MEC Software

PCL Barcode Font Kits

Downloadable barcode fonts for
PCL-compatible printers

User Manual

Published by Measurement Equipment Corporation, Evanston, Illinois USA
www.mecsw-usa.com

Introduction

While most laser printers do not include any internal barcode fonts, printers which are compatible with Hewlett-Packard's PCL-5 printer command language are capable of accepting downloaded fonts. One or more barcode fonts can be stored in the printer's memory and then used by embedding appropriate control sequences in the printer data stream. Downloaded barcode fonts offer some advantages:

- The cost per printer is much lower than hardware solutions
- Performance is faster than downloading barcode graphic images
- Barcodes can be printed from virtually any operating system

All of the fonts are created to produce precision barcodes on printers with 300 dpi or greater resolution. PCL fonts offer much greater precision than TrueType and other types of scaleable fonts; bar widths and spacings are accurate, producing consistently better scanning results.

Each Barcode Font Kit supports a single type of barcode and includes a selection of fonts with barcode heights from 0.25" to 2.00" in increments of 0.25", with narrow bar widths from 6.7 mils to 20 mils, and (where appropriate) with bar width ratios from the minimum to maximum allowed. Each font is identified by a unique number, so several barcode fonts can be downloaded to the printer; the total number is limited only by available printer memory.

Each kit includes a file with a list of all the fonts included in the kit:

C39LIST.TXT (Code 39)	UPCLIST.TXT (UPC-A)
C28LIST.TXT (Code 128)	EANLIST.TXT (EAN-13)
ITFLIST.TXT (Interleaved 2 of 5)	POSTLIST.TXT (Postnet)

The barcode fonts do not include readable text below the bars; the variety of available typefaces and sizes would create an overwhelming range of choices. It is much easier to print the readable text using a separate PCL instruction.

End User License Agreement

YOU MAY:

1. Install and use each font kit on only one printer.
2. Make one additional copy of the complete font kit for back-up purposes.
3. Transfer the complete font kit package to someone else only if you assign all of your rights under this license, cease all use of the fonts, remove all copies of the fonts from your printer and computers, and if the person to whom the font kit package is transferred agrees to the terms of this license.

YOU MAY NOT:

1. Use or make copies of the fonts except as permitted by this license.
2. Rent, sell, lease, assign or transfer the font kit package or any of its components except as set out above.
3. Modify the fonts or merge all or any part of the fonts into another software package.

This license shall continue for as long as you use the product. However, it will terminate if you fail to comply with any of its terms or conditions. You agree, upon termination, to destroy all copies of the product.

We warrant that the storage media for the fonts will be free from defects in materials and workmanship for 90 days from the date you acquire it. If such a defect occurs, return the product to us and we will replace it for free. This remedy is your exclusive remedy for breach of this warranty. It gives you certain rights and you may have other legislated rights which vary from jurisdiction to jurisdiction.

LIMITATION OF WARRANTIES AND LIABILITY: Except for the express warranties of merchantable quality, merchantability or fitness for a particular purpose, or those arising by law, statute, usage of trade or course of dealing, the entire risk as to the results and performance of the FONTS is assumed by you. Neither we nor our dealers or suppliers shall have any liability to you or any other person or entity for any indirect, incidental, special or consequential damages whatsoever including but not limited to loss of revenue or profit, lost or damaged data or other commercial or economic loss, even if we have been advised of the possibility of such damages or they are foreseeable, or for claims by a third party. Our maximum aggregate liability to you, and that of our dealers and suppliers shall not exceed the amount paid by you for the FONTS. The limitations in this section shall apply whether or not the alleged breach or default is a breach of a fundamental condition or term, or a fundamental breach. Some states/countries do not allow the exclusion or limitation of liability for consequential or incidental damages, so the above limitation may not apply to you.

Downloading Fonts to the Printer

There are several methods for sending a font to your printer:

copy

If you are using MS-DOS or Windows, you can simply Copy the font file to the printer port. In Windows, open a MS-DOS command line window. Use the /b (binary) command line option:

```
C:\> COPY /B BC_3of9_4pitch.dsf LPT1:
```

Note: If your system does not support long filenames you will need to rename the files. For example: `C39_4PIT.DSF`.

If you are in a MS-DOS command prompt under Windows and the destination printer is on the network, using the printer's network name may not work depending on your version of Windows. For example, this command should send the font to the printer under Windows/98 and Windows/2000, but may simply make a copy of the file on disk with Windows/95:

```
C:\> COPY /B BC_3of9_4pitch.dsf \\MAIN\LASER
```

You may first have to assign the printer to a local printer port with the NETUSE command. The port does not have to physically exist, and you should not use a port that actually has a local printer attached to it. For example, if you have local printers attached to LPT1 and LPT2 and a network printer named "\\Main\Laser," you could assign LPT3 to the network printer:

```
C:\> NETUSE LPT3: "\\MAIN\LASER"
C:\> COPY /B BC_3of9_4pitch.dsf LPT3:
```

cat

In Unix or Linux you can use the cat command to copy the file to the printer raw device. The "raw device" usually has the same name as the device you commonly use, but with an "r" on the end. Using the raw device avoids the possibility that a print driver or spooler will insert formatting codes into the font as it is sent to the printer:

```
#cat /usr/myfiles/BC_3of9_4pitch.dsf /dev/lpt1r
```

lpr

The lpr command can be an effective method for downloading fonts. Unix and Linux support lpr; Windows/NT, 2000, and XP also support lpr, but it may not be installed by default (lpr may be called Unix printing by the Windows setup process). The general form of the lpr command is:

```
# lpr -S<server> -P<printer> [-C<class>] [-J<job name>]
      [-O<option>] <filename>
```

As an example, let's assume that the destination printer is available at IP address 192.168.100.40 and the file BC_3of9_4pitch.dsf contains the downloadable PCL barcode font. We must use -O (option) followed by a lower case letter "L" to specify binary mode (Windows requires a lower case "o"):

```
# lpr -S192.168.100.40 -P192.168-100.40 -Ol          BC_3of9_4pitch.dsf
```

ftp

This method can be used on Unix, Linux, and Windows systems. At a command line prompt enter the ftp command and IP address of the printer:

```
C:\> ftp <IP address of printer>
```

You will be prompted for a user name and password; press the Enter key for each of these (no user name or password). Then use the PUT command to send each font file to the printer. Here is a sample session:

```
C:\> ftp 192.168.100.40
Connected to 192.168.100.40
220 PRINTER <Model> FTP server (Version) ready.
User (xxx.xxx.xxx.xxx:(none)):_
331 Password required for (none)
Password: _
230 User (none) logged in.
ftp> put BC_3of9_4pitch.dsf
200 PORT command successful.
150 Opening connection 'BC_3of9_4pitch.dsf'
226 Transfer complete.
ftp: xxxx bytes sent in xx.xSec
ftp> bye
```

After you have transferred the files, use the printer's control panel to print a PCL Configuration Page to verify that the fonts have been successfully installed.

Selecting Fonts with Escape Sequences

PCL-compatible printers understand and respond to Hewlett-Packard's Printer Command Language (PCL). The PCL Font Kits are compatible with PCL Version 5 and higher (higher being PCL-6 as of this writing). PCL uses Escape Sequences to control the printer. An escape sequence always begins with the ASCII Escape code, which has a decimal numeric value of 27 (33 in octal). The Escape code (shown below as <esc>) is followed by a series of parameters which tell the printer what to do.

An escape sequence may include more than one parameter. Each parameter generally consists of a value followed by a letter which identifies the type of parameter. If the parameter letter is lower case, it means that another parameter follows it. If the parameter letter is upper case, it means that it is the last parameter and it marks the end of this particular escape sequence.

It is certainly possible to print plain text on a PCL printer without using escape sequences, but the text will be plain and printed line-by-line (teletype style). We can produce more elaborate results by inserting escape sequences to control the type style, position of the text on the page, and so on:

```
<escape sequence to select big and bold type>
My Final Report to the Committee
<escape sequence to select small type>
<escape sequence to reposition cursor>
blah blah blah
123456
```

Once a barcode font has been downloaded, you can select it by embedding this PCL command in the data sent to the printer:

```
<esc>(nX
```

where *n* is the font ID number. For example, if the font ID number is 25501 (the name of the file in the font kit would be C3925501.DSF), this is the PCL command to select the font:

```
<esc>(25501X
```

To switch back to the printer's default font, send this command:

```
<esc>(3@
```

So we can print a barcode in our sample report just by adding a few simple escape sequences:

```
<escape sequence to select big and bold type>
My Final Report to the Committee
<escape sequence to select small type>
<escape sequence to reposition cursor>
blah blah blah
<esc>(25501X*123456*<esc>(3@
```

Here is a snippet of program code witten in C to produce the last line of this sample:

```
sprintf(mystring, "\033(25501X*123456*\033(3@")
```

Here is an example in Visual Basic:

```
mystring = Chr(27) & "(25501X*123456*" &   Chr(27) & "(3@"
```

Adding Readable Text to a Barcode

The fonts do not include readable text characters. Adding the text with a separate print command is easy to do and gives the programmer complete freedom in choosing the style and position of the text.

The simplest way to print text below a barcode is to print the barcode, start a new line, and then print the data again using a plain text font. Here is an example in C that uses the fprintf function to send formatted data to the printer (lp). In practice, programmer would probably write this as one or two lines of code; we have used several lines for clarity:

```
fprintf(lp, "\033(25501X"); /* select font */
fprintf(lp, "*123456*");    /* data */
fprintf(lp, "\033(3@");     /* default font */
fprintf("\n");              /* new line */
fprintf(lp, "123456");      /* text */
```

By using PCL cursor positioning commands, selecting specific typefaces, and controlling type size and boldness the programmer can create virtually any finished appearance desired. Details on the PCL commands are beyond the scope of this manual, but are discussed at length in Hewlett-Packard's publication *PCL5 Printer Language Technical Reference Manual*.

Formatting Barcode Strings

The last piece of the puzzle is to format the data which will be inserted in the barcode. This process can be simple or a bit complex depending on the type of barcode you have decided to use (Code 39 vs. Code 128, for example) and your comfort level with programming. Each PCL Barcode Font Kit contains a set of fonts for one type of barcode and sample files which demonstrate how to format the data and perform any necessary calculations. Details about formatting each type of barcode can be found on the following pages.

Code 39

Code 39 (also known as Code 3 of 9) is a widely-used alphanumeric barcode. The character set includes the upper case letters A-Z, the digits 0-9, and several symbols: hyphen (-), period (.), dollar sign ($), forward slash (/), plus sign (+), and percent (%).

Code 39 barcodes are variable length, and may include any number of characters. In practice, the number of characters in a single barcode is limited by the maximum physical space available on the document. Also, many barcode scanners in common use (CCD scanners) cannot read barcodes over a specific width, typically about 3 inches.

Each Code 39 barcode must begin and end with a special start/stop character which is represented by an asterisk (*). Formatting of the data prior to printing requires only that you add an asterisk at the beginning and at the end of the data. For example, let's say that the data to be encoded is PN334958. The finished string ready for printing as a Code 39 barcode would look like this:

PN334958

Here is a complete set of escape sequences to print the data as a barcode using font number 25501 and then print the information again as text using the printer's default font below the barcode:

<esc>(25501X*PN334958*<esc>(3@
PN334958

The result will look something like this:

PN334958

Here is a C function that illustrates how the escape sequences might be generated in code. The caller passes the desired font number (fontid), the data (partno), and a pointer to the output file or device (prn). The code \033 in C represents Octal 33 which is the escape character (octal 33 = decimal 27).

```
int SendBC(int fontid, char *partno, FILE *prn){
  fprintf(prn, "\033(%dX", fontid);  /*font*/
  fprintf(prn, "*%s*", partno);      /*partno*/
  fprintf(prn, "\033(3@\n");         /*dfltfont*/
  fprintf(prn, "%s\n", partno);      /*text*/
  return(0);                         /*done*/
}
```

Here is Visual Basic code that will return a string with similar results; Chr(27) is the escape code:

```
Function SendBC(fontid as integer, partno as string) as String
  Dim MyString as String
  MyString = Chr(27)& "(" & [fontid] & "X"
  MyString = MyString & "*" & partno & "*"
  MyString = MyString & Chr(27) & "(3@" & vbcrlf
  MyString = MyString & partno
  SendBC = MyString
End Function
```

Extended Code 39

At times it may be necessary to encode a character which is not part of the normal Code 39 character set. “Extended Code 39” or “Full ASCII Code 39” is a method that allows encoding of all 128 ASCII characters. These barcodes must be read using a scanner which has been configured for Extended Code 39. The official AIM specification for Code 39 lists Extended Code 39 as an Optional Characteristic and uses the following language:

“Readers can be programmed to respond to Code 39 symbols in non-standard ways to satisfy particular application requirements... Since use of these features requires special reader programming, they are not recommended for general applications where there would exist the possibility of ambiguity of interpretation with standard Code 39 symbols.”

The upper case alphabet, the digits 0 through 9, the space, the dash (-), and the period (.) are encoded just like standard Code 39. All others are encoded with a pair of barcode characters. The percent sign (%), dollar ($), slash (?), and plus sign (+) are followed by a second character; for example, the pair $M will be scanned as a carriage return code. The scanner must be configured to read Extended Code 39.

Extended Code 39 character codes

ASCII	C39	ASCII	C39	ASCII	C39	ASCII	C39
NUL	%U	SP	_	@	%V	‘	%W
SOH	$A	!	/A	A	A	a	+A
STX	$B	“	/B	B	B	b	+B
ETX	$C	#	/C	C	C	c	+C
EOT	$D	$	/D	D	D	d	+D
ENQ	$E	%	/E	E	E	e	+E
ACK	$F	&	/F	F	F	f	+F
BEL	$G	‘	/G	G	G	g	+G
BS	$H	(	/H	H	H	h	+H
HT	$I	)	/I	I	I	i	+I
LF	$J	*	/J	J	J	J	+J
VT	$K	+	/K	K	K	k	+K
FF	$L	,	/L	L	L	l	+L
CR	$M	-	-	M	M	m	+M
SO	$N	.	.	N	N	n	+N
SI	$O	/	/O	O	O	o	+O
DLE	$P	0	0	P	P	p	+P
DC1	$Q	1	1	Q	Q	q	+Q
DC2	$R	2	2	R	R	r	+R
DC3	$T	4	4	S	S	s	+S
DC4	$T	4	4	T	T	t	+T
NAK	$U	5	5	U	U	u	+U
SN	$V	6	6	V	V	v	+V
ETB	$W	7	7	W	W	w	+W
CAN	$X	8	8	W	W	w	+W
EM	$Y	9	9	Y	Y	y	+Y
SUB	$Z	:	/Z	Z	Z	z	+Z
ESC	%A	;	%F	[	%K	{	%P
FS	%B	<	%G	\	%L	|	%Q
GS	%C	=	%H	]	%M	}	%R
RS	%D	>	%I	^	%N	~	%S
US	%E	?	%J	_	%O		
DEL	%T, %X, %Y, %Z						

Modulo 43 Checksum for Code 39

A checksum is an extra character which is added to the end of a barcode just before the stop character. The value of the checksum is computed from the preceding characters in the barcode, so it will change depending on the data contained in the barcode. The software that creates the barcode is responsible for performing the calculation and adding the checksum character. The scanner reads the barcode, performs the same checksum calculation, and compares the result of this calculation to the checksum at the end of the barcode. If the two do not match, the scanner presumes that something is wrong and does not accept the scan.

In practice, the Modulo 43 checksum is seldom used. While it does provide an additional level of reliability, Code 39 has other checks built into its structure that assure a level of accuracy more than adequate for most applications. A Code 39 barcode is presumed not to include a checksum unless explicitly required. To calculate a Modulo 43 checksum, first assign each character in the barcode a numeric value according to the following table.

Char	Value	Char	Value	Char	Value
0	0	F	15	U	30
1	1	G	16	V	31
2	2	H	17	W	32
3	3	I	18	X	33
4	4	J	19	Y	34
5	5	K	20	Z	35
6	6	L	21	-	36
7	7	M	22	.	37
8	8	N	23	Space	38
9	9	O	24	$	39
A	10	P	25	/	40
B	11	Q	26	+	41
C	12	R	27	%	42
D	13	S	28		
E	14	T	29		

Sum the numeric values of the characters in the barcode (exclude the start/stop characters) and divide the result by 43; the remainder is the checksum value. Convert this to a character using the table above and add that character to the end of the barcode, just before the stop character.

In programming parlance, dividing and taking the remainder as the result is a Modulo division. In Basic, it would be expressed as:

```
Checksum = MySum Mod 43
```

In C/C++ it would be:

```
Checksum = MySum % 43.
```

Code 128

Code 128 has more features than Code 39. For example, the character set includes all of the printable ASCII characters (upper case, lower case, and all symbols); it includes the non-printable ASCII control codes; and there is a method for compressing numeric-only data for significant space savings. Those features come with a cost: more complexity for the programmer... but it isn't too bad if taken one step at a time.

Code 128 includes 103 character codes, but there are three ways to interpret the codes. Where most barcode symbologies define a single start code, Code 128 has three start codes to choose from: Subset A, Subset B, and Subset C. The meaning of a scanned character depends on the subset currently being used. Each subset includes codes to switch to either of the other two subsets, making it possible to mix subsets within a single bar-code. The ability to switch subsets provides a wide range of capabilities to satisfy almost any application.

Subset A contains the standard ASCII characters and control codes: printable symbols, upper case alphabetic characters, the digits 0 through 9, and control codes (NUL, SOH, STX, ETX, etc.). Subset B is similar to Subset A, but the control codes are replaced by lower case alphabetic characters. Subset C includes numbers only but encodes two digits into each barcode character, increasing the amount of data that can be printed in a limited space. Note that the data to be encoded in Subset C must have an even number of digits.

Since each subset switch requires insertion of a control character, it is a good idea to keep switching to a mini-mum. For instance, putting a numeric-only Subset C field in the middle of a barcode would cost two control characters (switch to Subset C and back again). It would be more efficient to put the numeric data at the end (or beginning) of the barcode so only one subset switch is needed.

Building and printing a Code 128 barcode

A Code 128 barcode begins with a start code for either Subset A, B, or C (see the code chart at the end of this section for the specific character value). This is followed by the data to be printed. Following the data is a checksum which is calculated based on all of the characters in the code from the start character through the last data character. The method for calculating the checksum is given below. The stop character is inserted after the checksum:

<start character> <data> <checksum> <stop character>

The Code 128 PCL Font Kit includes a C language module (HP_128) which provides basic functions to assist with building a barcode. The linkable modules are HP_128_S.OBJ (small memory model) and HP_128_L.OBJ (large memory model); the source code is in file HP_128.C. Here is a summary of the functions available:

The function hp128_start_subset(char subset) returns the start character for the desired subset. For example:

```
c = hp128_start_subset('B');
```

The function hp128_switch_subset(char from, char to) inserts the control code to change from one subset to another in the middle of a barcode:

```
c = hp128_switch_subset('B', 'A');
```

The function hp128_checksum(char *datastring) returns the checksum character for datastring. This character should then be appended to datastring, followed by the stop character:

```
c = hp128_checksum(mystring);
```

The function hp128_stop(void) returns the Code 128 stop character:

```
c = hp128_stop();
```

There are two other functions included in HP_128 that are useful. The first, pcl_select_font() returns a pointer to a PCL command string that will select the desired font. For example, if the barcode font has an ID number of 25801 and the file pointer for the printer is lp:

```
fprintf(lp, "%s", hp_select_font(25801));
```

The second function, pcl_transparent(), instructs the printer to print the next *n* characters without checking them for command codes. This will prevent barcode data, including the checksum, from being incorrectly interpreted as a command. Use it immediately before sending the barcode string to the printer. For example, if mystring contains the complete barcode (start character, data, checksum, and stop character) and the file pointer for the printer is lp:

```
fprintf(lp, "%s",   pcl_transparent(strlen(mystring)));
```

One other PCL instruction, which is not included in HP_128, selects the default font on the printer. This is the font which has been selected on the printer's control panel. After printing a barcode you will have to change back to a normal text font, and using the default may be preferable to specifying a particular font. This instruction will switch to the default font:

```
fprintf(lp, "\033(3@");
```

For a complete example of how to use these functions, see the source code for EXAMPLE.C which is included on the release disk for the Code 128 PCL Font Kit.

Character Value vs. Location in Font

The characters in Code 128 are numbered from 0 through 102 (plus the start and stop characters); this number is shown in the *Value* column of the chart at the end of this section and is used to calculate the checksum. The ASCII character codes recognized by PCL printers begin at 32 (space); these codes are shown in the *Location in Font* column.

Printable ASCII characters can be printed in Subset A or B simply by sending the character's ASCII code to the printer. For purposes of calculating the checksum, the value of the character is 32 less than its ASCII code. For example, the value of the letter A (ASCII code 65) when calculating the checksum is 33 (65 - 32).

Subset A permits printing of control characters, those with ASCII codes between 0 and 31. To print a control character, add 96 to its ASCII code and send the result to the printer. For example, to send a form feed (ASCII 12) add 96 and send the result: 12 + 96 = 108. To obtain the value of this character for the checksum calculation add 64 to its ASCII code. For example, 12 + 64 = 76.

Subset C prints pairs of digits as single barcode characters. To print a pair of digits, add 32 to the numeric value of the pair (00 - 99) and send the result to the printer. The value of the character for the checksum is the numeric value of the pair. For example, 55 would be sent to the printer as 87 (55 + 32) and its value for the checksum would be 55.

Calculating the Code 128 Checksum

Code 128 requires that a modulo 103 checksum be appended to the barcode. The checksum includes the start character and all subsequent data and control characters, but not the stop character. Calculate the checksum as follows:

1. Initialize the checksum with the value of the start character used (103, 104, or 105 for Subset A, B, or C).

2. Initialize a multiplier to 1.

3. Starting at the left end of the barcode, add the value of each character times the multiplier to the checksum. Increment the multiplier by one after each character.

4. Divide the result by 103. The remainder is the checksum.

For example, calculate the checksum for the word “HELLO” using Subset B:

Start Code B		104
H:	40 * 1	40
E:	37 * 2	74
L:	44 * 3	132
L:	44 * 4	176
O:	47 * 5	235

The total is 761, and 761 modulo 103 is 40 (761 / 103 = 7 with remainder of 40), and that is the value of the character that should be printed. The ASCII character code to be sent to the printer is 72 (40 + 32).

Special caution for C programmers: *Be careful when using string functions (sprintf, fprintf, etc.) when formatting barcodes. Checksums can come out to any value, and if they resolve to a backslash or percentage sign, the string functions will interpret them as formatting characters and give you incorrect results. Either avoid the standard string functions or test for \ and % characters.*

Code 128 Character Table

This table lists all of the characters available within Code 128. The *value* column is the numeric value of the character which should be used in calculation of the checksum. The *font* location column is the character code which should be sent to the PCL printer to print that character. The *Subset A, B, and C* columns give the meaning of the character in each subset.

Value	Font Loc	Sub A	Sub B	Sub C
0	32	SP	SP	00
1	33	!	!	01
2	34	“	“	02
3	35	#	#	03
4	36	$	$	04
5	37	%	%	05
6	38	&	&	06
7	39	‘	‘	07
8	40	(	(	08
9	41	)	)	09
10	42	*	*	10
11	43	+	+	11
12	44	,	,	12
13	45	-	-	13
14	46	.	.	14
15	47	/	/	15
16	48	0	0	16
17	49	1	1	17
18	50	2	2	18
19	51	3	3	19
20	52	4	4	20
21	53	5	5	21
22	54	6	6	22
23	55	7	7	23
24	56	8	8	24
25	57	9	9	25
26	58	:	:	26
27	59	;	;	27
28	60	<	<	28
29	61	=	=	29
30	62	>	>	30
31	63	?	?	31
32	64	@	@	32
33	65	A	A	33
34	66	B	B	34
35	67	C	C	35
36	68	D	D	36
37	69	E	E	37
38	70	F	F	38
39	71	G	G	39
40	72	H	H	40
41	73	I	I	41
42	74	J	J	42
43	75	K	K	43
44	76	L	L	44
45	77	M	M	45
46	78	N	N	46
47	79	O	O	47
48	80	P	P	46
49	81	Q	Q	49
50	82	R	R	50
51	83	S	S	51
52	84	T	T	52
53	85	U	U	53
54	86	V	V	54
55	87	W	W	55
56	88	X	X	56
57	89	Y	Y	57
58	90	Z	Z	58
59	91	[	[	59
60	92	\	\	60
61	93	]	]	61
62	94	^	^	62
63	95	_	_	63
64	96	NUL	‘	64
65	97	SOH	a	65
66	98	STX	b	66
67	99	ETX	c	67
68	100	EOT	d	68
69	101	ENQ	e	69
70	102	ACK	f	70
71	103	BEL	g	71
72	104	BS	h	72
73	105	HT	i	73
74	106	LF	j	74
75	107	VT	k	75
76	108	FF	l	76
77	109	CR	m	77
78	110	SO	n	78
79	111	SI	o	79
80	112	DLE	p	80
81	113	DC1	q	81
82	114	DC2	r	82
83	115	DC3	s	83
84	116	DC4	t	84
85	117	NAK	u	85
86	118	SYN	v	86
87	119	ETB	w	87
88	120	CAN	x	88
89	121	EM	y	89
90	122	SUB	z	90
91	123	ESC	{	91
92	124	FS	|	92
93	125	GS	}	93
94	126	RS	~	94
95	127	US	DEL	95
96	128	FNC3	FNC3	96
97	129	FNC2	FNC2	97
98	130	SHIFT	SHIFT	98
99	131	SUBC	SUBC	99
100	132	SUBB	FNC4	SUBB
101	133	FNC4	SUBA	SUBA
102	134	FNC1	FNC1	FNC1
103	135	Start Code A		
104	136	Start Code B		
105	137	Start Code C		
	138	Stop		

Interleaved 2 of 5

Interleaved 2 of 5 is a compact, numeric-only barcode that encodes a pair of digits on each barcode character. For example, “35” is encoded as a single character; when this character is read by the barcode scanner, it is separated into two digits before transmission to the computer. The character which represents the digit pair “00” is located where you would expect to find the zero character (ASCII character value 48). By adding the numeric value of the desired digit pair to the ASCII value of the zero character, you will obtain the ASCII value of the character that you should print. In Visual Basic, for example, Chr(48 + 57) would print the barcode character for 57.

The start and stop characters are represented by the left and right parentheses: “(“ and “)”.

An Interleaved 2 of 5 barcode may include a checksum, which is calculated using the Modulo 10 method. Since the final barcode string (including the checksum) must have an even number of digits, an extra character (typically a leading zero) may have to be added to the original data. The checksum is calculated on the string of digits before they are divided into pairs; the start character is not included in the calculation:

1. Starting with the digit in position 1 (the left-most digit), add the values of all the digits in the odd-numbered character positions (1, 3, 5, etc.).

2. Multiply the result of Step 1 by 3.

3. Add the values of the digits in the even-numbered positions to the result of Step 2.

4. Perform a Modulo 10 division on the result of Step 3; that is, divide the result of Step 3 by 10 and take the remainder.

5. Subtract the result of Step 4 from 10; if this result is 10, change it to zero. This is the checksum which should be appended to the end of the data string.

The sample Visual Basic function on the following page will return a formatted string for printing as an Interleaved 2 of 5 barcode

```
Function FmtITF(InString As String, Checksum As Boolean) As String
  Dim i As Integer
  Dim MySum As Integer
  Dim WorkString As String
  Dim FinishedString As String
  Dim PairVal As Integer
  Dim CharVal As Integer
  Dim PairString As String

  '
  '   check for illegal alpha characters
  '
  If (IsNumeric(InString) = False) Then
    FmtITF = ""
    Exit Function
  End If
  '
  '   insure even number of digits
  '
  i = Len(InString) Mod 2
  If ((Checksum = False And i <> 0) Or (Checksum   = True And i=0)) Then
    WorkString = "0" & InString
  Else
    WorkString = InString
  End If
```

```
  '
  '   calculate checksum if requested
  '
  If Checksum = True Then
    MySum = 0
    '
    '  Add up digits in odd-numbered positions
    '  then multiply result by 3
    '
    For i = 1 To Len(WorkString) Step 2
      MySum = MySum + Val(Mid(WorkString, i, 1))
    Next i
    MySum = MySum * 3
    '
    '  Add in digits in even-numbered positions
    '
    For i = 2 To Len(WorkString) Step 2
      MySum = MySum + Val(Mid(WorkString, i, 1))
    Next i
    '
    '   Finish the calculation
    '
    MySum = MySum Mod 10
    MySum = 10 - MySum
    If MySum = 10 Then
      MySum = 0
    End If
    WorkString = WorkString & Format(MySum)
  End If
  '
  '  Build the finished string
  '
  FinishedString = "("
  For i = 1 To Len(WorkString) Step 2
    PairString = Mid(WorkString, i, 2)
    PairVal = Val(PairString)
    CharVal = PairVal + Asc("0")
    FinishedString=FinishedString & Chr(CharVal)
  Next i
  FinishedString = FinishedString & ")"
  FmtITF = FinishedString
End Function
```

UPC-A

UPC-A is used in the United States and Canada to identify retail products for checkout scanning. The code is fixed-length (12 digits), numeric only. The Uniform Code Council (located in Dayton, Ohio) assigns identification numbers to manufacturers. The barcode begins with a leading “number system” digit followed by the manufacturer’s identification; both are assigned by the UCC. This is followed by digits, assigned by the manufacturer, which identify each of the manufacturer’s products. The final 12th digit is a check digit used to insure scanning accuracy. The barcode also includes left, center, and right guard characters.

The length of the manufacturer identification number assigned by the UCC may vary from 5 to 8 digits, depending on the anticipated number of products the manufacturer needs to identify. The longer the manufactuer ID, the fewer digits available for identifying products.

Character Position	*Function*
1	left guard
2	number system digit
3-7	5 digits (manufacturer ID)
8	center guard
9-13	5 digits (mfgr ID/product ID)
14	check digit
15	right guard

The left guard and right guard characters are identical and can be printed using an asterisk or left and right parentheses or square brackets: * ( ) [ ]. The center guard character may be printed using a hyphen or a vertical “pipeline” symbol: - |.

The bar pattern for digits in the left half of the barcode is different from the pattern used on the right side. In the PCL font, left-side digits are represented in the normal way by the characters 0 through 9. Right-side digits are represented by the letters A through J. For a programmer, encoding the right-side digits is simply a matter of adding the numeric value of each digit to the ASCII value for the character A (41).

Left	*Right*	*Left*	*Right*
0	A	5	F
1	B	6	G
2	C	7	H
3	D	8	I
4	E	9	J

Let’s say we wish to encode the number 0-00123-45678-4 (dashes are shown for clarity). Each of the following examples would produce the same result:

```
*000123-EFGHIE*
(000123-EFGHIE)
[000123|EFGHIE]
```

Applications designed to use the Hewlett-Packard “Bar Codes & More” Font Product already perform the required formatting and checksum calculation. Use of the downloadable UPC-A font requires no changes to existing applications.

To assist with the creation of new code or the modification of existing applications, sample code that illustrates how to calculate the checksum and format the data for a UPC-A barcode is provided on the following pages.

Note: The UPC-A font supports all of the common retail barcode symbols.

- UPC-A
- UPC-E
- EAN-13
- EAN-8
- 2-digit supplemental
- 5-digit supplemental

Format UPC-A with C

```
static char upc_string[20];

char *format_upc(char *instring)
{
  char *cpin, *cpout;
  int  check, i;

  // clear working string and set up pointers
  memset(upc_string, 0, sizeof(upc_string));
  check = upc_check(instring);
  cpin = instring;
  cpout = upc_string;

  // insert left guard
  *cpout++ = '*';

  // insert first 6 digits
  for (i = 1; i <= 6; i++)
    *cpout++ = *cpin++ - '0';

  // insert center guard
  *cpout++ = '-';

  // insert next 5 digits
  for (i = 1; i <= 5; i++)
    *cpout++ = 'A' + (*cpin++ - '0');

  // insert check character
  *cpout++ = 'A' + check;

  // insert right guard and return to caller
  *cpout = '*';
  return(upc_string);
}

int upc_check(char *instring)
{
  int  mysum, mycheck;
  char *cp;

  // start with zero sum and point at string
  mysum = 0;
  cp = instring;

  // add values of characters in odd positions
  while (*cp != '\0') {
    mysum += *cp - '0';
    cp += 2;
    }

  // multiply sum so far by 3
  mysum = mysum * 3;
```

```
  // set pointer back to start of string
  // and advance by 1 to first even position.
  cp = instring;
  cp++;

  // Add values of chars in even positions
  while (*cp != '\0') {
    mysum += *cp - '0';
    cp += 2;
  }

  // Calculate the Modulo 10 result.  If
  // result = 10, then force to zero.
  // Return result to caller
  mycheck = 10-(mysum % 10);
  if (mycheck == 10)
    mycheck = 0;
  return(mycheck);
}
```

Format UPC-A with Basic

```
Function format_upc(instring As String) As String
  Dim check As Integer
  Dim i As Integer
  Dim MyString as String
  Dim MyChar as Integer

  ' Insert left guard
  MyString = "*"

  ' Insert first 6 characters
  MyString = MyString & Left$(instring, 6)

  ' Insert center guard
  MyString = MyString & "-"

  ' Insert the next 5 characterse
  for i = 7 to 11        ' next 5 digits
    MyChar = Val(Mid(instring, i, 1))
    MyChar = MyChar + Asc("A")
    MyString = MyString & Chr(MyChar)
  next i

  ' Insert the checksum
  MyChar = upc_check(instring) + Asc("A")
  MyString = MyString & Chr(MyChar)

  ' Insert the right guard
  MyString = MyString & "*"

  ' Return result to the caller
  format_upc = MyString
End Function

Function upc_check(instring As String) As Integer
  Dim MySum As Integer
  Dim i As Integer

  ' Start with zero
  mysum = 0

  ' Add up values of digits in odd positions
  for i = 1 to 11 Step 2
    mysum = mysum + Val(Mid(instring, i, 1))
  next i

  ' Multiply result so far by 3
  mysum = mysum * 3

  ' Add values of digits in even positions
  for i = 2 to 10 Step 2
     mysum = mysum + Val(Mid(instring, i, 1))
  next i
```

```
  ' Calculate Modulo 10. If result = 10 then
  ' force result to zero
  MySum = MySum Mod 10
  MySum = 10 - MySum
  if MySum = 10 then
    MySum = 0
  endif

  ' Return to caller
  upc_check = MySum
End Function
```

EAN-13

EAN-13 is the retail barcode used everywhere in the world outside North America. While the first digit in a UPC-A code defines the "number system", the first 2 or 3 digits in an EAN-13 code identify the country in which the code is registered.

The EAN-13 symbol is almost identical to the UPC-A symbol but encodes 1 extra digit (a total of 13 digits) into the same number of bars and spaces. The first (left-most) digit in an EAN-13 data string is not explicity printed as a separate character in the barcode. Instead, EAN-13 takes advantage of the fact that any of the individual barcode characters can be printed using even or odd parity; the first character is encoded in the even/odd parity pattern in the six left-hand digits of the barcode.

So what does parity mean? Each EAN character is made up of 7 modules which are combined to form 2 bars and 2 spaces. If we use "0" to represent a white module and "1" to represent a dark module, "0001101" would indicate a space (3 modules wide), a bar (2 modules wide), a space (1 module wide, and a bar (1 module wide). This is odd parity, because the total number of dark modules (3) is odd. This particular pattern encodes a zero, which can also be printed with even parity as "01000111".

To format a string for printing as an EAN-13 symbol, begin with a start code (represented by an asterisk). The next step is to use the first character in the input string to decide what parity pattern to use. Number set A uses odd parity, and number set B uses even parity.

1st Digit	*Number Set to use*	*1st Digit*	*Number Set to use*
0	AAAAAA	5	BAABBA
1	BBABAA	6	AABBBA
2	BABBAA	7	BABABA
3	ABBBAA	8	ABBABA
4	BBAABA	9	ABABBA

For example, if the first digit in the data string is 4, then we should use the pattern BBAABA. The second digit in the data string (the first digit to be printed in the barcode) and the one after it should be printed using Number Set B; the 4th and 5th digits in the data string should use Number Set A; the 6th digit should use Number Set B; and the 7th digit should use Number Set A.

The ten characters for Number Set A begin at "0" in the font, and Number Set B begins at "K". To print a character using Set A, add the numeric value of the digit to the ASCII value for "0":

```
CHR(ASC("0") + VAL(mychar))        ... use set A
CHR(ASC("K") + VAL(mychar))        ... use set B
```

After digits 2 through 7 from the input string have been added to the finished string, add the center guard bars using a vertical pipe symbol (|) or a hyphen (-). Print the remaining characters from the input string (8 through 12) using Character Set C, which starts in the font at "A":

```
CHR(ASC("A") + VAL(mychar))        ... use set C
```

The next step is to calculate the checksum. Starting from the right end of the input data string, add the numeric values of the digits in the even-numbered positions; the right-most character is in position 1, so start with the second character from the right. Multiply the sum by 3. Then start back at the right end with the character in position 1 and add the numeric values of the digits in the odd-numbered positions into our sum.

Finally, perform a Modulo 10 division on the sum (divide by 10 and use the remainder). Subtract the result from 10; if the answer is 10, change it to zero. This is the checksum. Add this character to the finished barcode string using Number Set C. Finish the string with a guard bar pattern (asterisk). For example, let's print an EAN-13 barcode using the string 719954678901.

1. Start the output string with a guard bar pattern: "*"

2. The first digit in our string is a 7. Using our chart on the preceding page, we will use a pattern of number sets BABABA. Add the next 6 digits to the output string:

Input Digit	*Num Set*	*Compute Character value*
1	B	CHR(ASC("K") + VAL("1"))
9	A	CHR(ASC("0") + VAL("9"))
9	B	CHR(ASC("K") + VAL("9"))
5	A	CHR(ASC("0") + VAL("5"))
4	B	CHR(ASC("K") + VAL("4"))
6	A	CHR(ASC("0") + VAL("6"))

3. Insert the center guard bar pattern: "-"

4. Add the remaining digits using Number Set C:

Input Digit	*Num Set*	*Compute Character value*
7	C	CHR(ASC("A") + VAL("7"))
8	C	CHR(ASC("A") + VAL("8"))
9	C	CHR(ASC("A") + VAL("9"))
0	C	CHR(ASC("A") + VAL("0"))
1	C	CHR(ASC("A") + VAL("1"))

5. Calculate the checksum:
   a. Starting from the right, add up even-numbered positions:
      7 + 9 + 5 + 6 + 8 + 0 = 35
   b. Multiply the sum by 3:
      35 * 3 = 105
   c. Starting from the right, add odd-numbered positions:
      105 + 1 + 9 + 7 + 4 + 9 + 1 = 136
   d. Perform a Modulo 10 division:
      136 Mod 10 = 6
   e. Subtract the result from 10:
      10 - 6 = 4
   f. If the result is 10, change it to zero

6. Add the checksum to the output string using Number Set C:
   CHR(ASC("A") + VAL("4"))

7. Add the right guard bar pattern: "*"

The sample file ModuleEAN13.bas contains Visual Basic functions that will perform the formatting functions. The code is printed on the following pages.

```
Function format_ean(instring As String) As String
  Dim check, i As Integer
  Dim outstring, mychar, myset As String
  Dim setstr, mystring As String

  '
  '   Make sure that we have 12 digits
  '
  If IsNull(instring) = True Then
    mystring = "000000000000"
  ElseIf Len(instring) < 12 Then
    mystring = "
```

```
    For i = 1 To (12 - Len(instring))
      mystring = mystring & "0"
    Next i
    mystring = mystring & instring
  ElseIf Len(instring) > 12 Then
    mystring = Mid$(instring, 1, 12)
  Else
    mystring = instring
  End If
  '
  '   Determine what pattern of EAN character
  '   sets should be used.
  '
  setstr = ean_charset(Mid$(mystring, 1, 1))
  '
  '   Insert the left guard bar pattern and the
  '   next 6 digits, looking up the correct
  '   character based on the EAN character set
  '   for each position.  Do not insert the first
  '   character; it is encoded in the parity
  '   pattern of digits 2 through 7
  '
  outstring = "*"
  For i = 2 To 7
    mychar = Mid$(mystring, i, 1)
    myset = Mid$(setstr, i - 1, 1)
    outstring=outstring & ean_char(mychar, myset)
  Next i
  '
  '   Insert center guard and the next 6 digits
  '
  outstring = outstring & "|"
  For i = 8 To 12
    mychar = Mid$(mystring, i, 1)
    outstring = outstring & ean_char(mychar, "C")
  Next i
  '
  '   Calculate check digit, append to string,
  '   and add the right guard bar pattern
  '
  check = ean_check(mystring)
  outstring=outstring & ean_char(Chr(check), "C")
  outstring = outstring & "*"
  format_ean = outstring
End Function

Function ean_check(instring As String)As Integer
  Dim cp, sum, ck As Integer

  '
  '   Initialize sum and add up the values of
  '   even positions starting from the right
  '
  sum = 0
  For cp = 12 To 1 Step -2
    sum = sum + Val(Mid$(instring, cp, 1))
  Next cp
  '
```

```
  '   Multiply result by 3, then add values of
  '   characters in the odd positions
  '
  sum = sum * 3
  For cp = 11 To 1 Step -2
    sum = sum + Val(Mid$(instring, cp, 1))
  Next cp
  '
  '   Do a Modulo 10 division and subtract the result
  '   from 10.  If the result is 10, set it to zero.       '
  ck = 10 - (sum Mod 10)
  If ck = 10 Then
    ck = 0
  End If
  ean_check = ck
  End Function

Function ean_charset(leftchar As String) As String
    Dim Sets() As Variant
    Dim MyVal As Integer

  '
  '   This array defines the parity pattern to
  '   be used for digits 2 through 7.  The
  '   pattern encodes the first data character.
  '
  Sets = Array("AAAAAA", "BBABAA", "BABBAA",
  "ABBBAA", "BBAABA", "BAABBA", "AABBBA",
  "BABABA", "ABBABA", "ABABBA")

  '
  '   Get the numeric value of the character
  '   and return the correct string
  '   based on its value.
  '
  MyVal = Val(leftchar)
  If MyVal < 0 Or MyVal > 9 Then
    MyVal = 0
  End If
  ean_charset = Sets(MyVal)
End Function

Function ean_char(datachar As String, charset As String) As String
  Dim i As Integer

  '
  '   This function returns the character that
  '   should be printed based on the value of
  '   the character and the character set that
  '   should be used (A, B, or C)
  '
  i = Val(datachar)
  If charset = "A" Then
    i = Val(datachar) + Asc("0")
  ElseIf charset = "B" Then
    i = Val(datachar) + Asc("K")
  ElseIf charset = "C" Then
    i = Val(datachar) + Asc("A")
  End If
  ean_char = Chr(i)
End Function
```

UPC-E

UPC-E is a reduced-size retail barcode that encodes 6 digits.
Standard UPC-A barcodes can be compressed to the smaller UPC-E symbol through a method that squeezes out zeroes; the UPC-E symbol can only be used for number system 0 (the UPC code must begin with a zero). The number of different item numbers available depends on the number of zeroes in the manufacturer number. We are assuming a number system of 0 and a 5-digit manufacturer number. Here is the UPC-A layout:

<number system><manufacturer><item><checksum>
0-12000-00001-x

The quantity of product numbers available in UPC-E depends on the final few digits in the manufacturer number:

Manufacturer ends in 000, 100, or 200
Item numbers may range from 00000 to 00999 (1,000 numbers). The 6 digits of the UPC-E code are obtained from the first two digits of the manufacturer number, the last three digits of the item number, and the third digit of the manufacturer number.

Manufacturer ends in 300, 400, 500, 600, 700, 800, or 900
Item numbers may range from 00000 to 00099 (100 numbers). The 6 digits of the UPC-E code are obtained from the first three digits of the manufacturer number, the last two digits of the item number, and a fixed “3”.

Manufacturer ends in 10, 20, 30, 40, 50, 60, 70, 80, or 90
Item numbers may range from 00000 to 00009 (10 numbers). The 6 digits of the UPC-E code are obtained from the first four digits of the manufacturer number, the last digit of the item number, and a fixed “4”.
Manufacturer does not end in 0
Item numbers may range from 00005 to 00009 (5 numbers). The 5 digits of the UPC-E code are obtained from the five digits of the manufacturer number and the last digit of the item number.

Using the example UPC-A code shown above (0-12000-00001-x), the manufacturer number ends in 000, so the 6 UPC-E digits will be the first two digits of the manufacturer number (12), the last three digits of the item number (001), and the third digit of the manufacturer number (0): 120010.

The readable text printed below UPC-E barcodes would often seem to imply that the barcode contains 8, not 6, digits; our example might appear as 0-120010-0. The leftmost digit (0) represents number system 0; this number system is required for UPC-E and is not explicitly encoded. The rightmost digit (0) is the check digit; it is encoded in the parity pattern of the barcode characters which represent the 6 data digits.

Calculating the Checksum
The checksum is calculated based upon the full 11-digit UPC code using the same method as the UPC-A symbol. If only the 6-digit number is available, it must be enlarged to its uncompressed form before calculating the checksum. Use the last (6th) data digit to find an expansion pattern on the following table, then insert the pattern shown into the first five digits of the UPC-E data:

Last Digit	*Expansion Pattern*
0	**00000***
1	**10000***
2	**20000***
3	***00000**
4	****00000*
5	*****00005
6	*****00006
7	*****00007
8	*****00008
9	*****00009

For example, our six digits are 120010. The last digit (0) points us to the first line of the table and a pattern of **00000***. By replacing the asterisks in the pattern with our 5 digits of data we obtain **12**00000**010**. Number system 0 is assumed, so the final result can be shown as 0-12000-00010-x (dashes added for clarity, x represents the checksum). Now we can calculate the checksum using the standard UPC-A method and use that result in the finished UPC-E code.

The check digit is not printed in the barcode as a separate character; it is encoded by the parity pattern used to print the 6 data digits. Use the check digit to select a row in the following table and use the parity pattern on that row to encode the digits.

Check	*Character Position*					
Digit	*1*	*2*	*3*	*4*	*5*	*6*
0	E	E	E	O	O	O
1	E	E	O	E	O	O
2	E	E	O	O	E	O
3	E	E	O	O	O	E
4	E	O	E	E	O	O
5	E	O	O	E	E	O
6	E	O	O	O	E	E
7	E	O	E	O	E	O
8	E	O	E	O	O	E
9	E	O	O	E	O	E

Odd parity characters (normally used on the left side of UPC-A symbols) are mapped in the font from 0 - 9. Even parity characters (normally used on the right side of UPC-A symbols) are mapped from A - J (A = 0, B = 1, C = 2, etc.).

Calculating the check digit for 120010 produces a result of 0 which points us to the first line of the parity table and pattern EEEOOO. The first three digits (120) will be encoded with even parity and the next three digits (010) with odd parity.

Even parity digits are mapped in the font starting at A, so “120” would be printed as “BCA”. Odd parity digits are mapped in the font starting at 0, so “010” will be printed without change as “010”.

We need to add the left and right guard patterns to complete the barcode. The left guard is represented by a less-than symbol (<), and the right guard for UPC-E is represented by a greater-than symbol (>). The finished text string will look like this:

```
<BCA010>
```

When printed using the UPC/EAN font this will produce a finished UPC-E barcode.

EAN-8

The EAN-8 symbol encodes 8 digits. The first two digits represent the EAN country code; the next 5 digits are the product number assigned by the EAN authority; the rightmost digit is the Modulo 10 check digit based on the preceding 7 digits. The check digit is calculated using the same method as EAN-13.

Unlike the UPC-E code which is a compressed form of a UPC-A code, EAN-8 codes are assigned to specific products and do not have a matching EAN-13 version. Five digits limits the quantity of possible numbers to 100,000 within a single country, so EAN-8 codes are generally assigned only to products where physical space limitations prevent use of the EAN-13 symbol.

For example, assume a country code of 01 and an item number of 12001. The check digit for the complete number (0112001) would be 1. To print the barcode start with the left guard character (square brackets, parentheses, or asterisks can be used' we'll use the square brackets in this example). Next come the first 4 digits using odd-parity characters (mapped in the font from 0 - 9). These are followed by the center guard (hyphen - or vertical pipe |). Next come the last 4 digits printed with even-parity characters (mapped in the font from A - J). A right guard character finishes the barcode. If our data is 01120011, here is a finished string that, when printed with the UPC/EAN font, will produce an EAN-8 symbol.

```
[0112-AABB]
```

UPC/EAN Supplemental Codes

A 2-digit or 5-digit extension can be appended to the right of a UPC-A or EAN-13 symbol. These codes are used for specialized purposes such as indicating the issue number of a periodical (2-digit code) or the price of a book (5-digit code).

The parity pattern for a 2-digit code can be determined by performing a Modulus 4 division (divide by 4 and use the remainder) and taking the the parity pattern from the following table:

Modulus 4 Result	*Parity Pattern*
0	OO
1	OE
2	EO
3	EE

The supplemental code is separated from the main UPC-A or EAN-13 code by a space character. Next comes a special left guard character for the supplemental (plus sign), the first data character, a separator (slash character), and the second digit. There is no right guard character.

For example, if the value to be encoded in the 2-digit supplemental is 02 the parity pattern would be EO. Even parity supplemental characters are mapped in the font from a - j, so the first digit (0) would be printed as “a”. Odd parity characters are mapped from 0 - 9, so the second digit (2) would be printed as “2”. The finished supplemental string would be:

```
+a/2
```

Supplemental codes never appear by themselves. Here is an example of a complete string to print a UPC-A code and the supplemental code described above:

```
[100001-AAACA] +a/2
```

For a 5-digit supplemental code, calculate the checksum using the following method:

1. Starting at the left, sum the digits in positions 1, 3, and 5
2. Multiply the result by 3
3. Sum the digits in positions 2 and 4
4. Multiply the result by 9
5. Sum the results of Steps 2 and 4
6. Perform a modulus 10 division; the result is the check digit

For example, take the data value 33124:

1. 3 + 1 + 4 = 8
2. 8 * 3 = 24
3. 3 + 2 = 5
4. 5 * 9 = 45
5. 24 + 45 = 69
6. 69 Modulo 10 = 9

Take the parity pattern for the 5 barcode characters from the following table. In our example, a check digit of 9 indicates a parity pattern of OOEOE:

Check Digit	*Parity Pattern*
0	EEOOO
1	EOEOO
2	EOOEO
3	EOOOE
4	OEEOO
5	OOEEO
6	OOOEE
7	OEOEO
8	OEOOE
9	OOEOE

As with the 2-digit code, a space character separates the supplemental from the main barcode; the supplemental begins with a special left guard character (+), and the 5 digits are separated by a special character (/).

Even parity supplemental characters are mapped in the font from a - j. Odd parity characters are mapped from 0 - 9. In our example (33124) the check digit is 9, so the parity pattern is OOEOE. The first two digits are odd parity and printed without change as “33”; the third digit (1) is even parity and printed as “b” (“a” + 1 = “b”); the fourth digit (2) is odd parity and printed as “2”; the fifth digit (4) is even parity and printed as “e” (“a” + 4 = “e”).

Adding the left guard and the separator characters, the 5-digit supplemental code would be printed like this:

```
+3/3/a/2/e
```

This is what a complete UPC-A print string would look like with the supplemental attached:

```
[100001-AAAACA] +3/3/a/2/e
```

Postnet

The Postet barcode is the row of tall and short bars that often appear below or above an address on a letter. Postnet encodes the Zip Code so that it can be read by automatic sorting equipment. A Postnet barcode can include the 9-digit Zip+4 code or the 11-digit Delivery Point Code, which is the same as Zip+4 with two extra digits to define the destination in more detail. Each digit is represented by five bars, two tall and three short, and the complete barcode is constructed as follows:

- Start character (guard bar) represented by letter L or l (left)
- Numeric data (5, 9, or 11 digits)
- Check Digit
- Stop character (guard bar) represented by letter R or r (right)

For example, the following text string will produce a complete Postnet barcode:

```
L1234567895R
```

The check digit is used to insure accuracy when the barcode is scanned. The check digit is calculated using the 5, 9, or 11 digits of Zip Code data and must be calculated for each barcode. When the barcode is read, the scanning equipment performs the same calculation and compares its result with the check digit that was read from the barcode. If the two do not match, the scanner knows that there is something wrong with the data and can eject the letter for manual sorting.

Here is the general method for calculating the checksum. Starting from the left, sum all of the digits in the barcode. Using the example above:

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
```

For the next step, we need the units column of the sum. You can extract this information using Modulo 10 division, which is the same as dividing by 10 and taking the remainder.

```
Basic:     45 MOD 10 = 5
C:         45 % 10 = 5
```

Subtract this result from 10 to obtain the check digit; if the final result is 10, change it to zero. In the example above, the check digit would be 5. Here is a bit of sample code written in Basic:

```
ZipString = "123456789"
Sum = 0
FOR x = 1 to LEN(ZipString)
  Sum = Sum + VAL(MID$(ZipString, x, 1))
NEXT x
Check = 10 - (Sum MOD 10)
IF Check = 10 THEN
  Check = 0
ENDIF
PrintString = "L"+ZipString+Str$(Check)+"R"
```

Here is the same thing in C:

```
char ZipString[20], PrintStr[20], *cp;
int  Sum, Check;
 strcpy(ZipString, "123456789");
 Sum = 0;
 cp = ZipString;
 while (*cp != '\0')
   Sum += (*cp++) - '0';
 Check = 10 - (Sum % 10);
 if (Check == 10)
 Check = 0;
 sprintf(PrintStr,"L%s%1dR",ZipString, Check);
```

Planet Code

The Planet Code is a variation of the Postnet code used by the United States Postal Service. The height of the bars is inverted, so each Planet character has three tall bars and two short ones. The left and right frame bars are the same as Postnet, and the checksum calculation method is the same.

Planet Code is used for four applications, each with 12 digits:

- Mailpiece rate, pre-sort bureau origin, customer identification
- On-line notification and ID of person returning reply mail
- Sorting of foreign mail
- Automated address correction requests

Planet Code fonts begin with “PLA” and use the same formatting rules as the Postnet fonts.