywhere in printer memory after the image buffer has been cleared.

At PRBUF, the printer waits for image data to be received on the standard IN channel. Because PRBUF only works with binary transfers, XON/XOFF must be disabled. Optionally, you can set a timeout between characters (default is 12.7 seconds). When the number of bytes specified by PRBUF has been received, image data is processed directly into the image buffer and printed immediately.

PRBUF does not work if <*nexp1*> bytes cannot be allocated. If memory is low, it is possible to download the bitmap image in two or more blocks. Field settings (alignment, clipping, direction, xor mode, inverse image, magnification, x-position, and y-position) are handled by the current protocol, but only x- and y-positions, field clipping, and xor mode are handled. Other attributes are ignored.

If PRPOS x,y, then the real print position is PRPOS x,y+1.

<*new line*> is not part of the statement, but any combination of carriage return (ASCII 13 decimal,) and/or linefeed (ASCII 10 decimal) is allowed without interfering with the PRBUF protocol.

For more information on how to structure your image data, see PRBUF protocol.

Example

This example shows how the printer is instructed to receive and print 1,424 bytes of image data according to the PRBUF protocol:

PRBUF 1424
<binary image data>

PRBUF Protocol

Purpose

This protocol is used to send print image data from an application to the printer image buffer. The protocol consists of a two-byte header and a number of data bytes in the RLL (Run Length Limited) format.

Header Syntax

The first byte is the @ symbol (Unicode 0x0040), which indicates the start of the protocol header.

The second byte is a numeric value from 0 to 255:

- 0: Reserved (bitmap format)
- 1: Reserved (RLL image format)
- 2: RLL buffer format
- 3-255: Reserved

RLL Buffer Format

The RLL buffer format is optimized for use by Windows drivers, because image processing is more efficient on the host PC rather than the printer.

- Bytes 1 and 2 specify the pixel width (unsigned) of the data in big-endian format for one line.
- Bytes 3 and 4 specify the pixel height (unsigned) of the buffer when it is expanded, in big-endian format.
- The remaining bytes (5+) specify the bitmap in RLL format.

RLL Buffer Format Example

This shows a RLL buffer protocol header for a 515 × 212 pixel hexdump:

40 02 02 03 00 d4

RLL Format

The RLL format is good for black and white pixels, and compresses data in both dimensions. It works well with one-dimensional bar codes, but grayscales grow in size instead of shrinking. The format is symmetric, so all pixel runs begin and end with a white pixel, and lines can be repeated whenever possible. This makes the format possible to turn upside-down.

The overall RLL format uses these parameters:

<start><toggling pixel runs><stop>

The width of these parameters is equal to the total width of the RLL pattern. The parameters can contain additional information.

start
<line repetitions> or *<small white pixel run>*

stop
<start> or <empty>

toggling pixel runs
<white and black pixel runs> or *<black and white pixel runs>* or *<white pixel run>* or *<empty>*

white and black pixel run
<white pixel run><black pixel run> or *<small white pixel run><black pixel run>*

black and white pixel run
<black pixel run><white pixel run>

line repetitions
((-1)-(-128))*-1 number of equal lines

small white pixel run
0 through 127, the number of white pixels

black pixel run
0 through 255, the number of black pixels

white pixel run
0 through 255, the number of white pixels

empty
empty (used when the line fits in one pixel run)

If there is no line repetition, line repeats are not necessary. If the pixel run is out of range, it must be split into several runs.

RLL Format Examples

An eight-bit pattern:

```
 * * * *      1,1,1,1,1,1,1,1,0      Note the last 0, which ends the line with a white pixel run
* * * *       0,1,1,1,1,1,1,1,1      Begins with a white pixel run of 0 pixels
 **  **      2,2,2,2,0              Stopped with a white pixel run of 0 pixels
**  **        -2,0,2,2,2,2,-2        Line and pixel repetitions
**  **
```

A black square, 800 dots per side:

```
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-128,0,255,0,255,0,255,0,35,0,-128
-32,0,255,0,255,0,255,0,35,0,-32
```

Example

This example shows how the printer is instructed to receive and print 1,424 bytes of image data according to the PRBUF protocol:

PRBUF 1424
<binary image data>

PRDIAGONAL

Purpose

This command creates a diagonal line. This command can be abbreviated as PD.

Syntax

PRDIAGONAL<*nexp1*>, <*nexp2*>, <*nexp3*>, <*sexp1*>

Parameters

<*nexp1*>
The width of the imaginary box that the diagonal line creates in dots. Only values greater than 0 are accepted.

<*nexp2*>
The height of the imaginary box that the diagonal line creates in dots. Only values greater than 0 are accepted.

<*nexp3*>
The thickness of the diagonal line in dots. Only values greater than 0 are accepted.

<*sexp1*>
The orientation of the diagonal line. Enter "L" to specify a left leaning diagonal or "R" to specify a right leaning diagonal.

Notes

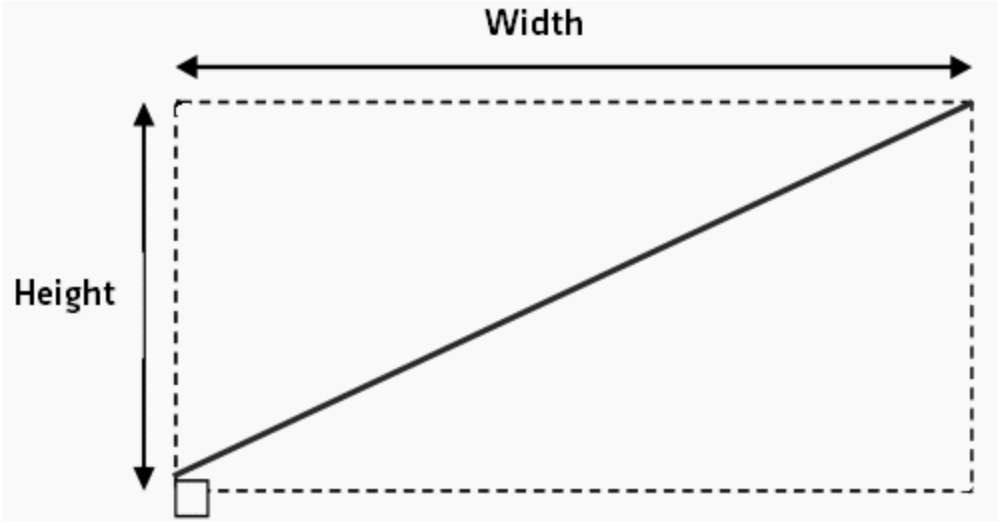
The alignment for the diagonal line can be set using the ALIGN (AN) command.

The print direction of the diagonal line can be set using the DIR command.

Examples

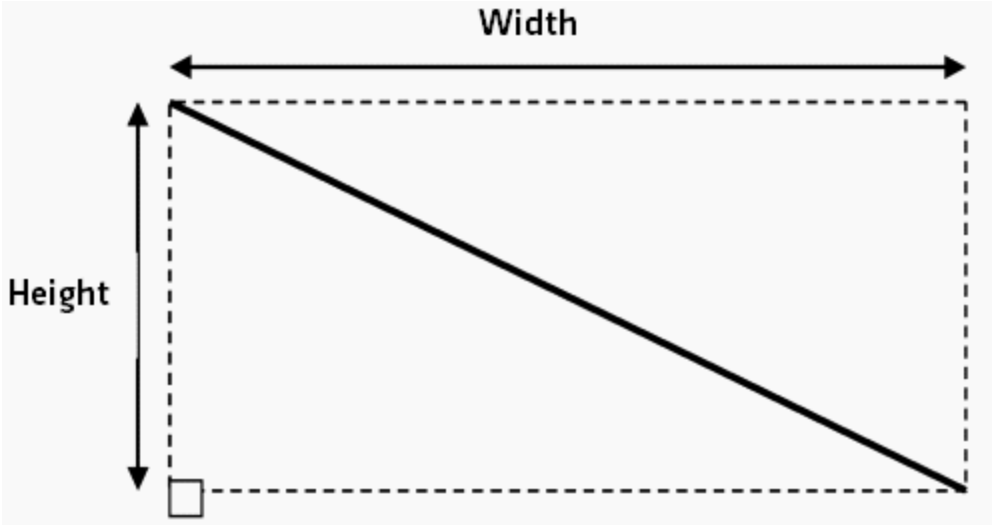
This example prints a right leaning diagonal line:

```
DIR 1
AN 1
PP 100, 300
PD 330, 183, 10,"R"
PF
```



This example prints a left leaning diagonal line:

```
DIR 1
AN 1
PP 100, 300
PD 330, 183, 10,"L"
PF
```



PRELLIPSE

Purpose

Creates an ellipse object in the image buffer.

Syntax

PRELLIPSE <*nexp1*>,<*nexp2*>,<*nexp3*>

Parameters

<*nexp1*>
The height of the ellipse at its widest point, in dots. Minimum value is 1, maximum value is 32767.

<*nexp2*>
The width of the ellipse at its widest point, in dots. Minimum value is 1, maximum value is 32767.

<*nexp3*>
The line weight of the ellipse, in dots. Minimum value is 1, maximum value is 32767.

Notes

Fingerprint does not support inverting an ellipse object.

A filled ellipse, created by using a large line weight, will not be completely black.

Examples

This example prints an ellipse contained within a rectangle:

```
10 PRPOS 200,200
20 DIR 1
30 ALIGN 1
40 PRELLIPSE 70,100,20
50 PRBOX 70,100,2
60 PRINTFEED
```

This example prints a circle:

```
10 PRPOS 200,200
20 DIR 1
30 ALIGN 1
40 PRELLIPSE 100,100,2
50 PRINTFEED
```

PRIMAGE

Purpose

Renders an image stored in the printer's memory to the label buffer. This command can be abbreviated as PM.

Syntax

PRIMAGE<*sexp*>

or

PM<*sexp*>

Parameters

<*sexp*>
Full name of the desired image including its extension.

Notes

An image is positioned according to preceding PRPOS, DIR, and ALIGN statements and can be magnified by means of MAG. For the best printout quality, create and download a larger version of the image instead of magnifying a smaller one. All images provided by Honeywell have an extension which indicates the intended image direction:

- Extension .1 indicates print directions 1 and 3.
- Extension .2 indicates print directions 2 and 4.

Although Fingerprint does not require these extensions, Honeywell strongly recommends that you follow the same convention when creating your own images so that selecting the correct image is easier.

Fingerprint supports black and white (no grayscale) images in PNG, PCX, GIF, and BMP formats.

Example

This example illustrates printing a label containing an image printed "upside down":

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

PRINT

Purpose

Prints data to the standard OUT channel.

This command can be abbreviated as ? (question mark).

Syntax

PRINT[<<*nexp*>|<*sexp*>>[<,|;><<*nexp*>|<*sexp*>>...][;]]

or

?[<<*nexp*>|<*sexp*>>[<,|;><<*nexp*>|<*sexp*>>...][;]]

Parameters

<<nexp>|<sexp>>
String or numeric expressions to be printed to the standard OUT channel.

Notes

If no expressions are specified after the PRINT statement, it yields a blank line. If one or more expressions are listed, the expressions are processed and the resulting values are presented on the standard OUT channel (such as the host screen).

Do not confuse PRINT with PRINTFEED.

Each line is divided into zones of 10 character positions each. These zones can be used for positioning the values:

- A comma (,) between the expressions causes the next value to be printed at the beginning of the next zone.
- A semicolon (;) between the expressions causes the next value to be printed immediately after the last value.
- A plus sign (+) between two string expressions causes the next value to be printed immediately after the last value. Plus signs cannot be used between numeric expressions.
- If the list of expressions is terminated by a semicolon, the next PRINT statement is added on the same line. Otherwise, a carriage return is performed at the end of the line. If the printed line is wider than the screen, the software automatically wraps to a new line and continues printing.

Printed numbers are always followed by a space character. Printed negative numbers are preceded by a minus sign.

Example

```
10 LET X%=10
20 LET A$="A"
30 PRINT X%;X%+1,X%+5;X%-25
40 PRINT A$+A$;A$,A$
50 PRINT X%;
60 ? "PIECES"
RUN
```

This results in:

```
10 11 15 -15
AAA A
10 PIECES
```

PRINT KEY ON/OFF

Purpose

Enables or disables printing of a label by pressing the **Print** key.

Syntax

PRINT KEY ON|OFF

Default is PRINT KEY OFF.

Notes

In Immediate mode and Direct Protocol, the **Print** key can be enabled to issue printing commands corresponding to PRINTFEED statements. This implies that each time **Print** is pressed, one single label, ticket, tag, or portion of continuous stock is printed and fed out.

PRINT KEY ON|OFF cannot be entered in Programming mode (use KEY ON and ON KEY GOSUB statements instead).

Example

This example shows how the Print key is enabled and a label is printed in Direct Protocol:

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

[Press the Print key]

INPUT OFF
```

PRINT#

Purpose

Prints data to a specified device or sequential file opened using the OPEN command, or controls the cursor in the front panel screen.

Syntax 1

Prints data to a device or file:

PRINT#*<nexp1>*[,*<<nexp2>*|*<sexp1>*[;]

Parameters 1

<nexp1>
Number assigned to the file or device when it was opened.

<<nexp2-n>|<sexp1-n>>
String or numeric expressions to be printed to the specified file or device.

[;]
If the line is appended by a semicolon, there is no carriage return appended to the line.

An alternate method to print on two lines in one statement is to send all of the data to the display in a singe PRINT# statement. Characters 1 through 16 display on the upper line, and characters 18 through 33 display on the lower line. Character 17 is ignored. The method also applies to strings that are not two full lines but still have characters covering more than the first line. You can also include a line break character in the text string using CHR$(10).

Syntax 2

Controls the cursor on the screen:

PRINT#, CHR$(155) + "(*<sexp1>*|*<sexp2>*|*<sexp3>*|*<sexp4>*|*<sexp5>*)";

Parameters 2

<sexp1>
Clears the front panel screen:

0J: From the active position to the end of the line (default)
1J: From the start of the line to the active position, inclusive
2J: Completely

<sexp2>
Selects the cursor type:

4p: Underscore
5p: Blinking block (default)

<sexp3>
Enables or disables the cursor:

2p: Enables cursor
3p: Disables cursor (default)

<sexp4>
Sets the absolute cursor position. Specified as <<*line*>>;<*position*>H>, where:

line: 1 for the upper line on the screen, or 2 for the lower line
position: Horizontal position in the line. Range is 1 to 16.

<sexp5>
Moves the cursor relative to its present location. Specified as <*distance*><*direction*>, where:

distance: Number of steps to move from current position. Default is 1.
direction: A - up, B - down, C - Right, D - left.

Movement must not place the cursor outside the front panel screen or the command is ignored.

Notes

Expressions can be separated by commas or semicolons according to the same rules for PRINT. The expressions must be separated properly so they can be read back when needed, or be presented correctly on the front panel screen.

PRINT# can only be used to print to sequential files, not to random files. When sending data to the printer screen ("console:"), PRINT# works the same way as PRINT does on the standard OUT channel. For example, the screen can be cleared by sending PRINT#*<nexp>* twice (see line 20 in the example below).

Examples of Printing to the Screen

The printer screen can show two lines of 16 characters each. Before sending any text, you must open the device (line 10) and clear both lines on the screen (line 20). Note the trailing semicolon on line 40:

```
10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT# 1:PRINT# 1
30 PRINT# 1,"OUT OF LABELS"
40 PRINT# 1,"PLEASE RELOAD!";
50 CLOSE# 1
RUN
```

Since the last line was appended by a semicolon, there is no carriage return and the text appears in the front panel screen as:

```
OUT OF LABELS
 PLEASE RELOAD!
```

An alternate method is to send all the data to the screen in a single PRINT# statement. Characters 1 through 16 are displayed on the upper line and characters 17 through 33 appear on the lower line, although character 17 is ignored.

The double-headed arrows in line 30 represent space characters. Note the trailing semicolon in line 30.

```
10 OPEN "console:" FOR OUTPUT AS 1
20 PRINT# 1: PRINT# 1
```

```
30 PRINT# 1,"OUT⇔OF⇔LABELS⇔⇔⇔⇔PLEASE⇔RELOAD!";
40 CLOSE# 1
RUN
```

Examples of Controlling the Cursor

This example clears all text from the screen:

```
10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT#1, CHR$(155) + "2J";
```

This example selects the underscore-type cursor:

```
10 OPEN "console:" FOR OUTPUT AS 1
20 PRINT#1, CHR$(155) + "4p";
```

This example enables the cursor:

```
10 OPEN "console:" FOR OUTPUT AS 1
20 PRINT#1, CHR$(155) + "2p";
```

This example sets the cursor in lower line, position 16:

```
10 OPEN "console:" FOR OUTPUT AS 1
20 PRINT#1, CHR$(155) + "2;16H";
```

This example moves the cursor from the first position in the upper line to the last position in the lower line:

```
10 OPEN "console" FOR OUTPUT AS 1
20 PRINT#1, CHR$(155) + "1B"
30 PRINT#1, CHR$(155) + "15C";
```

PRINTFEED

Purpose

Prints and feeds out one or a specified number of labels, tickets, tags, or portions of strip, depending on the printer setup.

This command can be abbreviated as PF.

Syntax

PRINTFEED[<*nexp1*>] | [-1,<*nexp2*>]

or

PF[<*nexp1*>] | [-1,<*nexp2*>]

Parameters

<*nexp1*>
Specifies number of copies to be printed.

<*nexp2*>
Specifies that <*nexp2*> number of identical copies of the last printed label should be reprinted. This function cannot be used with Direct Protocol.

Notes

Each time a PRINTFEED is executed without an appending value, one new label, ticket, tag, or portion of continuous stock is printed.

Optionally, PRINTFEED can be appended by a numeric expression specifying the number of copies to print. In Direct Protocol, possible counter, time, and date values are updated between copies printed using a predefined layout. Do not include any PRINTFEED statements in layouts in Direct Protocol.

If the number of copies is > 1, and LTS& and CUT are disabled , the BATCH optimizing strategy is automatically enabled, corresponding to an OPTIMIZE BATCH ON statement. Otherwise, BATCH optimizing strategy is automatically disabled.

It is also possible to reprint a specified number of copies of the last printed label, for example after an out-of-media condition (also see PRSTAT). Executing a PRINTFEED resets these statements to their default values:

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

Fields defined by statements executed before the PRINTFEED are not affected. Note that, when using PRINTFEED in a loop, all formatting parameters are reset to default each time the PRINTFEED is executed, and must therefore be included inside the loop.

The length of media to be fed out at execution of a PRINTFEED statement is determined by the choice of media type in the printer setup (label stock with gaps, tickets with gaps, fixed length strip, or variable length strip) and globally by the start and stop adjustment setup (positive or negative). The length of media to be fed out can be further modified by an additional positive or negative FORMFEED statement, either before or after the PRINTFEED.

Example 1

This example prints a single label with one line of text:

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

Example 2

This example prints five identical labels with one line of text:

```
10 FONT "Univers"
20 PRTXT "Hello!"
30 PRINTFEED 5
 RUN
```

Example 3

This example prints five labels using a FOR...NEXT loop. Note that formatting parameters are placed inside the loop:

```
10 FOR A%=1 TO 5
20 FONT "Univers"
30 PRPOS 200, 100
40 DIR 3
50 ALIGN 5
60 PRTXT "Hello!"
70 PRINTFEED
80 NEXT A%
RUN
```

Example 4

This example prints five labels in Direct Protocol, illustrating how the TICKS value is updated between labels (provided a predefined layout is used):

```
INPUT ON
FORMAT INPUT "#","@","&"
LAYOUT INPUT "tmp:LABEL1"
FT "Univers"
PP 100,100
PT TICKS
PP 100,200
```

```
PT VAR1$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
#See how time flies&@
PF 5
INPUT OFF
```

PRINTONE

Purpose

Prints characters (specified by their ASCII values) to the standard OUT channel.

Syntax

PRINTONE<*nexp*>[<,|;><*nexp*>...][;]

Parameters

<*nexp*>
ASCII decimal value of a character to be printed to the standard OUT channel.

Notes

When certain characters cannot be produced by the host computer, they can be substituted by the corresponding ASCII decimal values using PRINTONE. The characters are printed (per the currently selected character set defined by NASC) to the standard OUT channel (usually the host screen).

PRINTONE is similar to the PRINT statement and the use of commas and semicolons follows the same rules.

Example

PRINTONE 80;82;73;67;69;58,36;52;57;46;57;53

This results in:

PRICE: $49.95

PRINTONE#

Purpose

Prints characters specified by ASCII values to a device or sequential file.

Syntax

PRINTONE#*<nexp1>*[,*<nexp2>*[<,|;>*<nexp3>*...][;]]

Parameters

<nexp1>
Number assigned to the file or device when it is opened using the OPEN command.

<nexp2-n>
ASCII decimal value of the character to be printed to the specified file or device.

Notes

This statement is useful when the host cannot produce certain characters. The ASCII values entered produce characters according to the currently selected character set. The ASCII values can be separated by commas or semicolons according to the same rules as for the PRINT# statement.

PRINTONE# can only be used to print to sequential files, not to random files. When sending data to the printer display, PRINTONE# works in a way similar to PRINT#. The display can be cleared by sending PRINT#*<nexp>* twice (see line 20 in the example below).

Example

The printer display is able to show two lines of 16 characters each. Before sending any text, the device must be OPENed and the display cleared. Note the trailing semicolon sign on line 40.

```
10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT# 1:PRINT# 1
30 PRINTONE# 1,80;82;69;83;83
40 PRINTONE# 1,69;78;84;69;82;
50 CLOSE #1
RUN
```

Since the last line was appended by a semicolon, there is no carriage return. The text appears on the printer display as:

PRESS
ENTER

PRLINE

Purpose

Creates a line. This command can be abbreviated as PL.

Syntax

PRLINE<*nexp1*>,<*nexp2*>

or

PL<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
Length of the line in dots. Maximum is 6000.

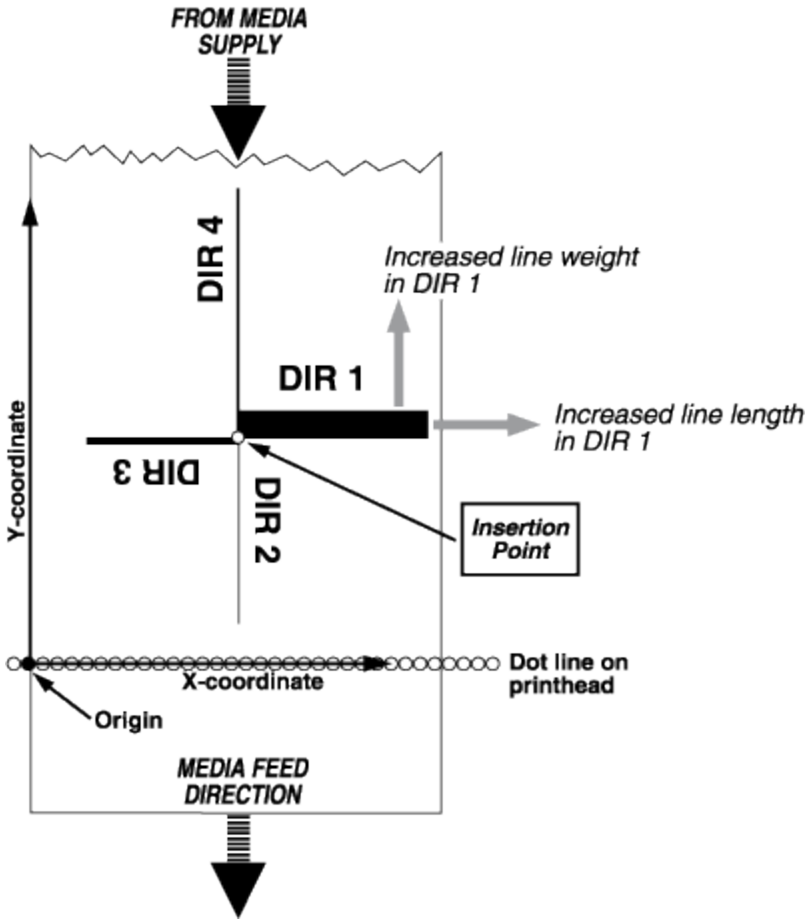
<*nexp2*>
Line weight in dots. Maximum is 6000.

Notes

The line is drawn from the insertion point and away according to the nearest preceding DIR and ALIGN statements (that is, the line runs in parallel with any text printed in the selected direction).

A line can be ALIGNed left, right or center. The anchor points are situated at the bottom of the line, which means that as the line weight increases, the line grows upward in relation to the selected direction.

In the illustration below, all lines are aligned left. Lines may cross (see XORMODE ON/OFF statement).



Example

This example draws a 2.5 cm (1 in) long and 10 dots thick line across the media in an 8 dots/mm printer:

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

PRPOS

Purpose

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

This command can be abbreviated as PP.

Syntax

PRPOS<*nexp1*>,<*nexp2*>

or

PP<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
X-coordinate (in number of dots from the origin).

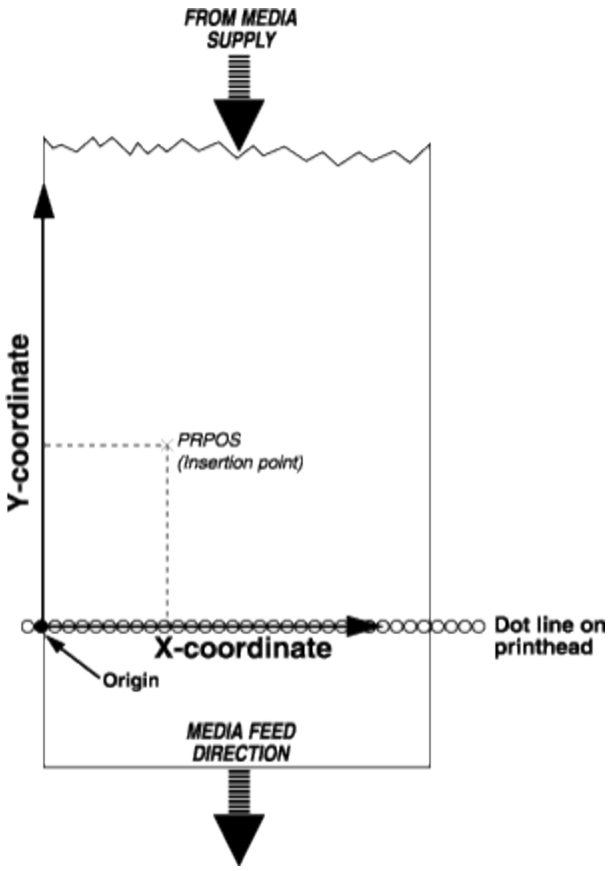
<*nexp2*>
Y-coordinate (in number of dots from the origin).

Default for both is 0. Reset to default by executing a PRINTFEED.

Notes

When the printer is set up, a "print window" is created. This involves specifying the location of the origin along the X-axis, setting the maximum print width along the X-axis from the origin, and setting the maximum print length along the Y-axis from the origin.

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



The insertion point must be selected so the field in question fits inside the print window. A field that does not fit entirely inside the print window causes error 1003 ("Field out of label") except when a CLIP ON statement is issued.

To find the present insertion point, use PRSTAT.

Example 1

This example prints a line of text:

```
10 FONT "Univers"
20 PRPOS 30,200
30 PRTXT "HELLO"
40 PRINTFEED
RUN
```

Example 2

Each text line is normally positioned separately by is own PRPOS statement. If no position is given for a printable statement, it is printed immediately after the preceding printable statement.

```
10 FONT "Univers"
20 PRPOS 30,200
30 PRTXT "SUMMER"
40 PRTXT "TIME"
50 PRINTFEED
RUN
```

This results in:

SUMMERTIME

Example 3

A program for fixed line-spacing of text may be composed this way (another way is to use the extended PRBOX statement):

```
10 FONT "Univers"
20 X%=30:Y%=500
30 INPUT A$
40 PRPOS X%,Y%
50 PRTXT A$
60 Y%=Y%-50
70 IF Y%>=50 GOTO 30
80 PRINTFEED
90 END
RUN
```

Enter the text for each line after the question mark shown on the host screen. The Y-coordinate is decremented by 50 dots for each new line until it reaches the value 50, which means that ten lines are printed.

PRSTAT

Purpose

Returns the current printer status or the current position of the insertion point.

Syntax

PRSTAT[(<*nexp*>)]

Parameters

<*nexp*>
Numeric expression which returns different status indicators:

1: X-position for the insertion point at DIR 1&3.
2: Y-position for the insertion point at DIR 2&4.
3: X-position of the corner with the lowest coordinates of the last object.
4: Y-position of the corner with the lowest coordinates of the last object.
5: Width along the X-axis of the last object.
6: Height along the Y-axis of the last object.
7: Print job identifier
8: Print job state. See the Notes for more information.
9: Print job error code.
10: Remaining number of copies to be printed in a batch print job.

Notes

PRSTAT returns a numeric expression, which is the sum of the values given by the following conditions at the moment when PRSTAT is executed:

Value	Condition
0	Ok
1	Printhead lifted
2	Label not removed (see note)
4	Label Stop Sensor (LSS) detects no label
8	Printer out of transfer ribbon (TTR) or ribbon installed (DT)
16	Printhead voltage too high
32	Printer is feeding
64	front arm lifted
128	Printer out of media

PRSTAT always returns 0 in printers not fitted with a label taken sensor.

If two error conditions occur simultaneously (for example, if the printhead is lifted and the printer is out of transfer ribbon), PRSTAT returns the sum of the two error values (in this example, (1+8) = 9). Every combination of errors results in a unique sum. You can use it to branch to a subroutine such as notifying the operator or interrupting the program. The label stop sensor detects no label if a gap or black mark is in front of the sensor, as well as when the printer is out of media.

PRSTAT(1) and PRSTAT(2) return the current position of the insertion point relative to either the X or the Y position, depending on the selected print direction. This is useful for measuring the length of a text string or a bar code.

PRSTAT(3) to PRSTAT(6) returns the position and size of the last object regardless of RENDER ON/OFF. Their values are not updated by the execution of a PRBUF statement.

PRSTAT(7) to PRSTAT(10) detects if a print job has been interrupted, so steps can be taken to reprint missing copies (see PRINTFEED).

PRSTAT (7) also returns a print job identifier that is automatically assigned to the print job by the firmware.

PRSTAT (8) returns the state of the print job as a numeric expression, which is the sum of the values given by the following conditions:

Value	Condition
0	Print cycle not set up for printing, perhaps due to out-of-ribbon
1	The previous print cycle never ended (timeout)
2	Print cycle has started
4	All lines successfully printed
8	Printing truncated (media shorter than print image)
16	Printhead strobing error or label length exceeded
32	Ribbon low

PRSTAT (8) = 6 or 22 indicates a successfully printed label (in the latter case error "next label not found" may have been detected).

PRSTAT (9) returns the error code detected by the print engine during printfeed. It is used together with PRSTAT(8) to determine the error cause when using OPTIMIZE "BATCH" ON.

PRSTAT (10) returns the number of copies that remain to be printed in an interrupted batch print job.

Example 1

This example shows how two error conditions are checked:

```
10 A% = PRSTAT
20 IF A% AND 1 THEN GOSUB 1000
30 IF A% AND 128 THEN GOSUB 1010
```

```
40 END
.....
1000 PRINT "Printhead is lifted":RETURN
1010 PRINT "Printer out of media":RETURN
RUN
```

Example 2

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

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

This results in:

```
380
```

PRTXT

Purpose

Provides input data for a text field.

This command can be abbreviated as PT.

Syntax

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

or

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

Parameters

<<nexp>|<sexp>>
Specifies one line of text (maximum 300 characters).

Notes

A text field consists of one line of text. The text field must be defined by FONT or FONTD and may be further defined and positioned by DIR, ALIGN, MAG, PRPOS, INVIMAGE, or NORIMAGE (or their default values).

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. For more information, see Complex Scripts.

PRTXT supports inline text formatting, such as boldface and italicization. For more information, see Inline Text Formatting.

Two or more expressions can be combined to form a text line. They must be separated by semicolons (;) and are printed adjacent to each other. Plus signs can also be used for the same purpose, but only between string expressions. A comma sign can be used for a similar purpose, but the second expression starts at the next tabulating zone. Each zone is 10 characters long.

String constants must be enclosed by quotation marks, whereas numeric constants or any kind of variables must not.

Example 1

This example prints a line of text:

```
10 FONT "Univers"
20 PRPOS 30,300
30 PRTXT "How do you do?"
40 PRINTFEED
RUN
```

Example 2

Several string constants and string variables can be combined into one line of text by the use of plus signs or semicolons:

```
10 FONT "Univers"
20 PRPOS 30,300
30 PRTXT "SUN";"SHINE"
40 A$="MOON"
50 B$="LIGHT"
60 PRPOS 30,200
70 PRTXT A$+B$
80 PRINTFEED
RUN
```

This results in:

```
SUNSHINE
MOONLIGHT
```

Example 3

Numeric constants and numeric variables can be combined by the use of semicolons, but plus signs cannot be used in connection with numeric expressions:

```
10 FONT "Univers"
20 PRPOS 30,300
30 PRTXT 123;456
40 A%=222
50 B%=555
60 PRPOS 30,200
70 PRTXT A%;B%
80 PRINTFEED
RUN
```

This results in:

```
123456
222555
```

Example 4

Numeric and string expressions can be mixed on the same line, for example:

```
10 FONT "Univers"
 20 PRPOS 30,300
 30 A$="December"
 40 B%=27
 50 PRTXT A$;" ";B%;" ";"2007"
 80 PRINTFEED
 RUN
```

This results in:

```
December 27 2007
```

Example 5

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

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

This results in:

HAPPY BIRTHDAY

PUT

Purpose

Writes a given record from the random buffer to a given random file.

Syntax

PUT[#]<*nexp1*>,<*nexp2*>

Parameters

#
(Optional) Indicates that whatever follows is a number.

<*nexp1*>
Number assigned to the file when it is opened using the OPEN command.

<*nexp2*>
Number of the record. Must be ≥ 1.

Notes

Use LSET or RSET statements to place data in the random buffer before issuing the PUT statement.

Example

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

This results in:

SMITH...JOHN.....12345630

RANDOM

Purpose

Generates a random integer within a specified interval.

Syntax

RANDOM(<*nexp1*>,<*nexp2*>)

Parameters

<*nexp1*>
First integer in the interval. Use positive integers only. Must not be 0 or equal to <*nexp2*>.

<*nexp2*>
Last integer in the interval.

Notes

The randomly generated integer will be:

- equal to or greater than <*nexp1*>.
- equal to or less than <*nexp2*>.

Example

This example produces ten random integers between 1 and 100:

```
10 FOR I%=1 TO 10
20 A% = RANDOM (1,100)
30 PRINT A%
40 NEXT I%
RUN
```

This results in:

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

RANDOMIZE

Purpose

Reseeds the random number generator, optionally with a specified value.

Syntax

RANDOMIZE[*<nexp>*]

Parameters

<nexp>
Integer (0 to 99999999) with which the random number generator will be reseeded.

Notes

If no value is specified, a message appears asking you to enter a value between 0 and 99,999,999.

Example 1

In this example, a prompt appears, asking you to specify a value to reseed the generator:

```
10 RANDOMIZE
20 A%=RANDOM (1,100)
30 PRINT A%
RUN
Random Number Seed (0 to 99999999) ?
Enter 555
```

This results in:

```
36
```

Example 2

In this example, the reseeding integer is specified, so there is no prompt:

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

This results in:

```
68
```

Example 3

A higher degree of randomization is obtained when the random integer generator is reseeded with another random integer, in this example provided by a TICKS function:

```
10 A%=TICKS
20 RANDOMIZE A%
```

```
30 B%=RANDOM(1,100)
40 PRINT B%
RUN
```

This results in:

```
42
```

READY

Purpose

Orders a ready signal, for example XON, CTS/RTS or PE, to be transmitted from the printer on the specified communication channel.

Syntax

READY[<*nexp*>]

Parameters

<*nexp*>
Specifies a communication channel:

1: "uart1:"
2: "uart2:"
3: "uart3:"
4: "centronics:"
6: "usb1:" (Supported when virtual COM port is enabled)
8: "uart4:"
9: "uart5:"

Notes

The selected communication protocol usually contains a "ready" signal, telling the host computer that the printer is ready to receive more data.

READY allows you to order a ready signal to be transmitted on the specified communication channel. If no channel is specified, the signal will be transmitted on the standard OUT channel (see SETSTDIO statement).

The READY signal is used to revoke a previously transmitted BUSY signal. However, the printer may still be unable to receive more data due to other reasons such as a full receive buffer.

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

Example

This example allows the printer to receive more data on "uart2:" after the process of printing a label is completed.

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

Running this example may require an optional interface board.

REBOOT

Purpose

Restarts the printer.

Syntax

REBOOT

Notes

This statement has exactly the same effect as cycling power to the printer.

REDIRECT OUT

Purpose

Redirects the output data to a created file.

Syntax

REDIRECT OUT[*<sexp>*]

Parameters

<sexp>
(Optional) Name of the file to be created and where the output will be stored.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

Normally the output data is transmitted on the standard output channel (in most cases, the host screen). However, by means of a REDIRECT OUT *<sexp>* statement, a file can be created to which the output will be redirected. That implies that no data is echoed back to the host. Normal operation, with the output being transmitted on the standard output channel again, is resumed when a REDIRECT OUT statement without any appending file name is executed.

Example

In this example, a file ("LIST.DAT") is created to which the names of the files in the printer permanent memory are redirected. The redirection is then terminated (line No. 30) and the file for input is opened using the OPEN command.

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

REM

Purpose

Adds headlines and comments to the program without including them in the execution.

This command can be abbreviated as ' (single quote).

Syntax

REM*<remark>*

or

'<remark>

Parameters

<remark>
String of text inserted in the program as a comment (maximum of 32,767 characters per line).

Notes

REM may be entered on a program line of its own or inserted at the end of a line with another command. In the latter case, REM must be preceded by a colon (":REM").

A shorthand form for REM is an apostrophe (ASCII 39 decimal). It is possible to branch to a line of REM statement. Execution continues at the first executable line after the REM line. Use REM statements carefully, because they slow down execution and transfer of data and also take up valuable memory space.

Example

This example includes both types of REM statements:

```
10 'Label format No. 1
20 FONT "Univers"
30 PRPOS 30,100
40 DIR 1 :REM Print across web
50 ALIGN 4 :REM Aligned left/baseline
60 MAG 2,2 :'Double height and width
70 PRTXT "HELLO"
80 PRINTFEED
RUN
```

REMOVE IMAGE

Purpose

Removes a specified image from the printer memory. Only removes images downloaded by means of a STORE statement.

Syntax

REMOVE IMAGE *<sexp>*

Parameters

<sexp>
Full name, including extensions, of the image to be removed.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

Use the REMOVE IMAGE command to delete unneeded images from the printer memory to save valuable space.

There is a difference between images and image files. Image files downloaded using other methods (such as a TRANSFER KERMIT statement) should be removed using a KILL statement.

REMOVE IMAGE cannot be undone, so use this statement carefully.

Example

```
10 REMOVE IMAGE "LOGOTYPE.1"
 RUN
```

RENDER ON/OFF

Purpose

Enables or disables the rendering of text, bar code, image, box, and line fields.

Syntax

RENDER ON|OFF

Parameters

ON (default) enables rendering and OFF disables rendering.

Notes

Use RENDER OFF with PRSTAT to get information regarding size and position of a field without actually rendering it; the field is not printed when the program is executed.

When rendering is disabled:

- PRTXT, PRBAR, PRIMAGE, PRLINE, and PRBOX statements give no result when a PRINTFEED statement is executed.
- statements other than PRPOS do not update the insertion point.
- field numbers (see FIELDNO) are not updated.
- statements such as CLIP ON/OFF, XORMODE ON/OFF, or BARSET retain their usual meanings.

PRBUF renders a field regardless of RENDER ON/OFF.

RENDER ON enables field rendering after a RENDER OFF statement.

Example

This example retrieves information on the size of a text field which was not rendered. Note that the actual result may vary depending on font, font size, and printer type.

```
10 RENDER OFF
20 PRTXT "Render off"
30 PRINT "Width:",PRSTAT(5),"Height:",
PRSTAT(6)
40 RENDER ON
50 PRINTFEED
RUN
```

This results in:

```
Width: 153 Height: 46
No field to print in line 50
Ok
```

RENUM

Purpose

Renumbers the lines of the program currently residing in the printer working memory.

Syntax

RENUM[<*ncon1*>][,[<*ncon2*>][,<*ncon3*>]]

Parameters

<*ncon1*>
First line number of the new sequence. Default is 10.

<*ncon2*>
Line in the current program at which renumbering is to start. Default is 1.

<*ncon3*>
Desired increment between line numbers in the new sequence. Default is 10.

Notes

Use RENUM to create space for more program lines when expanding an existing program, and for renumbering programs written without line numbers (for example, after an unnumbered programs is LISTed, LOADed, or MERGEd). Line references following GOTO statements are renumbered accordingly. Use a LIST statement to print the new numbers on the screen.

Example

This example renumbers a program:

```
10 FONT "Univers"
20 PRPOS 30,100
30 PRTXT "HELLO"
40 A%=A%+1
50 PRINTFEED
60 IF A%<3 GOTO 40
70 END
RENUM 100,20,50
LIST
```

This results in:

```
10 FONT "Univers"
100 PRPOS 30,100
150 PRTXT "HELLO"
200 A%=A%+1
250 PRINTFEED
300 IF A%<3 GOTO 200
350 END
```

Note that the line number in the GOTO statement on line 300 has changed. Line 10 is not renumbered, since line 20 was specified as the starting point. The new increment is 50.

REPRINT ON/OFF

Purpose

Enables or disables the reprinting of a label in Direct Protocol.

Syntax

REPRINT ON|OFF

Parameters

ON (default) enables reprinting and OFF disables reprinting.

Notes

REPRINT OFF also affects and overrides the behavior of the PRINT KEY ON statement. If REPRINT OFF is entered before PRINT KEY ON, no error message is shown, but PRINT KEY is set to ON and an empty label is printed when the **Print** key is pressed.

If REPRINT is set to OFF, there is no way to reprint an old print job. If a PRINTFEED statement is sent to the printer after a print job has been completed, a blank label is fed out and an error 1006 ("No field to print") occurs. However, the REPRINT OFF statement does not clear the print buffer, which only occurs after a PRINTFEED statement has been executed.

Leaving and re-entering Direct Protocol does not reset the REPRINT status. Restarting the printer resets the REPRINT status to its default value (ON). A REPRINT OFF statement prevents automatic reprinting after the error has been cleared for the following errors:

- 1005: "Out of paper"
- 1022: "Head lifted"
- 1031: "Next label not found"
- 1058: "Transfer ribbon is installed"

If REPRINT is set to OFF, an "Out of transfer ribbon" error does not cause the "Continue-Reprint" message to be shown on the printer's screen.

Example 1

This example disables reprinting:

```
INPUT ON        'Enter Direct Protocol
REPRINT OFF     'Disable reprinting
```

Example 2

This example shows what happens when the first and second PRINTFEED statements generate a printed label. After the second PRINTFEED, the print buffer is cleared and a "No field to print" error occurs.

```
INPUT ON          'Enter Direct Protocol
PRTXT "PRINT1"    'Print the text "Print1"
PRINTFEED         'Yields a print label
REPRINT OFF       'Disables reprinting
PRINTFEED         'Yields a printed label and clears the print buffer
PRINTFEED         'Yields a blank label and generates an error
```

RESUME

Purpose

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

Syntax

RESUME[(*<ncon>*|*<line label>*|<NEXT>|<0>)|<HTTP>]

Parameters

<ncon>|*<line label>*
Number or label of the line to which the program should return.

Notes

RESUME is used in connection with ON ERROR GOTO and ON HTTP GOTO.

There are five ways of using RESUME:

- RESUME or RESUME 0: Execution is resumed at the statement where the error occurred.
- RESUME NEXT: Execution is resumed at the statement immediately following the one that caused the error.
- RESUME *<ncon>*: Execution is resumed at the specified line.
- RESUME *<line label>*: Execution is resumed at the specified line label.
- RESUME *<HTTP>*: Execution is resumed at the point where it was branched by an ON HTTP GOTO statement. STDIN and STDOUT are restored to their original values.

Examples

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

```
10 ON ERROR GOTO 1000
20 FONT "Univers"
30 PRPOS 100,100
40 PRTXT "HELLO"
50 PRPOS 100, 300
60 PRIMAGE "GLOBE.1"
70 PRINTFEED
80 END
```

Alternate Subroutine 1

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

```
1000 IF ERR=15 THEN FONT "Univers":RESUME NEXT
1010 RESUME 80
```

Alternate Subroutine 2

An error message is printed and the execution goes on from the line following the error.

```
1000 IF ERR=15 THEN PRINT "Font not found"
1010 RESUME NEXT
```

RETURN

Purpose

Returns to the main program after having branched to a subroutine due to a GOSUB statement.

Syntax

RETURN[*<ncon>*|*<line label>*]

Parameters

<ncon>|*<line label>*
Number or label of a line in the main program to return to.

Notes

When RETURN is encountered during the execution of a subroutine, the execution returns to the main program. Execution continues from the statement immediately following the most recently executed GOSUB or from an optionally specified line.

If a RETURN statement is encountered without a GOSUB statement having been previously executed, error 28 ("Return without Gosub") occurs.

Example

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

This results in:

```
This is the main program
This is subroutine 1
This is subroutine 2
This is subroutine 3
You're leaving subroutine 3
You're back in subroutine 2
You're back in subroutine 1
You're back in the main program
```

REWINDCONTROL

Purpose

Controls the internal rewind motor in PM-series printers.

This command is applicable only for the PM23c, PM43, and PM43c printers.

Syntax

REWINDCONTROL *<nexp>*

Parameters

<nexp>
Controls the rewind motor as follows:

- Positive values specify how long the internal rewinder pulls the liner before the paper feed motor
  starts. Maximum value is 5000 dots.
- Negative values specify the distance (in dots) the paper is fed before the internal rewind motor starts pulling liner. A value of -1 turns the rewind motor off. Maximum value is label length.

Notes

PM-series printers can have two separate motors, one for feeding paper and one for rewinding liner. REWINDCONTROL allows for control of the internal rewind motor.

A positive value stretches the liner before printing to ensure better label dispensing. However, a positive value might cause the stepper motor to stall for certain liner surfaces. In such cases, contact your local support organization.

A negative value allows the label to be rewound with the liner using the self-strip mode. A value of -1 turns the internal rewind motor off, which makes for more silent operation when the printer runs without self-strip. This command cannot be used when REWINDVOID is enabled.

Examples

REWINDCONTROL 200
REWINDCONTROL -100

REWINDVOID

Purpose

Rewinds VOID labels with the liner in RFID-enabled PM-series printers.

This command is applicable only for the PM23c, PM43, and PM43c printers.

Syntax

REWINDVOID OFF|ON[<*nexp*>]

Parameters

<*nexp*>
Specifies the distance (in dots) the paper is fed before the internal rewind motor starts pulling liner. Default is 100. Maximum value is label length.

Notes

PM-series printers can have two separate motors, one for feeding paper and one for rewinding liner. This function causes a VOID tag/label to be rewound with the liner paper when using the printer in self-strip mode, preventing a label being from being dispensed (such as to an applicator). The value for <*nexp*> must be tested for each specific type of media and label length.

Example

REWINDVOID ON 200

RIGHT$

Purpose

Returns a specified number of characters from a given string, starting from the end (extreme right end) of the string.

Syntax

RIGHT$(*<sexp>*,*<nexp>*)

Parameters

<sexp>
String from which the characters will be returned.

<nexp>
Specifies the number of characters to be returned.

Notes

RIGHT$ is the complementary function for LEFT$. If the number of characters to be returned is greater than the number of characters in the string, then the entire string is returned. If the number of characters is set to zero, a null string is returned.

Example 1

```
PRINT RIGHT$("THERMAL_PRINTER",7)
```

This results in:

```
PRINTER
```

Example 2

```
10 A$="THERMAL_PRINTER":B$ = "LABEL"
20 PRINT RIGHT$(B$,5);RIGHT$(A$,8);"S"
RUN
```

This results in:

```
LABEL_PRINTERS
```

RSET

Purpose

Places data right-justified into a field in a random file buffer.

Syntax

RSET<*svar*>=<*sexp*>

Parameters

<*svar*>
String variable assigned to the field by a FIELD statement.

<*sexp*>
Holds the input data.

Notes

After OPENing a file and formatting it using a FIELD statement, you can enter data into the random file buffer using the RSET and LSET statements. The input data can only be stored in the buffer as string expressions. Numeric expressions must be converted to strings by the STR$ function before an LSET or RSET statement is executed.

If the length of the input data is less than the field, the data is right-justified and the remaining number of bytes are printed as space characters.

If the length of the input data exceeds the length of the field, the input data is truncated on the left side.

Example

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

This results in:

SMITH...JOHN.....12345630

RUN

Purpose

Starts the execution of a program.

Syntax

RUN[<<*scon*>|<*ncon*>>]

Parameters

<scon>
(Optional) Specifies an existing program to run.

<ncon>
(Optional) Specifies the number of a line in the current program where the execution will start.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

RUN starts the execution of the program currently residing in the printer working memory, or of a specified program residing elsewhere. The execution begins at the line with the lowest number, or from a specified line in the current program.

If the program is stored somewhere besides the current directory (see CHDIR statement) and has not been LOADed, its designation must be preceded by a reference to that device (such as "/c", "tmp:").

Never use RUN on a numbered line, or in a line without a number in Programming Mode, or error 40 ("Run statement in program") occurs.

A RUN statement executed in Direct Protocol makes the printer switch to Immediate Mode, and has the same effect as an INPUT OFF statement.

Examples

To execute the current program from its first line:

RUN

To execute the current program starting from line 40:

RUN 40

To execute the program "TEST.PRG" from its first line:

RUN "TEST"

To execute the program "TEST.PRG" from its first line.

RUN "TEST.PRG"

To execute the program "FILELIST.PRG", which is stored in temporary memory, from its first line:

RUN "/tmp/FILELIST.PRG"

To list all of the fonts and aliases installed on the printer:

RUN "ls -l /printer/fonts"

SAVE

Purpose

Saves a file to the specified location.

Syntax

SAVE<*scon*>[,P|L]

Parameters

<scon>
Name of the file. To save the file to somewhere other than the current directory, start this string with the desired directory. Maximum of 30 characters including extension, or 26 characters without an extension.

P
(Optional) Protects the file.

L
(Optional) Saves the file without line numbers.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

When a file is saved, its file name can be a maximum of 30 characters including an extension. By default, the program automatically appends the name with the extension .PRG and converts all lowercase characters to uppercase. The name must not contain any quotation marks (" ").

If you start the file name with a period character (.), it is not removed at a soft formatting operation (see FORMAT). Such a file is also listed differently (see FILES).

Files can only be saved in the printer permanent memory ("/c"), its temporary memory ("tmp:"), or to a USB drive ("usb:"). If a file with the selected name already exists in the selected directory, that file is deleted and replaced by the new file without warning. You can continue to work with a file after saving it until a NEW, LOAD, KILL, or you run the REBOOT command

A protected file (SAVE <filename>,P) is encrypted at saving and cannot be listed after it is loaded. Program lines cannot be removed, changed, or added. Once a file has been protected, it cannot be made unprotected again. Therefore, you should save an unprotected copy, should a programming error be detected later on. If you are going to use an electronic key to prevent unauthorized access to a file, you should protect it.

A saved program can be merged with the program currently residing in the printer working memory. To prevent automatically assigned line numbers from interfering with the line numbers in the current program, you can choose to save the program without line numbers (SAVE <filename>,L). See MERGE instruction.

Example 1

SAVE "Label14"

saves the file as "LABEL 14.PRG" in current directory.

Example 2

SAVE "/c/Label14",P

saves and protects the file "LABEL14.PRG" in the permanent memory.

Example 3

SAVE "d:Label14",L

saves "LABEL14.PRG" without line numbers on a USB storage device.

SET FAULTY DOT

Purpose

Marks one or several dots on the printhead as faulty, or marks all faulty dots as correct.

Syntax

SET FAULTY DOT<*nexp1*>[,<*nexpn*>...]

Parameters

<*nexp1*>
Number of the dot to be marked as faulty. Successive executions add more faulty dots. <*nexp1*> = -1 marks all dots as correct (default).

Notes

SET FAULTY DOT is closely related to HEAD and BARADJUST. You can check the printhead for possible faulty dots by means of HEAD, and mark them as faulty using SET FAULTY DOT. Use BARADJUST to automatically reposition horizontal bar codes sideways to place the faulty dots between the printed bars, preserving the readability.

Once a dot has been marked faulty by a SET FAULTY DOT statement, it remains marked as faulty until all dots are marked as correct by a SET FAULTY DOT -1 statement.

The HEAD function makes it possible to mark all faulty dots using a single command, instead of specifying each faulty dot using SET FAULTY DOT.

Example

This example illustrates how a bar code is repositioned by means of BARADJUST when a number of dots are marked as faulty by a SET FAULTY DOTS statement. Type RUN and send various numbers of faulty dots from the host a few times to see how the bar code moves sideways across the label.

```
10 INPUT "No. of faulty dots"; A%
20 FOR B% = 1 TO A%
30 C% = C% + 1
40 SET FAULTY DOT C%
50 NEXT
60 D% = A%+2
70 BARADJUST D%, D%
80 PRPOS 0, 30
90 BARTYPE "CODE39"
100 PRBAR "ABC"
110 SET FAULTY DOT -1
120 PRINTFEED
RUN
```

SETASSOC

Purpose

Sets a value for a tuple in a string association.

Syntax

SETASSOC *<sexp1>*, *<sexp2>*, *<sexp3>*

Parameters

<sexp1>
Name of the association (case-sensitive).

<sexp2>
Name of the tuple.

<sexp3>
Value of the tuple.

Notes

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

Example

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

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

This results in:

```
time=UNKNOWN LABEL=321 DESCRIPTION=DEF
time=153355
```

SETPFSVAR

Purpose

Registers variables to be saved at power failure.

Syntax

SETPFSVAR<*sexp*>[,<*nexp*>]

Parameters

<*sexp*>
Name of a numeric or string variable (uppercase characters only).

<*nexp*>
Size in bytes of a string variable. Maximum is 230.

Notes

When a program is loaded, it is copied to and executed in the printer temporary memory ("tmp:"). Should an unexpected power failure occur, the printer tries to save as much data as possible in the short time available before all power is lost. To minimize the risk of losing important variable data at a power failure, you can register numeric and string variables to be saved. There are 2176 bytes (including overhead) available for this purpose.

However, should the power failure occur while the printer is printing, there will be no power left to save the current variables.

When you register a string variable, you must also specify its size in bytes. The variable name is limited to a length of 20 characters.

Related instructions are GETPFSVAR, DELETEPFSVAR, and LISTPFSVAR.

Example 1

Example with string variable:

```
100 IF QA$="" THEN QA$="Hello":QA%=LEN(QA$)
110 SETPFSVAR "QA$",QA%
```

Example 2

Example with numeric variable:

```
200 SETPFSVAR    "QCPS%"
```

SETSTDIO

Purpose

Selects standard IN and OUT communication channel.

Syntax

SETSTDIO<*nexp1*>[,<*nexp2*>]

Parameters

<*nexp1*>
Desired input/output channel:

100: Auto-hunting enabled (default)
0: "console:"
1: "uart1:"
2: "uart2:"
3: "uart3:"
4: "centronics:"
5: "net1:"
6: "usb1:"
7: "uart4:"
8: "uart5:"
9: "usbhost:"
10: "bluetooth:"
11: "ftp1:"
12: "http1:"
13: "lpr1:"
14 - 19: Reserved for future use
20 - 28: NET2 - NET10

<*nexp2*>
(Optional) Specifies a different output channel:

0: "console:"
1: "uart1:"
2: "uart2:"
3: "uart3:"
5: "net1:"
6: "usb1:"
7: "uart4:"
8: "uart5:"
9: "usbhost:"

Notes

The printer is controlled from its host via a communication channel. By default, auto-hunting is selected, meaning that all available channels are continuously scanned for input. When data is received on a given channel, that channel is designated as the standard input/output channel. If no data is received on the present standard input channel within a 2-second timeout period, the firmware scans all other existing channels (except "console:") looking for input data. The channel where input data is

first found is then designated the new STDIN/STDOUT channel. The same procedure is repeated infinitely as long as auto-hunting is enabled.

These restrictions apply to auto-hunting:

- If "centronics:" is used as input channel and auto-hunting is enabled, "uart1:" is selected standard out channel.
- Auto-hunting does not work with "console:".
- Auto-hunting does not work with COMSET, INKEY$, INPUT$, or LINE INPUT.
- When using auto-hunting in the programming mode, the detected port stays as STDIO until the program breaks.

It is also possible to specify a certain channel as the permanent STDIN and/or STDOUT channel. If only one channel is specified, it serves as both the standard input and standard output channel. Alternatively, a different channel can be selected for the standard output channel.

For programming, Honeywell recommends using "uart1:" for both the standard input and standard output channels. If another channel is selected, use the same serial channel for both input and output. The "centronics:" channel can only be used for input to the printer and is not suited for programming.

Example 1

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

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

Example 2

This example enables auto-hunting for input and "uart1:" for output:

```
10 SETSTDIO 100,1
. . . .
. . . .
```

SETUP

Purpose

Enters Setup Mode, or changes the printer setup parameters, including network and communications settings and media handling. Supported settings and parameters depend on your printer and installed options.

There are specific SETUP commands for network settings, including Ethernet, 802.11 wireless, and Bluetooth. For more information, see SETUP Command Information for Network Parameters.

Many legacy SETUP command paths, such as "PRINT DEFS,HEAD RESIST", are no longer supported in Fingerprint. Honeywell recommends that you use current command paths instead. For more information on the setup parameters supported by your printer, see your printer user manual.

Syntax 1

SETUP

Parameters 1

None. Places the printer in Setup Mode.

Syntax 2

SETUP*<sexp>*

Parameters 2

<sexp>
Name of an existing setup file that is used to change the entire current printer setup, or a string used to change a single parameter in the current printer setup.

Syntax 3

SETUP *<sexp1>*,*<sexp2>*

Parameters 3

<sexp1>
Name of a setup section (see the printer user manual).

<sexp2>
Name of a file that will be used to change the specified setup section.

Syntax 4

SETUP *<sexp1>*,*<sexp2>*,*<sexp3>*

Parameters 4

<sexp1>
Name of a setup section (see the printer user manual). Not implemented for "prt".

<sexp2>
Name of the setup object (see the printer user manual).

<sexp3>
Specifies the new value (see the printer user manual).

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

To retrieve current Setup settings, use SETUP GET or SETUP WRITE.

A setup file may contain new values for one or several setup parameters, whereas a setup string only can change a single parameter. Creating a setup file requires several operations. Setup strings can be created in a single operation, making them suitable for use with Direct Protocol.

By default, setup parameters are saved as a file in the printer permanent memory. However, you can use SYSVAR (35) to prevent changes from being saved. See SYSVAR for more information.

When a SETUP*<sexp>* statement is encountered during program execution, the setup is changed accordingly and program execution is resumed. Note that some printing instructions (ALIGN, DIR, FONT, and PRPOS) may be changed when test labels are printed.

Below is an example of a Setup file. Note that Setup files or setup strings have a special syntax for each parameter that must be followed exactly:

- Variable numeric input data is indicated by "n"–"nnnnn", string data by "sssss", and alternative data by bold characters separated by vertical bars (|).

NETWORK,IP SELECTION,DHCP

NETWORK,IP ADDRESS,0.0.0.0

NETWORK,NETMASK,0.0.0.0

NETWORK,DEFAULT ROUTER,0.0.0.0

NETWORK,DHCP RESPONSE,Broadcast

NETWORK,IPV6 SELECTION,Automatic

NETWORK,IPV6 ADDRESS,2001:db8:0:1::1/64

MEDIA,MEDIA TYPE,LABEL (W GAPS)

MEDIA,PAPER TYPE,TRANSFER

MEDIA,MEDIA SIZE,WIDTH,840

MEDIA,MEDIA SIZE,LENGTH,1200

MEDIA,MEDIA SIZE,XSTART,24

PRINT DEFS,CLIP DEFAULT,Off

MEDIA,TESTFEED MODE,FAST

MEDIA,LEN (SLOW MODE),0

MEDIA,CONTRAST,+0%

COM,USB KEYBOARD,U.S./U.K.

PRINT DEFS,CMD LANGUAGE,Fingerprint

TRANSFER,RIBBON SENSOR,LOW DIAMETER,0

NETWORK,NAMESERVER,10.10.1.155

NET-COM,NET1,NEW LINE

Example 1

This example enables a key for branching to Setup Mode:

```
10 ON KEY(18) GOSUB 1000
20 KEY(18)ON
.....
1000 SETUP
1010 RETURN
```

Example 2

This example shows how a new file is opened for output and each parameter in the setup is changed by means of PRINT# statements. Then the file is closed. Any lines, except the first and the last line in the example, can be omitted. Finally, the printer setup is changed using this file.

```
10 OPEN "/tmp/SETUP.SYS" FOR OUTPUT AS #1
20 PRINT#1,"SER-COM,UART1,BAUDRATE,19200"
30 PRINT#1,"SER-COM,UART1,CHAR LENGTH,7"
40 PRINT#1,"SER-COM,UART1,PARITY,EVEN"
50 PRINT#1,"SER-COM,UART1,STOPBITS,2"
60 PRINT#1,"SER-COM,UART1,FLOWCONTROL,RTS/CTS,ENABLE"
70 PRINT#1,"SER-COM,UART1,FLOWCONTROL,ENQ/ACK,ENABLE"
80 PRINT#1,"SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA FROM
HOST,ENABLE"
90 PRINT#1,"SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA TO HOST,ENABLE"
100 PRINT#1,"SER-COM,UART1,NEW LINE,CR"
110 PRINT#1,"FEEDADJ,STARTADJ,-135"
120 PRINT#1,"FEEDADJ,STOPADJ,-36"
130 PRINT#1,"MEDIA,MEDIA SIZE,XSTART,50"
140 PRINT#1,"MEDIA,MEDIA SIZE,WIDTH,1000"
150 PRINT#1,"MEDIA,MEDIA SIZE,LENGTH,2000"
160 PRINT#1,"MEDIA,MEDIA TYPE,LABEL (w GAPS)"
170 PRINT#1,"MEDIA,PAPER TYPE,TRANSFER"
180 PRINT#1,"MEDIA,CONTRAST,-4%"
190 PRINT#1,"PRINT DEFS,PRINT SPEED,200"
```

```
200 CLOSE
210 SETUP "/tmp/SETUP.SYS"
```

Example 3

This example shows how a setup parameter is changed in the Immediate mode (or Direct Protocol) using a setup string.

SETUP"MEDIA,MEDIA TYPE,VAR LENGTH STRIP" .

This method can also be used in Programming Mode, as in this example:

10 SETUP"MEDIA,MEDIA TYPE,VAR LENGTH STRIP"

SETUP GET

Purpose

Gets the current setting for a single setup object.

Syntax

SETUP GET*<sexp1>*,*<sexp2>*,*<sexp3>*

Parameter

<sexp1>
Specifies the setup section.

<sexp2>
Specifies the setup object.

<sexp3>
Stores the result.

Notes

Refer to the printer user's guide for a list of setup sections and objects.

Example 1

```
SETUP GET "Communications,Wireless 802.11,Security,Association",A$
 ?A$
Ok

Open/WEP
Ok
```

Example 2

```
SETUP GET "Communications,Serial,COM1,Baud Rate",A$
 ?A$
 Ok

115200
 Ok
```

Example 3

```
SETUP GET "prt","MEDIA,MEDIA TYPE", B$

?B$

Ok

LABEL (W GAPS)
```

SETUP KEY

Purpose

Enables or disables access to Setup mode from the printer keypad.

Syntax

SETUP KEY ON|OFF

Parameters

ON enables entering Setup mode through the keypad, and OFF disables entering Setup mode through the keypad.

Notes

You can prevent operators or non-authorized personnel from changing printer settings from the printer keypad. SETUP KEY OFF allows the administrator to disable the SETUP key on the printer keypad, effectively preventing changes to the printer setup. SETUP KEY ON enables the SETUP key and access to the setup mode.

Setup mode can always be entered via a SETUP command.

SETUP WRITE

Purpose

Creates a file containing the current printer setup, or returns the setup file on a specified communication channel.

Syntax

SETUP WRITE ["*<sexp1>*",]*<sexp2>*

Parameters

<sexp1>
(Optional) specifies the setup section.

<sexp2>
Name of a file or device to which the current printer setup is written.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

SETUP WRITE is useful when you want to return to the current printer setup at a later time. You can make a copy of the current setup using SETUP WRITE<filename>, change the setup using a SETUP <filename> statement, and return to the original setup when needed by issuing a new SETUP<filename> statement containing the name of the copy you created.

Honeywell recommends that you create the file in the temporary memory ("tmp:"), as in SETUP WRITE "tmp:OLDSETUP". Once the file has been created in "tmp:", the file can be copied to the permanent memory ("/c") so it is not lost at power off.

When SETUP WRITE file is used to change the setup, the present TESTFEED adjustment is not affected.

Example 1

SETUP WRITE returns the printer setup in this example:

SETUP WRITE "Communications","net1:

This results in:

Communications,Ethernet,IPv4,IP Assignment Method,DHCP

Communications,Ethernet,IPv4,IP Address,0.0.0.0

Communications,Ethernet,IPv4,Subnet Mask,0.0.0.0

Communications,Ethernet,IPv4,Default Router,0.0.0.0

Communications,Ethernet,IPv4,DHCP Response,Broadcast

Communications,Ethernet,IPv6,IP Assignment Method,Automatic

Communications,Ethernet,IPv6,IP Address,2001:db8:0:1::1/64

Communications,Wireless 802.11,General,Network Name (SSID),INTERMEC

Communications,Wireless 802.11,General,Network Type,Infrastructure

Communications,Wireless 802.11,General,Roaming,Off

Communications,Wireless 802.11,General,Power Mode,Constant Awake

Communications,Wireless 802.11,General,Channel,1

Communications,Wireless 802.11,General,Hidden SSID,Disable

Communications,Wireless 802.11,Security,Security Type,None

Communications,Wireless 802.11,Security,Association,Open/WEP

Communications,Wireless 802.11,Security,Pre-Shared Key

Communications,Wireless 802.11,Security,Network Key Index,1

Communications,Wireless 802.11,Security,Network Key 1

Communications,Wireless 802.11,Security,Network Key 2

Communications,Wireless 802.11,Security,Network Key 3

Communications,Wireless 802.11,Security,Network Key 4

Communications,Wireless 802.11,Security,User Name

Communications,Wireless 802.11,Security,Password

Communications,Wireless 802.11,Security,Anonymous Name

Communications,Wireless 802.11,Security,Inner Authentication,MSCHAPv2

Communications,Wireless 802.11,Security,CA Certificate,intermec.pem

Communications,Wireless 802.11,Security,Client Certificate

Communications,Wireless 802.11,Security,Client Key

Communications,Wireless 802.11,Security,Server Common Name 1

Communications,Wireless 802.11,Security,Server Common Name 2

Communications,Wireless 802.11,Security,Validate Certificate,No

Communications,Wireless 802.11,Security,PAC,eap-fast.pac

Communications,Wireless 802.11,Security,Acquire PAC,On

Communications,Wireless 802.11,Security,Fast Roaming (CCKM),Disable

Communications,Wireless 802.11,Security,Mixed Mode (Group=TKIP),Disable

Communications,Wireless 802.11,Network,IPv4,IP Assignment Method,DHCP

Communications,Wireless 802.11,Network,IPv4,IP Address,0.0.0.0

Communications,Wireless 802.11,Network,IPv4,Subnet Mask,0.0.0.0

Communications,Wireless 802.11,Network,IPv4,Default Router,0.0.0.0

Communications,Wireless 802.11,Network,IPv4,DHCP Response,Broadcast

Communications,Wireless 802.11,Network,IPv6,IP Assignment Method,Automatic

Communications,Wireless 802.11,Network,IPv6,IP Address,::Communications,Bluetooth,Security,Disable

Communications,Bluetooth,Device Name,PC43-0000000411

Communications,Bluetooth,Discover,Enable

Communications,Bluetooth,Passkey,Communications,Bluetooth,Reserve,Disable

Example 2

In this example, the current setup is saved in the printer temporary memory under the name "SETUP1.SYS". Then the start adjustment is changed to "200" by the creation of a new setup file named "SETUP2. SYS." Finally, the created setup file is used to change the printer setup.

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

SGN

Purpose

Returns the sign (positive, zero, or negative) of a specified numeric expression.

Syntax

SGN(*<nexp>*)

Parameters

<nexp>
Numeric expression from which the sign will be returned.

Notes

The sign is returned as:

SGN(<nexp>) = -1 Negative
SGN(<nexp>) = 0 Zero
SGN(<nexp>) = 1 Positive

Example 1

Positive numeric expression:

```
10 A%=(5+5)
20 PRINT SGN(A%)
RUN
```

This results in:

```
1
```

Example 2

Negative numeric expression:

```
10 A%=(5-10)
 20 PRINT SGN(A%)
 RUN
```

This results in:

```
-1
```

Example 3

Zero numeric expression:

```
10 A%=(5-5)
20 PRINT SGN(A%)
RUN
```

This results in:

SORT

Purpose

Sorts a one-dimensional array.

Syntax

SORT<*nvar*>|<*svar*>,<*nexp1*>,<*nexp2*>,<*nexp3*>

Parameters

<*nvar*>|<*svar*>
Numeric (nvar) or string (svar) array to be sorted.

<*nexp1*>
Index of the first element to be sorted.

<*nexp2*>
Index of the last element to be sorted.

<*nexp3*>
For a string array, this value specifies how the array is sorted:
> 0: Ascending sorting
< 0: Descending sorting
= 0: Illegal value

Notes

A numeric or string array can be sorted, in its entirety or within a specified range of elements, in ASCII value order.

<*nexp3*> is used differently for numeric and string arrays. The sign always specifies ascending or descending order. For numeric arrays, the value is of no consequence, but for string arrays, the value specifies in which character position the elements will be sorted. <*nexp3*> = 0 results in error 41 ("Parameter out of range").

Example

In this example, one numeric and one string array are sorted in descending order. The string array is sorted in ascending order according to the third character position in each string:

```
10 ARRAY% (0) = 1001
20 ARRAY% (1) = 1002
30 ARRAY% (2) = 1003
40 ARRAY% (3) = 1004
50 ARRAY$ (0) = "ALPHA"
60 ARRAY$ (1) = "BETA"
70 ARRAY$ (2) = "GAMMA"
80 ARRAY$ (3) = "DELTA"
90 SORT ARRAY%,0,3,-1
100 SORT ARRAY$,0,3,3
110 FOR I% = 0 TO 3
120 PRINT ARRAY% (I%), ARRAY$ (I%)
```

```
130 NEXT
RUN
```

This results in:

```
1004 DELTA
1003 GAMMA
1002 ALPHA
1001 BETA
```

SOUND

Purpose

Makes the printer buzzer produce a sound with a specified frequency and duration.

Syntax

SOUND<*nexp1*>,<*nexp2*>

Parameters

<*nexp1*>
Frequency of the sound in Hz (maximum value 9999, a higher value will be ignored and give no sound).

<*nexp2*>
Duration of the sound in periods of 0.020 seconds each (no maximum limit).

Notes

SOUND allows you to include significant sound signals in your programs. For example, a sound signal can notify the operator that various errors have occurred.

A sound with approximately the specified frequency is produced for the specified duration. If the program encounters a new SOUND statement, it is not executed until the previous sound reaches the specified duration.

The SOUND statement even allows you to make melodies, although the musical quality may be somewhat limited. For more information on setting specific musical pitches, see KEY BEEP.

Example

The tune "Colonel Bogey" starts like this:

```
10 SOUND 392,10
20 SOUND 330,15
30 SOUND 330,10
40 SOUND 349,10
50 SOUND 392,10
60 SOUND 659,18
70 SOUND 659,18
80 SOUND 523,25
```

SPACE$

Purpose

Returns a specified number of space characters.

Syntax

SPACE$(*<nexp>*)

Parameters

<nexp>
Number of space characters to return.

Notes

SPACE is useful for complex formatting, such as in a table.

Example

This example prints two left-justified columns on the screen:

```
10 FOR Q%=1 TO 6
20 VERBOFF:INPUT "",A$
30 VERBON:PRINT A$;
40 VERBOFF:INPUT "",B$
50 VERBON
60 C$=SPACE$(25-LEN(A$))
70 PRINT C$+B$
80 NEXT Q%
90 END
RUN
```

Entering these strings:

```
January
February
March
April
May
June
July
August
September
October
November
December
```

results in:

January	February
March	April
May	June
July	August
September	October
November	December

SPLIT

Purpose

Splits a string into an array according to the position of a specified separator character and returns the number of elements in the array.

Syntax

SPLIT(*<sexp1>*,*<sexp2>*,*<nexp>*)

Parameters

<sexp1>
String to be split.

<sexp2>
String array in which the parts of the split string should be put.

<nexp>
Specifies the ASCII value for the separator.

Notes

The string is divided by a specified separating character which may found an infinite number of times in the string. Each part of the string becomes an element in the string array, but the separator character itself is not included in the array.

If the resulting array has more than four elements, you must first use a DIM statement to set the array size. If you do not, error 57 ("Subscript out of range") occurs.

Note that SPLIT does not return the number of elements in the array if the last substring is empty, as seen in Example 2 below.

Example 1

In this example a string is divided into five parts by the separator character # (ASCII 35 decimal). The result is an array of five elements numbered 0 to 4 as specified by DIM. Finally, the number of elements is also printed on the screen.

```
10 A$="ONE#TWO#THREE#FOUR#FIVE"
20 B$="ARRAY$"
30 DIM ARRAY$(4)
40 C%=SPLIT(A$,B$,35)
50 PRINT ARRAY$(0)
60 PRINT ARRAY$(1)
70 PRINT ARRAY$(2)
80 PRINT ARRAY$(3)
90 PRINT ARRAY$(4)
100 PRINT C%
RUN
```

This results in:

```
ONE
TWO
THREE
FOUR
FIVE
5
```

Example 2

```
10 A$="A#B#C#D" : GOSUB ZSUB
20 A$="A#B#C" : GOSUB ZSUB
30 END
40 ZSUB:
50 DIM ARRAY$(3): REM Maximum numbers of substrings
60 C% = SPLIT(A$,"ARRAY$",35)
70 IF RIGHT$(A$,1)= CHR$(35) THEN
80 ARRAY$(C%)= " ": C%=C%+1: REM Insert empty array entry.
90 END IF
100 FOR I%=0 to C%: PRINT ARRAY$(I%): NEXT: PRINT C%; "total member count"
110 RETURN
```

STOP

Purpose

Terminates program execution and returns the printer to Immediate Mode.

Syntax

STOP

Notes

When a STOP statement is encountered, the following message is returned to the Debug STDOUT channel (default is "uart1:"):

Break in line <line number>

You can resume execution where it was stopped by means of a CONT statement, or at a specified program line using a GOTO statement in the Immediate Mode.

STOP is usually used in conjunction with CONT for debugging. When execution is stopped, you can examine or change the values of variables using direct mode statements. You may then use CONT to resume execution.

CONT is invalid if the program has been edited during the break.

Related instructions are CONT and DBSTDIO.

Example

```
10 A%=100
20 B%=50
30 IF A%=B% THEN GOTO QQQ ELSE STOP
40 GOTO 30
50 QQQ:PRINT "Equal"
Ok
RUN
Break in line 30
Ok
PRINT A%
100
Ok
PRINT B%
50
Ok
B%=100
OK
CONT
Equal
Ok
```

STORE IMAGE

Purpose

Sets up parameters for storing an image in the printer memory.

Syntax

STORE IMAGE [RLL][KILL]<*sexp1*>,<*nexp1*>,<*nexp2*>,[<*nexp3*>],<*sexp2*>

Parameters

[RLL]
(Optional) Indicates RLL compression.

[KILL]
(Optional) Erases the image from the temporary memory at startup (recommended).

<*sexp1*>
Name of the image. Maximum 30 characters including extension.

<*nexp1*>
Width of the image in bits (= dots).

<*nexp2*>
Height of the image in bits (= dots).

[<*nexp3*>]
Size of the image in bytes (RLL only).

<*sexp2*>
Name of the protocol:

"INTELHEX"
"UBI00"
"UBI01"
"UBI02"
"UBI03"
"UBI10"

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

The name of the protocol must be entered in one sequence (for example, "INTELHEX"). Upper or lower case letters can be used.

STORE IMAGE RLL is used when the image to be received is compressed into RLL format. In this case the size of the image must be included in the list of parameters (<*nexp3*>).

STORE IMAGE KILL implies that the image will be stored in the printer temporary memory, which is erased at power off or REBOOT. Honeywell strongly recommends this option to improve performance.

To store the image permanently, copy it from the temporary memory to the permanent memory after the download is completed.

A STORE IMAGE statement must precede any STORE INPUT statement.

Example

This example shows how an Intelhex file is received via the standard input channel and stored in the printer temporary memory:

```
10 STORE OFF
20 INPUT "Name:", N$
30 INPUT "Width:", W%
40 INPUT "Height:", H%
50 INPUT "Protocol:", P$
60 STORE IMAGE N$, W%, H%, P$
70 INPUT "", F$
80 STORE F$
90 IF MID$(F$,8,2)<>"01" THEN GOTO 70
100 STORE OFF
```

STORE INPUT

Purpose

Receives and stores protocol frames of image data in the printer memory.

Syntax

STORE INPUT<*nexp1*>[,<*nexp2*>]

Parameters

<*nexp1*>
Timeout in ticks (0.01 sec) before the next character is received.

<*nexp2*>
(Optional) Number assigned to a device when it is opened for INPUT using the OPEN command. Default is the standard IN channel.

Notes

STORE INPUT receives and stores a protocol frame of image data as specified by preceding INPUT and STORE IMAGE statements, and also performs an end frame check.

STORE INPUT works differently for various types of protocol:

- INTELHEX: Receives and stores frames until timeout or end frame is received.
- UBI00-03: Receives and stores frames until timeout or required number of bytes are received.
- UBI10: Receives and stores frames until timeout or end frame is received.

Example 1

This example shows how an Intelhex file is stored using the STORE IMAGE statement. The number of input parameters may vary depending on the type of protocol (see STORE INPUT statement).

```
10 STORE OFF
20 INPUT "Name:", N$
30 INPUT "Width:", W%
40 INPUT "Height:", H%
50 INPUT "Protocol:", P$
60 STORE IMAGE N$, W%, H%, P$
70 STORE INPUT 100
80 STORE OFF
```

Example 2

To receive the input from another channel than standard IN channel, the device must be opened for INPUT using the OPEN command and a reference must be included in the STORE INPUT statement.

```
10 STORE OFF
20 OPEN "uart2:" FOR INPUT AS #9
30 INPUT "Name:", N$
40 INPUT "Width:", W%
50 INPUT "Height:", H%
60 INPUT "Protocol:", P$
70 STORE IMAGE N$, W%, H%, P$
80 STORE INPUT 100,9
90 CLOSE #9
100 STORE OFF
```

STORE OFF

Purpose

Terminates the storing of an image and resets the storing parameters.

Syntax

STORE OFF

Notes

After storing all protocol frames of an image, storing must be terminated by STORE OFF. To store another image, you must issue a STORE OFF statement before the parameters for the new image can be set up using a new STORE IMAGE statement.

Honeywell recommends that you always start an image storing procedure by issuing a STORE OFF statement to clear the parameters of any existing STORE IMAGE statement.

Example

This example shows how an Intelhex file is received via the standard IN channel and stored in printer memory:

```
10 STORE OFF
20 INPUT "Name:", N$
30 INPUT "Width:", W%
40 INPUT "Height:", H%
50 INPUT "Protocol:", P$
60 STORE IMAGE N$, W%, H%, P$
70 STORE INPUT 100
80 STORE OFF
```

STR$

Purpose

Returns the string representation of a numeric expression.

Syntax

STR$(*<nexp>*)

Parameters

<nexp>
Numeric expression from which the string representation is returned.

Notes

This is the complementary function for the VAL function.

Example

In this example, the value of the numeric variable A% is converted to string representation and assigned to the string variable A$:

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

This results in:

```
246
123123
```

STRING$

Purpose

Repeatedly returns the character of a specified ASCII value, or the first character in a specified string.

Syntax

STRING$(*<nexp1>*,*<nexp2>*|*<sexp>*)

Parameters

<nexp1>
Number of times the specified character should be repeated.

<nexp2>
ASCII decimal code of the character to be repeated.

<sexp>
String expression from which the first character will be repeated.

Notes

The character to be repeated is specified either by its ASCII decimal code according to the selected character set, or as the first character in a specified string expression.

Example

This example shows both ways of using STRING$. The asterisk character (*) is ASCII 42 decimal:

```
10 A$="*INTERMEC*"
20 LEADING$ = STRING$(10,42)
30 TRAILING$ = STRING$(10,A$)
40 PRINT LEADING$; A$; TRAILING$
RUN
```

This results in:

```
**********INTERMEC**********
```

SYSHEALTH

Purpose

Sets or retrieves the system health and controls the Ready-to-Work indicator on the printer front panel.

Syntax 1

SYSHEALTH=<*nexp1*>

Parameters 1

<*nexp1*>
Sets the application view of the system health and controls the Ready-to-Work indicator:

1: Indicator off
2: Indicator blinking
3: Indicator on (default)

Syntax 2

<*nvar1*>=SYSHEALTH

Parameters 2

<*nvar1*>
Returns the current system health status as shown by the Ready-to-Work indicator:

1: Indicator off
2: Indicator blinking
3: Indicator on

Notes

The readiness of the printer, individually or as a part of a solution, is indicated by the blue Ready-to-Work indicator.

If the indicator blinks or is switched off, the printer is not ready. In case of several errors or similar conditions occurring simultaneously, only the most significant error is displayed. Once this error has been cleared, the next remaining error is displayed.

Provided the printer is connected to a network, all conditions that prevent printing are reported to SmartSystems.

The SYSHEALTH variable adds more functionality to the Ready-to-Work indicator than is offered by the printer. However, it does not override the standard indicator handling. The worst case is always reported regardless if it is a system error or an application error.

Example

This example shows how the Ready-to-Work indicator can be made to show a "Connection refused" condition:

```
10 ON ERROR GOTO 1000
50 TRANSFER NET
"ftp://wrong.server.com/file","c/myfile"
60 PRINT "XXX"
100 END
1000 IF ERR=1833 THEN SYSHEALTH=2 ELSE SYSHEALTH=3
1010 RETURN
```

You can find out the health of the system this way:

```
A%=SYSHEALTH
PRINT A%
```

SYSHEALTH$

Purpose

Returns the error causing the current system health status.

Syntax

<*svar*>=SYSHEALTH$

Parameters

<svar>
Returns the error causing the current status, such as "Head lifted."

Notes

If SYSHEALTH = 3, SYSHEALTH$ returns "Operational."

Example

```
A$=SYSHEALTH$
PRINT A$
```

This results in:

```
Out of paper
```

SYSVAR

Purpose

System array for reading or setting system variables.

Settings in SYSVAR are reset to default values after you restart your printer.

Syntax

SYSVAR(*<nexp>*)

Parameters

<nexp>
Reference number of the system variable as described in the next table.

No.	Description
0-13	Reserved, obsolete or not implemented.
14	Reads number of errors detected since last power up.
15	Reads number of errors detected since last executed SYSVAR(15).
16	Reads the number of bytes received after the execution of a STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.
17	Reads the number of frames received after the execution of a STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

No.	Description
18	Read or Set verbosity level. In the Immediate and Programming Modes, all levels are enabled by default. In the Direct Protocol, all levels are disabled by default. Different verbosity levels can be selected: -1: All levels enabled (= VERBON) 0: No verbosity (= VERBOFF) 1: Echo received characters 2: "Ok" after correct command lines 4: Echo input characters from communication port 8: Error after failed lines The levels can be combined, so for example SYSVAR(18)=3 means both "Echo received characters" and "Ok after correct command line." The presently selected verbosity level can also be read and is returned as a numeric value, for example by PRINT SYSVAR(18).
19	Read or Set type of error message. Four types of error messages can be selected: 1: <string> in line <line> (default) for example "Invalid font in line 10" 2: Error <number> in line <line>: <string>, for example "Error 19 in line 10: Invalid font" 3: E<number>, for example "E19" 4: Error <number> in line <line>, for example "Error 19 in line 10" The presently selected type of error message can also be read and is returned as a numeric value (1-4), for example by PRINT SYSVAR(19).
20	Read if the printer is set up for direct thermal printing or thermal transfer printing, which is decided by your choice of paper type in the printer setup. The printer returns: 0: Direct thermal printing 1: Thermal transfer printing
21	Read the printhead density, expressed as number of dots per millimeter.
22	Read number of printhead dots.

No.	Description
23	Read status of transfer ribbon sensor in thermal transfer printers. The printer returns: 0: No ribbon detected 1: Ribbon detected
24	Returns whether the printer has been restarted since the last time SYSVAR(24) was used. This system variable is important when using the Direct Protocol. At power up, all data not saved as programs, files, fonts or images is deleted, and most instructions will be reset to their respective default values. SYSVAR(24) allows the host to poll the printer to see if a power up has occurred, for example because of a power failure and, if so, download new data and new instructions. The printer returns: 0: No power up since last SYSVAR(24) 1: Power up has occurred since last SYSVAR(24)
25	Obsolete
26	Read the status of the ribbon low sensor, assuming that the printer is fitted with a thermal transfer mechanism. In Setup mode (Media/Paper Type/Transfer/Low Diameter), you can specify a diameter in mm of the ribbon supply roll, when SYSVAR(26) will switch from 0 to 1. The printer returns: 0: Ribbon not low 1: Ribbon low By default, the Low Diameter is set to 0, which disables the ribbon low function. However, if the Low Diameter is set to a higher value than 0 and SYSVAR(26) returns 1, the error condition 1083 "Ribbon Low" occurs at every tenth PRINTFEED operation. Further actions must be taken care of by the running Fingerprint program.
27	Set condition for label reprinting at out-of-ribbon error. When printing a batch of labels using thermal transfer printing (OPTIMIZE "BATCH" ON or PRINTFEED<n>), a label is deemed erroneous and thus eligible for reprinting if the ribbon has been empty for a distance longer than specified in dots by SYSVAR(27). Default is 0. Non-negative integers only.

No.	Description
28	Set or read media feed data erase at printhead lift. Fingerprint keeps track of media between the label stop sensor and the dot line of the printhead. If the printhead is lifted, there is a large risk that the media is moved, so media feed will not work correctly before those labels have been fed out. This parameter allows you to decide or read whether these data should be cleared or not when the printhead is lifted. 0: Media feed data are not cleared at head lift 1: Media feed data are cleared at head lift (default) 2: Media feed data are cleared at head lift and the firmware looks for the first gap or mark and adjusts the media feed using the same data as before the head was lifted.
29	Read DSR (Data Send Ready)condition on "uart2:" The printer returns: 0: No 1: Yes
30	Read DSR (Data Send Ready) condition on "uart3:" The printer returns: 0: No 1: Yes
31	Read last sent ACK, NAK, or CAN character in the MUSE protocol. This parameter allows you to read the last control character sent from the MUSE protocol (special applications). The printer returns one of the following alternatives: NUL, ACK, NAK, CAN.
32	Returns the length of media feed past the printhead. Resolution: 10 meters.
33	Read DSR (Data Send Ready) condition on "uart1:".

No.	Description
34	Read or Set positioning mode for TrueType characters This parameter allows you to select one of three modes for the positioning of TrueType characters and also to read for which mode the printer is set. The modes are: 0: Standard mode (default). 1: Compatible mode. This mode is compatible with Fingerprint 7.xx earlier than version 7.2. 2: Adjusted mode.
35	This parameter allows you to decide whether a change in the printer setup is to be saved as a file (which is effective after you restart the printer) or not be saved (volatile). You can also read for which alternative the printer is set. Note that the SYSVAR (35) setting at the moment when the new setup is entered decides whether it will be saved or not. The alternatives are: 0: Setup saved to file (Default). 1: Setup not saved to file.
36	Print changes of program modes. This parameter is used with the Fingerprint debugger and controls whether changes of program modes should be printed to the Debug Standard Out port (see DBSTDIO). The options are: 0: Disable printout (default) 1: Enable printout
37	Set minimum gap length. The media may have perforations or marks that are not intended to be interpreted as gaps or black marks by the LSS. Using this SYSVAR parameter, it is possible to make the LSS ignore gaps or marks that are shorter that a specified value. (In this context, long and short are related to the media feed direction.) The minimum gap length is specified in dots within a range of 1-32. Default value is 1 mm (0.039 inches). Note that SYSVAR(37) affects PRINTFEED and FORMFEED.
38	Obsolete and has no effect, even if it does not cause an error if used.

No.	Description
39	Enable/disable slack compensation. Forces start adjust. Label slack compensation is a method of eliminating slack in the belts after having fed the media back. At a negative FORMFEED, the printer will pull back the media slightly more than specified by the FORMFEED statement and then feed the media forward the same distance. If slack compensation is enabled, and FORMFEED -100 is specified, the printer will pull back the media for example -112 dots and then feed the media forward +12 dots to take out the slack. Performing a positive FORMFEED normally feeds out paper even if start adjust is set to a negative value. Using narrow labels and liner rewinding may cause the printout position to differ on the next printed label after a FORMFEED. Forcing a start adjust when doing FORMFEED improves printout precision in such cases. The options are: 0: Disable slack compensation and disable force start adjust. 1: Enable slack compensation (default). 2: Force start adjust at FORMFEED. 3: Force start adjust at FORMFEED and enable slack compensation.
40	Not implemented.
41	"Next label not found" at predefined feed length. The automatic detection of the error condition "Next label not found" (error 1031) by the label stop sensor can be overridden by specifying a fixed length in dots. The length should preferably correspond to at least the distance between the tops of two consecutive labels. During printing, error 1031 occurs if the media does not come loose from the core (media glued to core) or if a label is missing on the liner. Especially useful for short labels (10–40 mm/0.4–1.5 inches long). Default value is 0.
42	Stop media feed in the middle of label gaps: 0: The media feed stops so the middle of a 3 mm (0.12 in) gap becomes aligned with the tear bar when using labels (w gaps). This is the default setting. 1: The media feed stops so the middle of the gap becomes aligned with the tear bar, regardless of gap size.

No.	Description
43	Enable/disable file name conversion, meaning lowercase characters will be converted to uppercase and the extension .PRG will be added if an extension is missing: 0: File name conversion is enabled (default) 1: File name conversion is disabled.
44	Enable/disable filtering of NUL characters in background communication (see COMBUF$): 0: Enables filtering (default) 1: Disables filtering
45	Returns the resolution of the printhead, expressed in dots per inch (dpi).
46	Read status of paper low sensor: 0: Indicates that the diameter of the media supply is larger than specified in the Setup Mode. 1: Indicates that the detected diameter of the media supply roll is equal or less than the diameter specified in Setup mode (Media/Paper/Low Diameter). The error condition 1084 "Paper low" will occur. This error does not stop the printing, but interrupts any program that does not handle it.
47	Enable/disable use of start adjust-stop adjust together with positive and/or negative formfeeds: 0: Use of start adjust/stop adjust OR negative positive Formfeed values (default). 1: Enable the use of start adjust/stop adjust values together with negative/positive Formfeed values.

No.	Description
48	Enable/disable bidirectional direct protocol: 0: Disables use of direct commands (default) 1: Scans standard IN channel for direct commands. Can only be set in Direct Protocol. The supported immediate/bidirectional Direct Protocol commands are: ^s: Status request. The printer will respond with its current status in the form: Response: cnt: <INT_CNT> left:<INT_LEFT> prstat:<INT_PRSTAT> pause:<INT_PAUSE> errno:<INT_ERRNO> recbuf:<INT_RECBUF> Description: <INT_CNT> is the total number of labels to be printed in the current print job. <INT_LEFT> is the total number of lables left in the current print job. <INT_PRSTAT> is the standard PRSTAT, except for LTS status, which has to be enabled by the command LTS&ON to detect a remaining label. <INT_PAUSE> is 1 if the current print job is paused either by console or by command, 0 otherwise. <INT_ERRNO> is the latest error code that has occurred in FP/DP. <INT_RECBUF> is the remaining number of bytes in the receive buffer. ^q: Status query. Same action and response as ^s, but the command is not added to the receive buffer. ^a: Abort current print job.
49	Set temporarily lower speed after negative start adjust. This is used to avoid hard stretches at the beginning of a label with a full roll. Specify the percentage of the negative start adjust value that should be fed slower (70 mm/s). For example SYSVAR(49)=150 will cause the printer to run slower for 150% of the negative start adjust value, before speeding up.
50	Set lower speed for given length after lowering printhead. To avoid hard stretches after loading new media roll, this can lower the speed (to 70 mm/s) for the specified length after lowering the printhead. The value is given in dots. Default is 0.
51	Set the value in meters for SYSVAR(49) and SYSVAR(50) that will be in effect after lowering the printhead.

No.	Description
53	Set or read the highest allowed outer diameter (in mm) of the ribbon supply. Range is 40 to 1000 mm. Default is 83. Note that a larger value causes out of ribbon detection to require a longer feed.
54	Modified the DNS timeout value in increments of 30 seconds. Default is 5 (150 sec).
58	Enable/disable RFID retract for negative values. This is useful when going beyond the printhead.
59	Show or hide the two line text window used in a Fingerprint program.
84	Set or read start character used with inline string formatting commands. The character is specified with a decimal representation of the byte code (0-255) desired. To disable the functionality (disabled by default), set the start character to -1. This variable is only active if SYSVAR(85) is set to a valid value.
85	Set or read stop character used with PRTXT/PRBOX inline formatting commands. The character is specified with a decimal representation of the byte code (0- 255) desired. To disable the functionality (disabled by default), set the start character to -1. This variable is only active if SYSVAR(84) is set to a valid value.
105	Set/Read label verification status 0 = Label verification not completed 1 = Label verification completed
106	Read label verification result 0 = Label verification pass Non-Zero = Label verification failed, and error code is returned

Example 1

Reading the value of a system variable, in this case the transfer ribbon sensor:

```
PRINT SYSVAR(23)
```

Example 2

Setting the value of a system variable. In this case verbosity is disabled:

```
SYSVAR(18)= 0
```

TAGFIELD

Purpose

Defines a field in the RFID tag memory area available for RFID operations.

Syntax

TAGFIELD[<*sexp1*>,]<*sexp2*>[,<*nexp3*>[,<*nexp4*>]]

Parameters

<*sexp1*>
(Optional) Name of the field to make available.

<sexp2>
Name of the segment to make available. Allowed values are described in the next table.

Tag Type	Values for <sexp2>
EPC Class 1 Version 1	"@ID": Tag identification segment for EPC data. Default.
ISO 18000-6B or EPC UCode 1.19	"@ID": Tag identification segment (default) "@DATA": Optional data segment "@ALL": Tag's complete memory structure
EPC Class 1 Gen 2	"@RESERVED": Access and kill password segment. "@EPC": EPC specific data (default) "@TID": Tag identifier. "@USER": User defined data.

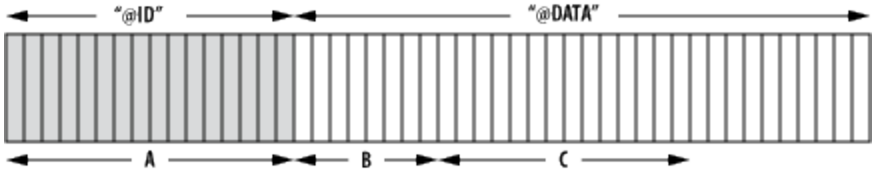
<*nexp3*>
Field starting byte in the chosen segment. Default varies according to RFID tag standard and segment in use. Must be an even number for Generation 2 RFID tags.

<*nexp4*>
Length of the field in bytes. Default varies according to the RFID tag standard and EPCGlobal tag format in use. Must be an even number for Generation 2 RFID tags.

Notes

TAGFIELD specifies the field available for subsequent RFID commands, such as TAGWRITE or TAGREAD. The memory structure of RFID depends on the tag type. Class 1 tags only allow the "@ID" segment, where EPC data is stored. ISO18000-6B or EPC

UCode 1.19 tags contain two segments, "@ID" and "@DATA", as seen in the next illustration:



Generation 2 (Gen 2) RFID tags have four segments, "@RESERVED", "@EPC", "@TID", and "@USER".

The *<nexp3>* and *<nexp4>* parameters specify a subset of a segment as the currently available field. It is possible to name the field to use with *<sexp1>*. This name may not start with an @-character. A possible representation of tag segments can be seen in the illustration.

The default values for *<nexp1>* and *<nexp2>* may be modified by a subsequent TAGFORMAT command (see Chapter 6). Values for *<nexp3>* and *<nexp4>* do not normally need to be set when writing EPC information, as these are automatically adjusted for the EPC tag format specified. The only exception is ISO 18000-6B tags.

TAGFIELD resets TAGFORMAT to default. Related commands are TAGFORMAT, TAGREAD and TAGWRITE.

Example

The fields A, B, and C in the illustration above are defined by:

A: TAGFIELD "@ID"
B: TAGFIELD "@DATA",0,8
C: TAGFIELD "C","@DATA",8,14

The field name "C" can then be reused:

C: TAGFIELD "C"

TAGFORMAT

Purpose

Specifies the format of the data to be read from or written to an RFID tag. For more information, see Supported RFID Tag Formats.

Syntax

TAGFORMAT<*sexp*>

Parameters

<sexp>
Format of the data. Available formats are:

"ASCII" 8-bit ASCII string.
"HEX" (Default) Hexadecimal string. Values 0-9 and a-f allowed. Hex characters must be entered in pairs.
"NUM" Integer. Valid range is 0 to 2147483647. Always uses 4 bytes to represent data, unless a smaller field has been defined, and the number fits in that field. Not allowed for Class1 tags.

The following formats are supported by EPCglobal tags only:

"SGTIN-64" "Filter","Company Prefix","Item Reference","Serial Number"
"SGTIN-96" "Filter","Company Prefix","Item Reference","Serial Number"
"SSCC-64" "Filter","Company Prefix","Serial Reference"
"SSCC-96" "Filter","Company Prefix","Serial Reference"
"SGLN-64" "Filter","Company Prefix","Location Reference","Serial Number"
"SGLN-96" "Filter","Company Prefix","Location Reference","Serial Number"
"GRAI-64" "Filter","Company Prefix","Asset Type","Serial Number"
"GRAI-96" "Filter", "Company Prefix", "Asset Type", "Serial Number"
"GIAI-64" "Filter", "Company Prefix", "Individual Asset Reference"
"GIAI-96" "Filter", "Company Prefix", "Individual Asset Reference"
"GID-96" "General Manager Number", "Object Class", "Serial Number"
"USDOD-64" "Filter", "Government Managed Identifier","Serial Number"
"USDOD-96" "Filter", "Government Managed Identifier", "Serial Number"
"EPC-HEX64" Hex values for all bits on a 64-bit tag's memory area.
"EPC-HEX96" Hex values for all bits on a 96-bit tag's memory area.
"EPC-URN" Uniform Resource Name string. Standardized format for entry of EPC identity.

Notes

The TAGFORMAT command specifies the format of the data to be read from or written to a tag with a subsequent TAGREAD or TAGWRITE command. The format applies to the field defined by the most recent TAGFIELD command.

TAGFORMAT is reset to default by a TAGFIELD command. Other related commands are TAGREAD and TAGWRITE.

Example 1

```
10 FILTER$ = "3"
20 PREFIX$ = "0614141"
30 ITEM$ = "100734"
40 SERIAL$ = "2"
50 TAGFIELD "@EPC"
60 TAGFORMAT "SGTIN-96"
70 TAGWRITE FILTER$, PREFIX$, ITEM$, SERIAL$

80 TAGFIELD "@ID"
90 TAGFORMAT "EPC-URN"
100 TAGREAD MyURI$
110 PRINT MyURI$
RUN
```

This results in:

urn:epc:sgtin-96:3.0614141.100734.2

Example 2

```
10 TAGFIELD "@USER",10,4
20 TAGFORMAT "ASCII"
30 TAGWRITE "RFID"
40 TAGREAD A$
50 PRINT A$
RUN
```

This results in:

RFID

TAGPROTECT

Purpose

Protects tag data from being overwritten.

Syntax 1

(EPCglobal Class 1, ISO 18000-6B, and EPC UCode v1.19 Tags only)

TAGPROTECT*<sexp1>*

Parameters 1

<sexp1>
Specifies the tag protection level for a field or segment. Available options are:

"ON" = Tag is protected.
"OFF" = Tag remains unprotected.

Syntax 2

EPC Gen 2 tags only)

TAGPROTECT*<sexp1>[,<sexp2>]*

Parameters 2

<sexp1>
Specifies the tag protection level for a field or segment. Available options are:

"LOCK" = The field is locked and the data cannot be overwritten until the lock is revoked.
"LOCKP" = The field is permanently locked.
"UNLOCK" = The field is unlocked and data stored within the field is not protected.
"UNLOCKP" = The field is permanently unlocked.

<sexp2>
Consists of a 4-byte access password stored in the "@RESERVED" segment on the tag. Only the HEX format can be used for the password (see TAGFORMAT). Password is set with TAGWRITE.

Notes

TAGPROTECT protects the most recently referenced tag from being overwritten by an RFID operation. The protection is applied during the next TAGWRITE operation, since TAGPROTECT does not align the tag over the antenna.

Related commands are TAGFIELD, TAGFORMAT and TAGWRITE.

Examples

Gen 1 example:

```
10 TAGFIELD "@ID"
20 TAGFORMAT "SSCC-64"
30 TAGPROTECT "ON"
40 TAGWRITE "1","12345","123456"
```

Gen 2 example (assumes that the access password "12345678" is stored in the @RESERVED memory bank of the tag):

```
TAGFIELD "@EPC",4,12
TAGFORMAT "SGTIN-96"
TAGPROTECT "LOCK","12345678"
TAGWRITE "3","2747561","158310","74"
```

TAGREAD

Purpose

Reads an RFID tag field.

Syntax

TAGREAD<*nvar*>|<*svar1*>[,<*svar2*>,<*svar3*>,<*svar4*>]

Parameters

<*nvar*>
Numeric variable that stores the data. Only used with the "NUM" field in TAGFORMAT.

<*svarN*>
String variable that stores the data. The number of arguments depends on TAGFORMAT.

Notes

TAGREAD reads the data from the field specified by the latest TAGFIELD command into the variable <*nvar*> or variables <*svarN*>.

The format of the data is defined by the latest TAGFORMAT command. If you state a numeric variable <*nvar*>, an error is returned unless the data is in the "NUM" format.

Related commands are TAGFIELD, TAGFORMAT and TAGWRITE.

Example 1

Using numeric variables:

```
10 TAGFIELD "@DATA",2,2
20 TAGFORMAT "NUM"
30 TAGWRITE 19562
40 TAGFIELD "@DATA",2,2
50 TAGFORMAT "NUM"
60 TAGREAD A%
RUN
```

This results in:

```
19562
```

Example 2

Reading a SGTIN-96 tag from a Gen 2 tag:

```
10 TAGFIELD "@EPC"
20 TAGFORMAT "SGTIN-96"
30 TAGREAD FILTER$, PREFIX$, ITEM$, SERIAL$
40 PRINT FILTER$, PREFIX$, ITEM$, SERIAL$
RUN
```

This results in:

3 0614141 100734 2

TAGWRITE

Purpose

Writes to an RFID tag field.

Syntax

TAGWRITE<*nvar*>|<*svar1*>[,<*svar2*>,<*svar3*>,<*svar4*>]

Parameters

<*nvar*>
Numeric variable to be written. Only used with the "NUM" field in TAGFORMAT.

<*svarN*>
String variable to be written. The number of arguments depends on TAGFORMAT.

Notes

TAGWRITE writes the data from <*nvar*> or <*svar1*> to the field specified by the latest TAGFIELD command. The format of the data is specified by the latest TAGFORMAT command. If the data written is shorter than the field, the field is padded with zeroes.

The exception to this rule occurs in some EPC tag formats where the exact number of digits must be entered, even if it means adding non-significant digits (for example, writing 00234 instead of 234). If the data is too long to fit in the specified field, an error is returned. For more information, see Supported RFID Tag Formats.

Related commands are FORMAT$, TAGFIELD, TAGFORMAT and TAGREAD.

Examples

Writing an SGTIN-96 to a Gen 2 tag:

```
10 TAGFIELD "@EPC"
20 TAGFORMAT "SGTIN-96"
30 TAGWRITE "3","0614141","100734","2"
```

Two examples of writing the same SSCC-64 info to an ISO18000-6B tag:

```
10 TAGFIELD "@DATA",10,8
20 TAGFORMAT "SSCC-64"
30 TAGWRITE "1","12345","123456"
```

```
10 TAGFIELD "@DATA",10,8
20 TAGFORMAT "EPC-URN"
30 TAGWRITE
"urn:epc:tag:sscc-64:1.12345.123456"
```

TESTFEED

Purpose

Adjusts the label stop, ribbon end/low and paper low sensors, and RFID module while running the media and ribbon feed mechanisms.

Syntax

TESTFEED[*<nexp>*]

Parameters

<nexp>
(Optional) Feed length in dots.

Notes

TESTFEED feeds *<nexp>* dots while calibrating the label stop/black mark sensor (LSS) for the characteristics of the media loaded in the printer. The statement is needed to detect media, gaps, black marks, and out-of-paper conditions, and should be done for all media types.

In the setup, TESTFEED MODE can be set to SLOW, which might be necessary when using media with pre-printed lines. This is done with a SETUP command or by placing the printer in Setup Mode. When in SLOW mode, TESTFEED samples the media length plus 10 mm. Alternately, the length sampled can be set using the MEDIA,LEN (SLOW MODE) option, the minimum being the number of dots corresponding to 10 mm. This value is ignored when TESTFEED MODE is set to FAST.

If an RFID module is installed, and RFID ON is set, TESTFEED attempts to identify the RFID tag. This is always performed in SLOW mode.

If *<nexp>* is omitted, it is automatically set to 1.5 times the media length specified in the setup. For the TESTFEED to be successful, at least one gap or black mark must pass the LSS. Best results for "Ticket w Mark" are obtained with a *<nexp>* value of 1200 or any other reasonable number.

When a TESTFEED is executed, the ribbon end/low and paper low sensors are also calibrated (if installed). However, this does not apply when the testfeed is ordered using the testfeed option in Setup Mode.

In Immediate Mode, a TESTFEED is performed when the **Shift** and **Feed** keys are pressed simultaneously.

Since TESTFEED is essential for a proper media load, some facility for issuing a TESTFEED statement should be included in all custom-made label-printing programs as seen in the example.

Example

This program performs a TESTFEED statement when the **Shift** and **Feed** keys are pressed simultaneously on the printer keypad:

```
10 ON KEY (117) GOSUB QTESTFEED
20 KEY (117) ON
```

```
30 QLOOP:
40 GOTO QLOOP
. . . . .
1000 QTESTFEED:
1010 TESTFEED
1020 RETURN
```

TICKS

Purpose

Returns the elapsed time since the last power up in the printer, expressed in number of "TICKS" (1 TICK = 0.01 sec).

Syntax

TICKS

Notes

TICKS allows you to measure time more exactly than the TIME$ variable, which cannot handle time units smaller than 1 second. The TICKS counter is reset to zero at power up.

Example

```
10 A%=TICKS
20 PRINT A%
RUN
```

This results in:

1081287

In this case, the time which has passed since the printer was started is 10812.87 seconds, or 3 hours 12.87 seconds.

TIME$

Purpose

Sets or returns the current time.

Syntax 1

Setting the time:

TIME$=<*sexp*>

Parameters 1

<*sexp*>
Sets the current time by a 6-digit number. By default, time is always entered and returned as HHMMSS, where:

HH = Hours: Two digits (00-23)
MM = Minutes: Two digits (00-59)
SS = Seconds: Two digit (00-59)

Time is entered as a 24-hour cycle (for example, 8 P.M. is entered as "200000").

Syntax 2

Reading the time:

<*svar*>=TIME$[(<*sexp*>)]

Parameters 2

<*svar*>
Returns the current time according to the printer clock.

<*sexp*>
(Optional) Flag "F", indicating that the time will be returned according to the format specified by FORMAT TIME$.

Notes

TIME$ works best if the printer has a real-time clock (RTC). The RTC keeps track of the time even if the printer is restarted. Honeywell mobile printers do not have an RTC.

If no RTC is installed, the internal clock is used. After startup, an error occurs when trying to read the date or time if the internal clock has been not been manually set using either a DATE$ or TIME$ variable.

If only the date is set, the internal clock starts at 00:00:00, and if only the time is set, the internal clock starts at Jan 01 1980. After setting the internal clock, you can use the DATE$ and TIME$ variables the same way as when an RTC is fitted, until a power off or REBOOT causes the date and time values to be lost.

The format for how the printer returns time from a TIME$("F") variable can be changed using a FORMAT TIME$ statement.

Example

This example sets and reads the time, then prints it on the host screen:

```
10 TIME$ = "154300"
20 FORMAT TIME$ "HH.MM"
30 PRINT "Time is "+TIME$("F")
RUN
```

This results in:

```
Time is 15.43
```

TIMEADD$

Purpose

Returns a new time after a number of seconds have been added to or subtracted from the current time or from a specified time.

Syntax

TIMEADD$([<*sexp1*>,]<*nexp*>[,<*sexp2*>])

Parameters

<*sexp1*>
Any time given according to the TIME$ format, which a certain number of seconds should be added to or subtracted from.

<*nexp*>
Number of seconds to be added to (or subtracted from) the current time, or (optional) the time specified by <*sexp1*>.

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

Notes

This function works best if a real-time clock circuit (RTC) is fitted on the printer's CPU board. Honeywell mobile printers do not have an RTC.

The original time (<*sexp1*>) should always be entered according to the TIME$ format (HHMMSS). Time is entered as a 24-hour cycle (for example, 8 P.M. is entered as "200000").

The number of seconds to be added or subtracted from the original time should be specified as a positive or negative numeric expression respectively.

If no "F" flag is included in the TIMEADD$ function, the result is returned according to the TIME$ format.

If the TIMEADD$ function includes an "F" flag, the result is returned in the format specified by FORMAT TIME$.

Example 1

```
10 A%=30
20 B$=TIMEADD$ ("133050",A%)
30 PRINT B$
RUN
```

This results in:

133120

Example 2

```
10 TIME$="133050"
20 FORMAT TIME$ "hh.mm.ss pp"
30 A% = -40
40 PRINT TIMEADD$(A%,"F")
RUN
```

This results in:

```
01.30.10 pm
```

TIMEDIFF

Purpose

Returns the difference between two specified times in number of seconds.

Syntax

TIMEDIFF(*<sexp1>*,*<sexp2>*)

Parameters

<sexp1>
First point in time (time 1).

<sexp2>
Second point in time (time 2).

Notes

This variable works best if a real-time clock circuit (RTC) is fitted on the printer's CPU board. Honeywell mobile printers do not have an RTC.

To get the result as a positive value, the two points of time should be entered with the earlier moment (time 1) first and the later moment (time 2) last, as seen the first example. If the later moment (time 2) is entered first, the resulting value is negative, as seen in the second example.

Enter the time according to the standard TIME$ format HHMMSS. The resulting difference in seconds is returned.

Example 1

PRINT TIMEDIFF ("133050","133120")

This results in:

30

Example 2

PRINT TIMEDIFF ("133120","133050")

This results in:

-30

TRANSFER KERMIT

Purpose

Transfers data files using the KERMIT communication protocol.

Syntax

TRANSFER K[ERMIT]<*sexp1*>[,<*sexp2*>]

Parameters

<*sexp1*>
Specifies the direction of the transmission by the expression "S" (= send) or "R" (= receive).

<*sexp2*>
(Optional) Name of the file transmitted from the printer. Default is "KERMIT.FILE".

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

Kermit is a protocol for serial binary transfer of a complete file, and is included in HyperTerminal and other communication programs. For more information, consult the application program documentation.

TRANSFER KERMIT can only handle a single file at a time. When transmitting files from the printer to the host, carefully observe possible restrictions on the number of characters in the file name that may be imposed by the host operating system.

When receiving a file, you must start the transmission within 30 seconds of completing the TRANSFER KERMIT "R" statement. The printer stores the file in the current directory ("/c", "tmp:"). Files cannot be received into "/rom". If a file in the current directory has the same name as the one to be transferred, the existing file will be replaced by the new file.

Thus, you need to keep track of the files already stored in the current directory (see FILES statement). Give the new file a name not already used by an existing file, unless you want to replace the existing file. Downloaded fonts and images are auto-installed.

Example 1

This example sets up the printer to receive a file on the standard IN channel:

```
TRANSFER KERMIT "R"
```

Example 2

This example sends "FILE1.TXT" from the printer to the host on a channel other than the standard OUT channel:

```
TRANSFER K "S","FILE1.TXT","uart2:","uart2:"
```

TRANSFER NET

Purpose

Transfers files to and from the printer using FTP.

Syntax

TRANSFER N[ET] <*sexp1*>,<*sexp2*>[,<*sexp3*>]

Parameters

<*sexp1*>
Source file. If the source is a local file, this file is sent from the printer to the destination specified by <*sexp2*>. If the source is a URI, this file is fetched from the server, sent to the printer, and stored at the location specified by <*sexp2*>.

<*sexp2*>
Destination of the file transfer.

<*sexp3*>
(Optional) An account secret.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

TRANSFER NET is not a complete FTP client. It only supports file transfer to and from the printer in binary format. Only one file can be transferred per command. File transfer between two local or two remote files is not supported.

A local file is a path to an existing file (when sending from the printer) or to the file that is created (when sending to the printer). If a local file already exists when sending a file to the printer, the existing file is replaced if it is not write-protected. A read-protected file is not sent. If the destination is a local directory, the sent file gets the same name as the source file.

Enter a URI in the format:

ftp://[<user>:<password>@]<server>[:port]/<path>

Entries inside square brackets [...] are optional. The following default values are used:

- user: anonymous
- password: nopass@<ip address>
- port: 21

If the destination is a URI specifying a directory, the sent file gets the same name as the source file.

Example

This example shows how the file README.uploads is fetched from the sunet ftp server and stored as UPLOAD.TXT in the current directory. The default user, password and port number are used.

```
TRANSFER NET "ftp://ftp.sunet.se/README.uploads",
 "UPLOAD.TXT"
```

TRANSFER STATUS

Purpose

Checks the last TRANSFER KERMIT or TRANSFER ZMODEM operation.

Syntax

TRANSFER S[TATUS]<*nvar*>,<*svar*>

Parameters

<*nvar*>
Five-element, one-dimensional numeric array where the elements return:

0: Number of packets. (Kermit only)
1: Number of NAK's. (Kermit only)
2: ASCII value of last status character. (Kermit only)
3: Last error. (Kermit and ZMODEM)
4: Block check type used. (Kermit only)

<*svar*>
Two-element ,one-dimensional string array where the elements return:

0: Type of protocol. ("KERMIT" or "ZMODEM")
1: Last file name received.

Notes

After a file transfer using the Kermit or ZMODEM protocol has been performed (using TRANSFER KERMIT or TRANSFER ZMODEM), you can check how the transfer was performed. Note that the numeric array requires the use of a DIM statement, since the array contains more than four elements.

Example

```
10 TRANSFER KERMIT "R"
20 DIM A%(4)
30 TRANSFER STATUS A%, B$
40 PRINT A%(0), A%(1), A%(2), A%(3), A%(4)
50 PRINT B$(0), B$(1)
.....
.....
```

TRANSFER ZMODEM

Purpose

Transfers data files using the ZMODEM communication protocol. Only the standard IN and OUT channels are supported.

Syntax

TRANSFER Z[MODEM]<*sexp1*>[,<*sexp2*>[,<*sexp3*>[,*sexp4*>]]]

Parameters

<*sexp1*>
Specifies the direction of the transmission by the expression "S" (= send) or "R" (= receive).

<*sexp2*>
(Optional) Name of the file transmitted from the printer (default "ZMODEM.FILE").

<*sexp3*>
(Optional) Specifies the input device as "uart1:", "uart2:", "uart3:", "uart4:", or "uart5:". Default is the standard IN channel.

<*sexp4*>
(Optional) Specifies the output device as "uart1:", "uart2:", "uart3:", "uart4:", or "uart5:". Default is the standard OUT channel.

Notes

Directory names are case sensitive. You must first set SYSVAR (43) to 1 before running this command in order for the printer to recognize the directory you specify.

ZMODEM is a protocol for serial transfer of a complete file. For more information on the ZMODEM protocol, please refer to .www.omen.com. Related instructions are the external commands RZ (receive data using the ZMODEM protocol) and SZ (send data using the ZMODEM protocol).

TRANSFER ZMODEM handles a single file at a time.

When transmitting files from the printer to the host, carefully observe possible restrictions on the number of characters in the file name that may be imposed by the operating system of the host. When receiving a file, you must start the transmission within 30 seconds of completing the TRANSFER ZMODEM "R" statement. The printer stores the file in the current directory. If a file in the current directory has the same name as the one to be transferred, the existing file is replaced by the new file.

Thus, you need to keep track of the files already stored in the current directory (see FILES statement). Before transfer, give the new file a name not already occupied by an existing file, unless you want to replace the existing file. If you use TRANSFER ZMODEM to download a font or image file, the font or image is automatically installed after the download is completed. You do not need to restart your printer.

Example 1

This example sets up the printer to receive a file on the standard IN channel:

```
TRANSFER ZMODEM "R"
```

Example 2

This example sends the file "FILE1.TXT" from the printer to the host on the IN channel:

```
TRANSFER Z "S","FILE1.TXT"
```

TRANSFER$

Purpose

Executes a transfer from source to destination as specified by a TRANSFERSET statement.

Syntax

TRANSFER$(*<nexp>*)

Parameters

<nexp>
Character timeout in ticks (10 ms).

Notes

TRANSFER$ executes the transfer from source to destination as specified by the TRANSFERSET statement. It also checks the transfer and breaks it if no character has been transmitted before the specified timeout has expired, or if any break character (as specified by the break character string in the TRANSFERSET statement) is encountered.

If the transmission was interrupted because a character in the break set was encountered, that character is returned.

If the transmission was interrupted because of a timeout error, an empty string is returned.

If the transmission was interrupted because of the reception of a character on any other communication channel than the source (as specified by TRANSFERSET statement), an empty string is returned.

Example

In this example, the transfer is executed by the TRANSFER$ function in line 60, and possible interruptions are indicated by a break character or empty string (" ") in the string variable C$.

```
10 OPEN "LABEL1.PRG" FOR INPUT AS #1
20 OPEN "UART1:" FOR OUTPUT AS #2
30 A$=CHR$(13)
40 B$=CHR$(10)
50 TRANSFERSET #1, #2, A$+B$
60 C$=TRANSFER$(100)
.....
.....
.....
```

TRANSFERSET

Purpose

Enters setup for the TRANSFER$ function.

Syntax

TRANSFERSET[#]<*nexp1*>,[#]<*nexp2*>,<*sexp*>[,<*nexp3*>]

Parameters

#
(Optional) Number sign.

<*nexp1*>
Number of the source (the file or device opened for input using the OPEN command).

<*nexp2*>
Number of the destination file (the file or device opened using the OPEN command for output or append).

<*sexp*>
A set of break characters

<*nexp3*>
(Optional) Enables or disables break on any other channel than the source:
<nexp> = 0, Break disabled
<nexp> ≠ 0, Break enabled
Default is the standard I/O with no break characters. Break on any other channel is enabled.

Notes

This statement sets up the transfer of data from a file or device opened for input using the OPEN command to another file or device opened for output or append. The transfer is interrupted if any character in a string of break characters specified in this statement is encountered (optionally on another specified channel). The actual transfer is executed by means of a TRANSFER$ function that also returns the break character that caused any possible interruption.

Example

In this example, the data transfer from a file in the current directory to an external device connected to the communication port "uart1:" is interrupted as soon as a carriage return or a line feed character is encountered in the file.

```
10 OPEN "LABEL1.PRG" FOR INPUT AS #1
20 OPEN "uart1:" FOR OUTPUT AS #2
30 A$=CHR$(13)
40 B$=CHR$(10)
50 TRANSFERSET #1, #2, A$+B$
60 C$=TRANSFER$(100)
.....
```

.....
.....

TRON/TROFF

Purpose

Enables or disables tracing of the program execution.

Syntax

TRON|TROFF where TRON enables tracing and TROFF disables tracing (default).

Notes

This statement is useful for debugging purposes. When tracing is enabled, each line number of the program is displayed on the screen within parentheses as the execution goes on.

Tracing is disabled when a TROFF statement is executed.

Example

```
10 PRINT "HELLO"
20 INPUT"Enter Text"; A$
30 PRINT A$
TRON
RUN
```

This results in:

```
(10) HELLO
(20) Enter test? (Operator enters "WORLD")
(30) WORLD
```

VAL

Purpose

Returns the numeric representation of a string expression.

Syntax

VAL(*<sexp>*)

Parameters

<sexp>
String expression from which the numeric representation is returned.

Notes

VAL is the complementary function for STR$. VAL ignores space characters from the argument string to determine the result.

If the first character in the string expression is anything else but a digit, a plus sign, or a minus sign, the VAL function returns the value 0.

Example

In this example, the values of the string variables A$ and B$ are read and assigned to the numeric variables A% and B%:

```
10 A$="123, MAIN STREET"
20 A%=VAL(A$)
30 B$="PHONE 123456"
40 B%=VAL(B$)
50 PRINT A$
60 PRINT A%
70 PRINT B$
80 PRINT B%
RUN
```

This results in:

```
123, MAIN STREET
123
PHONE 123456
0
```

VERBON/VERBOFF

Purpose

Specifies the verbosity level of the communication from the printer on the standard OUT channel.

Syntax

VERBON|VERBOFF where VERBON enables all verbosity levels (default) and VERBOFF disables all verbosity levels.

Notes

By default, when a character is received on the standard IN channel (see SETSTDIO statement), the corresponding character is echoed back on the standard OUT channel. As the serial channel "uart1:" is by default selected as the standard IN and OUT channel, when you enter a character on the keyboard of the host the same character appears on the screen after being transmitted to the printer and back.

When an instruction is successfully executed, "Ok" appears on the screen. Otherwise an error message is returned. Since this requires two-way communication, verbosity has no meaning when using the parallel "centronics:" communication protocol.

VERBON corresponds to SYSVAR(18) = -1.

Other verbosity levels can be selected using SYSVAR(18), and the type of error message can be selected using SYSVAR (19).

Fingerprint is silent while VERBOFF is active, which means that no characters are echoed, and no acknowledgements or error messages are sent. VERBOFF statements do not affect question marks or prompts displayed as a result of an INPUT statement. Instructions like DEVICES, FILES, FONTS, IMAGES, LIST, and PRINT also work normally.

VERBOFF corresponds to SYSVAR(18) = 0.

Example

This example shows how VERBOFF suppresses the printing of INPUT data in lines 20 and 40 during the actual typing on the host, and VERBON allow printing of the resulting string variables on the screen:

```
10 FOR Q%=1 TO 6
20 VERBOFF:INPUT "", A$
30 VERBON:PRINT A$;
40 VERBOFF:INPUT "", B$
50 VERBON
60 C$=SPACE$(25-LEN(A$))
70 PRINT C$+B$
80 NEXT Q%
90 END
```

VERIFIER RESULT

Purpose

Retrieve or print out a summary of the label verification results.

Syntax

VERIFIER RESULT [PRINT]

Note

This command will only print out information that is currently stored in the verifier result database.

Example

A typical VERIFIER RESULT might be:

Verified: 104 labels

Failed: 3 labels

Failure Rate: 2.9%

Ok

VERSION$

Purpose

Returns the firmware version, printer family, or type of CPU board.

Syntax

VERSION$[(*<nexp>*)]

Parameters

<nexp>
(Optional) Type of information to be returned:

0: Version of firmware (default)
1: Printer family
2: Type of CPU board

Notes

The name of the firmware depends on whether or not the printer is running in Immediate or Programming Mode, or in Direct Protocol.

The printer family is returned as one of the following:

PM23c
PM43
PM43c
PC23d
PC43d
PC43t

The CPU board type is returned as a string of text such as "hardware version 4.0".

Example 1

PRINT VERSION$(0)

A typical response might be:

P10.03.006424

Example 2

PRINT VERSION$(1)

A typical response might be:

PM43

Example 3

PRINT VERSION$(2)

A typical response might be:

Platform version 1.0

WEEKDAY$

Purpose

Returns the name of the weekday from a specified date.

Syntax

WEEKDAY$(<*sexp*>)

Parameters

<*sexp*>
Date for which the name of the weekday is returned.

Notes

This function returns the name of the weekday from a list of names specified by a NAME WEEKDAY$ statement or, if the name is missing, the full English name in lowercase characters (for example, "friday").

The date should be entered according to the syntax for the DATE$ variable:

YY = Year: Last two digits (for example 2007 = 07)

MM = Month: Two digits (01-12)

DD = Day: Two digits (01-28|29|30|31)

For example, December 1, 2007 is entered as "071201". The built-in calendar corrects incorrect values for the years 1980-2048 (for example, the incorrect date 031232 is corrected to 040101).

Example

This example shows how to make the printer return the name of the weekday as a three-letter English abbreviation in connection with a formatted date:

```
10 FORMAT DATE$ ", MM/DD/YY"
20 DATE$="071201"
30 NAME WEEKDAY$ 1, "Mon"
40 NAME WEEKDAY$ 2, "Tue"
50 NAME WEEKDAY$ 3, "Wed"
60 NAME WEEKDAY$ 4, "Thu"
70 NAME WEEKDAY$ 5, "Fri"
80 NAME WEEKDAY$ 6, "Sat"
90 NAME WEEKDAY$ 7, "Sun"
100 PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN
```

This results in:

MON, 12/01/07

WEEKDAY$

Purpose

Returns the name of the weekday from a specified date.

Syntax

WEEKDAY$(<*sexp*>)

Parameters

<*sexp*>
Date for which the name of the weekday is returned.

Notes

This function returns the name of the weekday from a list of names specified by a NAME WEEKDAY$ statement or, if the name is missing, the full English name in lowercase characters (for example, "friday").

The date should be entered according to the syntax for the DATE$ variable:

YY = Year: Last two digits (for example 2007 = 07)

MM = Month: Two digits (01-12)

DD = Day: Two digits (01-28|29|30|31)

For example, December 1, 2007 is entered as "071201". The built-in calendar corrects incorrect values for the years 1980-2048 (for example, the incorrect date 031232 is corrected to 040101).

Example

This example shows how to make the printer return the name of the weekday as a three-letter English abbreviation in connection with a formatted date:

```
10 FORMAT DATE$ ", MM/DD/YY"
20 DATE$="071201"
30 NAME WEEKDAY$ 1, "Mon"
40 NAME WEEKDAY$ 2, "Tue"
50 NAME WEEKDAY$ 3, "Wed"
60 NAME WEEKDAY$ 4, "Thu"
70 NAME WEEKDAY$ 5, "Fri"
80 NAME WEEKDAY$ 6, "Sat"
90 NAME WEEKDAY$ 7, "Sun"
100 PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN
```

This results in:

MON, 12/01/07

WEEKNUMBER

Purpose

Returns the number of the week for a specified date.

Syntax

WEEKNUMBER(*<sexp>*[,*<nexp*])

Parameters

<sexp>
Date for which the week number will be returned. Range is 1 to 53.

<nexp>
Specifies the calculating function as described in the next table:

<nexp>	Week #1 starts on:
0 (default)	Per ISO 8601 (European standard): • Week #1 starts on the last Monday at or before the New Year, if January 1 occurs on a Monday, Tuesday, Wednesday, or Thursday • Week #1 starts on the first Monday after the New Year, if January 1 occurs on a Friday, Saturday, or Sunday
1	Sunday in the first week with 7 days in the actual year.
2	January 1st, with each following week starting on a Sunday.
3	Monday in the first week with 7 days in the actual year.
4	January 1st, with each following week starting on a Monday.
5	Tuesday in the first week with 7 days in the actual year.
6	January 1st, with each following week starting on a Tuesday.
7	Wednesday in the first week with 7 days in the actual year.
8	January 1st, with each following week starting on a Wednesday.
9	Thursday in the first week with 7 days in the actual year.
10	January 1st, with each following week starting on a Thursday.
11	Friday in the first week with 7 days in the actual year.
12	January 1st, with each following week starting on a Friday.
13	Saturday in the first week with 7 days in the actual year.
14	January 1st, with each following week starting on a Saturday.

Notes

The date should be entered according to the syntax for the DATE$ variable that is in the following order:

YY = Year: Last two digits (for example 2013 = 13)

MM = Month: Two digits (01-12)

DD = Day: Two digits (01-28|29|30|31)

For example, December 1, 2013 is entered as "131201". The built-in calendar corrects incorrect values for the years 1980-2048 (for example, the incorrect date 031232 is corrected to 040101).

Example

This example returns the week number of December 29, 2013 using calculating function 2:

```
PRINT WEEKNUMBER ("131229",2)
```

This results in:

```
53
```

WHILE...WEND

Purpose

Executes a series of statements in a loop, providing a given condition is true.

Syntax

WHILE *<nexp>*
<stmt>[...*<stmt>*]
WEND

Parameters

<nexp>
Numeric expression that is either TRUE (-1) or FALSE (0).

<stmt>
Statement, or a list of statements on separate lines, that are executed provided *<nexp>* is TRUE.

Notes

If *<nexp>* is TRUE, all following statements are executed successively until a WEND statement is encountered. The program execution then goes back to the WHILE statement and repeats the process, provided *<nexp>* still is TRUE.

If *<nexp>* is FALSE, the execution resumes at the statement following the WEND statement.

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

Example

In this example, the WHILE...WEND loop is executed only if the character "Y" (ASCII 89 decimal) is entered on the host keyboard.

```
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
```

This results in:

```
Want to exit? Press Y=Yes or N=No N
Want to exit? Press Y=Yes or N=No Y
The answer is Yes
You will exit the program
```

XORMODE ON/OFF

Purpose

Enables or disables the xor/flip mode of Fingerprint in connection with graphical operations.

Syntax

XORMODE ON|OFF

Notes

When XORMODE is set ON, dots are reversed (as opposed to set) by all graphical operations except bar codes. For example, if two black lines cross, the intersection is white. If XORMODE is OFF, the intersection is black.

Default is XORMODE OFF. XORMODE is automatically set to default when a PRINTFEED statement is executed or a Fingerprint program has been successfully run.

Example

The following program illustrates the difference between XORMODE ON and XORMODE OFF. The two lines to the left are drawn with XORMODE disabled and the lines to the right with XORMODE enabled.

```
10 XORMODE OFF
20 PRPOS 0,50
30 PRLINE 300,30
40 DIR 4
50 PRPOS 100,0
60 PRLINE 200,30
70 XORMODE ON
80 DIR 1
90 PRPOS 400,50
100 PRLINE 300,30
110 DIR 4
120 PRPOS 500,0
130 PRLINE 200,30
140 PRINTFEED
RUN
```

CHAPTER

3

SETUP COMMAND INFORMATION FOR NETWORK PARAMETERS

802.11 wireless settings are supported only by printers with an 802.11 wireless radio installed.
Bluetooth settings are supported only by printers with a Bluetooth radio installed. For more information, see your printer user manual.

Click below to see SETUP command information for these network parameters:

- Wireless
- Bluetooth
- Ethernet

You must be logged in as itadmin to change network parameters.

802.11 Wireless Parameters

Use the SETUP commands to set and retrieve hardware, network, and security settings for a wireless radio. You can also use SETUP GET to retrieve all parameters within the section at once. To run these commands, you must first authenticate as itadmin.

802.11 wireless settings are supported only by printers with an 802.11 wireless radio installed.

The 802.11 wireless parameters consist of:

- 802.11 wireless radio parameters
- 802.11 wireless network parameters
- 802.11 wireless security parameters

802.11 Wireless Network Parameters

This topic describes how to use SETUP commands to set and retrieve network settings for a wireless radio. You can also use SETUP GET to retrieve all parameters within the section at once. In order to run these commands, you must first authenticate as itadmin.

802.11 wireless settings are supported only by printers with an 802.11 wireless radio installed.

802.11 wireless network parameters include:

- (IPv4) IP Assignment Method - Specifies how the printer obtains its IPv4 IP address.
- (IPv4) IP Address - Specifies the unique IPv4 address assigned to the printer on a TCP/IP network.
- Subnet Mask - Specifies the range of IP addresses in the subnet, or local network.
- Default Router - Specifies the IP address of a router used to send information to devices outside the subnet.
- DHCP Response - Specifies whether the printer receives DHCP responses using broadcast or unicast.
- (IPv6) IP Assignment Method - Specifies how the printer obtains its IPv6 IP address.
- (IPv6) IP Address - Specifies the unique IPv6 address assigned to the printer on a TCP/IP network.

Set a Parameter

To set a parameter, send the SETUP command to the printer with this syntax:

SETUP "*path*,*value*"

Each parameter has a specific path you must use. The path, accepted values, and default value are explained with each parameter.

Some parameters have a Legacy Path for backwards compatibility with Fingerprint syntax from previous versions. If a parameter does not have a Legacy Path available, you must use the Path instead.

Retrieve a Parameter

To retrieve a parameter, send a SETUP GET command with this syntax:

SETUP GET "*path*",*variable*

For more information about variables, see Fingerprint Variable Names.

(IPv4) IP Assignment Method

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv4,IP Assignment Method
Default Value	DHCP
Values	DHCP: The printer receives all settings automatically from a DHCP server. BOOTP: The printer receives all settings automatically from a BOOTP server. DHCP+BOOTP: The printer receives all settings automatically from the first DHCP or BOOTP server it finds. Manual: You must specify the IP address, subnet mask, default router, and DNS server manually.

This example sets the IPv4 IP assignment method to manual:

SETUP "Communications,Wireless 802.11,IPv4,IP Assignment Method,Manual"

(IPv4) IP Address

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv4,IP Address
Default Value	0.0.0.0
Value	An IP address in the format *a.b.c.d*.

This example sets the IPv4 IP address to 10.10.1.105:

SETUP "Communications,Wireless 802.11,IPv4,IP Address,10.10.1.105"

Subnet Mask

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv4,Subnet Mask

Default Value	0.0.0.0
Value	A subnet mask in the format *a.b.c.d*.

This example sets the subnet mask to 255.255.255.0:

SETUP "Communications,Wireless 802.11,Subnet Mask,255.255.255.0"

Default Router

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv4,Default Router
Default Value	0.0.0.0
Value	An IP address in the format *a.b.c.d*.

This example sets the default router to 10.10.1.1:

SETUP "Communications,Wireless 802.11,IPv4,Default Router,10.10.1.1"

DHCP Response

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv4,DHCP Response
Default Value	Broadcast
Values	Broadcast Unicast

This example sets the DHCP Response method to Unicast:

SETUP "Communications,Wireless 802.11,IPv4,DHCP Response,Unicast"

(IPv6) IP Assignment Method

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv6,IP Assignment Method
Default Value	Automatic

Values	Automatic: The printer receives all settings from the network automatically, but you can change the DNSv6 server address. Automatic+DHCP: The printer receives all settings from the network or a DHCP server automatically, but you can change the DNSv6 server address. DHCP: The printer receives all settings automatically from a DHCP server. Manual: You must specify the IPv6 address. If the IPv6 network is not active, the IPv6 address is set to 0 (::/128).

This example sets the IPv6 IP Assignment Method to DHCP:

SETUP "Communications,Wireless 802.11,IPv6,IP Assignment Method,DHCP"

(IPv6) IP Address

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Network,IPv6,IP Address
Default Value	2001:db8:0:1::1/64
Value	An IP address in the format *a:b:c:d:e:f:g/h*.

This example sets the IPv6 IP Address to fe80::202:b3ff:fe1e:8329/128:

SETUP "Communications,Wireless 802.11,IPv6,IP Address,fe80::202:b3ff:fe1e:8329/128"

802.11 Wireless Hardware Parameters

This topic describes how to use SETUP commands to set and retrieve hardware settings for a wireless radio. You can also use SETUP GET to retrieve all parameters within the section at once. In order to run these commands, you must first authenticate as itadmin.

802.11 wireless settings are supported only by printers with an 802.11 wireless radio installed.

802.11 wireless radio parameters include:

- Network Name (SSID) - Specifies the name of the wireless network.
- Network Type - Specifies whether to use Ad-Hoc or Infrastructure as the 802.11 wireless radio behavior.
- Roaming - Specifies whether the printer should switch to access points with higher signal strength.
- Power Mode - Specifies the power management settings for the 802.11 wireless radio.

- Channel - Specifies the current wireless channel.
- Hidden SSID - Specifies whether the printer can search for hidden wireless networks.

Set a Parameter

To set a parameter, send the SETUP command to the printer with this syntax:

SETUP "*path*,*value*"

Each parameter has a specific path you must use. The path, accepted values, and default value are explained with each parameter.

Some parameters have a Legacy Path for backwards compatibility with Fingerprint syntax from previous versions. If a parameter does not have a Legacy Path available, you must use the Path instead.

Retrieve a Parameter

To retrieve a parameter, send a SETUP GET command with this syntax:

SETUP GET "*path*",*variable*

For more information about variables, see Fingerprint Variable Names.

Network Name (SSID)

The Network Name (SSID) parameter indicates the name of a wireless network. All access points and Honeywell devices must use the same network name. A network name is case-sensitive.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,General,Network Name (SSID)
Default Value	HONEYWELL
Value	Valid SSIDs are from 1 to 32 characters long, and can include alphanumeric or hexadecimal values. Each hexadecimal character in an SSID must include a percent symbol (%) and two hex digits. Hex digits can be uppercase or lowercase. For example, %0f or %0F.

This example sets the network name to "MyCompany":

SETUP "Communications,Wireless 802.11,General, Network Name (SSID),MyCompany"

Network Type

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,General,Network Type
Default Value	Infrastructure

Values	Ad Hoc Infrastructure

This example sets the wireless network type to ad-hoc:

SETUP "Communications,Wireless 802.11,General,Network Type,Ad Hoc"

Roaming

The Roaming parameter indicates if the printer should switch access points based on received signal strength.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,General,Roaming
Default Value	Off
Values	Off Level 1 Level 2 Level 3

This example sets the printer to switch access points when signal strength is low:

SETUP "Communications,Wireless 802.11,General,Roaming,Level 2"

Power Mode

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,General,Power Mode
Default Value	Constant Awake
Values	Constant Awake - Highest throughput, but uses the most power. Fast Power Saving - Switches between Constant Awake and Power Saving mode. Use Fast Power Saving when power consumption is a concern, but you need more throughput than the default settings. Power Saving - Lowest throughput, but uses the least amount of power.

This example sets the printer to use Power Saving mode:

SETUP "Communications,Wireless 802.11,General,Power Mode,Power Saving"

Channel

The Channel parameter is read-only when Network Type is set to Infrastructure. To change the Channel parameter, set the Network Type to Ad Hoc.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,General,Channel
Default Value	1
Value	A valid channel number for your region.

This example sets the wireless network type to ad-hoc, and then sets the channel to 7:

SETUP "Communications,Wireless 802.11,General,Network Type,Ad Hoc"
SETUP "Communications,Wireless 802.11,General,Channel,7"

Hidden SSID

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,General,Hidden SSID
Default Value	Disable
Value	Enable Disable

This example sets the printer to look for hidden wireless networks:

SETUP "Communications,Wireless 802.11,General,Hidden SSID,Enable"

802.11 Wireless Security Parameters

This topic describes how to use SETUP commands to set and retrieve security settings for a wireless radio. You can also use SETUP GET to retrieve all parameters within the section at once. In order to run these commands, you must first authenticate as itadmin.

802.11 wireless settings are supported only by printers with an 802.11 wireless radio installed.

802.11 wireless security parameters include:

- Security Type - Specifies the encryption protocol used by your wireless network.
- Association - Specifies the security method used to get access to a wireless network.
- Pre-Shared Key - Specifies the password required for WPA or WPA2 encryption.
- Network Key *N* - These parameters specify the four keys required for WEP encryption.
- Network Key Index - Specifies the active key required for WEP encryption.
- User Name - Specifies a user name for use with EAP networks.
- Password - Specifies the password for use with EAP networks.
- Anonymous Name - Specifies the outer EAP user name used for authentication.

- Inner Authentication - Specifies the authentication method used for some EAP networks.
- CA Certificate - Specifies the name of the CA certificate used to verify server certificates.
- Client Certificate - Specifies the name of the client certificate used for authentication.
- Client Key - Specifies the private key used for EAP-TLS encryption.
- Server Common Name 1 - Specifies the primary authentication server.
- Server Common Name 2 - Specifies the secondary authentication server.
- Validate Certificate - Specifies whether to check the identify of the authentication server.
- PAC - Specifies the name of the Protected Access Credential used for EAP-FAST authentication.
- Acquire PAC - Specifies whether to set the PAC automatically.
- Fast Roaming (CCKM) - Specifies whether the printer can move to a new access point without authenticating again.

Set a Parameter

To set a parameter, send the SETUP command to the printer with this syntax:

SETUP "*path*,*value*"

Each parameter has a specific path you must use. The path, accepted values, and default value are explained with each parameter.

Some parameters have a Legacy Path for backwards compatibility with Fingerprint syntax from previous versions. If a parameter does not have a Legacy Path available, you must use the Path instead.

Retrieve a Parameter

To retrieve a parameter, send a SETUP GET command with this syntax:

SETUP GET "*path*",*variable*

For more information about variables, see Fingerprint Variable Names.

Security Type

Honeywell recommends that you use encryption on your wireless network.

Most security types require you to set other parameters. For example, if you choose WEP (static), you must also set the Association, Network Key N, and Network Key Index parameters.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Security Type

Default Value	None
Values	None WEP (static) Pre-shared key PEAP TLS TTLS LEAP EAP-FAST

This example changes the security type to Pre-shared key:

SETUP "Communications,Wireless 802.11,Security,Security Type,Pre-shared key"

Association

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Association
Default Value	Open/WEP
Values	Open/WEP Shared/WEP WPA/TKIP WPA2/AES

This example changes the association to WPA2/AES:

SETUP "Communications,Wireless 802.11,Security,Association,WPA2/AES"

Pre-Shared Key

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Pre-Shared Key
Default Value	No default value

Value	A password from 8 to 64 characters. If the password is exactly 64 characters long, it is interpreted as a hexadecimal string which specifies the 256-bit key. To clear the current password, set this parameter to an empty string.

This example clears the current pre-shared key:

SETUP "Communications,Wireless 802.11,Security,Pre-Shared Key,"

Network Key *N*

This parameter refers to four WEP key parameters: Network Key 1, Network Key 2, Network Key 3, and Network Key 4.

Syntax	SETUP "*path*,*value*"
Paths	Communications,Wireless 802.11,Security,Network Key 1 Communications,Wireless 802.11,Security,Network Key 2 Communications,Wireless 802.11,Security,Network Key 3 Communications,Wireless 802.11,Security,Network Key 4
Default Value	No default value
Value	An alphanumeric password 5 characters long (WEP64) An alphanumeric password 13 characters long (WEP128) All WEP keys must be in hexadecimal format, and prefaced with 0x.

This example sets Network Key 3 to "h3ll0" using the WEP64 format:

SETUP "Communications,Wireless 802.11,Security,Network Key 3,0x68336c6c30"

Network Key Index

The values for this parameter correspond to the Network Key N WEP keys.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Network Key Index
Default Value	1
Values	1 2 3 4

This example sets Network Key 2 as the active WEP network key:

SETUP "Communications,Wireless 802.11,Security,Network Key Index,2"

User Name

Honeywell recommends that you use a user name longer than eight (8) characters.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,User Name

Default Value	Anonymous
Value	A user name from 1 to 96 characters. To clear the current user name, set this parameter to an empty string.

This example clears the current user name:

SETUP "Communications,Wireless 802.11,Security,User Name,"

Password

Honeywell recommends that you use a password longer than eight (8) characters.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Password
Default Value	Anonymous
Value	A password from 1 to 96 characters. To clear the current password, set this parameter to an empty string.

This example sets the password to "ex@mpl3pas$":

SETUP "Communications,Wireless 802.11,Security,User Name,ex@mpl3pas$"

Anonymous Name

Honeywell recommends that you use an anonymous name longer than eight (8) characters.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Anonymous Name
Default Value	(null)
Value	A user name from 1 to 96 characters.

This example sets the anonymous name to "accounting":

SETUP "Communications,Wireless 802.11,Security,Anonymous Name,accounting"

Inner Authentication

These authentication types are used only when the EAP type is EAP-FAST, PEAP, or TTLS.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Inner Authentication
Default Value	MSCHAPv2

Value	PAP* MSCHAPv2 EAP/MSCHAPv2 EAP/MD5 EAP/GTC EAP/TLS

* The PAP authentication type can be used only with TTLS.

This example sets the inner authentication to EAP/MD5:

SETUP "Communications,Wireless 802.11,Security,Inner Authentication,EAP/MD5"

CA Certificate

The CA certificate and the server certificate must be signed by the same CA.

This parameter is not used when the Security Type is set to LEAP.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,CA Certificate
Default Value	honeywell.pem
Value	The name of a CA certificate stored on the printer. To use the default certificate again, set this parameter to an empty string.

This example sets the CA certificate to mycompany.pem:

SETUP "Communications,Wireless 802.11,Security,CA Certificate,mycompany.pem"

Client Certificate

This parameter is used only when the Security Type is set to EAP-TLS.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Client Certificate
Default Value	No default value
Value	The name of a client certificate stored on the printer. The maximum length of the filename is 128 characters. To stop using a client certificate, set this parameter to an empty string.

This example sets the client certificate to myserver.pem:

SETUP "Communications,Wireless 802.11,Security,Client Certificate,myserver.pem"

Client Key

This parameter is used only when the Security Type is set to EAP-TLS.

You must use the certinstall.sh script to install your private key.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Client Key
Default Value	No default value
Value	The name of a private key file stored on the printer. The maximum length of the filename is 128 characters. To stop using a private key, set this parameter to an empty string.

This example sets the printer to stop using a private key file:

SETUP "Communications,Wireless 802.11,Security,Client Key,"

Server Common Name 1

If this value is not set, your printer automatically attempts to use Server Common Name 2. If both values are not set, your printer accepts any certificate name. Honeywell recommends that you use server common names longer than eight (8) characters.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Server Common Name 1
Default Value	No default value
Value	The certificate name for the primary authentication server. The maximum length of the server name is 96 characters. To remove the server common name, set this parameter to an empty string.

This example sets the printer to use web1.example.com as the primary authentication server certificate name:

SETUP "Communications,Wireless 802.11,Security,Server Common Name 1,web1.example.com"

Server Common Name 2

If this value is not set, your printer automatically attempts to use Server Common Name 1. If both values are not set, your printer accepts any certificate name. Honeywell recommends that you use server common names longer than eight (8) characters.

This parameter is not used when the Security Type is set to LEAP.

Syntax	SETUP "*path*,*value*"

Path	Communications,Wireless 802.11,Security,Server Common Name 2
Default Value	No default value
Value	The certificate name for the secondary authentication server. The maximum length of the server name is 96 characters. To remove the server common name, set this parameter to an empty string.

This example sets the printer to use web2.example.com as the secondary authentication server certificate name:

SETUP "Communications,Wireless 802.11,Security,Server Common Name 2,web2.example.com"

Validate Certificate

Honeywell recommends that you change this parameter to **Yes** if you use an authentication server on your wireless network. You must install and use a valid CA certificate to use this feature.

If your printer has a real-time clock, this clock is used to check certificate validity.

This setting is used when the Security Type is set to EAP-TTLS, PEAP, or TLS. This setting is not used when the Security Type is set to LEAP.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Validate Certificate
Default Value	No
Values	Yes No

This example sets your printer to check the validity of the authentication server CA certificate:

SETUP "Communications,Wireless 802.11,Security,Validate Certificate,Yes"

PAC

A PAC file is used only when Acquire PAC is set to On.

This parameter is used only for EAP-FAST authentication, when the first authentication attempt fails.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,PAC
Default Value	eap-fast.pac
Value	The name of the PAC (Protected Access Credential) file to use.

Acquire PAC

This parameter is used only for EAP-FAST authentication, when the first authentication attempt fails.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Acquire PAC
Default Value	On
Values	On Off

This example turns off automatic PAC provisioning:

SETUP "Communications,Wireless 802.11,Security,Acquire PAC,Off"

Fast Roaming (CCKM)

The Fast Roaming feature only operates on wireless networks that use the LEAP security type.

Syntax	SETUP "*path*,*value*"
Path	Communications,Wireless 802.11,Security,Fast Roaming (CCKM)
Default Value	Disable
Values	Enable Disable

This example turns on Fast Roaming:

SETUP "Communications,Wireless 802.11,Security,Fast Roaming (CCKM),Enable"

Bluetooth Parameters for SETUP

Use the SETUP command to set and retrieve network settings for a Bluetooth radio. You can also use SETUP GET to retrieve all parameters within the section at once. In order to run these commands, you must first authenticate as itadmin.

Bluetooth parameters you can set include:

- Security - Specifies the level of security to use when pairing a Bluetooth device with the printer.
- Power Saving Mode - Specifies the power management settings to use for the Bluetooth radio.
- Device Name - Specifies the name for your printer on the Bluetooth network.
- Discover - Specifies whether other Bluetooth devices can detect the printer.
- Passkey - Specifies the password used when pairing with a Bluetooth device.
- Reserve - Specifies whether the printer can connect to more than one Bluetooth device.

Set a Parameter

To set a parameter, send the SETUP command to the printer with this syntax:

SETUP "*path*,*value*"

Each parameter has a specific path you must use. The path, accepted values, and default value are explained with each parameter.

Some parameters have a Legacy Path for backwards compatibility with Fingerprint syntax from previous versions. If a parameter does not have a Legacy Path available, you must use the Path instead.

Retrieve a Parameter

To retrieve a parameter, send a SETUP GET command with this syntax:

SETUP GET "*path*",*variable*

For more information about variables, see Fingerprint Variable Names.

Security

Path	Communications,Bluetooth,Security
Default Value	Disable
Values	Disable - Any device can pair with the printer. Authentication - Require a password to pair with the printer. Auth+Encrypt - Require a password to pair with the printer, and encrypt network traffic.

This example sets the Bluetooth security method to Authentication:

SETUP "Communications,Bluetooth,Security,Authentication"

Power Saving Mode

Enabling power saving reduces data throughput.

This parameter is not supported for PM43 or PM43c printers.

Path	Communications,Bluetooth,Power Saving Mode
Default Value	Disable
Values	Disable Enable

This example enables power saving for the Bluetooth radio:

SETUP "Communications,Bluetooth,Power Saving Mode,Enable"

Device Name

Path	Communications,Bluetooth,Device Name
Default Value	(device model number + serial number)
Value	The name of your printer on the Bluetooth network. The maximum name length is 16 characters.

This example sets the device name to "PM43_5A":

SETUP "Communications,Bluetooth,Device Name,PM43_5A"

Discover

Path	Communications,Bluetooth,Discover
Default Value	Enable
Values	Disable Enable

This example prevents other Bluetooth devices from finding the printer:

SETUP "Communications,Bluetooth,Discover,Disable"

Passkey

Changing this parameter removes all paired devices from the Bluetooth network.

If a password has been set, retrieving this parameter with SETUP GET results in "****".

Path	Communications,Bluetooth,Passkey
Default Value	1234
Value	A case-sensitive password used to pair the printer with a Bluetooth device. The maximum password length is 16 characters.

This example sets the password to 46837632:

SETUP "Communications,Bluetooth,Passkey,46837632"

Reserve

After you enable this parameter and one device pairs successfully, additional pairing attempts from other Bluetooth devices are ignored.

Path	Communications,Bluetooth,Reserve
Default Value	Disable
Values	Enable Disable

This example prevents pairing attempts from other Bluetooth devices after the first successful pairing:

SETUP "Communications,Bluetooth,Reserve,Enable"

Ethernet Network Parameters

Use the SETUP command to set and retrieve network settings for a wired network card. You can also use SETUP GET to retrieve all parameters within the section at once. In order to run these commands, you must first authenticate as itadmin.

Ethernet network parameters you can set include:

- (IPv4) IP Assignment Method - Specifies how the printer obtains its IPv4 IP address.
- (IPv4) IP Address - Specifies the unique IPv4 address assigned to the printer on a TCP/IP network.
- Subnet Mask - Specifies the range of IP addresses in the subnet, or local network.
- Default Router - Specifies the IP address of a router used to send information to devices outside the subnet.
- DHCP Response - Specifies whether the printer receives DHCP responses using broadcast or unicast.
- (IPv6) IP Assignment Method - Specifies how the printer obtains its IPv6 IP address.
- (IPv6) IP Address - Specifies the unique IPv6 address assigned to the printer on a TCP/IP network.

Set a Parameter

To set a parameter, send the SETUP command to the printer with this syntax:

SETUP "*path*,*value*"

Each parameter has a specific path you must use. The path, accepted values, and default value are explained with each parameter.

Some parameters have a Legacy Path for backwards compatibility with Fingerprint syntax from previous versions. If a parameter does not have a Legacy Path available, you must use the Path instead.

Retrieve a Parameter

To retrieve a parameter, send a SETUP GET command with this syntax:

SETUP GET "*path*",*variable*

For more information about variables, see Fingerprint Variable Names.

(IPv4) IP Assignment Method

Path	Communications,Ethernet,IPv4,IP Assignment Method
Legacy Path	NETWORK,IPnSELECTION
Default Value	DHCP

Values	DHCP: The printer receives all settings automatically from a DHCP server. BOOTP: The printer receives all settings automatically from a BOOTP server. DHCP+BOOTP: The printer receives all settings automatically from the first DHCP or BOOTP server it finds. Manual: You must specify the IP address, subnet mask, default router, and DNS server manually.

This example sets the IPv4 IP assignment method to manual:

SETUP "Communications,Ethernet,IPv4,IP Assignment Method,Manual"

(IPv4) IP Address

Path	Communications,Ethernet,IPv4,IP Address
Legacy Path	NETWORK,IPnADDRESS
Default Value	0.0.0.0
Value	An IP address in the format *a.b.c.d*.

This example sets the IPv4 IP address to 10.10.1.105:

SETUP "Communications,Ethernet,IPv4,IP Address,10.10.1.105"

Subnet Mask

Path	Communications,Ethernet,IPv4,Subnet Mask
Legacy Path	NETWORK,NETMASK
Default Value	0.0.0.0
Value	A subnet mask in the format *a.b.c.d*.

This example sets the subnet mask to 255.255.255.0:

SETUP "Communications,Ethernet,IPv4,Subnet Mask,255.255.255.0"

Default Router

Path	Communications,Ethernet,IPv4,Default Router
Legacy Path	NETWORK,DEFAULT ROUTER
Default Value	0.0.0.0
Value	An IP address in the format *a.b.c.d*.

This example sets the default router to 10.10.1.1:

SETUP "Communications,Ethernet,IPv4,Default Router,10.10.1.1"

DHCP Response

Path	Communications,Ethernet,IPv4,DHCP Response
Legacy Path	NETWORK,DHCP RESPONSE
Default Value	Broadcast
Values	Broadcast Unicast

This example sets the DHCP Response method to Unicast:

SETUP "Communications,Ethernet,IPv4,DHCP Response,Unicast"

(IPv6) IP Assignment Method

When this parameter is set to Automatic, PM23c, PM43 and PM43c printers use a Link Local Address until they receive a router prefix from an IPv6 router.

Path	Communications,Ethernet,IPv6,IP Assignment Method
Legacy Path	NETWORK,IPV6 SELECTION
Default Value	Automatic
Values	Automatic: The printer receives all settings from the network automatically, but you can change the DNSv6 server address. Automatic+DHCP: The printer receives all settings from the network or a DHCP server automatically, but you can change the DNSv6 server address. DHCP: The printer receives all settings automatically from a DHCP server. Manual: You must specify the IPv6 address. If the IPv6 network is not active, the IPv6 address is set to 0 (::/128).

This example sets the IPv6 IP Assignment Method to DHCP:

SETUP "Communications,Ethernet,IPv6,IP Assignment Method,DHCP"

(IPv6) IP Address

Path	Communications,Ethernet,IPv6,IP Address
Legacy Path	NETWORK,IPV6 ADDRESS
Default Value	2001:db8:0:1::1/64 Link Local Address (PM43 and PM43c)

Value	An IP address in the format *a1*:*a2*:*a3*:*a4*:*a5*:*a6*:*a7*:*a8/p* (where p is the router prefix). An IP address in the format *a1*:*a2*:*a3*:*a4*::*a5/*64.

This example sets the IPv6 IP Address to fe80::202:b3ff:fe1e:8329/128:

SETUP "Communications,Ethernet,IPv6,IP Address,fe80::202:b3ff:fe1e:8329/128"

CHAPTER

4

SUPPORTED BAR CODE INFORMATION

Supported Symbologies
List of symbologies supported by Fingerprint, with links to specific bar code parameter and settings information.

About AddOn Codes
Describes how to configure add-on bar codes for the EAN and UPC symbologies by using the PRBAR command.

About Composite Bar Codes
Describes how to print composite bar codes. Includes links to specific composite bar code parameter and settings information.

Supported Symbologies

Fingerprint supports the following bar code symbologies. For parameter ranges and settings, click the name of the symbology below.

Symbology	BARTYPE Name
Aztec	AZTEC
Code 39	CODE39
Code 39 full ASCII	CODE39A
Code 39 with checksum	CODE39C
Code 128	CODE128
Code 128 subset A	CODE128A
Code 128 subset B	CODE128B
Code 128 subset C	CODE128C
Data Matrix	DATAMATRIX
Dotcode	DOTCODE
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 128 Composite with CC-C	EAN128_CCC
Gridmatrix	GRIDMATRIX
Hanxin	HANXINCODE
HIBC39	HIBC39
HIBC128	HIBC128
Interleaved 2 of 5	INT2OF5
Interleaved 2 of 5 Symbology Information	INT2OF5C
ISBT 128	ISBT128

Symbology	BARTYPE Name
MaxiCode	MAXICODE
MicroPDF417	MICROPDF417
PDF 417	PDF417
QR Code	QRCODE
RSS-14	RSS14
RSS-14 Composite	RSS14
RSS-14 Truncated	RSS14T
RSS-14 Truncated Composite	RSS14T
RSS-14 Stacked	RSS14S
RSS-14 Stacked Composite	RSS14S
RSS-14 Stacked Omnidirectional	RSS14SO
RSS-14 Stacked Omnidirectional Composite	RSS14SO
RSS-14 Limited	RSS14L
RSS-14 Limited Composite	RSS14L
RSS-14 Expanded	RSS14E
RSS-14 Expanded Composite	RSS14E
RSS-14 Expanded Stacked	RSS14ES
RSS-14 Expanded Stacked Composite	RSS14ES
UPC-A	UPCA
UPC-A Composite with CC-A or CC-B	UPCA_CC
UPC-E	UPCE
UPC-E Composite with CC-A or CC-B	UPCE_CC
USPS intelligent mail	USPS4CB

Additional Symbology Support

Fingerprint also supports these symbologies:

Symbology	BARTYPE Name
Codabar	CODABAR
Code 11	CODE11
Code 16K	CODE16K
Code 49	CODE49
Code 93	CODE93
DUN-14/16	DUN
Five-Character Supplemental Code	ADDON5
Industrial 2 of 5	C2OF5IND
Industrial 2 of 5 with checksum	C2OF5INDC
Interleaved 2 of 5 A	INT2OF5A
Matrix 2 of 5	C2OF5MAT
MSI (modified Plessey)	MSI
Planet	PLANET
Plessey	PLESSEY
Postnet	POSTNET
Straight 2 of 5	C2OF5
Two-Character Supplemental Code	ADDON2
UCC-128 Serial Shipping Container Code	UCC128
UPC-5 digits Add-On Code	SCCADDON
UPC-D1	UPCD1
UPC-D2	UPCD2
UPC-D3	UPCD3
UPC-D4	UPCD4
UPC-D5	UPCD5
UPC Shipping Container Code	UPCSCC

Aztec Symbology Information

The tables below include parameter information for the Aztec symbology.

Command	Ranges and Settings
BARTYPE	"AZTEC"
BARRATIO	Not applicable.
BARMAG	< 128. Default is 2.
BARHEIGHT	Not applicable.
BARFONT	Not applicable.

BARSET Parameters

The next table lists parameter ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<*sexp*>	Bar code type.	"AZTEC"
<*nexp1-7*>	Not applicable.	
<*nexp8*>	Error correction or fixed symbol.	0 (default): 23%+3 codewords. Dynamic symbol size with fixed error correction. 1-99: Static error correction level (in percent). Dynamic symbol size. 101-104: Compact Format symbol, 1 to 4 layers (+100). Error correction level depends on spare bits in chosen symbol size. Static symbol size. 201-232: Full Range symbol, 1-32 layers (+200). Error correction level depends on spare bits in chosen symbol size. Static symbol size. 300: Simple Aztec rune.
<*nexp9*>	Menu symbol.	0 (default): No menu symbol. 1: Menu symbol.
<*nexp10*>	Symbol append.	1 (default): Symbol append OFF. 2-26: Append # of symbols. ID field used if present.
<*nexp11*>	Enables or disables ECI support.	0 (default): ECI support disabled. 1: ECI support enabled.

PRBAR Parameters

The next table lists parameter ranges and settings for use with the PRBAR command (with Symbol Append ON).

Parameter	Ranges and Set-tings	Notes
Number of characters	0 to 24	
Input char-acters	0x00-0xFF 0x20 not allowed.	Most efficient if all uppercase characters (ASCII 0x41-0x5a). Do not use space character (ASCII(0x20)).

Remarks

When Aztec is selected and ECI support is enabled (<nexp11> = 1), the strings "\<*nnnnnn*>" and "\\" are interpreted as follows:

- \<*nnnnnn*>: The backslash indicates that the character set should be changed, and *nnnnnn* is the number of the chosen character set. The string is decoded as an ECI character followed by one to three digit numbers in the bar code. All ECI characters are printed in ASCII encodation, meaning that this string itself can be encoded in any mode. The algorithm switches to ASCII mode to encode the ECI characters, then encodes the other characters in the most efficient modes.
- \\: Two backslashes are decoded as a single backslash character ("\") in the bar code. All double backslashes are printed in the most efficient modes as specified and are decoded the same way when ECI support is not enabled.

Error 1112 (ECI syntax error) occurs if ECI mode is enabled and the PRBAR input string is invalid.

Code 39 Symbology Information

The table below includes parameter information for the Code 39 symbology.

Command	Ranges and Settings
BARTYPE	"CODE39", "CODE39A", or "CODE39C"
BARRATIO	2:1-3:1. Default is 3:1.
BARMAG	If the narrow element is less than 4 dots wide, then the ratio must be larger than 2.25:1 (9:4). Otherwise, no restrictions. Default is 2.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	No restrictions.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	CODE39: Unlimited CODE39A: 195 CODE39C: 199	
Check digit	CODE39: None CODE39A: None CODE39C: 1 digit	
Digits	0 to 9.	
Uppercase letters	A to Z.	No national characters.
Lowercase letters	Not supported.	
Punctuation marks	- . space $ / + %	
Start character	*	Added automatically. Never shown in the human-readable interpretation and not transmitted as part of decoded data.
Stop character	*	Added automatically. Never shown in the human-readable interpretation and not transmitted as part of decoded data.

Remarks

Code39A allows for the entire ASCII set to be encoded (128 characters). Code39C allows characters that are not in the input set to be sent into the symbol without error. These characters are not printed.

Code 128 Symbology Information

The table below includes parameter information for the Code 128 symbology.

Command	Ranges and Settings
BARTYPE	"CODE128", "CODE128A", "CODE128B", "CODE128C"
BARRATIO	Fixed. BARRATIO statement ignored.

Command	Ranges and Settings
BARMAG	= 2 (default).
BARHEIGHT	No restrictions. Default is 100.
BARFONT	No restrictions.

BARSET Parameters

The next table includes parameter information for use with the BARSET command.

BARSET	Description	Ranges and Settings
<*sexp*>	Bar code type.	"CODE128", "CODE128A", "CODE128B", "CODE128C"
<*nexp1-2*>	Not applicable.	
<*nexp3*>	Barmag/Enlargement.	<128. Default is 2.
<*nexp4*>	Barheight.	No restriction. Default is 100.
<*nexp5*>	Not applicable.	
<*nexp6*>	Character exclusion in bar code. Does not affect human readable.	0: Disables parentheses and spaces in the bar code data. Non-zero: Bar code and human readable field will include exactly the same data.
<*nexp7-10*>	Not applicable.	

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	Unlimited.	
Check digit	One.	Added automatically. Never shown in the human-readable interpretation and not transmitted as part of encoded data.
Input characters	ASCII 0 to 127.	According to Roman 8 character set.

Parameter	Ranges and Settings	Notes
Functional characters	FNC1 (ASCII 128 decimal.) FNC2 (ASCII 129 decimal.) FNC3 (ASCII 130 decimal) FNC4 (ASCII 131 decimal)	See Note 1.
Start characters	See Note 2.	
Code characters	See Notes 1 and 2.	
Shift characters	See Notes 1 & 2.	
Stop characters	Always.	Added automatically.

Note 1

Function characters FNC1-4, code characters, and shift characters require either an 8-bit communication protocol (remapping to an ASCII value between 0-127 decimal) or the use of a CHR$ function.

FNC2-4 are not allowed in Subset C.

Note 2

Code 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data).

Remarks

Code 128A, Code 128B, and Code 128C select subsets A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character. Shift character: «« (« = ASCII 171 decimal).
- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset: Code character: « + A|B|C (« = ASCII 171 decimal).

DataMatrix Symbology Information

The next table includes parameter ranges and settings for the DataMatrix symbology.

Command	Ranges and Settings
BARTYPE	"DATAMATRIX"
BARRATIO	Fixed. Values will be interpreted as BARMAG.
BARMAG	<128. Default is 2.
BARHEIGHT	Not applicable.
BARFONT	Set to ON to print the human readable. If ECI support is disabled, the human readable is printed as is. If ECI support is enabled, the human readable is printed without the ECI-related characters. For example, a bar code string of "\000009abc\\123" is printed as "abc\123".

BARSET Data

The next table includes parameters for use with the BARSET command.

Parameter	Instruction	Ranges and Settings
<sexp>	Bar code type.	"DATAMATRIX"
<nexp1-2>	Not applicable.	
<nexp3>	Barmag/Enlargement.	< 128. Default is 2.
<nexp4>	Not applicable.	
<nexp5>	Bar code shape.	0: Square (default) 1: Rectangular
<nexp6>	Bar code size.	See the next section.
<nexp7-10>	Not applicable.	
<nexp11>	Enables or disables ECI support.	0: Disables ECI support. 1: Enables ECI support.

Bar code size (<*nexp6*>)

By default, the number of characters determines the size of the symbol. For example, the data "123456" generates a 10 row × 10 column DataMatrix symbol, and 72 digits generates a 24 row × 24 column DataMatrix symbol. Fewer characters can be used for all symbols (10 × 10, 12 × 12, and so on up to 144 × 144) if the data includes non-numeric characters.

If the number of characters in the bar code exceeds the size you specify, the error "Too many characters in barcode" appears. Specify a larger size for the bar code.

If you specify an invalid value for <*nexp6*>, such as 10 for a rectangular bar code, Fingerprint uses the default value.

Sizes for square bar codes:

Value	Dimensions
0	Default
1	10 × 10
2	12 × 12
3	14 × 14
4	16 × 16
5	18 × 18
6	20 × 20
7	22 × 22
8	24 × 24
9	26 × 26
10	32 × 32
11	36 × 36
12	40 × 40
13	44 × 44
14	48 × 48
15	52 × 52
16	64 × 64
17	72 × 72
18	80 × 80
19	88 × 88
20	96 × 96

Value	Dimensions
21	104 × 104
22	120 × 120
23	132 × 132
24	144 × 144

Sizes for rectangular bar codes:

Value	Dimensions
0	Default
1	8 x 18
2	8 x 32
3	12 x 26
4	12 x 36
5	16 x 36
6	16 x 48

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	2,335 ASCII characters.	
Check digit	Yes.	Added automatically.
Input characters	ASCII 0-255 decimal.	

- ECC 200 type is used for this DataMatrix symbol. ECC 200 data may be encoded using any of the encodation schemes ASCII, C40, Text, X12, EDIFACT or Base 256. ASCII encodation is the default scheme. All other encodation schemes are invoked from ASCII encodation and return to this scheme.
- If you are encoding a function code using ASCII characters, make sure to include the function code before the encoded numerical data. For example, to encode "Function Code 1 followed by "240123456789" in a barcode, enter "CHR$(26) + "1"+"240123456789".

Remarks

When DataMatrix is selected and ECI support is enabled (<*nexp11*> = 1), the strings "\<nnnnnn>" and "\\" are interpreted as follows:

- \<nnnnnn>: The backslash indicates that the character set should be changed, and nnnnnn is the number of the chosen character set. The string is decoded as an ECI

character followed by one to three digit numbers in the bar code. All ECI characters are printed in ASCII encodation, meaning that this string itself can be encoded in any mode. The algorithm switches to ASCII mode to encode the ECI characters, then encodes the other characters in the most efficient modes.

- \\: Two backslashes are decoded as a single backslash character ("\") in the bar code. All double backslashes are printed in the most efficient modes as specified and are decoded the same way when ECI support is not enabled.

Error 1112 (ECI syntax error) occurs if ECI mode is enabled and the PRBAR input string is invalid.

Dotcode Symbology Information

The table below includes parameter information for the Dotcode symbology.

Command	Ranges and Settings
BARTYPE	"DOTCODE"
BARRATIO	Not applicable.
BARMAG	1-127. Default is 2.
BARHEIGHT	Not applicable.
BARFONT	Default is OFF.

BARSET Parameters

The next table lists parameter ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<*sexp*>	Bar code type.	"DOTCODE"
<*nexp1-2*>	Not applicable.	
<*nexp3*>	Dot magnification factor	1 to 127. Default is 2.
<*nexp4*>	Not applicable.	
<*nexp5*>	Aspect height ratio.	1 to 10. Default is 2. Bar code generator internally remaps 1 to 2.
<*nexp6*>	Aspect width ratio.	1 to 10. Default is 3.
<*nexp7*>	Barcode height in elements.	5 to 127. Ignored by default.
<*nexp8*>	Barcode width in elements.	5 to 127. Ignored by default.

Parameter	Description	Ranges and Settings
<*nexp9*>	Reflectance reversal.	0 (default): Dark dot on light background 1: Light dot on dark background
<*nexp10-11*>	Not applicable.	

EAN-8 Symbology Information

The table below includes parameter ranges and settings for the EAN-8 symbology.

Command	Ranges and Settings
BARTYPE	"EAN8"
BARRATIO	Fixed ratio. BARRATIO statement ignored.
BARMAG	Maximum 8. Default is 2.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	Font selection not possible. Font for human-readable interpretation is set automatically according to the specification on a BARFONT ON command.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	7	
Check digit	One.	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start character	None.	
Stop character	None.	

To learn more about using the PRBAR statement to generate an add-on bar code, see About AddOn Codes.

EAN-13 Symbology Information

The table below includes parameter ranges and settings for the EAN-13 symbology.

Command	Ranges and Settings
BARTYPE	"EAN13"
BARRATIO	Fixed ratio. BARRATIO statement ignored.
BARMAG	Maximum 8. Default is 2.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	Font selection not possible. Font for human-readable interpretation is set automatically according to the specification on a BARFONT ON command.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	12	
Check digit	One	Added automatically.
Digits	0 to 9	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start character	None.	
Stop character	None.	

To learn more about using the PRBAR statement to generate an add-on bar code, see About AddOn Codes.

EAN-128 Symbology Information

The table below includes parameter ranges and settings for use with the EAN-128 symbology.

Command	Ranges and Settings
BARTYPE	"EAN128", "EAN128A", "EAN128B", "EAN128C"
BARRATIO	Fixed ratio. BARRATIO statement ignored.
BARMAG	= 2 (default).
BARHEIGHT	No restrictions. Default is 100.
BARFONT	No restrictions.

BARSET Data

The next table includes parameter ranges and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"EAN128", "EAN128A", "EAN128B", "EAN128C"
<nexp1-2>	Not applicable.	
<nexp3>	Barmag/Enlargement.	<128. Default is 2.
<nexp4>	Barheight.	
<nexp5>	Not applicable.	
<nexp6>	Character exclusion in bar code. Does not affect human readable.	0: Disables parentheses and spaces in the bar code data. Non-zero: Bar code and human readable field will include exactly the same data.
<nexp7-10>	Not applicable.	

Input Data

The next table includes input data parameters for EAN-128.

Parameter	Ranges and Settings	Notes
Number of characters	Unlimited.	
Check digit	Trailing symbol check character.	Added automatically. Never shown in the human-readable interpretation and not transmitted as part of encoded data.
Input characters	ASCII 0-127.	According to Roman 8 character set.
Start characters	See Note 2.	
Code characters	See Notes 1 & 2.	
Shift characters	See Notes 1 & 2.	

Parameter	Ranges and Settings	Notes
Stop characters:	Always.	Added automatically.

Note 1

Code characters and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 decimal, or the use of a CHR$ function.

Note 2

EAN 128 automatically selects the optimal start character, and handles shift and changes of the character subset depending on input data content.

Remarks

To learn more about using the PRBAR statement to generate an add-on bar code, see About AddOn Codes.

EAN 128A, EAN 128B, and EAN 128C select subsets A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character. Shift character: «« (« = ASCII 171 decimal)
- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset. Code character: « + A|B|C (« = ASCII 171 decimal).

Gridmatrix Symbology Information

The table below includes parameter information for the Gridmatrix symbology.

Command	Ranges and Settings
BARTYPE	"GRIDMATRIX"
BARRATIO	Not applicable.
BARMAG	1-127. Default is 2.
BARHEIGHT	Not applicable.
BARFONT	Default is OFF.

BARSET Parameters

The next table lists parameter ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<*sexp*>	Bar code type.	"GRIDMATRIX"
<*nexp1-2*>	Not applicable.	
<*nexp3*>	Dot magnification factor	1 to 127. Default is 2.
<*nexp4*>	Not applicable.	
<*nexp5*>	User-selectable ECI	1 to 5. Default is 1.
<*nexp6*>	Enable or disable ECI support	0: disabled (default) 1: enabled
<*nexp7-11*>	Not applicable.	

HANXIN Symbology Information

The table below includes parameter information for the HANXIN symbology.

Command	Ranges and Settings
BARTYPE	"HANXINCODE"
BARRATIO	Not applicable.
BARMAG	1-30. Default is 2.
BARHEIGHT	Not applicable.
BARFONT	Default is OFF.

BARSET Parameters

The next table lists parameter ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<*sexp*>	Bar code type.	HANXINCODE
<*nexp1*>	Not used	Not used
<*nexp2*>	Not used	Not used
<*nexp3*>	Version number	1-30, Default = 2
<*nexp4*>	Not used	Not used
<*nexp5*>	Error correction	1-4, Default=2
<*nexp6*>	Dot element magnification factor	1-6, Default = 2
<*nexp7*>	Mask pattern selection	0-3, Default =1

Remarks

A secondary format data structure requires a link character which is the checksum of the primary format data structure. Therefore, a secondary data structure is only applicable after a primary format data structure. Otherwise an uninitialized link character is used.

HIBC Symbology Information

The table below includes parameter information for the HIBC symbology.

Command	Ranges and Settings
BARTYPE	"HIBC39" or "HIBC128"
BARRATIO	2:1, 3:1, 7:3, or 5:2. Default is 3:1.
BARMAG	1-127. Default is 2.
BARHEIGHT	1-500. Default is 100.
BARFONT	Not applicable.

BARSET Parameters

The next table lists parameter ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<*sexp*>	Bar code type.	"HIBC39" or "HIBC128"
<*nexp1*>	Ratio of large bar (Not applicable for HIBC128)	See "Ranges and Settings" for BARRATIO, above.
<*nexp2*>	Ratio of small bar (Not applicable for HIBC128)	See "Ranges and Settings" for BARRATIO, above.
<*nexp3*>	Bar magnification	1 to 127. Default is 2.
<*nexp4*>	Bar height	1 to 500. Default is 100.
<*nexp5*>	Data structure	0: Primary Format (default) 1: Supplier Alternative Format 2: Secondary Format 3: Provider Primary Format 4: Provider Secondary Format

Parameter	Description	Ranges and Settings
<*nexp6*>	Start and stop characters	0: No extra characters printed. 1: Print the start and stop characters. (default)
<*nexp7-11*>	Not applicable.	

Remarks

A secondary format data structure requires a link character which is the checksum of the primary format data structure. Therefore, a secondary data structure is only applicable after a primary format data structure. Otherwise an uninitialized link character is used.

Interleaved 2 of 5 Symbology Information

The next table includes parameter settings for the Interleaved 2 of 5 symbology.

Command	Ranges and Settings
BARTYPE	"INT2OF5"
BARRATIO	2:1-3:1/ Default: 3:1
BARMAG	No restrictions. Default is 2.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	No restrictions.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	Maximum of 195.	
Check digit	None.	
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None	
Start character	Yes.	Added automatically.
Stop character	Yes.	Added automatically.

Interleaved 2 of 5c Symbology Information

The next table includes parameter settings for the Interleaved 2 of 5 symbology.

Command	Ranges and Settings
BARTYPE	"INT2OF5c"
BARRATIO	2:1-3:1/ Default: 3:1
BARMAG	No restrictions. Default is 2.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	No restrictions.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	Maximum of 195.	
Check digit	1-digit.	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start character	Yes.	Added automatically.
Stop character	Yes.	Added automatically.

ISBT 128 Symbology Information

The table below includes parameter information for the ISBT 128 symbology.

Command	Ranges and Settings
BARTYPE	"ISBT128"
BARRATIO	Fixed. BARRATIO statement ignored.
BARMAG	= 2 (default).
BARHEIGHT	No restrictions. Default is 100.
BARFONT	No restrictions.

BARSET Parameters

The next table includes parameter information for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"ISBT128"

BARSET	Description	Ranges and Settings
<nexp1-2>	Not applicable.	
<nexp3>	Barmag/Enlargement.	<128. Default is 2.
<nexp4>	Barheight.	No restriction. Default is 100.
<nexp5>	Flag character.	0: Do not print. 1: (Default) Print flag character. Rotate 90 degrees clockwise, data from host. Calculate and print flag character if between 01 and 59. For type 3, do not print if between 60 and 96.
<nexp7>	Human-readable text scaling.	0: No scaling. 2: (Default) Increase font size by 2 between pipe (|) characters (serial number). N: Increase font size by N points between pipe (|) characters (serial number).
<nexp8>	Keyboard entry check character. Not part of bar code data.	0: Disabled, not printed. 1: (Default) Calculated and printed. Enabled for all data structures.
<nexp9>	Processing of spaces and vertical bar in the bar code data.	0: Bar code will not contain space or pipe (|) characters. Space characters are printed in the human-readable text. Pipe characters are removed. Non-zero: Print all data in the input string except pipe characters.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	Unlimited.	
Check digit	One.	Added automatically.
Input characters	ASCII 0 to 127.	According to Roman 8 character set.

Parameter	Ranges and Settings	Notes
Functional characters	FNC1 (ASCII 128 decimal) FNC2 (ASCII 129 decimal) FNC3 (ASCII 130 decimal) FNC4 (ASCII 131 decimal)	See Note 3.
Start characters	See Note 4.	
Code characters	See Notes 3 and 4.	
Shift characters	See Notes 3 and 4.	
Stop characters	Always.	Added automatically.

Note 1

Flag character printing only applies to Donation Identification Number data structure.

Note 2

If the flag character value is set between 60 and 96, the printer considers it as type 3 and calculates the new value by adding 60 to the check character value automatically.

Note 3

Function characters FNC1-4, code characters, and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 decimal, or the use of an CHR$ function.

FNC2-4 are not allowed in Subset C.

Note 4

Code 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data).

Remarks

Code 128B is used as the basis for ISBT 128.

Code 128A, Code 128B, and Code 128C select subsets A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character. Shift character: «« (« = ASCII 171 decimal).
- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset: Code character: « + A|B|C (« = ASCII 171 decimal).

DataMatrix Symbology Information

The next table includes parameter ranges and settings for the DataMatrix symbology.

Command	Ranges and Settings
BARTYPE	"DATAMATRIX"
BARRATIO	Fixed. Values will be interpreted as BARMAG.
BARMAG	<128. Default is 2.
BARHEIGHT	Not applicable.
BARFONT	Set to ON to print the human readable. If ECI support is disabled, the human readable is printed as is. If ECI support is enabled, the human readable is printed without the ECI-related characters. For example, a bar code string of "\000009abc\\123" is printed as "abc\123".

BARSET Data

The next table includes parameters for use with the BARSET command.

Parameter	Instruction	Ranges and Settings
<sexp>	Bar code type.	"DATAMATRIX"
<nexp1-2>	Not applicable.	
<nexp3>	Barmag/Enlargement.	< 128. Default is 2.
<nexp4>	Not applicable.	
<nexp5>	Bar code shape.	0: Square (default) 1: Rectangular
<nexp6>	Bar code size.	See the next section.
<nexp7-10>	Not applicable.	
<nexp11>	Enables or disables ECI support.	0: Disables ECI support. 1: Enables ECI support.

Bar code size (<*nexp6*>)

By default, the number of characters determines the size of the symbol. For example, the data "123456" generates a 10 row × 10 column DataMatrix symbol, and 72 digits generates a 24 row × 24 column DataMatrix symbol. Fewer characters can be used for all symbols (10 × 10, 12 × 12, and so on up to 144 × 144) if the data includes non-numeric characters.

If the number of characters in the bar code exceeds the size you specify, the error "Too many characters in barcode" appears. Specify a larger size for the bar code.

If you specify an invalid value for <*nexp6*>, such as 10 for a rectangular bar code, Fingerprint uses the default value.

Sizes for square bar codes:

Value	Dimensions
0	Default
1	10 × 10
2	12 × 12
3	14 × 14
4	16 × 16
5	18 × 18
6	20 × 20
7	22 × 22
8	24 × 24
9	26 × 26
10	32 × 32
11	36 × 36
12	40 × 40
13	44 × 44
14	48 × 48
15	52 × 52
16	64 × 64
17	72 × 72
18	80 × 80
19	88 × 88
20	96 × 96

Value	Dimensions
21	104 × 104
22	120 × 120
23	132 × 132
24	144 × 144

Sizes for rectangular bar codes:

Value	Dimensions
0	Default
1	8 x 18
2	8 x 32
3	12 x 26
4	12 x 36
5	16 x 36
6	16 x 48

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	2,335 ASCII characters.	
Check digit	Yes.	Added automatically.
Input characters	ASCII 0-255 decimal.	

- ECC 200 type is used for this DataMatrix symbol. ECC 200 data may be encoded using any of the encodation schemes ASCII, C40, Text, X12, EDIFACT or Base 256. ASCII encodation is the default scheme. All other encodation schemes are invoked from ASCII encodation and return to this scheme.
- If you are encoding a function code using ASCII characters, make sure to include the function code before the encoded numerical data. For example, to encode "Function Code 1 followed by "240123456789" in a barcode, enter "CHR$(26) + "1"+"240123456789".

Remarks

When DataMatrix is selected and ECI support is enabled (<*nexp11*> = 1), the strings "\<nnnnnn>" and "\\" are interpreted as follows:

- \<nnnnnn>: The backslash indicates that the character set should be changed, and nnnnnn is the number of the chosen character set. The string is decoded as an ECI

character followed by one to three digit numbers in the bar code. All ECI characters are printed in ASCII encodation, meaning that this string itself can be encoded in any mode. The algorithm switches to ASCII mode to encode the ECI characters, then encodes the other characters in the most efficient modes.

- \\: Two backslashes are decoded as a single backslash character ("\") in the bar code. All double backslashes are printed in the most efficient modes as specified and are decoded the same way when ECI support is not enabled.

Error 1112 (ECI syntax error) occurs if ECI mode is enabled and the PRBAR input string is invalid.

MaxiCode Symbology Information

The next table includes parameter settings for the MaxiCode symbology.

Command	Ranges and Settings
BARTYPE	"MAXICODE"
BARRATIO	Not applicable. Input ignored.
BARMAG	Not applicable. Input ignored.
BARHEIGHT	Not applicable. Input ignored.
BARFONT	Not applicable. Input ignored.

Remarks

MaxiCode requires 8 fields of data separated by an LF character (entered as CHR$(10)).

- Maximum number of characters: 84 (Modes 2 and 3) or 138 (Mode 4).
- Check character is automatic (Reed-Solomon algorithm).
- Data type: ASCII 0 to 255.

For all modes, each field must be present in final in data string and must contain valid data as described in the next table.

Field	Mode	Data Type and Range	Notes
1	2, 3	Mode 2 - 5 numeric characters. Mode 3 - 6 alpha-numeric characters.	
2	2	4 numeric characters [0-9999].	

Field	Mode	Data Type and Range	Notes
3	2, 3	3 numeric characters [0-999].	Country code.
4	2, 3	3 numeric characters [0-999].	Service class.
5	2, 3, 4	User-defined message.	
6		1 numeric character [2|3|4].	Mode selector.
7		1 numeric character [1-8].	Position in structured append. See the next section, "About Structured Append Mode."
8		1 numeric character.	Total number of symbols in structure.

About Structured Append Mode

F8 in structured append is the trigger for structured append mode:

- If F8 >1, the two first codewords in secondary message will be a pad followed by position codeword. F8 has higher precedence than F7.
- If F7 >1 when F8 = 1, the two first codewords do not signal structured append.
- If F8 >1, F7 may be >F8 without error and structured append codeword will signal given values.

Primary and Secondary Message Data Elements

The primary and secondary message data elements are described in the next tables. All primary message data elements are required for Mode 2 (structured data message for U.S. destinations) and Mode 3 (structured data message for international destinations).

Primary Message Data Elements

Element	Mode and Length	Sample
Zip code + 4-digit extension	Mode 2 (Numeric): 9 Mode 3 (Alphanumeric): 6	152392802 B1050
Country code	Mode 2 (Numeric): 3 Mode 3 (Numeric): 3	840
Service class	Mode 2(Numeric): 3 Mode 3 (Numeric): 3	001

Secondary Message Data Elements

Element	Mode and Length	Sample
Secondary message	Mode 2 (Alphanumeric): 84 Mode 3 (Alphanumeric): 84 Mode 4 (Standard symbol): 138	This is the secondary message.
Header (optional)	Mode 2 (Numeric): 9 Mode 3 (Numeric): 9	[]RS01GS97

Example

This example creates a MaxiCode symbol based on the following information:

Zip Code:	84170
Zip Code Extension:	1280
Country Code:	840
Class of Service:	001
Message Header:	[]><RS>01
Year:	07
Tracking Number:	1Z12345675
SCAC:	UPSN
UPS Shipper Number:	12345E
Julian Day of Pickup	089
Shipment ID:	1324567
Package:	1/1
Weight:	10.1
Address Validation:	Y
Ship to Street	1 Main ST
Ship to City:	PITTSBURGH
Ship to State:	PA

The syntax for this information is:

```
10 PRPOS 100,100
20 DIR 1
30 ALIGN 1
40 a$="84170"+CHR$(10)+"1280"+CHR$(10)+"840"+
CHR$(10)+"001"+CHR$(10)+"[)>"+CHR$(30)+

"01"+CHR$(29)+"07"+"1Z12345675"+CHR$(29)+"UPSN"+CHR$(29)+"12345E"+
CHR$(29)+"089"
```

```
+CHR$(29)+"1234567"+CHR$(29)+"1/1"+CHR$(29)+"10.1"+CHR$(29)+"Y"+CH
R$(29)+"1 MAIN ST"
50 b$=
CHR$(29)+"PITTSBURGH"+CHR$(29)+"PA"+CHR$(29)+CHR$(30)+CHR$(4)+CHR$
(10)+"2"+CHR$(10)+"1"+CHR$(10)+"1"
60 BARTYPE "MAXICODE"
70 PRBAR a$;b$
80 PRINTFEED
RUN
```

MicroPDF417 Symbology Information

MicroPDF417 is a multi-row symbology based on PDF417. A limited set of symbol sizes is available where each size has a fixed level of error correction. Up to 250 alphanumeric characters or 366 numeric digits can be encoded in a symbol.

Enhanced applications such as Extended Channel Interpretation (ECI), structured append, reader initialization, Code 128 emulation, and macro characters are not supported.

BARSET Data

The next table includes parameter ranges and settings to use with the BARSET command.

Parameter	Instruction	Ranges and Settings
<sexp>	Bar code type.	"MICROPDF417"
<nexp1-2>	Not applicable.	
<nexp3>	Element width in dots.	1 to 21. Default is 2.
<nexp4>	Element height in dots.	1 to 127. Default is 100.
<nexp5-7>	Not applicable.	
<nexp8>	Number of rows.	0, 4 to 44. Default is 0 (=automatic).
<nexp9>	Number of columns.	0 to 4. Default is 0 (=automatic).
<nexp10>	Not applicable.	

Setting the Number of Rows and Columns

The symbol size is defined by specifying the number of rows and columns. Not all combinations of rows and columns are allowed. The next table illustrates the valid combinations.

Total Columns	Valid Number of Rows for Total Columns										
1	11	14	17	20	24	28	-	-	-	-	-
2	8	11	14	17	20	23	26	-	-	-	-

Total Columns	Valid Number of Rows for Total Columns										
3	6	8	10	12	15	20	26	32	38	44	-
4	4	6	8	10	12	15	20	26	32	38	44

If the number of rows is set to a value that does not match the valid values for the given number of columns, the printer automatically chooses a larger number from the list of valid values.

Automatic Selection

The number of columns and rows can be set automatically by the printer. If the number of columns is set to 0, the printer sets the number of columns and rows automatically, regardless of the number of rows specified. The printer tries to fit the given data into a symbol with as few columns as possible. If the number of columns is non-zero and the number of rows is set to 0, the printer automatically sets the number of rows to the lowest number required to encode the given data.

Examples

This example shows how a MicroPDF417 bar code is specified using the BARTYPE and BARSET statements.

Bar width: 2 dots
Bar height: 8 dots
Number of rows: 26
Number of columns: 3

BARTYPE "MICROPDF417"

BARSET #4,2,8,1,1,1,26,3

The bar width and bar height can also be set using BARMAG and BARHEIGHT respectively.

The number of columns and rows are set using the BARSET statement. Parameters number 9 and 10 are the number of rows and columns respectively. Examples A and B below set the number of rows to 12 and the number of columns to 3. The type of bar code is set to MicroPDF417. Not all parameters of the BARSET command are applicable to the MicroPDF417 implementation. The parameters ignored by the implementation (large bar ratio, small bar ratio, security level, aspect height, and aspect width) are set to '1' in example B.

Example A (Direct Protocol):

BARTYPE "MICROPDF417"

BARSET #9, 12

BARSET #10, 3

Example B (Direct Protocol)

BARSET "MICROPDF417",1,1,2,8,1,1,1,12,3

The example code below prints a small MicroPDF417 bar code containing the string "MicroPDF417." The number of rows and columns is set by the printer based on the input string since the number of columns is set to 0.

```
10 BARSET "MICROPDF417",1,1,4,8,1,1,1,0,0
20 PRPOS 50, 50
30 PRBAR "MICROPDF417"
40 PRINTFEED
```

PDF417 Symbology Information

The next table includes parameter information for the PDF417 symbology.

Command	Ranges and Settings
BARTYPE	"PDF417"
BARRATIO	Fixed. Values are interpreted as BARMAG.
BARMAG	2 (default) to 128.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	Not applicable.

BARSET Data

The next table includes PDF417 parameter ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<sexp>	Bar code type.	"PDF417"
<nexp1>	Light bar ratio.	No restrictions. Default is 3.
<nexp2>	Narrow bar ratio.	No restrictions. Default is 1.
<nexp3>	Barmag/Enlargement.	< 128. Default is 2.
<nexp4>	Barheight.	< 500. Default is 100.
<nexp5>	Security level.	1-5. Default is 2.
<nexp6>	Aspect height ratio.	No restrictions. Default is 3.
<nexp7>	Aspect width ratio.	No restrictions. Default is 1.
<nexp8>	Number of rows.	0, and 3-90. Default is 0.
<nexp9>	Number of columns.	0-30. Default is 0.
<nexp10>	Truncate.	0 or not equal to 0. The default 0 prints a normal symbol.

Input Data

Parameter	Ranges or Settings	Notes
Number of characters	1,800 ASCII characters or 2,700 digits.	
Check digit	Yes.	Added automatically.
Input characters	ASCII 0-255 decimal.	

Example

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

Parameter	Data or Setting	Description
Data Identifier/Separator	Data	Field Name
[]>{RS}		Message Header
06{GS}		Format Header
P	12345678	Part Number
{GS}		Group Separator
Q	160	Quantity
{GS}		
1J	UN123456789A2B4C6D8E	License Plate
{GS}		
20L	LA6-987	Material Handling Code
{GS}		
21L	S4321ZES	Plant/Dock Code
{GS}		
K	GM1234	PO Number
{GS}		
15K	G1155	Kanban Number
{GS}		
B	KLT3214	Container Type
{GS}		
7Q	10GT	Gross Weight
{RS}		Record Separator
{EOT}		Message Trailer

The syntax is as follows:

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

+CHR$(29)+"KGM1234"+CHR$(29)+"15KG1155"+CHR$(29)+"BKLT3214"+CHR$(2
9)+"7Q10GT"+CHR$(30)+CHR$(4)
60 PRINTFEED
RUN
```

QR Code Symbology Information

Command	Ranges and Settings
BARTYPE	"QRCODE"
BARRATIO	To be defined.
BARMAG	1-30.
BARHEIGHT	Not applicable.
BARFONT	To be defined.

BARSET Parameters

The next table includes QR Code ranges and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"QRCODE"
<nexp1-2>	Not applicable.	
<nexp3>	Element size in dots.	1 to 27. Default is 2.
<nexp4>	QR Code model.	1 (default) or 2 (recommended)
<nexp5>	Security level.	1: L 2: M (default) 3: Q 4: H
<nexp6>	Mask pattern selection.	Not currently supported. Reserved.
<nexp7-10>	Not applicable.	

Input Data Capacity

Security Level	Correction Level	Numeric	Alphanumeric	8-bit Byte Data	Kanji
L	7%	Model 1: 1176 Model 2: 7089	707 4296	486 2953	299 1817
M	15%	Model 1: 877 Model 2: 5596	531 3391	365 2331	225 1435
Q	25%	Model 1: 738 Model 2: 3993	447 2420	307 1663	189 1024
H	30%	Model 1: 498 Model 2: 3057	302 1842	207 1273	127 784

The unit is number of characters. Mixed mode is supported for all combinations except or combinations containing both 8-bit byte and Kanji data. The type of data is set automatically by the implementation based on the input characters.

Example

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

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

or

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

RSS-14 Symbology Information

The next table includes parameter information for the RSS-14 symbology.

Command	Ranges and Settings
BARTYPE	"RSS14"

Command	Ranges and Settings
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	Not applicable.

BARSET Data

The next table includes RSS-14 ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<sexp>	Bar code type.	"RSS14"
<nexp1>	Placeholder.	Value insignificant.
<nexp2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Placeholder.	Value insignificant.
<nexp7>	Placeholder.	Value insignificant.
<nexp8-10>	Not applicable.	

Input Data

Parameter	Ranges and Set-tings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are auto-matically added so the string is 13 digits long.

Remarks

There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. For RSS-14, the width is 96X and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET*<nexp3>*.

For RSS used in composite bar codes, see the information for that RSS Composite symbology.

RSS-14 Expanded Symbology Information

The next table includes parameter settings for the RSS-14 Expanded symbology.

Command	Ranges and Settings
BARTYPE	"RSS14E"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	No restrictions.

BARSET Data

The next table includes RSS-14 Expanded ranges and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"RSS14L"
<nexp1>	Placeholder.	Value insignificant.
<nexp2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Placeholder.	Value insignificant.
<nexp7>	Placeholder.	Value insignificant.
<nexp8-10>	Not applicable.	

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	Max. 71 numeric or 41 alphanumeric characters.	Allowed characters: 0-9, A-Z, a-z ! " % & ' ( ) * + , - . / : ; < = > ? _ space FNC1 [CHR$(128)]

Remarks

Use RSS-14 Expanded for intelligent encoding of the input data. RSS-14E bar codes can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSS-14E, see the most current AIM specification.

RSS-14 Expanded Stacked Symbology Information

The next table includes parameter information for the RSS-14 Expanded Stacked symbology.

Command	Ranges and Settings
BARTYPE	"RSS14ES"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	No restrictions.

BARSET Data

The next table includes RSS-14 Expanded Stacked ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<sexp>	Bar code type.	"RSS14ES"
<nexp1>	Placeholder.	Value insignificant.
<nexp2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Segments per row.	2-22 (default 2). Multiples of 2 only.
<nexp6>	Placeholder.	Value insignificant.
<nexp7>	Height of a separator row.	The height for the pattern rows between the linear rows. If too low a height is entered, the height will be changed to the smallest legal height for the selected magnification. Minimum 1X, maximum 2X.
<nexp8-10>	Not applicable.	

Input Data

Bar codes in RSS-14 Expanded Stacked can be a maximum of 71 numeric or 41 alphanumeric characters long. Allowable characters include:

0-9

A-Z

a-z

Other characters: ! " % & ' ( ) * + , - . / : ; < = > ? _ space FNC1 [CHR$(128)]

Remarks

There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs because the bar code rows do not have the same height. BARHEIGHT or BARSET<*nexp4*> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14ES, the width depends on input. The minimum height is 34X per row + 3*1X per separator, where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

RSS-14ES can be used for intelligent encoding of the input data. RSS-14ES bar codes can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSS-14ES, see the most current AIM specification.

Example

This example is of RSS-14 Expanded Stacked for a variable weight item (0.001 kilogram increments) and with recommended minimum height selected:

Description	Value
Start parameter	#4
Most narrow element width in dots	2
Height in dots	68
Segments per row	4
Place holder <nexp6>	1
Separator pattern row height	2

Description	Value
Data	0190012345678900310300175O|9876543210 where: 01 is the Application Identifier (AI). 900123456780 is the AI 01 item ID. 0 is used as a placeholder. 3103 is the Application Identifier (AI). 001750 is the AI3103 variable weight element string. 9876543210 is the 2D data string,

BARTYPE "RSS14ES"

BARSET #4,2,68,4,1,2

PRBAR "0190012345678900310300175O"

RSS-14 Limited Symbology Information

The next table includes parameter information for the RSS-14 Limited symbology.

Command	Ranges and Settings
BARTYPE	"RSS14L"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	No restrictions.

BARSET Data

The next table includes RSS-14 Limited ranges and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"RSS14L"
<nexp1-2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.

BARSET	Description	Ranges and Settings
<*nexp4*>	Barheight.	See BARHEIGHT.
<*nexp5-7*>	Placeholder.	Value insignificant.
<*nexp8-10*>	Not applicable.	

Input Data

Parameter	Ranges and Set-tings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are auto-matically added so the string is 13 digits long.

Remarks

There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs because the bar code rows do not have the same height. BARHEIGHT or BARSET<*nexp4*> specifies the height of the lower row, and the height of the upper row is automatically calculated from the height of the lower row.

For RSS-14L, the width is 71X and the minimum height is 10X, where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

RSS-14 Stacked Symbology Information

The next table includes parameter settings for the RSS-14 Stacked symbology.

Command	Ranges and Settings
BARTYPE	"RSS14S"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	No restrictions.

BARSET Data

The next table includes RSS-14 Stacked ranges and settings for use with the BARSET command.

Parameter	Description	Ranges and Settings
<sexp>	Bar code type.	"RSS14S"
<nexp1-2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5-6>	Placeholder.	Value insignificant.
<nexp7>	Placeholder.	
<nexp8-10>	Not applicable.	

Input Data

Parameter	Ranges and Set-tings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are added automatically so the string is 13 digits long.

Remarks

There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp4> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14S, the width is 50X and the minimum height is 13X (upper 5X + lower 7X + separator 1X minimum.), where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp3>.

Example

This example creates an RSS14S bar code with the following characteristics and with recommended minimum height selected:

Place holder (nexp1): 1
Place holder (nexp2): 1
Most narrow element width in dots: 3
Height in dots: 21
Place holder (nexp5): 1
Place holder (nexp6): 1
Separator pattern row height: 4
Data: 1234567890123

```
BARSET "RSS14S",1,1,3,21,1,1,4
PRBAR "1234567890123"
```

RSS-14 Stacked Omnidirectional Symbology Information

The next table includes parameter settings for the RSS-14 Stacked Omnidirectional symbology.

Command	Ranges and Settings
BARTYPE	"RSS14SO"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	No restrictions.

BARSET Data

The next table includes RSS-14 Stacked Omnidirectional parameters and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"RSS14SO"
<nexp1>	Placeholder.	Value insignificant.
<nexp2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Placeholder.	Value insignificant.
<nexp7>	Placeholder.	
<nexp8-10>	Not applicable.	

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are added automatically so the string is 13 digits long.

Remarks

There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp4> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14SO, the width is 50X and the minimum height is 69X (upper 33X + lower 33X + separator 3*1X minimum.).

RSS-14 Truncated Symbology Information

The next table includes parameter settings for the RSS-14 Truncated symbology.

Command	Ranges and Settings
BARTYPE	"RSS14T"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See Remarks for more information.
BARFONT	No restriction.

BARSET Data

The next table includes parameter ranges and settings for use with the BARSET command.

Parameter	Description	Range and Settings
<sexp>	Bar code type.	"RSS14T"
<nexp1>	Placeholder.	Value insignificant.
<nexp2>	Placeholder.	Value insignificant.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Placeholder.	Value insignificant.
<nexp7>	Placeholder.	Value insignificant.
<nexp8-10>	Not applicable.	

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes will automatically be added so the string will be 13 digits long.

Remarks

There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs because the bar code rows do not have the same height. BARHEIGHT or BARSET<*nexp4*> specifies the height of the lower row, and the height of the upper row is automatically calculated from the height of the lower row.

For RSS-14 Truncated, the width is 96X and the minimum height is 13X, where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

UPC-A Symbology Information

The next table includes parameter information for the UPC-A symbology.

Command	Ranges and Settings
BARTYPE	"UPCA"
BARRATIO	Fixed ratio. BARRATIO statement ignored.
BARMAG	Maximum 8. Default is 2.
BARHEIGHT	No restrictions. Default is 100.
BARFONT	Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF work.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	11	
Check digit	One	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start character	None.	

Parameter	Ranges and Settings	Notes
Stop character	None.	

To learn more about using the PRBAR statement to generate an add-on bar code, see About AddOn Codes.

UPC-E Symbology Information

The next table includes parameter information for the UPC-E symbology.

Command	Ranges and Settings
BARTYPE	"UPCE"
BARRATIO	Fixed ratio. BARRATIO statement ignored.
BARMAG	Maximum 8. Default is 2.
BARHEIGHT	No restriction. Default is 100.
BARFONT	Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF work.

Input Data

Parameters	Ranges and Settings	Notes
Number of characters	6	
Check digit	1	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start character	None.	
Stop character	None.	

To learn more about using the PRBAR statement to generate an add-on bar code, see About AddOn Codes.

USPS 4-State

The table below includes parameter information for the USPS intelligent mail barcode symbology.

Command	Ranges and Settings
BARTYPE	"USPS4CB"
BARRATIO	Fixed. Input ignored.
BARMAG	Fixed. Input ignored.
BARHEIGHT	Fixed. Input ignored.
BARFONT	No restrictions.

BARSET Parameters

The next table includes parameter information for use with the BARSET command.

BARSET	Description	Ranges and Settings
<*sexp*>	Bar code type.	No default.
<*nexp1-11*>	Not applicable.	

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	20, 25, 29, or 31	
Input characters	Numeric.	

About Composite Bar Codes

An EAN.UCC Composite symbol consists of a linear component, an adjacent 2D Composite Component, a separator pattern, and an optional human readable section.

The Composite symbol always includes a linear component so that the primary identification is readable by all scanning technologies. 2D imagers can use the linear component as a finder pattern for the adjacent 2D Composite Component.

EAN.UCC Composite Symbology is dependent on several other symbologies. Some minor adjustment of implementation of PDF417 and MicroPDF417, nine new symbologies and a logical way of combining nine linear and three 2D symbologies in different composite combinations have been implemented.

- Learn more about the linear components
- Learn more about the 2D composite components

About the Human Readable Section

For all combinations of the EAN.UCC Composite symbology, the human readable section is an optional presentation of the information in either or both bar codes. The linear human readable section is presented below the linear component, within the same rules as for the single bar code. The human readable for the 2D composite component part is presented under the human readable for the linear bar code.

EAN/UPC bar codes have fixed fonts for the human readable bar code interpretation.

Supported Component Combinations

Based upon the width of the linear component, a choice of "best fit" 2D Composite Component is specified. The table below lists all of the permissible combinations.

Linear Component	CC-A/CC-B	CC-C	Number of columns
UPC-A & EAN-13	Yes/Yes	No	4
EAN-8	Yes/Yes	No	3
UPC-E	Yes/Yes	No	2
UCC/EAN-128	Yes/Yes	Yes	4|variable width (1-30)
RSS-14	Yes/Yes	No	4
RSS-14 Stacked	Yes/Yes	No	2
RSS-14 Stacked omnidirectional	Yes/Yes	No	2

Linear Component	CC-A/CC-B	CC-C	Number of columns
RSS Limited	Yes/Yes	No	3
RSS Expanded	Yes/Yes	No	4
RSS Expanded Stacked	Yes/Yes	No	4

About the 2D Composite Components

The 2D Composite Component, abbreviated as CC, encodes supplementary data, such as a batch number or expiration date. The CC is chosen based on the selected linear component and on the amount of complementary data to be encoded. The choice of linear symbol determines the name of the composite symbol, such as EAN-13 Composite symbol or UCC/EAN-128 Composite symbol.

There are three types:

- CC-A, a variant of MicroPDF417, designed for efficient encoding of supplemental application identifier data.
- CC-B, a MicroPDF417 symbol with codeword 920 in the first data codeword position as a linkage flag denoting EAN.UCC Composite Symbology data compaction.
- CC-C, a PDF417 symbol with a codeword 920 in the first data codeword position as a linkage flag denoting EAN.UCC Composite Symbology data compaction.

These types are described in the next section.

About the CC-A, CC-B, and CC-C Composite Symbols

The CC-A, CC-B and CC-C composite symbols are almost the same bar code as PDF417 and MicroPDF417.

CC-A

The CC-A component is a variant of MicroPDF417, and the smallest of the 2D composite components. CC-A can encode up to 56 digits and has 3 to 12 rows and 2 to 4 columns.

CC-B

The CC-B component is a variant of MicroPDF417 symbol uniquely identified by the codeword "920" as the first code word in the symbol. Encoding systems select CC-B when the data to be encoded exceeds the capacity for CC-A. CC-B can encode up to 338 digits and has from 10 to 44 rows and 2 to 4 columns.

CC-C

The CC-C component is the same as a PDF417 bar code except for the following:

- CC-C has the code word "920" in the second symbol character position (immediately following the Symbol Length Descriptor).
- High-level data encodation: after the first data codeword (920), the second data codeword is a latch to Byte Compaction mode.

- Structured append is not used.
- Reader initialization: code word 921 does not appear in CC-C.
- No quiet zones are required above and under the bar code.
- Reference decode algorithm: an incorrect composite 2D bar code.
- Error correction: CC-C meets or exceeds the minimum error correction level recommended for PDF417.
- Symbology identifiers: special for CC-C.

About the CC Escape Mechanism

After the first Byte mode codeword 901 or 924 in the 2D Composite component, if another codeword greater than 899 occurs, the composite "symbol" is logically terminated at that point. The remaining codewords are encoded or decoded according to special rules.

Example

This example corresponds to the illustration in About Composite Bar Codes:

```
BARFONT ON
BARSET "EAN128_CCC",1,1,5,100,0,0,0,0,0,1
PRBAR "(01)93812345678901|(10)ABCD123456(410)389
8765432108"
PRINTFEED
```

About the Linear Components

The linear bar code component encodes the item's primary identification in one of the following barcode types:

- UCC/EAN-128
- EAN/UPC: 8 or 13
- UPC-A or E
- Any RSS-family symbology that includes a separator character in between the data for the linear component and the 2D component of the input string.

For more information on linear components described by symbology, see the next section.

Linear Components by Symbology

UPC-A

There is no linkage flag in the UPC-A symbol to indicate the presence of an associated 2D Composite Component. UPC-A linear component may only be linked to four-column CC-A or CC-B components.

EAN-13

There is no linkage flag in the EAN-13 symbol to indicate the presence of an associated 2D Composite Component. EAN-13 linear component may only be linked

to four-column CC-A or CC-B components.

EAN-8

There is no linkage flag in the EAN-8 symbol to indicate the presence of an associated 2D Composite Component. EAN-8 linear component may only be linked to three-column CC-A or CC-B components.

UPC-E

There is no linkage flag in the UPC-E symbol to indicate the presence of an associated 2D Composite Component. UPC-E linear component may only be linked to two-column CC-A or CC-B components.

UCC/EAN-128

UCC/EAN-128 is a Code 128 with a FNC1 character in the first position after the start character. When UCC/EAN-128 is the linear component of an EAN.UCC Composite Symbol, the bar code has a Code Set character as a linkage flag in the last symbol character position before the check character. The printer will add the linkage flag automatically.

	Link Flag Character	
Active Code Set	CC-A or CC-B	CC-C
A	CODE B	CODE C
B	CODE C	CODE A
C	CODE A	CODE B

UCC/EAN-128 Composite Symbol may be combined with any of the 2D bar code components created for the EAN.UCC Composite Symbology. The choice between using CC-A/B or CC-C is made by the user. This table describes how an UCC/EAN-128 Composite symbol is built.

Parameter	Notes
Start character	Added by the printer.
FNC1	Added by the printer.
Data	Data is added by the user. If it is necessary to change Code Set the printers add Code- or Shift-characters to change the subset.
Link flag character	Added by the printer.
Check digit	Calculated and added by the printer.
Stop character	Added by the printer.

It is possible to choose to filter out spaces, parentheses, and carriage returns for the bar code data and display them in the human readable field. Carriage returns make it possible to display the human readable field in multiple rows.

- UCC/EAN-128 With a 2D Composite Component (CC-A or CC-B): The length of the data determines if a CC-A or a CC-B is used. Only a 4-column wide CC-A or a CC-B is used.
- UCC/EAN-128 With a 2D Composite Component (CC-C): The number of columns of the CC-C is selectable by the user. The separator pattern is a complement of the linear symbol. The patterns above the UCC/EAN-128 component's quiet zones are light. The 2D bar code is placed above the separator pattern on the following position: The first interior space module of the CC-C component is aligned with the second module of the linear components star character.

RSS

When RSS symbols are used as a Composite Component, the encoded value includes a linkage flag indicating the presence of an adjacent 2D Composite Component.

RSS-14

RSS-14 linear component may only be linked to four-column CC-A or CC-B components.

RSS-14 Truncated

RSS-14 Truncated linear component may only be linked to four-column CC-A or CC-B components.

RSS-14 Stacked

RSS-14 Stacked linear component may only be linked to two-column CCA or CC-B components.

RSS-14 Stacked Omnidirectional

RSS-14 Stacked Omnidirectional linear component may only be linked to two-column CC-A or CC-B components.

RSS Limited

RSS-14 Limited linear component may only be linked to three-column CC-A or CC-B components.

RSS Expanded

RSS-14 Expanded linear component may only be linked to four-column CC-A or CC-B components.

RSS Expanded Stacked

RSS-14 Expanded Stacked linear component may only be linked to four-column CC-A or CC-B components. If linked to a 2D Composite Component, the top row of the linear component must contain at least four symbol characters.

EAN-8 Composite with CC-A or CC-B Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the EAN-8 symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"EAN8_CC"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2. Maximum is 8.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET<nexp10> is set to 1, the human readable for the 2D bar code is printed also. OFF: No human readable is printed.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Instruction	Ranges and Settings
<sexp>	Bar code type.	"EAN8_CC"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human readables).	Always 1 (default). The bar code and the human readable field will include exactly the same data.
<nexp7>	Height of separator pat-tern row in dots.	Value insignificant.
<nexp8>	Height of each row in 2D bar code in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default is ASCII 124 decimal. ASCII 48-57 decimal not allowed.

BARSET	Instruction	Ranges and Settings
<*nexp10*>	2D human readables On/Off.	0: No human readable 2D field. 1: Print human readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	7	
Check digit	1	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start characters	None.	
Stop characters	None.	

EAN-13 Composite with CC-A or CC-B Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the EAN-13 symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"EAN13_CC"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2. Maximum is 8.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET<nexp10> is set to 1, the human readable for the 2D bar code is printed also. OFF: No human readable is printed.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Instruction	Ranges and Settings
<sexp>	Bar code type.	"EAN13_CC"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human readables).	Always 1 (default). The bar code and the human readable field will include exactly the same data.
<nexp7>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 x BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal Default is ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human readables On/Off.	0: No human readable 2D field. 1: Print human readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	12	
Check digit	1	Added automatically.
Digits	0 to 9	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start characters	None.	
Stop characters	None.	

EAN.UCC 128 Composite with CC-A or CC-B Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the EAN.UCC 128 symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"EAN128_CCAB"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET<nexp10> is set to 1, the human readable for the 2D bar code is printed also. OFF: No human readable is printed.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<sexp>	Bar code type.	"EAN128_CCAB"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human readables).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1: (Default) The bar code and the human readable field will include exactly the same data.
<nexp7>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default is ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human readables On/Off.	0: No human readable 2D field. 1: Print human readable 2D field.

For input data information and other remarks, see the Input Data section in the EAN-128 symbology information.

EAN.UCC 128 Composite with CC-C Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the EAN.UCC 128 symbology with a CC-C composite component.

Command	Ranges and Settings
BARTYPE	"EAN128_CCC"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET<*nexp10*> is set to 1, the human readable for the 2D bar code is printed also. OFF: No human readable is printed.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Ranges and Settings
<*sexp*>	Bar code type.	"EAN128_CCC"
<*nexp1-2*>	Placeholder.	Value insignificant. 0 is not allowed.
<*nexp3*>	Barmag/Enlargement.	See BARMAG.
<*nexp4*>	Barheight.	See BARHEIGHT.
<*nexp5*>	Number of columns/row in 2D bar code.	0 (default): Selects the largest number of columns for the 2D code to fit within the linear bar code including its quiet zones.
<*nexp6*>	Character exclusion in bar code (does not affect human readables).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1: (default) The bar code and the human readable field will include exactly the same data.
<*nexp7*>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.

BARSET	Description	Ranges and Settings
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default is ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human readables On/Off.	0: No human readable 2D field. 1: Print human readable 2D field.

For input data information and other remarks, see the Input Data section in the EAN-128 symbology information.

RSS-14 Composite Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the RSS-14 symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"RSS14"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	Not applicable.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<sexp>	Bar code type.	"RSS14"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.

BARSET	Description	Notes
<*nexp6*>	Character exclusion in bar code (does not affect human-readable fields).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human-readable field. Carriage returns make it possible to print human-readable fields in multiple rows. 1: (Default) The bar code and the human-readable field will include exactly the same data.
<*nexp7*>	Height of separator pattern row in dots.	Value must be 1–2 × BARMAG. For example, if BARMAG = 5, 5–10 is OK.
<*nexp8*>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<*nexp9*>	Separator character between data to linear bar code and 2D bar code.	ASCII 2–255 decimal. Default: ASCII 124 decimal. ASCII 48–57 decimal not allowed.
<*nexp10*>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are automatically added so the string is 13 digits long.

Remarks

For RSS symbologies used in composite bar codes, width and height standards are different because the bar code rows do not have the same height. BARHEIGHT or BARSET<*nexp4*> specifies the height of the lower row. The height of the upper row is automatically calculated from the height of the lower row.

For RSS-14, the width is 96X and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

RSS-14 Expanded Composite Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the RSS-14 Expanded symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"RSS14E"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	Not applicable.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<sexp>	Bar code type.	"RSS14E"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human-readable fields).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human-readable field. Carriage returns make it possible to print human-readable fields in multiple rows. 1: (Default) The bar code and the human-readable field will include exactly the same data.
<nexp7>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default: ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are automatically added so the string is 13 digits long.

Remarks

For RSS symbologies used in composite bar codes, width and height standards are different because the bar code rows do not have the same height. BARHEIGHT or BARSET<*nexp4*> specifies the height of the lower row. The height of the upper row is automatically calculated from the height of the lower row.

For RSS-14E, the width depends on input and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

RSS-14 Expanded Stacked Composite Symbology Information

The next table includes parameter information for the RSS-14 Expanded Stacked symbology.

Command	Ranges and Settings
BARTYPE	"RSS14ES"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	No restrictions.

BARSET Data

The next table includes RSS-14 Expanded Stacked ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<*sexp*>	Bar code type.	"RSS14ES"
<*nexp1*>	Placeholder.	Value insignificant.
<*nexp2*>	Placeholder.	Value insignificant.
<*nexp3*>	Barmag/Enlargement.	See BARMAG.
<*nexp4*>	Barheight.	See BARHEIGHT.

BARSET	Description	Notes
<nexp5>	Segments per row.	2-22 (default 2). Multiples of 2 only.
<nexp6>	Placeholder.	Value insignificant.
<nexp7>	Height of a separator row.	The height for the pattern rows between the linear rows. If too low a height is entered, the height will be changed to the smallest legal height for the selected magnification. Minimum 1X, maximum 2X.
<nexp8>	Height (in dots) of each row in the 2D bar code.	Default: 0 (=3 x BARMAG)
<nexp9>	Separator character between data in linear bar code and 2D bar code.	ASCII 2-255 decimal. ASCII 48-57 decimal. not allowed. Default is ASCII 124 decimal
<nexp10>	2D human-readable fields On/Off.	0: Do not print human-readable field. 1: Print human-readable field.

Input Data

Bar codes in RSS-14 Expanded Stacked can be a maximum of 71 numeric or 41 alphanumeric characters long. Allowable characters include:

0-9

A-Z

a-z

Other characters: ! " % & ' ( ) * + , - . / : ; < = > ? _ space FNC1 [CHR$(128)]

For More Information

See the Remarks and Example in the RSS-14 Expanded Stacked symbology information.

RSS-14 Limited Composite Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the RSS-14 Limited symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"RSS14L"
BARRATIO	Not applicable.

Command	Ranges and Settings
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	Not applicable.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<sexp>	Bar code type.	"RSS14L"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human-readable fields).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human-readable field. Carriage returns make it possible to print human-readable fields in multiple rows. 1: (Default) The bar code and the human-readable field will include exactly the same data.
<nexp7>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default: ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits	If less than 13 digits are entered, leading zeroes are automatically added so the string is 13 digits long.

Remarks

For RSS symbologies used in composite bar codes, width and height standards are different because the bar code rows do not have the same height. BARHEIGHT or BARSET<*nexp4*> specifies the height of the lower row. The height of the upper row is automatically calculated from the height of the lower row.

For RSS-14L, the width is 71X and the minimum height is 10X, where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

RSS-14 Stacked Composite Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the RSS-14 Stacked symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"RSS14S"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	Not applicable.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<*sexp*>	Bar code type.	"RSS14S"
<*nexp1-2*>	Placeholder.	Value insignificant. 0 is not allowed.
<*nexp3*>	Barmag/Enlargement.	See BARMAG.
<*nexp4*>	Barheight.	See BARHEIGHT.
<*nexp5*>	Placeholder.	Value insignificant.

BARSET	Description	Notes
<*nexp6*>	Character exclusion in bar code (does not affect human-readable fields).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human-readable field. Carriage returns make it possible to print human-readable fields in multiple rows. 1: (Default) The bar code and the human-readable field will include exactly the same data.
<*nexp7*>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<*nexp8*>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<*nexp9*>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default: ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<*nexp10*>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are automatically added so the string is 13 digits long.

Remarks

For RSS symbologies used in composite bar codes, width and height standards are different because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp4> specifies the height of the lower row. The height of the upper row is automatically calculated from the height of the lower row.

For RSS-14S, the width is 50X and the minimum height is 13X (upper row 5X + lower row 7X + separator row 1X minimum), where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp3>.

Example

The example below produces an EAN.UCC composite bar code using RSS-14 and with these characteristics:

Placeholder (<*nexp1*>)	1
Placeholder (<*nexp2*>)	1
Narrowest element width in dots (<*nexp3*>)	3
Height in dots (<*nexp4*>)	21
Placeholder (<*nexp5*>)	1
Character exclusion (<*nexp6*>)	1
Height of separator pattern row (<*nexp7*>)	4
Height of rows in 2D bar code (<*nexp8*>)	4
Separator character (<*nexp9*>)	124
2D human-readable fields (<*nexp10*>)	1
Data for bar code	1234567890123|987654321

BARSET "RSS14S",1,1,2,21,1,1,4,4,124,1

PRBAR "1234567890123|987654321"

RSS-14 Stacked Omnidirectional Composite Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the RSS-14 Stacked Omnidirectional symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"RSS14SO"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100. See the Remarks for more information.
BARFONT	Not applicable.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<*sexp*>	Bar code type.	"RSS14SO"
<*nexp1-2*>	Placeholder.	Value insignificant. 0 is not allowed.
<*nexp3*>	Barmag/Enlargement.	See BARMAG.

BARSET	Description	Notes
<*nexp4*>	Barheight.	See BARHEIGHT.
<*nexp5*>	Placeholder.	Value insignificant.
<*nexp6*>	Character exclusion in bar code (does not affect human-readable fields).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human-readable field. Carriage returns make it possible to print human-readable fields in multiple rows. 1: (Default) The bar code and the human-readable field will include exactly the same data.
<*nexp7*>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<*nexp8*>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<*nexp9*>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default: ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<*nexp10*>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are automatically added so the string is 13 digits long.

Remarks

For RSS symbologies used in composite bar codes, width and height standards are different because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp4> specifies the height of the lower row. The height of the upper row is automatically calculated from the height of the lower row.

For RSS-14SO, the width is 50X and the minimum height is 69X (upper row 33X + lower row 33X + separator row 3*1X minimum), where X is the width of the most narrow element as specified by BARMAG or BARSET<*nexp3*>.

RSS-14 Truncated Composite Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the RSS-14 Truncated symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"RSS14T"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	Not applicable.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Description	Notes
<sexp>	Bar code type.	"RSS14T"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human-readable fields).	0: Bar code will not include space, comma (,), or CR ([CHR$(13)]) characters, but they are printed in the human-readable field. Carriage returns make it possible to print human readable fields in multiple rows. 1: (Default) The bar code and the human-readable field will include exactly the same data.
<nexp7>	Height of separator pat-tern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 × BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default: ASCII 124 decimal. ASCII 48-57 decimal not allowed.

BARSET	Description	Notes
<nexp10>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Set-tings	Notes
Number of characters	13 digits.	If less than 13 digits are entered, leading zeroes are auto-matically added so the string is 13 digits long.

UPC-A Composite with CC-A or CC-B Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the UPC-A symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"UPCA_CC"
BARRATIO	Not applicable.
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2. Maximum is 8.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET<nexp10> is set to 1, the human readable for the 2D bar code is printed also. OFF: No human readable is printed.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Instruction	Ranges and Settings
<sexp>	Bar code type.	"UPCA_CC"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.

BARSET	Instruction	Ranges and Settings
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human-readable fields).	Always 1 (default). The bar code and the human-readable field will include exactly the same data.
<nexp7>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 x BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default is ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	11	
Check digit	1	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None.	
Start characters	None.	
Stop characters	None.	

UPC-E Composite with CC-A or CC-B Symbology Information

The next table includes parameter settings for an EAN.UCC Composite symbol using the UPC-E symbology with a CC-A or CC-B composite component.

Command	Ranges and Settings
BARTYPE	"UPCE_C"
BARRATIO	Not applicable.

Command	Ranges and Settings
BARMAG	Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default is 2. Maximum is 8.
BARHEIGHT	Specifies the height of the linear bar code. Default is 100.
BARFONT	ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET<nexp10> is set to 1, the human readable for the 2D bar code is printed also. OFF: No human readable is printed.

BARSET Data

The next table includes ranges and settings for use with the BARSET command.

BARSET	Instruction	Ranges and Settings
<sexp>	Bar code type.	"UPCE_CC"
<nexp1-2>	Placeholder.	Value insignificant. 0 is not allowed.
<nexp3>	Barmag/Enlargement.	See BARMAG.
<nexp4>	Barheight.	See BARHEIGHT.
<nexp5>	Placeholder.	Value insignificant.
<nexp6>	Character exclusion in bar code (does not affect human-readable fields).	Always 1 (default). The bar code and the human-readable field will include exactly the same data.
<nexp7>	Height of separator pattern row in dots.	Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK.
<nexp8>	Height of each row in 2D bar code in dots.	Default: 0 (= 3 x BARMAG)
<nexp9>	Separator character between data to linear bar code and 2D bar code.	ASCII 2-255 decimal. Default is ASCII 124 decimal. ASCII 48-57 decimal not allowed.
<nexp10>	2D human-readable fields On/Off.	0: No human-readable 2D field. 1: Print human-readable 2D field.

Input Data

Parameter	Ranges and Settings	Notes
Number of characters	6	

Parameter	Ranges and Settings	Notes
Check digit	1	Added automatically.
Digits	0 to 9.	
Uppercase letters	None.	
Lowercase letters	None.	
Punctuation marks	None	
Start characters	None.	
Stop characters	None.	

About AddOn Codes

The EAN and UPC bar code standards allow for two- and five-character supplemental bar codes to be printed together with the normal code. It is possible to print these bar codes simply by adding the desired characters to the PRBAR statement, separated by a period (.). The placement of the AddOn bar code is done automatically.

Examples

The AddOn characters are added to the PRBAR statement by use of a period (.) character as separator:

```
10 PRPOS 100, 100
20 BARSET "EAN8"
30 ALIGN 1
40 DIR 1
50 BARFONT ON
60 PRBAR "1234567.12345"
70 PRINTFEED
RUN
```

resulting in:

The AddOn code can be used in combination with any EAN and UPC bar code set. The following example prints a composite bar code, with a two character AddOn code. For more information, see About Composite Bar Codes.

```
10 PRPOS 100,100
20 BARSET "EAN8_CC"
30 ALIGN 1
40 DIR 1
50 BARFONT ON
60 PRBAR "1234567.12|987654321"
70 PRINTFEED
RUN
```

resulting in:

CHAPTER

5

SUPPORTED RFID TAG FORMATS

Detailed information for supported RFID tag formats. Use with the TAGREAD and TAGWRITE commands.

Fingerprint supports the tag formats listed in the next table. For specific format information, click the format name:

Type	Formats
General Identifier	GID-96
Global Individual Asset Identifier	GIAI-96
Global Returnable Asset Identifier	GRAI-96
Serialized Global Location Number	SGLN-96
Serialized Global Trade Identification Number	SGTIN-96
Serial Shipping Container Code	SSCC-96
United States Department of Defense	USDOD-96

These tag format types conform to revision 1.27 of the EPC Tag Data Standards, version 1.1. Fingerprint supports 64-bit tags, although these tags are not mentioned in the Gen 2 standard.

- Learn about other EPC tag input methods
- See an example of writing to an EPC tag
- Learn about tag memory allocation

About Tag Formats

Fingerprint includes commands to read, write, and format RFID tags. When writing to tags, you use the same information from EAN.UCC encoding schemes normally printed on bar codes.

The tag format information includes only the input data necessary to program the tags by using Fingerprint. Header and partition information is contained by the tags:

- The header is uniquely defined by the tag format and not included in the input.
- Partition information is defined by the length of other fields (for example, "company prefix" and "item reference") and is automatically calculated by Fingerprint.

All values entered for EPC formats must be integers entered as strings. Be sure to enter the correct number of digits in each field, including leading zeros as necessary.

United States Department of Defense (USDOD) formats may differ slightly from EPC norms. More information on EPC and USDOD tag formats can be obtained from:
www.epcglobalinc.org
www.dodrfid.org

About the Uniform Resource Identifier (URI)

To simplify encoding, the EPC standard provides a standardized method for tag writing. The Uniform Resource Identifier (URI) representations for EPC tag formats contain just the fields necessary to distinguish objects from each other. EPC is a namespace in the Uniform Resource Name (URN) encoding scheme. In order to use the URIs, the tag format "EPC-URN" must be previously specified by the TAGFORMAT command.

The next example shows how to specify the URI for an SSCC-96 tag format:

```
10 TAGFIELD "@EPC"
20 TAGFORMAT "EPC-URN"
30 TAGWRITE "urn:epc:tag:sscc-96:0.12345678.987654321"
```

Using Non-Standard Tag Formats

Fingerprint allows users to program RFID tags with other data than EPC tag data. The available tag formats are "NUM", "HEX", and "ASCII", representing numeric, hexadecimal, and ASCII data respectively. When reading tags, the data is assumed to be in the format last specified by the TAGFORMAT command. The next example demonstrates how the ASCII characters written as "RFID" are read as "52464944" if the last format is "HEX".

```
10 TAGFIELD "@ID",2,4
20 TAGFORMAT "NUM"
30 TAGWRITE 1234
10 TAGFIELD "@DATA",10,12
20 TAGFORMAT "HEX"
30 TAGWRITE "11223344556677889900AABB"
10 TAGFIELD "@DATA",10,4
20 TAGFORMAT "ASCII"
30 TAGWRITE "RFID"
40 TAGFORMAT "HEX"
50 TAGREAD A$
60 PRINT A$
RUN
```

resulting in:

```
52464944
```

RFID Tag Formats: GIAI-96

Field	Number of Digits	Range
Filter	Not applicable	0-7
Company Prefix Index (CoPrefIndex)	6-12	Not applicable
Individual Asset Reference (IndAsset)	18-12	Not applicable

URI Representation

"urn:epc:tag:giai 96:Filter.CoPrefIndex.IndAsset.SerialNo"

Notes

Total number of digits in Company Prefix and Individual Asset Reference must be 24.

RFID Tag Formats: GID-96

Field	Number of Digits	Range
General Manager Number (GMNo)	Not applicable	0-268 435 455
Object Class (Object)	Not applicable	0-16 777 215
Serial Number (SerialNo)	Not applicable	0-549 755 813 887

URI Representation

"urn:epc:tag:gid-96:GMNo.Object.SerialNo"

RFID Tag Formats: GRAI-96

Field	Number of Digits	Range
Filter	Not applicable	0-7
Company Prefix Index (CoPrefIndex)	6-12	Not applicable
Asset Type (Asset)	6-0	Not applicable
Serial Number (SerialNo)	Not applicable	0-274 887 906 943

URI Representation

"urn:epc:tag:sgln-96:Filter.CoPrefIndex.Asset.SerialNo"

Notes

Total number of digits in Company Prefix and Asset Type must be 12.

An empty Asset Type must be entered as an empty string ("").

RFID Tag Formats: SGLN-96

Field	Number of Digits	Range
Filter	Not applicable	0-7
Company Prefix Index (CoPrefIndex)	6-12	Not applicable
Location Reference (LocRef)	6-0	Not applicable
Serial Number (SerialNo)	Not applicable	022 199 023 255 551

URI Representation

"urn:epc:tag:sgln-96:Filter.CoPrefIndex.LocRef.SerialNo"

Notes

Total number of digits in Company Prefix Index and Location Reference must be 12.

An empty Location Reference must be entered as an empty string (";").

The serial number can be left out when writing to a tag, as EAN.UCC specifications do not yet allow the use of serial numbers in SGLN tags.

RFID Tag Formats: SGTIN-96

Field	Number of Digits	Range
Filter	Not applicable	0-7
Company Prefix Index (CoPrefIndex)	6-12	Not applicable
Item Reference (ItemRef)	7-1	Not applicable
Serial Number (SerialNo)	Not applicable	0-274 877 906 943

URI Representation

"urn:epc:tag:sgtin-96:Filter.CoPrefIndex.ItemRef.SerialNo"

Notes

Number of digits in Company Prefix and Item Reference must total 13.

RFID Tag Formats: SSCC-96

Field	Number of Digits	Range
Filter	Not applicable	0-7
Company Prefix (CoPref)	6-12	Not applicable
Serial Reference (SerialRef)	11-5	Not applicable

URI Representation

"urn:epc:tag:sscc-96:Filter.CoPref.SerialRef"

Notes

Number of digits in Company Prefix and Serial Reference must total 17.

RFID Tag Formats: USDOD-96

Field	Number of Digits	Range
Filter	Not applicable	0-15
Government Managed Identifier (GMID)	Exactly 5 characters	0-9, SPACE, and A-Z (not I or O)
Serial Number (SerialNo)	Not applicable	0-68 719 476 735

URI Representation

"urn:epc:tag:usdod-64:Filter.GMID.SerialNo"

About Tag Memory Allocation Standards

Fingerprint supports the following tag memory standards. Click the standard name to see more information.

- EPCglobal Class 1 Gen 2
- ISO 18000-6B
- EPC UCode 1.19
- EPC Class 1

Because these tags have different memory structures, you may need to use different input parameters for the TAGFIELD and TAGFORMAT statement.

Class 1 (EPC Class 1 Version 1) Tag Memory Allocation

EPC 64 Tag Memory Allocation

Byte	Row	Pointer value (bit)	@ID	EPC 96
0	0	0		Tag CRC - no Read nor Write
1				
2	1	16	0	Tag ID most significant bytes (EPC data) - R/W
3			1	
4	2	32	2	Tag ID (EPC data) - R/W
5			3	
6	3	48	4	Tag ID (EPC data)- R/W
7			5	
8	4	64	6	Tag ID (EPC data)- R/W
9			7	
10	5	80	8	Tag Id (EPC data) - R/W
11			9	
12	6	96	10	Tag ID least significant bytes (EPC data) - R/W
13			11	
14	7	112	12	Kill Passcode - R/W
15				Lock bits --/-

EPC 96 Tag Memory Allocation

Byte	Row	Pointer value (bit)	@ID	EPC 96
0	0	0		Tag CRC - no Read nor Write
1				
2	1	16	0	Tag ID most significant bytes (EPC data) - R/W
3			1	
4	2	32	2	Tag ID (EPC data) - R/W
5			3	
6	3	48	4	Tag ID (EPC data)- R/W
7			5	
8	4	64	6	Tag ID (EPC data)- R/W
9			7	
10	5	80	8	Tag Id (EPC data) - R/W
11			9	
12	6	96	10	Tag ID least significant bytes (EPC data) - R/W
13			11	
14	7	112	12	Kill Passcode - R/W
15				Lock bits --/-

Notes

For Class 1 tags, EPC data is stored in the @ID segment, bytes 0-7 for 64-bit and bytes 1-11 for 96-bit tags. The values for the parameters <nexp2> and <nexp3> for the TAGFIELD command do not need to be entered when writing EPC data, as these are assigned automatically.

The Kill passcode can be addressed as byte 12 in EPC-96 (byte 8 in EPC-64). It must be addressed independently with a TAGFIELD,"@ID",12,1 command.

The Class 1 standard defines a tag CRC (cyclic redundancy check) for tag ID bits, and if incorrect, the tag will not be detected by a TAGREAD operation.

Erasing all tag information is done by the following command:

```
TAGFIELD "@ID",0,0
TAGFORMAT "HEX"
TAGWRITE ""
```

EPCglobal Class 1 Gen 2 Tag Memory Allocation

RESERVED Field

@Reserved	Content	Read/Write
0-3	Kill Password	Yes
4-7	Access Password	Yes
8-n	Optional	Yes

EPC Field

@EPC	Content	Read/Write
0-1	CRC-16	Read-only
2-3	Protocol Control Bits	Yes
4-15	EPC data	Yes
16-n2	Optional data	Yes

TID Field

@TID	Content	Read/Write
0	0xE2	Read-only
1	Tag mask designer identifier + vendor-defined tag model number	Read-only
2-n3	Optional	Read-only

USER Field

@USER	Content	Read/Write
0-n4	Optional User Data	Yes

Notes

The @RESERVED field contains kill and access passwords. These are not always implemented, and the tag acts as if they were zero-valued passwords.

The @EPC segment is specifically designed to hold EPC values, and is at least 96 bits long. When writing EPC information to the @EPC segment, values for the <*nexp2*> and <*nexp3*> values in the TAGFIELD command need not be defined.

For EPCglobal applications, the @TID segment contains special tag and vendor-specific data. This segment may contain an identifier for ISO/IEC 15963, but this is not an EPCglobal application.

The @USER bank allows for user-specific data storage. The organization of user memory for EPCglobal applications is vendor-defined.

ISO 18000-6B (UCode HSL) Tag Memory Allocation

Byte		@ID	@DATA	@ALL	Read/Write	Comment
0	ID	0		0	Read-only	Unique ID
1		1		1		
2		2		2		
3		3		3		
4		4		4		
5		5		5		
6		6		6		
7		7		7		

Byte		@ID	@DATA	@ALL	Read/Write	Comment
8	Data		0	8	Read-only	Tag Manufacturer
9			1	9		
10			2	10	Read-only	Tag Hardware Type
11			3	11		
12			4	12	Read/Write	Tag Memory Layout
13			5	13		
14			6	14		
15			7	15		
16			8	16		
17			9	17		
18			10	18	Read/Write	User Data: Definition and format determined by Tag Memory Layout
19			11	19		
20			12	20		
	. . . .			. . . .	. . . .	
223			215	223		

The actual byte length of the tag may vary.

When writing to an ISO 18000-6B tag, EPC data must be written to the @DATA field, and valid start byte and field length parameters must be entered in the TAGFIELD instruction.

Example

```
10 FILTER$ = "3"
20 PREFIX$ = "0614141"
30 ITEM$ = "100734"
40 SERIAL$ = "2"
50 TAGFIELD "@DATA",10,12
60 TAGFORMAT "SGTIN-96"
70 TAGWRITE FILTER$, PREFIX$, ITEM$, SERIAL$
```

UCode EPC 1.19 Tag Memory Allocation

Byte		@ID	@DATA	Read/Write	Comment
0	ID	0		Read only	EPCGlobal (HEX EF04)
1		1			
2		2		Read/Write	EPC Data
3		3			
4		4			
5		5			
6		6			
7		7			
8	Data		0	Read/Write	User Data
9			1		
10			2		
11			3		
12			4		
13			5		
14			6		
15			7		

Byte		@ID	@DATA	Read/Write	Comment
16	ID	8		Read/Write	EPC Data
17		9			
18		10			
19		11			
20		12			
21		13			
22		14			
23		15			
24	Data		8	Read/Write	User Data
25			9		
26			10		
... ...			... ...		
47			31		

Notes

When writing EPC data to the @ID segment of a UCode EPC 1.19 tag, Fingerprint ignores input parameters (and other defaults) for the TAGFIELD command, and automatically starts writing at byte 2 of the @ID segment. When writing to the @DATA segment, valid start byte and field length parameters must be entered in the TAGFIELD command.

Other EPC Tag Input Methods

Although Honeywell recommends that you use the syntax described for each format to write to an EPC tag, it is possible to write the full length of the 64-bit or 96-bit chip manually in hex format. You can use the tag formats "EPC-HEX64" and "EPC-HEX96" to manually write to the tag.

EPC-HEX64 Format

Number of bytes: 8
Allowed values: 0 to 9, A to F in hex format

Example:

10 TAGFIELD "@ID"

20 TAGFORMAT "EPC-HEX64"

30 TAGWRITE "1122334455667788"

EPC-HEX96 Format

Number of bytes: 12
Allowed values: 0 to 9, A to F in hex format

Example:

10 TAGFIELD "@ID"

20 TAGFORMAT "EPC-HEX96"

30 TAGWRITE "11223344556677889900AABB"

EPC Tag Writing Example

The following example shows how to include EAN.UCC encoding schemes (as currently used in printed bar codes) in writing to an RFID tag. This example uses a Serialized Global Trade Item Number (SGTIN) formed by the GTIN 10614141007346 and serial number 2. The GTIN is formed by an indicator, company prefix and item reference.

GTIN Indicator: 1
Company Prefix: 0614141
Item reference: 00734
Check digit: 6
Serial Number: 2

This would normally be encoded as an EAN128 bar code:

Callout	Description	Value in Example
A	Application identifiers	(01)
B	GTIN indicator	1
C	Company prefix index	0614141
D	Item reference	00734
E	Check digit	6
F	Application identifier	(21)
G	Serial number	2

For the correct syntax, the application identifiers (AI) are dropped because this information is included in the header. The check digit is dropped. The GTIN indicator is repositioned at the leftmost position of the item reference, to form a new item reference. A filter value of 3 is chosen for this example (signifying Standard Trade Item Group, though filter definitions are non-normative at this time). This gives us the necessary values to program an SGTIN-96 tag, in this order:

Filter: 3
Company Prefix Index: 0614141
Item reference: 100734
Serial number: 2

We could use the following Fingerprint programs to program an SGTIN-96 tag with this information, The first example assumes a Gen 2 tag, and the second an ISO 18000-6B chip:

Example 1: EPCglobal Class 1 Gen 2 Tag

```
10 FILTER$ = "3"
20 PREFIX$ = "0614141"
30 ITEM$ = "100734"
40 SERIAL$ = "2"
50 TAGFIELD "@EPC"
60 TAGFORMAT "SGTIN-96"
70 TAGWRITE FILTER$, PREFIX$, ITEM$, SERIAL$
```

Example 2: ISO 18000-6B Tag

```
10 TAGFIELD "@DATA",10,12
20 TAGFORMAT "EPC-URN"
30 TAGWRITE "urn:epc:tag:sgtin-96:3.0614141.100734.2"
```

CHAPTER

6

REFERENCE INFORMATION

This section includes useful information used with a variety of Fingerprint commands:

- Fonts
- Character Sets
- Printer Keypad Layouts
- Illustrations of printer keypad layouts, with ID numbers and ASCII keypress values. Use with the KEYBMAP$, KEY ON/OFF, and ON KEY GOSUB commands.
- Error Codes
- Fingerprint Syntax Conventions
- Fingerprint Variable and Line Label Names

Fonts

Fingerprint includes a variety of commands you can use to manage fonts and font printing.

This section includes information on Fingerprint font support including:

- Default fonts
- How to install fonts
- How to create font aliases
- Complex scripts

Default Fonts

Fonts included by default with Honeywell printers include:

- Andale Mono (Regular and Bold)
- CG Times (Regular and Bold)
- Century Schoolbook Roman
- Letter Gothic
- OCR-A
- OCR-B
- Univers (Regular, Bold, Condensed Bold, and Extra Condensed)

Bitmap fonts included by default with Honeywell printers include:

- IPLFONT0
- IPLFONT1
- IPLFONT2
- IPLFONT7

You can purchase additional fonts from Monotype. For more information, see Installing Fonts.

Legacy Font Information

Many fonts included with legacy Honeywell printers are now automatically mapped to new fonts as font aliases:

Legacy Font	Alias
Swiss 721 BT	Univers

Legacy Font	Alias
DingDings SWA	
OCR-A BT	OCR-A
OCR-B 10 Pitch BT	OCR-B
Prestige 12 Pitch Bold BT*	
Century Schoolbook BT	Century Schoolbook
Dutch 801 Roman BT	CG Times
Dutch 801 Bold BT	CG Times Bold
Letter Gothic 12 Pitch BT	Letter Gothic
Monospace 821 BT	Andale Mono
Monospace 821 Bold BT	Andale Mono Bold
Swiss 721 Bold BT	Univers Bold
Swiss 721 Bold Condensed BT	Univers Condensed Bold
Extra Condensed BT	Univers Extra Condensed
Futura Light BT*	
UPC11.1, UPC11.2, UPC21.1, UPC21.2, UPC31.1, UPC31.2, UPC51.1, UPC51.2, UPC71.1, UPC71.2	OCR-B
OB035RM1, OB035RM1.1, OB035RM1.2	OCR-B
SW030RSN, SW030RSN.1, SW030RSN.2, SW050RSN, SW050RSN.1, SW050RSN.2	Univers
SW020BSN, SW020BSN.1, SW020BSN.2, SW060BSN, SW060BSN.1, SW060BSN.2, SW080BSN, SW080BSN.1, SW080BSN.2, SW120BSN, SW120BSN.1, SW120BSN.2	Univers Bold
MS030RMN, MS030RMN.1, MS030RMN.2, MS050RMN, MS050RMN.1, MS050RMN.2	Andale Mono
MS060BMN, MS060BMN.1, MS060BMN.2	Andale Mono Bold

* Prestige and Futura Light can be purchased from Monotype.

For more information on font mapping, see Font Aliases.

How to Install Fonts

You can install additional fonts using these methods:

- PrintSet
- Printer web page
- USB storage device
- FTP
- SmartSystems

User fonts can be stored in /home/user/fonts/, or its subdirectories. For more information on the font installation process, see your printer user guide.

Getting More Fonts

Fingerprint can use any TrueType® or True-Type-based OpenType® font.

You can also purchase additional fonts from Monotype at www.fonts.com that have been tested for use with Fingerprint applications:

Monotype Font	Language/Locale	Supported NASC Values
Helvetica World	Eastern Europe	CP 1250, UTF-8
M Sung PRC	Simplified Chinese	CP 936, UTF-8
M Sung HK	Traditional Chinese	CP 950, UTF-8, BIG5HKSCS
HY Gothic Medium	Korean	CP 949, UTF-8
TypeBank Mincho Light	Japanese	CP 932, UTF-8, Shift-JIS
Narkis	Hebrew, Arabic	CP 1255, CP 1256
Helvetica World	Mediterranean	CP 1257
WorldType Collection J	Japanese	UTF-8
WorldType Collection S	Simplified Chinese	UTF-8
WorldType Collection T	Traditional Chinese	UTF-8
WorldType Collection K	Korean	UTF-8

How to Create Font Aliases

Purpose

Font aliasing provides flexibility when moving from a legacy or competitive printer installation to aHoneywell printer installation in two ways:

- It removes the need to change font names in the data stream to match those resident in the printer.
- It enables you to adjust the size, width, and height of a font to fine-tune the text fit for any need.

An alias provides a link from the font name in a data stream to a currently installed font. By using font aliasing, you can create a font alias that points an Arial font to use Univers instead.

There are three ways to create aliases:

- Use the RUN command alias to create a single alias.
- Use the RUN command alias to create multiple aliases from an alias batch file.
- Manually copy an alias batch file to the printer and then restart the printer.

Syntax

RUN "alias [FONT] *alias_namereference_font*[*font_size*] [slant] [width]"

Parameters

FONT

(Optional)

alias_name

The name of the font alias. Spaces in the alias name must be escaped, or preceded by a backslash (\).

reference_font

The filename of the font to use in place of the alias. Spaces in the font filename must be escaped, or preceded by a backslash (\).

font_size

(Optional) The size of the font.

slant

(Optional) The slant of the font.

width

(Optional) The width of the font.

If values are not supplied in the alias for optional parameters, they may be specified in the data stream and will be used. If values are included in the alias and also specified in the data stream, the values specified in the data stream will be ignored.

Aliases are available for immediate use.

Example

This command creates an alias for Courier New to Andale Mono, at 10 point, 90 slant, and 94 width:

RUN "alias FONT Courier\ New Andale\ Mono 10 90 94"

Create Multiple Aliases from a Batch File

To create multiple font aliases at a time, use a RUN command to specify a text file:

RUN "alias [FONT] *batch.txt*"

This alias text file must have the following structure:

"*alias_1_name*", "*reference_font*", [*size*], [*slant*], [*width*]
"*alias_2_name*", "*reference_font*", [*size*], [*slant*], [*width*]

Example:

"FONT0101.0", "Univers", 10, 0, 60
"FONT0101.1", "Univers", 12, 0, 60

Copy an Alias Batch File to the Printer

You can copy a file named .FONTALIAS (with aliases defined using the batch alias structure) to the /home/user/fonts folder on your printer and the aliases defined within the file will be created. Here are the main differences between using the .FONTALIAS method to create font aliases and using the RUN command:

- The printer automatically creates <AliasName>.REF files in the /home/user/fonts folder for each alias in .FONTALIAS.
- The file .FONTALIAS will no appear in the /home/user/fonts folder in a FILES lising or if you view the folder with an FTP client.
- You must restart the printer before you can use aliases created using this method.

Complex Scripts

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.

Script types currently supported by Fingerprint include:

- Arabic
- Bopomofo (Taiwan)
- Cyrillic
- Devanagari (Hindi)
- Greek
- Hebrew
- Hiragana (Japanese)
- Johab (Korean)
- Katakana (Japanese)
- Kanji (Japanese)
- Simplified Chinese

- Tajik
- Thai
- Traditional Chinese
- Uhangul (Korean)
- Wansung (Korean)

Character Sets

This section includes the character sets supported by Fingerprint and describes using the UTF-8 character set.

The character sets in this manual are listed as follows

- numerically by ANSI or NASC number

Character Sets Listed by ANSI or NASC Names

Roman 8 Character Set NASC 1

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

French Character Set NASC 33

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	°	ç	§	^	_	µ	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	é	ù	è	¨		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Spanish Character Set NASC 34

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	¡	Ñ	¿	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	°	ñ	ç	~		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	*ƒ*	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	*µ*	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Italian Character Set NASC 39

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	°	ç	é	^	_	ù	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	à	ò	è	ì		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

English (UK) Character Set NASC 44

	0	1	2	3	4	5	6	7	8	9
30				!	"	£	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	¯		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Swedish Character Set NASC 46

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	¤	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	É	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Ä	Ö	Å	Ü	_	é	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	ä	ö	å	ü		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Norwegian Character Set NASC 47

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Æ	Ø	Å	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	æ	ø	å	¯		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

German Character Set NASC 49

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Ä	Ö	Ü	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	ä	ö	ü	ß		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Japanese Latin Character Set NASC 81

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	¥	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

Portuguese Character Set NASC 351

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	§	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	Ã	Ç	Õ	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	ã	ç	õ	°		€	□
130	□	□	□	□	□	□	□	□	□	□
140	□	□	□	□	□	□	□	□	□	□
150	□	□	□	□	□	□	□	□	□	□
160		À	Â	È	Ê	Ë	Î	Ï	´	`
170	^	¨	~	Ù	Û	£	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Ð	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

PCMAP Character Set NASC-1

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	å	ç	ê	ë	è	ï
140	î	ì	Ä	Å	É	æ	Æ	ô	ö	ò
150	û	ù	ÿ	Ö	Ü	¢	£	¥	§	ƒ
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	`
170	^	½	¼	¡	«	»	¯	Ý	ý	°
180	Ç	ç	Ñ	ñ	¡	¿	¤	£	¥	§
190	ƒ	¢	â	ê	ô	û	á	é	ó	ú
200	à	è	ò	ù	ä	ë	ö	ü	Å	î
210	Ø	Æ	å	í	ø	æ	Ä	ì	Ö	Ü
220	É	ï	ß	Ô	Á	Ã	ã	Đ	ð	Í
230	Ì	Ó	Ò	Õ	õ	Š	š	Ú	Ÿ	ÿ
240	Þ	þ	·	µ	¶	¾	—	¼	½	ª
250	º	«	■	»	±	□				

ANSI Character Set NASC-2

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ	□	Ž	□	□	‘	’	“	”	•
150	–	—	˜	™	š	›	œ	□	ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	2	3
180	´	µ	¶	·	¸	1	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

MS-DOS Latin 1 Character Set NASC 850

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	å	ç	ê	ë	è	ï
140	î	ì	Ä	Å	É	æ	Æ	ô	ö	ò
150	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	®
170	¬	½	¼	¡	«	»	▒	▓	▓	|
180	┤	Á	Â	À	©	╣	║	╗	╝	¢
190	¥	┐	└	┴	┬	├	─	┼	ã	Ã
200	╚	╔	╩	╦	╠	═	╬	¤	ð	Ð
210	Ê	Ë	È	ı	Í	Î	Ï	┘	┌	▉
220	▄	¦	Ì	▀	Ó	ß	Ô	Ò	õ	Õ
230	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´
240	-	±	‗	¾	¶	§	÷	¸	°	¨
250	·	¹	³	²	■					

MS-DOS Greek 1 Character Set NASC 851

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	Ά	ç	ê	ë	è	ï
140	î	Έ	Ä	Ή	Ί	□	Ό	ô	ö	Ύ
150	û	ù	Ώ	Ö	Ü	ά	£	έ	ή	ί
160	ï	ΐ	ó	ú	A	B	Γ	Δ	E	Z
170	H	½	Θ	I	«	»	░	▒	▓	|
180	┤	K	Λ	M	N	╣	║	╗	╝	Ξ
190	O	┐	└	┴	┬	├	─	┼	Π	P
200	╚	╔	╩	╦	╠	═	╬	Σ	T	Y
210	Φ	X	Ψ	Ω	α	ß	γ	┘	┌	█
220	▄	δ	ε	■	ζ	η	θ	ι	κ	λ
230	μ	ν	ξ	ο	π	ρ	σ	ς	τ	΄
240	-	±	υ	φ	χ	§	ψ	΅	°	¨
250	ω	ϋ	ΰ	ώ	■					

MS-DOS Latin 2 Character Set NASC 852

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	ů	ć	ç	ł	ë	Ő	ő
140	î	Ź	Ä	Ć	É	Ĺ	Í	ô	ö	Ľ
150	ľ	Ś	ś	Ö	Ü	Ť	ť	Ł	×	č
160	á	í	ó	ú	Ą	ą	Ž	ž	Ę	ę
170	¬	ź	Č	ş	«	»	░	▒	▓	|
180	┤	Á	Â	Ě	Ş	╣	║	╗	╝	Ż
190	ż	┐	└	┴	┬	├	─	┼	Ă	ă
200	╚	╔	╩	╦	╠	═	╬	¤	đ	Đ
210	Ď	Ë	ď	Ň	Í	Î	ě	┘	┌	█
220	▄	Ţ	Ů	▀	Ó	ß	Ô	Ń	ń	ň
230	Š	š	Ŕ	Ú	ŕ	Ű	ý	Ý	ţ	´
240	-	˝	˛	ˇ	˘	§	÷	¸	°	¨
250	˙	ű	Ř	ř	■					

MS-DOS Cyrillic Character Set NASC 855

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		ђ	Ђ
130	ѓ	Ѓ	ё	Ё	є	Є	s	S	i	I
140	ї	Ї	j	J	љ	Љ	њ	Њ	ћ	Ћ
150	ќ	Ќ	ў	Ў	џ	Џ	ю	Ю	ъ	Ъ
160	а	А	б	Б	ц	Ц	д	Д	е	Е
170	ф	Ф	г	Г	«	»	░	▒	▓	|
180	┤	х	Х	и	И	╣	║	╗	╝	й
190	Й	┐	└	┴	┬	├	─	┼	к	К
200	╚	╔	╩	╦	╠	═	╬	¤	л	Л
210	м	М	н	Н	о	О	п	┘	┌	█
220	▄	П	я	▀	Я	р	Р	с	С	т
230	Т	У	у	ж	Ж	в	В	ь	Ь	№
240	-	ы	Ы	з	З	ш	Ш	э	Э	щ
250	Щ	ч	Ч	§	■					

MS-DOS Turkish Character Set NASC 857

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ç	ü
130	é	â	ä	à	å	ç	ê	ë	è	ï
140	î	ı	Ä	Å	É	æ	Æ	ô	ö	ò
150	û	ù	İ	Ö	Ü	ø	£	Ø	Ş	ş
160	á	í	ó	ú	ñ	Ñ	Ğ	ğ	¿	®
170	¬	½	¼	¡	«	»	░	▒	▓	|
180	┤	Á	Â	À	©	╣	║	╗	╝	¢
190	¥	┐	└	┴	┬	├	─	┼	ã	Ã
200	╚	╔	╩	╦	╠	═	╬	¤	º	ª
210	Ê	Ë	È	□	Í	Î	Ï	┘	┌	▉
220	▄	¦	Ì	▀	Ó	ß	Ô	Ò	õ	Õ
230	µ	□	×	Ú	Û	Ù	ì	ÿ	¯	´
240	-	±	□	¾	¶	§	÷	¸	°	¨
250	˙	¹	³	²	■					

Windows Latin 2 Chracter Set NASC 1250

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	□	„	…	†	‡	□	‰	Š	‹
140	Ś	Ť	Ž	Ź	□	‘	’	“	”	•
150	–	—	□	™	š	›	ś	ť	ž	ź
160		ˇ	˘	Ł	¤	Ą	¦	§	¨	©
170	Ş	«	¬	-	®	Ż	°	±	˛	ł
180	´	µ	¶	·	¸	ą	ş	»	Ľ	˝
190	ľ	ż	Ŕ	Á	Â	Ă	Ä	Ĺ	Ć	Ç
200	Č	É	Ę	Ë	Ě	Í	Î	Ď	Đ	Ń
210	Ň	Ó	Ô	Ő	Ö	×	Ř	Ů	Ú	Ű
220	Ü	Ý	Ţ	ß	ŕ	á	â	ă	ä	Í
230	ć	ç	č	é	ę	ë	ě	í	î	ď
240	đ	ń	ň	ó	ô	ö	ö	÷	ř	ů
250	ú	ű	ü	ý	ţ	˙				

Windows Cyrillic Character Set NASC 1251

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		Ђ	Ѓ
130	‚	ѓ	„	…	†	‡	€	‰	Љ	‹
140	Њ	Ќ	Ћ	Џ	ђ	‘	’	“	”	•
150	–	—		™	љ	›	њ	ќ	ћ	џ
160		Ў	ў	Ј	¤	Ґ	¦	§	Ё	©
170	Є	«	¬	-	®	Ї	°	±	І	і
180	ґ	µ	¶	·	ё	№	є	»	ј	Ѕ
190	ѕ	ї	А	Б	В	Г	Д	Е	Ж	З
200	И	Й	К	Л	М	Н	О	П	Р	С
210	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы
220	Ь	Э	Ю	Я	а	б	в	г	д	е
230	ж	з	и	й	к	л	м	н	о	п
240	р	с	т	у	ф	х	ц	ч	ш	щ
250	ъ	ы	ь	э	ю	я				

Windows Latin 1 Character Set NASC 1252

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ		Ž			‘	’	“	”	•
150	–	—	˜	™	š	›	œ		ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	2	3
180	´	µ	¶	·	¸	1	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Windows Greek Character Set NASC 1253

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	ƒ	„	…	†	‡	□	‰	□	‹
140	□	□	□	□	□	‘	’	“	”	•
150	–	—	□	™	□	›	□	□	□	□
160		΅	Ά	£	¤	¥	¦	§	¨	©
170	□	«	¬	-	®	―	°	±	²	³
180	΄	μ	¶	·	Έ	Ή	Ί	»	Ό	½
190	Ύ	Ώ	ΐ	Α	Β	Γ	Δ	Ε	Ζ	Η
200	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π	Ρ
210	□	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	Ϊ	Ϋ
220	ά	έ	ή	ί	ΰ	α	β	γ	δ	ε
230	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
240	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω
250	ϊ	ϋ	ό	ύ	ώ	□				

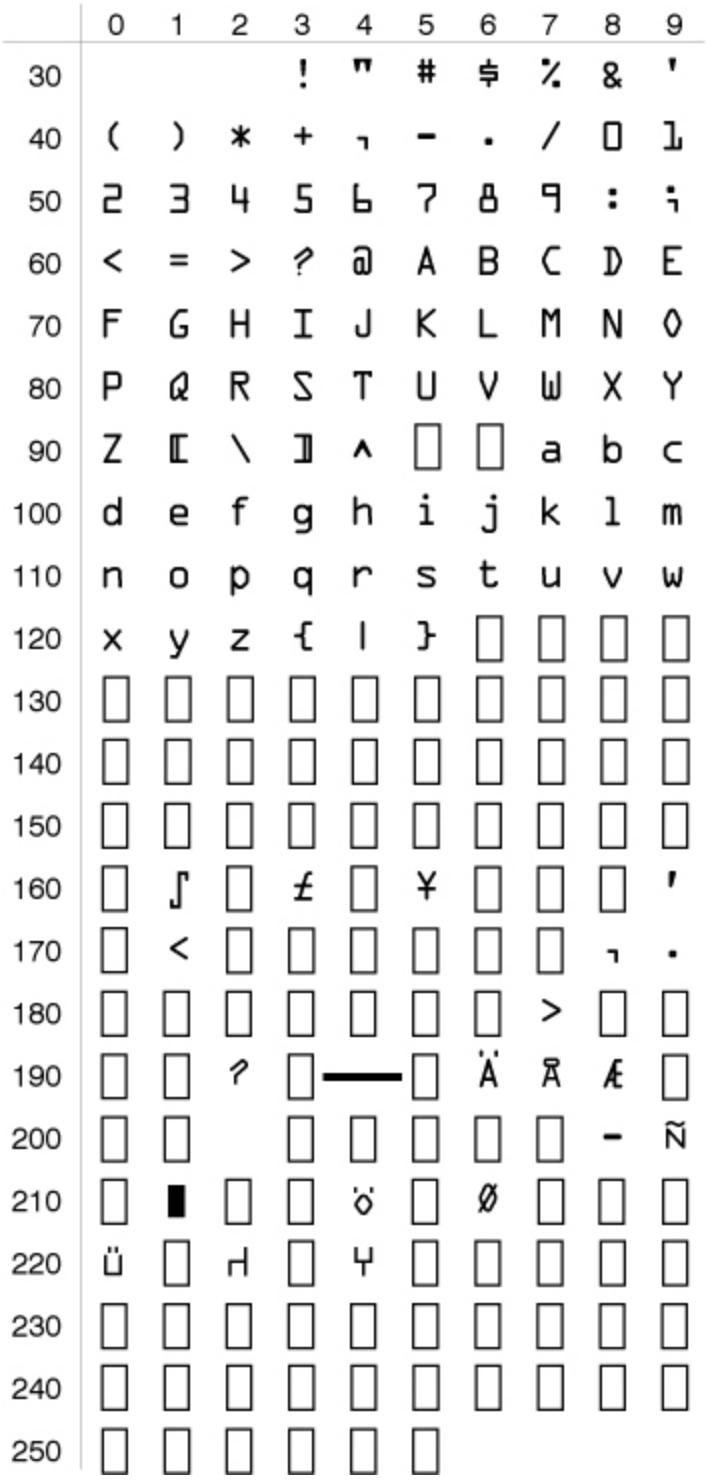
Windows Latin 5 Character Set NASC 1254

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ					‘	’	“	”	•
150	–	—	˜	™	š	›	œ			Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ğ	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	İ	Ş	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ğ	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ı	ş	ÿ				

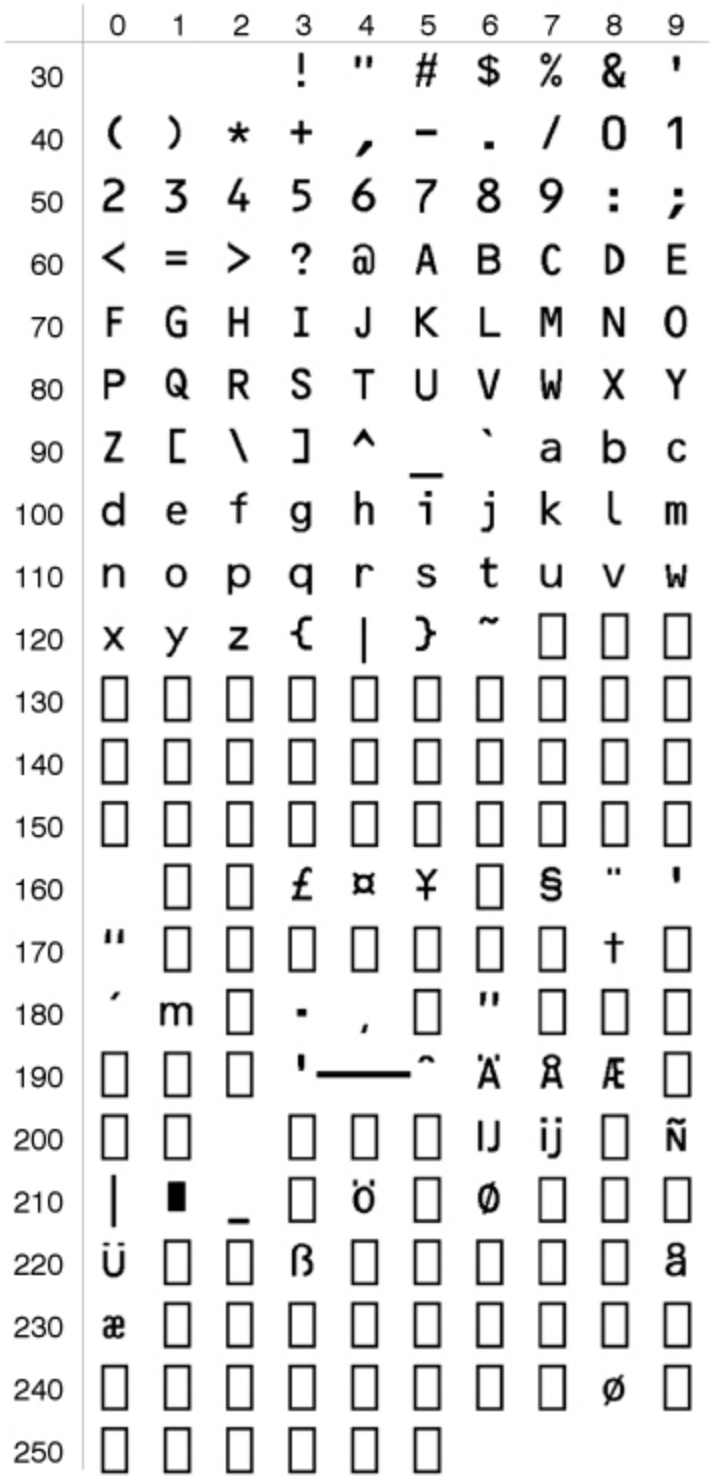
Windows Baltic Rim Character Set NASC 1257

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	□
130	‚	□	„	…	†	‡	□	‰	□	‹
140	□	¨	ˇ	¸	□	‘	’	“	”	•
150	–	—	□	™	□	›	□	¯	˛	□
160		□	¢	£	¤	□	¦	§	Ø	©
170	Ŗ	«	¬	-	®	Æ	°	±	²	³
180	´	µ	¶	·	ø	¹	ŗ	»	¼	½
190	¾	æ	Ą	Į	Ā	Ć	Ä	Å	Ę	Ē
200	Č	É	Ź	Ė	Ģ	Ķ	Ī	Ļ	Š	Ń
210	Ņ	Ó	Ō	Õ	Ö	×	Ų	Ł	Ś	Ū
220	Ü	Ż	Ž	ß	ą	į	ā	ć	ä	å
230	ę	ē	č	é	ź	ė	ģ	ķ	ī	ļ
240	š	ń	ņ	ó	ō	õ	ö	÷	ų	ł
250	ś	ū	ü	ż	ž	˙				

OCR-A Character Set



OCR-B Character Set



NASC Encoding Format

You can create an encoding for use with the NASC command. The format is different for single-byte or double-byte formats. Each encoding file contains a sequence of two Unicode code points, or encoding value pairs.

Single-Bye Encoding Example

```
# Fingerprint NASC 1
 Roman 8
 1

 00 0000
 01 0001
 02 0002
 03 0003
 04 0004
 05 0005
 06 0006
 07 0007
 ...
 F8 00BD
 F9 00AA
 FA 00BA
 FB 00AB
 FC 25A0
 FD 00BB
 FE 00B1
 FF 0000
```

Double-Byte Encoding Example

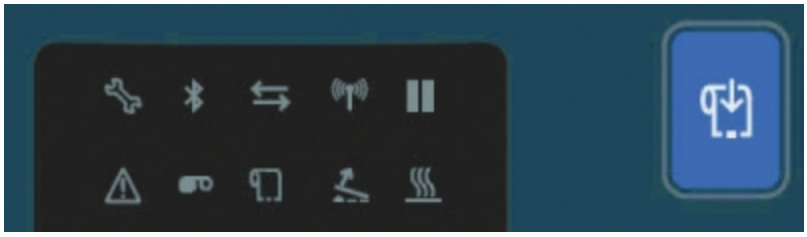
```
# Fingerprint NASC 1361
 Johab (Korean)
 2

 20 0020
 21 0021
 22 0022
 23 0023
 24 0024
 25 0025
 26 0026
 27 0027
 ...
 F9F7 7199
 F9F8 71B9
 F9F9 71BA
 F9FA 72A7
 F9FB 79A7
 F9FC 7A00
```

F9FD 7FB2
F9FE 8A70

Printer Keypad Layouts

Click the printer name in the list below for illustrations that include LED position numbers, ID values, and ASCII values. Use these values with the KEY ON/OFF, KEYBMAP$, or ON KEY GOSUB commands.

PC23d, PC43d, and PC43t Icon Layout



PC23d, PC43d, and PC43t Icon Layout

Icon/Button	Label	LED ID
	Wi-Fi Communications	N/A
	Connectivity	0
	Bluetooth	N/A
	Media	3
	Configuration Error	7
	Ribbon	4
	Printhead Hot	6

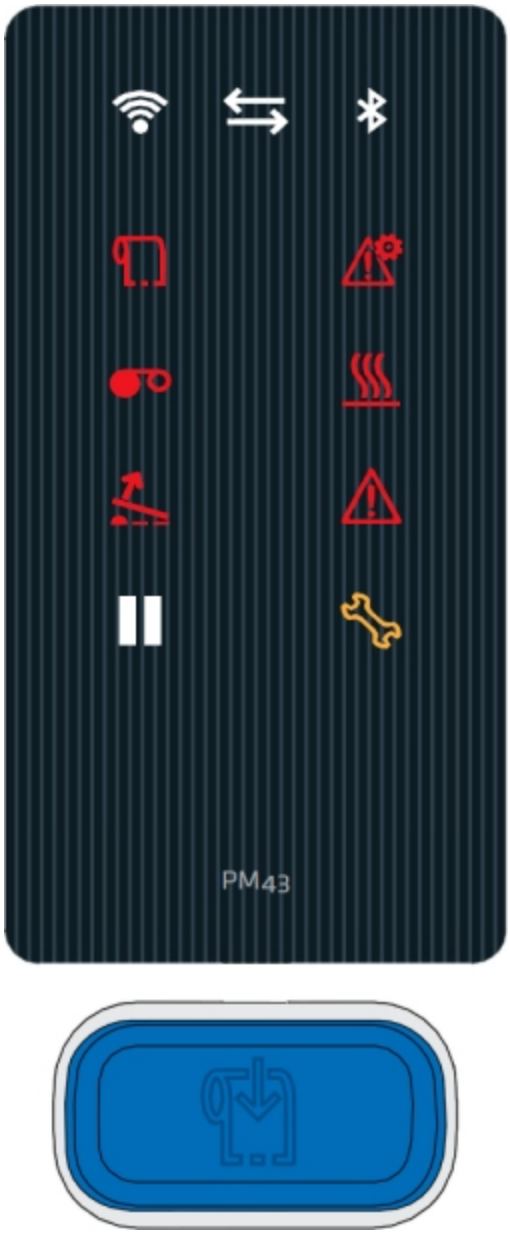
Icon/Button	Label	LED ID
	Printhead Lifted	5
	General Error	1
	Pause	8
	Maintenance	9
	Print/Feed Button	17

PC23d, PC43d, PC43t, PD43, PD43c, and PM42 Keypad Layout

PC23d, PC43d, and PC43t Keypad Layout

Icon	Key Label	Key ID	ASCII value (Shift: +128)
	Menu/Home	10	1
	Up	12	3
	Down	14	5
	Left	11	2
	Right	13	4
	Select	16	13
	Back	20	8
	Print/Feed	17	31

PM23c, PM43, and PM43c Printer Icon Layout



PM23c, PM43, and PM43c Printer Icon Layout

Icon/Button	Label	LED ID
	Wi-Fi Communications	N/A
	Connectivity	0

Icon/Button	Label	LED ID
	Bluetooth	N/A
	Media	3
	Configuration Error	7
	Ribbon	4
	Printhead Hot	6
	Printhead Lifted	5
	General Error	1
	Pause	8
	Maintenance	9
	Print/Feed Button	17

PM23c, PM43, and PM43c Printer Keypad Layout

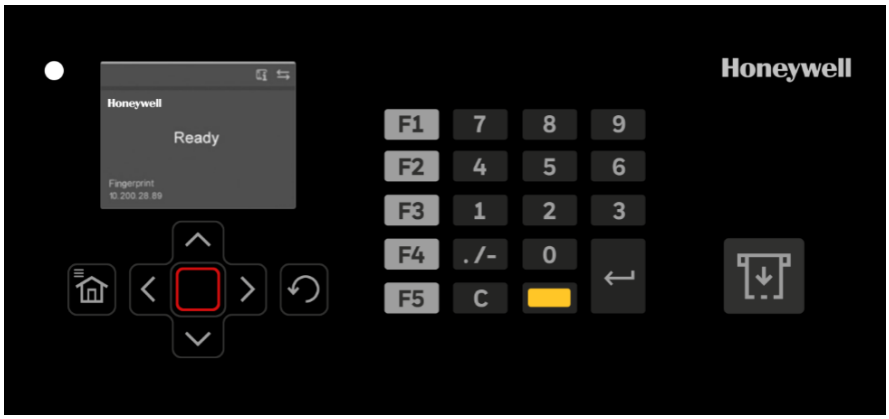
PM43 Printer Keypad Layout

PM23c and PM43c Printer Keypad Layout

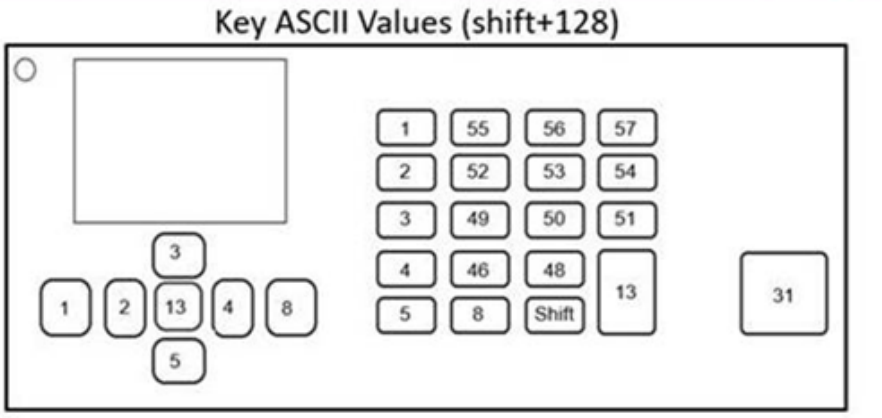
Key	Label	Key ID	ASCII value (Shift:+128)
	Shift	N/A	N/A

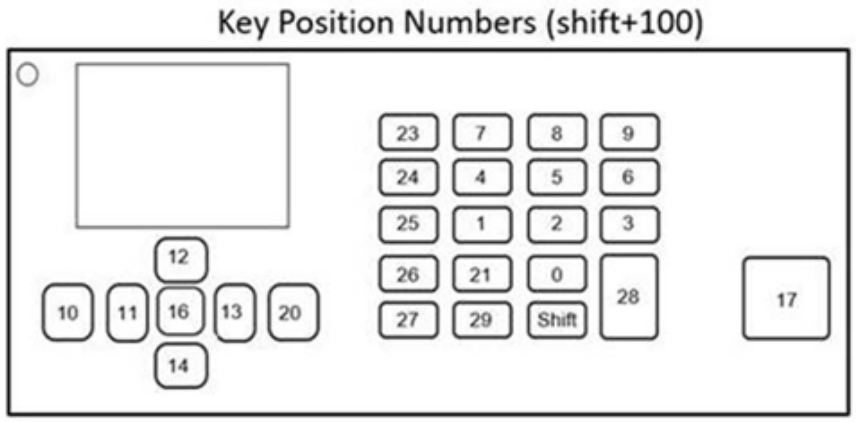
Key	Label	Key ID	ASCII value (Shift:+128)
	F1	10	1
	F2	11	2
	F3	12	3
	F4	13	4
	F5	14	5
	1	1	49
	2	2	50
	3	3	51
	4	4	52
	5	5	53
	6	6	54
	7	7	55
	8	8	56
	9	9	57
	0	0	48
	./-	21	46
	C	20	8
	Enter	16	13
	Print/Feed	17	31

PX4ie and PX6ie Printer Keypad Layout



PX4ie and PX6ie Keyboard Layout





Key/Label	Key Position	Key ID	ASCII Value
Shift	Shift	N/A	N/A
F1	23	10	1

Key/Label	Key Position	Key ID	ASCII Value
F2	24	11	2
F3	25	12	3
F4	26	13	4
F5	27	14	5
1	1	1	49
2	2	2	50
3	3	3	51
4	4	4	52
5	5	5	53
6	6	6	54
7	7	7	55
8	8	8	56
9	9	9	57
0	0	0	48
./-	21	21	46
C	29	20	8
Enter	28	16	13
Print/Feed	17	17	31
Left	11	11	2
Right	13	13	4
Up	12	12	3
Down	14	14	5
Home	10	10	1
Select	16	16	13
Back	20	20	8

PX940 Printer Icon Layout

Icon/Button	Label	LED ID
	Wi-Fi Com-munications	N/A
	Connectivity	0
	Bluetooth	N/A
	Media	3
	Ribbon	4
	Printhead Lifted	5
	Pause	8
	Configuration Error	7
	General Error	1
	Printhead Hot	6
	Maintenance	9
	Home	N/A
	Save	N/A
	Back	N/A
	Menu	N/A
	Printer Information	N/A

RTW LED

Ready-to-Work State	Description
On	The printer is Ready-to-Work
Off	The printer is in a state where it is being used for purposes other than printing or communicating as during a firmware upgrade or while it is booting. It might also mean that there is a severe error that requires a reboot. It might be off if the printer is off.
Blink	A blinking light (50% duty cycle @ 0.75Hz) indicates that the printer is not ready to work and there is a printing or communication error that requires attention.

Error Codes

Click the following link to view a list of error codes:

- Error Codes 1 - 90
- Error Codes 1001-1095
- Error Codes 1101-1370
- Error Codes 1401-1728
- Error Codes 1801-1958

Code	Message or Explanation	Code	Message or Explanation
Error Codes 1 - 90			
0	No error.	46	Store already in progress.
1	Syntax error.	47	Unknown store protocol.
2	Unbalanced parentheses.	48	No store defined.
3	Feature not implemented.	49	NEXT without FOR.
4	Evaluation syntax error.	50	Bad store record header.
5	Unrecognized token.	51	Bad store address.
6	Tokenized line too long.	52	Bad store record.
7	Evaluation stack overflow.	53	Bad store checksum.
8	Error in exectab.	54	Bad store record end.
9	Undefined token.	55	Remove in ROM.
10	Non-executing token.	56	Illegal communication channel.
11	Evaluation stack underflow.	57	Subscript out of range.
12	Type mismatch.	58	Field overflow.
13	Line not found.	59	Bad record number.
14	Division with zero.	60	Too many strings.
15	Font not found.	61	Error in setup file.
16	Bar code device not found.	62	File is list protected.
17	Bar code type not implemented.	63	ENTER function.
18	Disk full.	64	FOR without NEXT.
19	Error in file name.	65	Evaluation overflow.
20	Input line too long.	66	Bad optimizing type.

Code	Message or Explanation	Code	Message or Explanation
21	Error stack overflow.	67	Error from communication channel.
22	RESUME without error.	68	Unknown execution identity.
23	Image not found.	69	Not allowed in immediate mode.
24	Overflow in temporary string buffer.	70	Line label not found.
25	Wrong number of parameters.	71	Line label already defined.
26	Parameter too large.	72	IF without ENDIF.
27	Parameter too small.	73	ENDIF without IF.
28	RETURN without GOSUB.	74	ELSE without ENDIF.
29	Error in startup file.	75	ELSE without IF.
30	Assign to a read-only variable.	76	WHILE without WEND.
31	Illegal file number.	77	WEND without WHILE.
32	File is already open.	78	Not allowed in execution mode.
33	Too many files open.	79	Not allowed in a layout.
34	File is not open.	80	Download timeout.
35	Guru mode on.	81	Exit to system.
36	Guru mode off.	82	Invalid cont environment.
37	Cutter device not found.	83	ETX Timeout.
38	User break.	84	Application terminated by system.
39	Illegal line number.	85	XML format not found.
40	Run statement in program.	86	Comset conflict.
41	Parameter out of range.	87	Recursive call error.
42	Illegal bar code ratio.	88	Operation not allowed. Applicator port disabled.
43	Memory overflow.	89	Operation not allowed. Applicator port enabled.
44	File is write protected.	90	Operation not allowed in Direct Protocol.
45	Unknown store option		

Error Codes 1001 - 1095

Code	Message or Explanation	Code	Message or Explanation

Code	Message or Explanation	Code	Message or Explanation
1001	Not implemented.	1048	Too many processes.
1002	Memory too small.	1049	Invalid process.
1003	Field out of label.	1050	Invalid child.
1004	Wrong font to chosen direction.	1051	Dot resistance measure out of limits.
1005	Out of media.	1052	Error in printhead.
1006	No field to print.	1053	Unable to complete a dot measurement.
1007	Test feed failed.	1054	Error writing to device.
1008	Test feed failed.	1055	Error reading from device.
1009	Invalid parameter.	1056	O_BIT open error.
1010	Hardware error.	1057	File exists.
1011	I/O error.	1058	Ribbon installed.
1012	Too many files opened.	1059	Cutter not responding.
1013	Device not found.	1060	Ribbon save motor did not start/stop.
1014	File not found.	1061	Wrong type of media.
1015	File is read-only.	1062	Not allowed.
1016	Illegal argument.	1063	E2BIG
1017	Results are too large.	1064	Device or resource is busy.
1018	Bad file descriptor.	1065	Resource deadlock would occur.
1019	Invalid font.	1066	Bad address.
1020	Invalid image.	1067	Is a directory.
1021	Too large argument for MAG.	1068	Too many links.
1022	Printhead lifted.	1069	Exec format error.
1023	Incomplete label.	1070	No record locks available.
1024	File too large.	1071	No space left on device.
1025	Permission denied.	1072	Out of streams resources.
1026	Label pending.	1073	Directory not empty.
1027	Out of ribbon.	1074	Not a typewriter.
1028	Paper type is not selected.	1075	No such device or address.
1029	Printhead voltage too high.	1076	Operation not permitted.

Code	Message or Explanation	Code	Message or Explanation
1030	Character is missing in chosen font.	1077	Broken pipe.
1031	Next label not found.	1078	Illegal seek.
1032	File name too long.	1079	Cross-device link.
1033	Too many files are open.	1080	Address family not supported.
1034	Not a directory.	1081	Timer expired.
1035	File pointer is not inside the file.	1082	Unsupported protocol type.
1036	Subscript out of range.	1083	Ribbon low.
1037	No acknowledge received within specified time.	1084	Media low.
1038	Communication checksum error.	1085	Connection timed out.
1039	Not mounted.	1086	Secret not found.
1040	Unknown file operating system.	1087	Paper jam.
1041	Error in file system structure.	1088	Printhead too hot.
1042	Internal error in mcs.	1089	Error in XML attribute.
1043	Timer table full.	1091	XML unbalanced tag-close.
1044	Low battery in memory card.	1092	XML empty tag-definition.
1045	Media was removed.	1093	XML expecting more data.
1046	Memory checksum error.	1094	XML data overflow.
1047	Interrupted system call.	1095	Front arm lifted
Error Codes 1101 - 1370			

Code	Message or Explanation	Code	Message or Explanation
1101	Illegal character in bar code.	1221	Map table was not found.
1102	Illegal bar code font.	1222	Double byte map table is missing.
1103	Too many characters in bar code.	1223	Single byte map table is missing.
1104	Bar code too large.	1224	Character map function is missing.
1105	Bar code parameter error.	1225	Double byte font is not selected.

Code	Message or Explanation	Code	Message or Explanation
1106	Wrong number of characters.	1301	Index is out of collection bounds.
1107	Illegal bar code size.	1302	Collection could not be expanded.
1108	Number or rows out of range.	1303	Parameter is not a collection.
1109	Number of columns out of range.	1304	Item is not a member of the collection.
1110	No separator is entered.	1305	Missing or faulty compare function.
1111	Barcode parameter out of range.	1306	Item is already a member of the collection.
1112	Invalid or not supported ECI function.	1307	Over-current warning.
1201	Insufficient font data loaded.	1320	RFID not installed.
1202	Transformation matrix out of range.	1321	No RFID tag found.
1203	Font format error.	1322	Access outside tag memory.
1204	Requested specs not compatible with output module.	1323	Tag format error.
1205	Intelligent transformation requested but not supported.	1324	RFID inactive.
1206	Unsupported output mode requested.	1325	Not supported by tag type.
1207	Extended font loaded but only compact fonts supported.	1326	RFID module too hot.
1208	Font specs not set prior to use of font.	1327	RFID duty cycle exceeded.
1209	Track kerning data not available.	1328	RFID lock error.
1210	Pair kerning data not available.	1329	RFID access error.
1211	Other Speedo error.	1340	Applicator error 1.
1212	Bitmap or outline device not specified.	1341	Applicator error 2.
1213	Speedo error six.	1342	Applicator error 3.
1214	Squeezing or clipping not supported.	1343	RTW external error.
1215	Character data not available.	1344	Dataready not enabled.
1216	Unknown font.	1360	Battery empty.
1217	Font format is not supported.	1361	Battery charging error.
1218	Correct mapping table is not found.	1362	Battery low.

Code	Message or Explanation	Code	Message or Explanation
1219	Font is in the wrong direction.	1370	Motor too hot.
1220	Error in external map table.		
Error Codes 1401 - 1730			
Code	Message or Explanation	Code	Message or Explanation
1401	crpap: Illegal print speed.	1710	Power supply Generic Error.
1402	crpap: Illegal base value.	1711	Power supply Pending.
1403	crpap: Illegal profile number.	1712	Power Status OK.
1404	crpap: Illegal regulation factor.	1713	Power supply power fail.
1405	crpap: Syntax error in parameter file.	1714	Power supply over volt V24.
1406	crpap: Illegal print head resolution.	1715	Power supply under volt V24.
1407	crpap: Illegal print head version.	1716	Power supply over volt VSTM.
1601	Referenced Font Not Found.	1717	Power supply under volt VSTM.
1602	Error in Wand-Device.	1718	Power supply too hot.
1603	Error in Slave Processor.	1719	Power supply error.
1604	Print Shift Error.	1721	Verification Failed
1605	No Hardware Lock.	1722	Replace Verifier Head
1606	Testfeed not done.	1723	Clean Verifier Head
1608	Access Denied.	1724	Verifier Calibration
1609	Specified Feed Length Exceeded.	1725	Verifier hardware Error
1610	Illegal Character Map File.	1725	Verifier Calibration Expired
1611	Compression failed.	1727	Exceed Maximun
1612	Decompression failed.	1728	Unsupported Barcode
1613	Font alias could not be created.	1729	License Expiring Soon
1701	Cutter jammed.	1730	License expired
Error Codes 1801 - 1958			

Code	Message or Explanation	Code	Message or Explanation
1801	Socket operation on non-socket.	1919	Authentication failed.
1802	Destination address required.	1920	Authenticating.
1803	Protocol wrong type for socket.	1921	Printer in maintenance mode.

Code	Message or Explanation	Code	Message or Explanation
1804	Protocol not available.	1922	Printer in maintenance mode.
1805	Protocol not supported.	1923	Info mode entered.
1806	Operation not supported on socket.	1924	Pairing.
1807	Protocol family not supported.	1925	Pairing failed.
1808	Address already in use.	1926	Permission denied.
1809	Can't assign requested address.	1927	Connection timeout.
1810	Socket is not connected.	1928	Print job complete.
1811	No buffer space available.	1929	Odometer (user resettable).
1812	Software caused connection abort.	1930	Odometer (current printhead).
1813	Connection reset by peer.	1931	Odometer (printer).
1814	Network dropped connection on reset.	1932	Replace printhead.
1815	Network is down.	1933	Clean printhead.
1816	Too many references: can't splice.	1934	Faulty dot.
1817	Operation already in progress.	1935	Printing.
1818	Operation now in progress.	1936	USB flash drive connected.
1819	Host is down.	1937	USB flash drive disconnected.
1820	No route to host.	1938	USB peripheral connected.
1821	Disc quota exceeded.	1939	USB host disconnected.
1833	Connection refused.	1940	Bluetooth device connected.
1838	Duplicate IP detected.	1941	Bluetooth device disconnected.
1901	No system light.	1942	Scanning.
1902	Printer is rebooting.	1943	Associating.
1903	Ready.	1944	Disconnected.
1904	Printer in pause mode.	1945	Completed.
1905	Printer in menu mode.	1946	Error in upgrading.
1906	Upgrading firmware.	1947	Printer in sleep mode.
1907	Application load error.	1948	Serial hardware connected.
1908	IP not acquired.	1949	Serial hardware disconnected.
1909	Network link error.	1950	Parallel hardware connected.

Code	Message or Explanation	Code	Message or Explanation
1910	Printhead not detected.	1951	Parallel hardware dis-connected.
1911	Application break.	1952	Parallel hardware connected
1912	Shutting down printer.	1953	Parallel hardware dis-connected
1913	Press feed not done.	1954	Duplicate IP address
1914	Ready-to-Work blink by application.	1955	Wireless Duplicate IP address
1915	Ready-to-Work off by application.	1956	Memory full
1916	Started.	1957	Memory Low
1917	Press a key to continue.	1958	USB flash drive mounted
1918	Label not taken.		

Fingerprint Syntax Conventions

In the command syntax, certain punctuation marks are used to indicate various types of data. Do not include punctuation marks unless noted for that command.

Convention	Description
[]	Entries within brackets are optional.
keyword1 | *keyword2*	Indicates "either-or." Choose only one of the keywords.
<>	Grouping of data.
.....	Repetition of the preceding data type.
"	Quotation mark (ASCII 34 decimal).
↵	Carriage return or linefeed.
<*scon*>	String constant.
<*ncon*>	Numeric constant.
<sexp>	String expression.
<*nexp*>	Numeric expression.
<*svar*>	String variable.
<*nvar*>	Numeric variable.
<*stmt*>	Statement.
<*linelabel*>	Line label.

Fingerprint Variable and Line Label Names

The naming rules for Fingerprint variables and line labels are more restrictive than some other languages:

- Fingerprint variables and line labels must end with either a dollar sign ($) for string variables, or a percent sign (%) for integer variables.
- Fingerprint variables and line labels may not start with Fingerprint keywords, including abbreviations of Fingerprint commands.

Honeywell recommends that you start Fingerprint variables and line labels with the letter "q". No Fingerprint commands or keywords will ever begin with this letter.

Invalid examples

PrintTest$ = "This is a test"
The variable begins with "Print", a Fingerprint command name.

pDispName$ = "Hello World"
The variable starts with a command name abbreviation, "pD" (PD/PRDIAGONAL).

Valid example

qPrintTest$ = "This is a test"
While the variable includes a command name, there are no commands that begin with "q" or "test".

Inline Text Formatting

This functionality allows the user to adjust formatting on-the-fly inside the data stream, using inline commands.

Preparing for Inline Formatting

To enable inline text formatting, specify start and stop characters using two SYSVARs. Each character is the decimal representation of the byte code (0-255).

SYSVAR	Description
SYSVAR(84)	Sets the start character.
SYSVAR(85)	Sets the end character.

If either character is set to -1, inline formatting is disabled.

Using Inline Formatting

Text modifiers cause or discontinue either bold or italic attributes to be applied to following text, according to the following syntax:

<*sexp1*>[/]b|i<*sexp2*>

where:

Parameter	Description
<sexp1>	The character previously assigned to SYSVAR(84).
[/]	Optional forward slash character (“/”) turns off an active modifier, following text to the immediately preceding rendering. Turning off a non-existent modifier is ignored.
b|i	b: bold-face following text i: italicize following text Only one modifier (b or i) may be specified between a pair of <sexp1> and <sexp2>, and the modifier must be lower-case.
<sexp2>	The character previously assigned to SYSVAR(85).

Behavior

Modifiers can appear anywhere in the text string, and may be nested in either order to create bold and italicized text.

Once a modifier is specified, all following text renders according to the modifier until one of the following occurs:

1. The end of the text string is encountered (i.e. a modifier only has effect within the current DP command).

2. A second modifier is specified (in which case all following text is modified with all modifiers currently in effect).
3. The modifier is turned off.

Error Handling

Inline formatting commands not following the syntax are ignored and rendered as normal text. An example of invalid syntax is using the space character inside the formatting command.

Supported Fonts

Boldness can be applied to fonts which have a corresponding bold version named "<regular font name> Bold" (for example the resident fonts: Andale Mono, CG Times, Univers).

Italic can be applied to fonts supporting slant, and is implemented as a slant of 17 degrees.

Examples

The following examples are using start-character '<' and stop-character '>' for readability. They illustrate the different styles supported, as well as different syntactic alternatives. It also illustrates the resulting number of fields, which will be dependent on the number of formatting switches within a PRTXT command.

Description	Code	Resulting Text
Enable inline text formatting.	SYSVAR(84)=60 SYSVAR(85)=62	Not applicable.
Apply boldface to text.	CLL PRTXT "Hello World" PRINT FIELDNO 2	HELLO **WORLD**
Apply italicization to text.	CLL PRTXT "Hello World" PRINT FIELDNO 2	HELLO *WORLD*
Apply boldface and italicization to text.	CLL PRTXT "Hello World" PRINT FIELDNO 2	HELLO *WORLD*

Description	Code	Resulting Text
Automatically end boldface text.	CLL PP 1,35 PRTXT "Hello <b>World" PP 1,1 PRTXT "Normal"	HELLO **WORLD** NORMAL

Authenticate as Privileged User

Purpose

To run certain commands, you must first authenticate yourself as a privileged user. Fingerprint uses two administrator accounts. Use the itadmin account to configure network settings, including Ethernet, 802.11 wireless, and Bluetooth. Use the admin account for all other commands which require privileges.

Syntax

run"su -p <sexp1> <sexp2>"

Parameters

<*sexp1*> is the password. Default is "pass."

<*sexp2*> is either "admin" or "itadmin".

Examples

run"su -p pass admin"

run"su -p pass itadmin"

Honeywell
9680 Old Bailes Road
Fort Mill, SC 29707

www.honeywellaidc.com

FPL-EN-CR Rev C
03-21