dLSoft

Barcode Fonts

By dLSoft

This manual was produced using *ComponentOne Doc-To-Help.*™

Contents

dLSoft Barcode Fonts

Introduction

Barcodes are patterns of light and dark that may be scanned by an optical scanner and decoded into characters. A complete barcode is referred to as a barcode Symbol and the characters represented by the symbol are called the Code.

Most barcode symbols are made up of a series of parallel, adjacent bars and spaces, where the bars present a dark image and the spaces a light, reflecting image. In most circumstances a human-readable form of the code is printed directly under the barcode symbol - to allow for manual deciphering in case the symbol has been damaged.

Barcodes fall into two main categories - discrete and continuous. Discrete barcodes are those for which each character in the code translates into a character in the symbol, and each character in the symbol is separated from its neighbour by a gap containing no information. A continuous code has no inter-character gaps - each character starts with a bar and ends with a space, and the start of the following character's first bar is immediately after that space. While continuous codes are more space efficient (i.e. no gaps), they do impose far greater tolerance requirements on barcode printing fonts and the printing process.

Some barcodes types support only numbers; others support uppercase letters and numbers, and some support the entire 128 characters of the ASCII character set (i.e. Characters 0 - 127).

Most barcode types employ a start character at the beginning of the symbol and a stop character at the end. In general these start and stop characters are not reproduced in human readable form under the barcode symbol. For example, in the symbol above the start and stop characters are * characters, but these do not appear under the symbol.

Most barcode types also employ one or more Check digits, which are used by the scanning equipment to ensure that the code has been read correctly. In some cases these check digits are optional, in others mandatory (i.e. the symbol will not scan if the check digit is omitted). Where present, check digits are usually the last characters in the code, just before the stop character, although they are not always shown in human readable form under the symbol.

A "self-checking" barcode symbology is one in which a printing defect cannot cause an incorrect character to be substituted for a misread character.

Barcode fonts

Barcode fonts enable you to print barcodes on graphics printers that can accept fonts (i.e. a Windows supported graphics printer or a PostScript printer which can accept downloaded fonts). Multi-user versions of dLSoft fonts are also licenced for embedding – so that the barcodes may be embedded in web pages or Portable Document Format (pdf) files. (See the application notes on our support website for information about font embedding www.dlsoft.com/support

However, the user needs to be aware of a number of factors that determine whether printed barcodes can actually be scanned correctly.

1. The thickness of bars and spaces in barcodes is important. Some types of barcode use only two thicknesses of bar, others use three thicknesses, and others more. Even when you print a barcode using a dLSoft barcode font, you need to ensure that the barcode has not been printed too small - so that within the resolution of the printer a single thickness bar has been printed at the same size as a double thickness bar. Consequently it is essential that you check that a printed barcode is readable using an appropriate scanner or reader.

   Barcodes printed by laser printer will, in general, be printed correctly, but codes printed by matrix printers must be reproduced at a large enough scale that the barcodes unit size is at least as large as the printer's pins.

2. Bar thickness reduction: Most dLSoft barcode fonts are supplied in three bar thicknesses. The Wide font (and its variants - names ending in W) should be suitable for most 600 and 1200 dpi laser printers - it has the bar/space ratio defined at its correct value. The Regular font (names ending in R) has all bars reduced by 8% and will probably be a better choice for 300 dpi laser printers and good quality ink-jets. The Narrow font (names ending in N) has all bars reduced by 16% and is supplied for users who will be creating master copy which will subsequently be printed using a wet ink technique (in which the ink spreads, so making each bar thicker than in the master). The narrow fonts should only be used if you know that a bar thickness reduction is required. Picking the wrong font usually produces unreadable images! If greater control of bar thickness is needed then an image creating system, such as dBarcode, will be required.

3. Many barcode types may use codes only of a specific length. (e.g. EAN13 requires 13 digits in the code – including the mandatory check digit). Some barcode types use specific digits of the code as a checksum - so not every combination of digits can form a legal barcode. dLSoft barcode fonts display as barcode characters the characters you specify. If your barcode type requires start and stop characters and a check digit character you must provide these character within the string of characters you wish to print as a barcode symbol. Furthermore most coding schemes are limited to 32 characters or less. The barcode types support by dLSoft barcode fonts are described in the remainder of this document, along with details of the Start/Stop character and check digit calculations.

4. Users should be aware that it is possible to print barcodes of a specific type and find that normal retail scanners are unable to decode the images. This does not necessarily mean that there is anything wrong with the barcode symbol. Most scanners aimed at the retail market are not programmed to interpret barcode codes reserved for other (e.g. military) use.

5. Space characters are treated as a special case in a number of Windows applications - the font character being ignored and a gap being placed where the space character was expected. In some applications the same behaviour is seen with the non-break space character (ASCII 160). dLSoft barcode fonts that contain a space character reproduce this character at ASCII 159, and this should be used whenever space characters give rise to a gap in barcodes. Even where the space character is not deliberately encoded it may appear as part of a check digit sequence.

6. The ASCII DEL character (ASCII 127) does not appear in the Windows character set. Those barcode fonts that need to use this character may use the repeated character at ASCII 223.

Installing your barcode font

To install your font place the disk containing the font in your disk drive. If your fonts are compressed into a self-extracting EXE file double-click on the file name to extract the fonts and supporting software into a directory of your choice. The dLSoft Setup program will install ONE group of TrueType fonts into your Windows system. If you require other fonts – such as lighter weight TrueType fonts, OpenType or PostScript fonts, then these must be installed using the Windows Control Panel.

Start the Windows Control Panel and double click on the Fonts icon. Then choose Add New Font or Install New Font and select the Fonts folder in the dFont installation directory that contains the extracted fonts.

A list of the fonts in this folder will be displayed and you can select the ones you wish to install. Unless you specifically require the OpenType or PostScript fonts (e.g. For Adobe Type Manager or downloading to a PostScript printer) select only the TrueType fonts for installation. Also, unless you know that you will be requiring your barcodes to be wet-ink printed, you will probably not need to install the Narrow fonts. If you are using a modern laser printer then you may need only the Wide fonts (those named DxWxx), while if your printer has less than 600 dpi resolution, or if your printer makes the bars of your barcodes too thick, you will need to install the Regular fonts (those name DxRxx).

Many font types are supplied in three different height/width ratios - to give you some choice over the aspect ratio of the printed barcode. Because many Windows programs were created when the choice of fonts styles was relatively limited (and may not be able to handle alternative names) we have used the common style names “Normal”, “Bold” and “Italic” to represent these styles.

The Normal style gives a height to width ratio that should be acceptable for most purposes. The Bold style reduces the height of the barcode symbol while keeping the width the same as with the Normal style. The Italic style reduces the width of the symbol while keeping the height that specified by the chosen point size.

Note that the Bold and Italic styles cannot be used at the same time - even if your Windows application permits the selection of such attributes.

Some fonts, such as the clocked fonts PostNet and RM4SCC, will not scan if the aspect ratio is not correct. These fonts do not have Bold or Italic components.

Getting started

UNLESS YOU READ THESE INSTRUCTIONS YOU WILL NOT PRODUCE BARCODES THAT SCAN CORRECTLY.

Once your fonts are installed you may like to try to print a barcode. Before you can print a barcode that has a significant chance of being scanned successfully you will probably need to read the part of this document which refers to the barcode type you need. However, for the moment lets just print a single barcode.

Start a suitable Windows application; anything capable of printing and displaying WYSIWYG will do, if in doubt try the Write or WordPad word processor that comes with Windows.

Into a new (or spare) document type 1234 then press return.

Select the text you have typed, and then change its font to the dLSoft barcode font listed in your fonts list, and its size to something like 36 point.

You should see a barcode image, and you should be able to print it.

Unfortunately you will not be able to scan the printed barcode at this stage because it does not contain the relevant start and stop characters, and in some cases you will also require a check digit. The characters required for these may be determined by running the dFont Helper program. The source code for this program is included for users who wish to use sections of the code to automate this procedure in programmable applications, such as Microsoft Office, Visual FoxPro, Borland Paradox, etc. A number of supporting tools are also provided for automating the process.

For some of our fonts the ` (grave symbol - top left on most UK keyboards) will work as the start character and the ¬ (shifted grave) will work as the stop character. For others the open bracket (will act as the start character and) will act as the stop character. For all of our fonts (except EAN/UPC) the section symbol § (Chr$(167)) will act as the start character and ¬ (Chr$(172)) will act as the stop character [For fonts which encode all the characters on the keyboard (e.g. Code 93 and the 128s) the special characters such as start and stop may be entered by holding down the <Alt> key and type 0xyz using the numeric keypad, where xyz is the value of the character. When these special characters are being created by programming, then use Chr$(xyz) in Basic or the equivalent in other languages.]

Position the cursor before the beginning of the barcode image on screen and type the start character. DO NOT press the enter key. Then position the cursor just after the end of the barcode and type the stop character.

You barcode will be a little longer now, but when its printed this time it should scan - even if the scanner complains that the check digit is wrong. To get that right you will need to start reading the section on your barcode type.

Note that there are some characters that cannot be printed by Windows. For example ASCII 128 and 129 do not print in all versions of Windows. Barcode fonts which would otherwise use such characters have second copies of the characters at alternative ASCII values. See the section dealing with individual barcode types for details.

Note also that a number of Windows programs do not actually use the space character, ASCII 32, for spaces typed at the keyboard. Some word-processors micro space characters by specifying the position of the start of the next word instead of sending the printer a space character. When a barcode font is used spaces will show up as spaces in such cases, and not as the barcode symbol representing ASCII 32. To overcome this problem the space character is reproduced at an alternative ASCII value. See the section dealing with individual barcode types for details.

dFont Helper

The dFont Helper program enables you to see which characters need to be used to create a barcode representing a particular Code, including any start, stop and check characters.

To use dFont follow this simple procedure:

From the drop down list of barcode types select the code type you require.

If you require the normal check digits in your symbol, check the “Include check digits” checkbox. [Note: for code types which require mandatory check digits the check digits will be included whether you check this box or not.]

Type the code you require into the box labelled “Enter Code”.

Push the recalculate button. The string you need to create the symbol with your dLSoft barcode font will appear in the box labelled “String required”.

If you wish you can copy the string required to the clipboard for pasting into another Windows application.

For characters which are not normally visible on screen, one of the alternative (visible) characters used in the dLSoft barcode font will be displayed along with the xyz value which may be used to produce that character - either using Chr$(xyz) in Basic or by holding down the Alt key and type 0xyz on the numeric keypad.

Note that Code 128 and EAN 128 barcode types have three subtypes. The dFont Helper application does not automatically handle subtype changes – you need to insert the subtype characters yourself. The dFont Developers Kit is available for developers who need to automate this process.

Automation

If you are planning to use many barcodes within programmable spreadsheet or database applications, then you will probably wish to automate the process of inserting the start and stop codes and the check digits.

This should prove relatively straightforward because most applications have a programming language. You can copy sections of the dFont helper program supplied with your font into the macro or module section of your application.

The basic principle is to take the string you wish to convert into a barcode (STRING1$) and add to it the start character, check digit (if required) and stop character, to create a second string (STRING2$) which then appears on forms or reports and is set to display in your barcode font at a suitable size. In most cases the font characteristics are set permanently (e.g. within a properties window), while the creation of the string to display is handled by a macro or module.

So your module may contain code such as

STRING2$=start$ + STRING1$ + check$ + stop$

and STRING2$ becomes the content of the target field or cell.

Note that Code 128 and EAN 128 barcode types have three subtypes. The simple dFont Helper application does not automatically handle subtype changes – you need to insert the subtype characters yourself. The software described below is suitable for developers who need to automate this process.

Several additonal items of supporting software are provided with your font. The font package include both a DLL which can be called from most languages, and an Active-X control designed primarily for use with Visual Basic, VBA or managed code in Visual Studio.NET, and a UFL designed for calling within Crystal Reports. These items are described in detail below.

Barcode types

Code 39

Code 39 is a discrete “self-checking” code (i.e. it has inter-character gaps) and is undoubtedly the most commonly used code outside retail labelling.

The standard Code 39 symbology supports upper case letters, numbers and the following additional characters; SPACE, HYPHEN/MINUS (-), POINT(.), PLUS(+), DIVIDE(/), DOLLAR($) and PERCENT(%). No other characters are allowed!

The Extended Code 39 symbology supports the full ASCII character set - although it does this at a price by using two barcode characters to represent each code character (the a is represented in the barcode by +A). Extended Code 39 symbols can thus become rather long, creating difficulties for the scanner.

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing in Code 39 [but NOT in Extended Code 39] the open bracket (may be used as the start character, and the close bracket) may be used as the stop character.

Code 39 symbols are self-checking so are normally used without a check digit. However, they may employ an optional check digit as the last character before the stop character.

The check digit is calculated using a modulo 43 algorithm, using the value of each character as follows:

character	value
0 - 9	0-9
A-Z	10 - 35
-	36
.	37
(space)	38
$	39
/	40
+	41
%	42
*	43 (i.e. ignored)

The check digit is calculated by summing the character values to give Sum, and using the formula

checkdigitvalue = Sum Mod 43

The check digit is then the corresponding character in the table above.

The dFont program provided with the dLSoft Barcode fonts calculates check digits and for both Code 39 and Extended Code 39, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Because the space character does not print from some Windows applications the space character is reproduced at ASCII 159 and 160.

If Extra1 is true the start and stop characters are shown as an asterisk in font which includes the human-readable form.

Note that Code39 is supplied as a different font from Extended Code 39 - even though the upper case letters and numbers are the same in both fonts. This is because some characters are not the same. For example, + in Code 39 is represented as /K in Extended Code 39 - to avoid the sequence +A being misinterpreted.

For convenience of handling Extended Code 39 characters with ASCII values of less than 32 are reproduced in the font at the character positions chr$(192) - chr$(223), and ASCII 127 is reproduced at ASCII 191.

Code 93

Code 93 is a continuous symbology and is much more compact than Code 39 to which it is closely related.

The standard Code 93 symbology supports upper case letters, numbers and the following additional characters; SPACE, HYPHEN/MINUS (-), POINT(.), PLUS(+), DIVIDE(/), DOLLAR($) and PERCENT(%). No other characters are allowed!

The Extended Code 93 symbology supports the full ASCII character set - although it does this at a price by using two barcode characters to represent each code character (the a is represented in the barcode by a special character followed by A).

Code 93 symbols are generally contain two check digits calculated using a modulo 47 algorithm, using the value of each character as follows:

character	value
0 - 9	0-9
A-Z	10 - 35
-	36
.	37
(space)	38
$	39
/	40
+	41
%	42
shift 1 Chr$(168) ¨	43
shift 2 Chr$(169) ©	44
shift 3 Chr$(170) ª	45
shift 4 Chr$(171) «	46

The check digits are calculated by summing the weighted character values to give sums t1 and t2,

```
j = ((n - i ) Mod 20) + 1   ' check digits weighted sums into t1 and t2
t1 = t1 + j * chn
j = ((n - i + 1) Mod 15) + 1
t2 = t2 + j * chn
```

where n is the number of characters in the string (not including the check digits), i is the position in the string (starting at 1 on the left), and chn is the value of the ith character taken from the table above.

Once the sums have been calculated the modulo 47 remainders are the values of the two check digits, the character for which are given in the table above

```
t1 = (t1 Mod 47)              ' modulo 47 checksum
t2 = ((t2 + t1) Mod 47)
```

The two check digit characters must be appended to the code, which becomes

§code(t1)(t2)¬

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

The dFont program provided with the dLSoft Barcode fonts calculates check digits and for both Code 93 and Extended Code 93, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Because the space character does not print from some Windows applications the space character is reproduced at ASCII 159 and 160.

For convenience of handling characters with ASCII values of less than 32 are reproduced in the font at the character positions chr$(192) - chr$(223), and ASCII 127 is reproduced at ASCII 191.

EAN and UPC

The EAN and UPC codes are the most common retail codes and one of these is likely to be found on virtually every item in a supermarket.

The standard EAN-13 code contains 13 digits and is a superset of the UPC-A code (which contains 12 digits). Confusingly the 13th digit of the EAN code is actually the leftmost digit, and this is not encoded in bars and spaces, but in the combination of coding rules used to encode the next six digits. The UPC coding of the first six digits uses only one of the coding rules of the EAN standard. As a result, EAN and UPC codes are very similar and can be read by the same scanners.

One of the problems with this approach is that there is not a one-to-one correspondence between a digit and a pattern of bars and spaces; for example, the digit 0 may be encoded in three different patterns in an EAN symbol. The three different patterns are called Sets A, B and C. Consequently three different characters are required to represent 0 within a barcode font - the dLSoft EAN/UPC font uses **A**, **a** and **0**, for set A, B and C respectively as shown in the Character sets Table below.

One of the cosmetic features of EAN and UPC symbols is that the start and stop characters, and the “centre marker” character (which merely separates the left hand six characters from the right hand six) are usually printed slightly longer than the encoded characters. The elongation of these bars has no effect on the machine readability of the symbol.

The dLSoft EAN/UPC font provides two sets of characters, one which contains just the bars and spaces and in which the start, stop and centre marker characters are the same height as the coded characters, and one set in which the barcode characters have their numeric values reproduced underneath and in which the start, stop and centre marker are elongated.

In both cases the normal start and stop characters are [and] respectively and the centre marker is /. For the special case of UPC-E codes the start and stop characters are { and } respectively. UPC-E does not use a centre marker.

EAN and UPC codes both require a check digit, the value of which may be obtained as follows:

Starting with the rightmost digit (excluding the check digit), find the sum of the alternate digits; multiply this sum by 3 and add to the sum of the remaining digits. The check digit is the number which when added to this result gives a multiple of 10.

The right-hand six characters in both EAN and UPC symbols are encoded using characters from Set C. The left hand six digits in UPC codes are encoded using Set A. The left-hand six digits (excluding the 13th) in EAN codes are encoded using a combination of Sets A and B - and the combination of sets is used to encode the 13th digit as follows:

Digit	n1	n2	n3	n4	n5	n6
0	A	A	A	A	A	A
1	A	A	B	A	B	B
2	A	A	B	B	A	B
3	A	A	B	B	B	A
4	A	B	A	A	B	B
5	A	B	B	A	A	B
6	A	B	B	B	A	A
7	A	B	A	B	A	B
8	A	B	A	B	B	A
9	A	B	B	A	B	A

The dFont program provided with the dLSoft Barcode fonts calculates check digits and encoding patterns for both EAN and UPC-A codes, and their minimal size relatives (EAN-8 and UPC-E), and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Users wishing to use their own encoding schemes should note that the UPC-E stop character is NOT the same as UPC-A or EAN stop characters (because it needs to indicate the direction of scanning). The UPC-E stop character is provided in the dLSoft EAN/UPC fonts as }.

Supplementary codes

EAN and UPC-A codes may include 2 digit or 5 digit supplementary codes (Add-on codes) that may be encoded using character sets A and B. However, these are normally reproduced with their number values above the bars - rather than underneath the bars as in the case of the main code. In order to accommodate this feature, character sets A and B are also provided in the dLSoft barcode font as character sets ExtA and ExtB as shown in the Character sets Table below.

2-digit supplementaries do not use a check digit for the supplementary, but the encoding of the 2 digits does depend on the value of the add-on. The character sets for the two digits depends on the remainder of dividing the value of the add-on by 4:

Remainder	lefthand digit	righthand digit
0	A	A
1	A	B
2	B	A
3	B	B

5-digit supplementaries do use a check digit calculated by taking three times the sum of digits 1, 3 and 5, plus nine times the sum of digits 2 and 4, and using the units value of the result. Encoding of the 5 digits is then determined by the value of the check digit as follows:

Check digit	n1	n2	n3	n4	n5
0	B	B	A	A	A
1	B	A	B	A	A
2	B	A	A	B	A
3	B	A	A	A	B
4	A	B	B	A	A
5	A	A	B	B	A
6	A	A	A	B	B
7	A	B	A	B	A
8	A	B	A	A	B
9	A	A	A	A	A

This table (and the check value calculation) differs from that used for the main EAN/UPC barcode symbol.

Light margin indicators

EAN barcodes are normally prefixed by the human-readable character form of the 13th digit (which is actually the leftmost digit) and this does not appear under the bars, but in the light margin before the barcode symbol - where it indicates the region in which no other printing should appear. The dLSoft barcode font includes the numerals 0-9 without bars in the ASCII characters 33-41 (the Prefix character set, see below), so that the 13th digit may be printed in the light margin. The right-hand light margin is indicated by a > symbol.

The Prefix character set may also be used to print the leading 0 and the check digit for UPC-A codes. These are normally printed at a smaller size than the other human readable characters, so are provided in a UPC Prefix character set (ASCII 161-170). To obtain the correct spacing, the check digit human readable character should be preceded by a space.

The light margin indicators for EAN-8 codes are < on the left and > on the right, and the light margin indicator for EAN supplementary codes is > at the level of the supplementary characters, and provided in the font as . (dot). To obtain the correct margin position this should be preceded by a space.

EAN/UPC Character Sets Table

Code	Formatted	Unformatted (ASCII)
Start	[	219
Stop	]	221
Centre Marker	/	175
Add-on start	+	171
Add-on inter-character	-	173
EAN Prefix	ASCII 33-42	
Margin Indicators	< > and .	
UPC-E start	{	251
UPCE-Stop	}	253
UPC Prefix/Check	ASCII 161-170	

Fully formatted Codes

Digit	Set A	Set B	Set C	ExtA	ExtB	Prefix
0	A	a	0	M	m	!
1	B	b	1	N	n	"
2	C	c	2	O	o	#
3	D	d	3	P	p	$
4	E	e	4	Q	q	%
5	F	f	5	R	r	&
6	G	g	6	S	s	'
7	H	h	7	T	t	(
8	I	I	8	U	u	)
9	J	j	9	V	v	*

Unformatted Codes (ASCII values)

Digit	Set A	Set B	Set C	ExtA	ExtB
0	193	225	176	205	237
1	194	226	177	206	238
2	195	227	178	207	239
3	196	228	179	208	240
4	197	229	180	209	241
5	198	230	181	210	242
6	199	231	182	211	243
7	200	232	183	212	244
8	201	233	184	213	245
9	202	234	185	214	246

Code B

Code B is a numeric only code and does not use a check digit, so there is no Code B entry in the dFont program.

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing the open bracket (may be used as the start character, and the close bracket) may be used as the stop character.

Code 11

Code 11 encodes only the number and a minus sign.

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing the open bracket (may be used as the start character, and the close bracket) may be used as the stop character.

Code 11 provides for a single check digit when the code length is less than 12 characters, and two check digits when the code length is 12 characters or more.

In each case the check digit is calculated using a weighted modulo 11 algorithm. For the first check digit the integer value of each character is multiplied by it the rightmost digit of the character's position (i.e. ranging from 1 to 9 and then restarting) in the code (starting from the right) and with the - sign given the value 10. The sum of these values is calculated and then divided by 11 and the remainder is used as the check digit.

```
‘ xx$ is the code  First check digit calculation
chn1=0:  chd$=““
n= len(xx$)
For i = n To 1 Step -1
   y$ = Mid$(xx$, i, 1)
   If y$ = "-" Then z = 10 Else z = Asc(y$) - 48'  value of character
   chn1 = chn1 + j * z
   j = j + 1
   If j > 10 Then j = 1
   Next I
t = chn1 Mod 11
If t < 10 Then
   chd$ = chd$ + Chr$(t + 48)' convert to ASCII character 0-9
   Else
   chd$ = chd$ + "-"
   End If
```

The second check digit is calculated in the same way except that the character position weightings range from 1 to 8 and then restarted. The first check digit character IS included as the rightmost character in the calculation for the second check digit.

The dFont program provided with the dLSoft Barcode fonts calculates check digits for Code 11 codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Codabar

Codabar is a self-checking, discrete symbology which encodes the numbers and the $: / . + - characters. Codabar is unusual in that the barcode characters have differing widths. Also the symbology allows for several combinations of start and stop character.

Start and stop characters are usually A for start and C for stop,

This code must begin with a start character and must end with a stop character. Start and stop characters are usually A for start and C for stop, and dFont reproduces these as the start character § (Chr$(167)) and the stop character ¬ (Chr$(172)).

However, any alternative may be chosen from the list below - although not all Codabar readers recognise all these combinations.

Allowed start/stop codes: A B C D E N T *

All codes prepared to create a Codabar symbol must include a start and stop character.

Codabar does not use check digits, so there is no entry for Codabar in the dFont program.

Interleaved 2-of-5

Interleaved 2-of-5 (I 2/5) is a high-density continuous symbology that encodes numeric digit pairs only. As this covers the sequence 00 - 99, clearly all digits pairs must be translated into single characters for this symbology to be represented by a font.

It should be noted that I 2/5 symbols require an even number of digits. The convention is that if an odd number of digits is to be encoded a LEADING 0 is attached to the code. This will of course decode when the symbol is subsequently scanned.

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing the open curly bracket { may be used as the start character, and the close curly bracket } may be used as the stop character.

Digit pairs are represented in the dLSoft fonts by characters with an ASCII value corresponding to value of the digit pair. Thus the digit pair 65 is represented by A, because this has an ASCII value of 65.

ASCII values below 32 are not represented by readable characters and so for convenience of handling characters with ASCII values in the range 0 - 32 are reproduced in the font at the character positions chr$(192) - chr$(223). ASCII 32 is reproduced at ASCII 159 and 160.

I 2/5 is often used with a Modulo 10 check digit in the final position.

The dFont program provided with the dLSoft Barcode fonts calculates the check digit for I 2/5 codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Unfortunately I 2/5 suffer from the fact that a partial scan is likely to decode as a valid (although shorter) code, and the presence of a check digit does little to overcome this. However, some scanners may be set to accept a fixed number of digits and record an error if less than that number decode.

Interleaved 2-of-5 is the same encoding scheme and bar pattern as the ITF and ITF-6 outer case markers used in distribution, although the latter codes are normally printed with large bearer bars (although this is to spread the pressure during printing and has no effect on the scanning).

MSI/Plessey

MSI, also know as Modified Plessey Code, is a relatively weak code which is inefficient in its use of space.

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing the open bracket (may be used as the start character, and the close bracket) may be used as the stop character.

Normally this code has a single Modulo 10 check digit, somewhat peculiar algorithm used for this is include in the dFont program. However, there are two variations of a double check digit form in common use. One uses a (standard) Modulo 11 check digit before the Modulo 10 check digit, the other uses two Modulo 10 check digits.

If Extra1 is set then a Modulo 10 check digit is calculated and inserted before the normal checkdigit.

If Extra2 is set then a Modulo 11 check digit is calculated and inserted before the normal checkdigit.

Some scanning equipment cannot read both forms. (in fact some scanning equipment cannot read either of the two check digit forms). Check your scanners documentation to ensure that you choose an appropriate combination.

The dFont program provided with the dLSoft Barcode fonts calculates the single Modulo 10 check digit for MSI codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Royal Mail RM4SCC

The RM4SCC is a clocked code which uses a central track made up of evenly spaced small bars, with the data encoded by the bars extending above and/or below the clocking bars.

The character set for RM4SCC consists of upper case letters and numbers only. The dLSoft barcode font includes a zero width space character, so that text copied from conventionally spaced postcodes may be translated into a barcode symbol.

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing the open bracket (may be used as the start character, and the close bracket) may be used as the stop character.

The check digit is normally obtained by a table-lookup procedure, but a simple algorithm may also be used. The character values are shown in the table.

Character	Value
0 - 9	0 - 9
A - Z	10 - 35

The character values may be determined directly from the corresponding ASCII values, and then two parameters calculated (tu and tl) representing the row and column of the 6*6 table-lookup.

```
For i = 1 to len(code$)
   z=ASC(MID$(code$,I,1))
   If (z < 65) Then
        z = z - 48
      Else
        z = z - 55
       End If
```

```
    ii = Int(z / 6)
    If (ii >= 5) Then k = 0 Else k = ii + 1
    tu = tu + k' row ref
    ii = Int(z - ii * 6)
    If (ii >= 5) Then k = 0 Else k = ii + 1
    tl = tl + k' col ref

  Next i
```

Then the check digit value is calculated

```
  tu = tu Mod 6: If (tu = 0) Then tu = 6    ' checksum
  tl = tl Mod 6: If (tl = 0) Then tl = 6
  k = (tu - 1) * 6 + tl - 1
```

And finally the value is converted to ASCII

```
  If (k < 10) Then
     chn = (k + 48)
  Else
     chn = (k + 55)
checkchar$ = Chr$(chn)
```

The dFont program provided with the dLSoft Barcode fonts calculates check digits for RM4SCC codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Some other European countries use a virtually identical 4 state clocked code for barcoding the mail. In some of these countries the start and stop characters are NOT used, and in some case the check digit is not used. In such cases the text of the postcode may be reproduced directly using the dLSoft barcode font.

In all cases the size of the printed barcode is important, and a font size of 20 point should always be used.

PostNet

PostNet codes are the clocked codes used in the US mail system. There are three types of PostNet code (identified as A, C and C') that differ in the number of characters encoded. These codes are based on the US ZIP code system and use numbers only.

Code type	number of code characters
A	5
C	9
C'	11

This code must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

For convenience in typing the open bracket (may be used as the start character, and the close bracket) may be used as the stop character.

For convenience of use the dLSoft barcode font also allows the (to used as a start character, and the) to be used for the stop character.

PostNet codes require a single check digit that may be calculated by adding up the numerical value of each character in the code and setting the check digit equal to the character that represents the number that must be added to the sum to give a multiple of 10.

The dFont program provided with the dLSoft Barcode fonts calculates check digits for PostNet codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

Code 128/ EAN-128

Code 128 and EAN-128 are modern very high density coding schemes that are somewhat more complex than most other schemes. They have three coding schemes each and permit the inclusion of special characters not present on the keyboard. In general, because of the presence of non-printing characters, 128 type codes are best reproduced using the fonts that do NOT include text under the bars, such human readable text being added separately.

Spaces may be stripped from the text provided for input by setting the Extra1 property to true.

There are 105 characters in the character set, but each one may be used to represent more than one human readable character - see the table below. For example, the character with a Code 128 value of 77 represents a carriage return (CHR$(13)) in coding scheme A, the lower case letter m in scheme B, and the digit pair 77 in scheme C. This may be created in Basic using CHR$(77+32) = CHR$(109) [the coding scheme starts at ASCII 32].

Generally scheme B is used by default. For EAN-128 scheme C is used for any code that has numbers in the first four digits (as recommended by the ANA). The scheme must be specified by making the first character one of the start characters specified below. The stop character is always the same and is provided as ¬ (CHR$(172)) in the dLSoft barcode font.

In an attempt to simplifying the task of encoding characters the dLSoft 128 font provides the entire ASCII character set; obviously some of the 105 Code 128 characters are duplicated. ASCII values from 0 to 31 are available within the font as duplicates of the ASCII values 192 to 223. So setting CHR$(27) into the barcode font will produce the correct pattern, as will setting the character Û. The character Û has an ASCII value of 219 = 192 + 27.

Some of the control characters (values above 127) are not defined in some text fonts - so tend to show up as identical blocks. To make strings containing these characters easier to read before they become barcode symbols, the characters from CHR$ 128 to 137 are duplicated at 161 to 170. The latter are defined, as you can verify by holding down the ALT key and typing 0161 in Notepad.

The space character, ASCII 32, is reproduced at ASCII 159 and 160, and the ASCII 127 character is reproduced at ASCII 191.

The 128 coding standard allows the scheme to be changed within the symbol. Thus a symbol which starts with scheme C may be changed to scheme B part way through by using one of the special function codes (Code B). It is beyond the scope of this document to provide a tutorial in the use of 128 codes, and the user who needs more information should contact his national Article Numbering Association for details of the schemes and practices adopted locally.

The Code 128 character set is reproduced in the table below.

128 codes have a mandatory check digit that is calculated using a Modulo 103 of the weighted sum of the other characters in the code, where the weightings are determined by the character position (starting from the left). The result must have 32 added to it to enable the check digit character to be inserted as CHR(x).

The dFont program provided with the dLSoft Barcode fonts calculates check digits for Code 128 codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

The dFont Helper application does not automatically handle subtype changes – you need to insert the subtype characters yourself. The dFont Developers Kit is available for developers who need to automate this process.

EAN-128

EAN-128 is based on the Code 128 symbology, and scanning requirements are identical. The EAN-128 code is distinguished from Code 128 by having a Function 1 character immediately after the start character.

The value of the Function 1 character IS taken into account in creating the check digit.

The dFont program provided with the dLSoft Barcode fonts calculates check digits for EAN-128 codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

The dFont Helper application does not automatically handle subtype changes – you need to insert the subtype characters yourself. The dFont Developers Kit is available for developers who need to automate this process.

Code 128 character code, ASCII Table and font values

A	B	C	Value	ASCII	additional
space	space	00	0	32	159
!	!	01	1	33	
“	“	02	2	34	
#	#	03	3	35	
$	$	04	4	36	
%	%	05	5	37	
&	&	06	6	38	
‘	‘	07	7	39	
(	(	08	8	40	
)	)	09	9	41	
*	*	10	10	42	
+	+	11	11	43	
,	,	12	12	44	
-	-	13	13	45	
.	.	14	14	46	
/	/	15	15	47	
0	0	16	16	48	
1	1	17	17	49	
2	2	18	18	50	
3	3	19	19	51	
4	4	20	20	52	
5	5	21	21	53	
6	6	22	22	54	
7	7	23	23	55	
8	8	24	24	56	
9	9	25	25	57	
:	:	26	26	58	
;	;	27	27	59	

<	<	28	28	60	
=	=	29	29	61	
>	>	30	30	62	
?	?	31	31	63	
@	@	32	32	64	
A	A	33	33	65	
B	B	34	34	66	
C	C	35	35	67	
….. all uppercase letters					
X	X	56	56	88	
Y	Y	57	57	89	
Z	Z	58	58	90	
[	[	59	59	91	
\	\	60	60	92	
]	]	61	61	93	
^	^	62	62	94	
_	_	63	63	95	
NUL	`	64	64	0	192
SOH	a	65	65	1	193
STX	b	66	66	2	194
ETX	c	67	67	3	195
EOT	d	68	68	4	196
END	e	69	69	5	197
ACK	f	70	70	6	198
BEL	g	71	71	7	199
BS	h	72	72	8	200
HT	i	73	73	9	201
LF	j	74	74	0	202
VT	k	75	75	11	203
FF	l	76	76	12	204
CR	m	77	77	13	205
SO	n	78	78	14	206
SI	o	79	79	15	207
DLE	p	80	80	16	208
DC1	q	81	81	17	209
DC2	r	82	82	18	210
DC3	s	83	83	19	211
DC4	t	84	84	20	212
NAK	u	85	85	21	213
SYN	v	86	86	22	214
ETB	w	87	87	23	215
CAN	x	88	88	24	216

EM	y	89	89	25	217
SUB	z	90	90	26	218
ESC	{	91	91	27	219
FS	|	92	92	28	220
GS	}	93	93	29	221
RS	~	94	94	30	222
US	DEL	95	95	31	223 and 191
Func3	Func3	96	96	128	161
Func2	Func2	97	97	129	162
Shift	Shift	98	98	130	163
Code C	Code C	99	99	131	164
Code B	Func4	CodeB	100	132	165
Func4	Code A	Code A	101	133	166
Func1	Func1	Func1	102	134	167
StartA	StartA	StartA	103	135	168
StartB	StartB	StartB	104	136	169
StartC	StartC	StartC	105	137	170
Stop	Stop	Stop	106	138	172

Telepen

The Telepen coding scheme has a number of variants - Telepen Numeric, Telepen ASCII, and either with various begin and end codes. You will need to determine which variant you are attempting to create barcodes for.

The Telepen ASCII scheme provides the full ASCII character set. For convenience of handling characters with ASCII values of less than 32 are reproduced in the font at the character positions chr$(192) - chr$(223). The space character, ASCII 32, is reproduced at ASCII 159 and 160.

Some Telepen symbols require the first character (after start) be an ASCII Shift In character, and the last character (before stop) to be a Shift Out character. The ASCII ESC character is required on some Telepen Numeric systems as the first character after to the start character. If you are using a Telepen system your system manual describes the scheme you require.

All Telepen codes must begin with a start character and must end with a stop character. The start character is § (Chr$(167)) and the stop character is ¬ (Chr$(172)).

Telepen ASCII normally uses a Modulo 127 check digit as shown below. However, there seem to be a wide variety of schemes for calculating Telepen check digits and the user is recommended to consult his Telepen system documentation.

```
n = Len(xx$)  ‘ xx$ is the code
t1 = 0
For i = 1 To n
   y$ = Mid$(xx$, i, 1)
      z = Asc(y$) ' ASCII value of character
      If (z > 127) Then z = z - 128
      t1 = t1 + z
      Next i
```

```
chn = t1 Mod 127  ' checksum
If chn > 0 Then chn = 127 - chn
If (chn < 32) Then chn = chn + 192 ‘ make it visible in font
chd$ = Chr$(chn)   ‘ set check character
```

The dFont program provided with the dLSoft Barcode fonts check digit for Telepen codes, and the source code is provided and may be adapted to the users needs - for example, for automating the calculation in a spreadsheet or database.

The extra Telepen_N barcode type provides the begin data end sequence required on some systems for Telepen Numeric.

Supporting software

The dFont DLL

The dFont DLL is a Windows Dynamic Link Library which may be called from a variety of programs and environments. It accepts 4 parameters. Its basic role is to be given the characters you wish to turn into a barcode, and return the character which when printed in the appropriate font, will represent that barcode - including the start and stop characters and the check digit.

The 32 bit version DFONT32.DLL is designed to be called by 32 bit applications (such as programs created with Visual Basic). The DLL must be placed either in the directory that contains the application or in the Windows system directory (for Windows 9x) or the Windows System32 directory for Windows NT/2000/XP.

If the application you are using cannot find the required DLL an error message will be generated when calls are made.

Three groups of function calls are supported:

Group 1 calls generate a barcode string directly from data and barcode properties totally contained within the function call.

Group 2 calls generate a barcode string from a data string, using barcode properties specified in advance that are remembered as default properties.

Miscellaneous calls serve other functions but do not generate a barcode string.

Group 1 calls

The Group 1 calls generate barcode strings that require the barcode properties (code type, automatic check digit calculation and Extra flag settings) to be passed as function parameters for each call. This allows the programmer full control over the barcode characteristics.

dBarFont() function

The DLL contains a function named dBarFont()

Most applications which will call this function require the function name and the name of the DLL to be declared within the application.

VB declaration

Declare Function dBarFont Lib "DFONT32.DLL" (ByVal icode As Integer, ByVal xin As String, ByVal xout As String, ByVal flags As Integer) As Integer

C declaration

int FAR PASCAL dBarFont(int icode, LPSTR sztin, LPSTR sztout, int Flags);

to appear at the start of the program (or in the declarations section/header file).

Note that some systems are case sensitive and require the function name to be specified as dBarFont - not dbarfont.

The function returns 0 if successful, or an errorcode number if an error is detected. The errorcode numbers are listed in the table below.

Parameters

dBarFont has four parameters, two of which may be obtained automatically by calling the dBFAsk() function.:

icode - type Integer (or int)

This specifies the type of barcode to be created. The dFont DLL supports 16 different barcode types (24 if supplementaries are included). The type and the corresponding values of icode are shown in the Barcode type table at the end of this manual.

Xin - type String (or LPSTR)

This parameter is a pointer to the text (a null terminated string) which will be turned into characters which will make the barcode when displayed or printed in the appropriate font. While the string actually allows 100 characters to be present, the chances of scanning a normal barcode with so many characters is rather slim.

Xout - type String (or LPSTR)

This parameter is a pointer to the text variable (a null terminated string) which will receive the characters which will make the barcode when displayed or printed in the appropriate font. This variable must have been created and initialised BEFORE the call, with enough character positions to receive the output. A safe rule of thumb is to allow 8 characters more than that provided as Xin.

In C/C++ the Xout parameter can be the name of a char array. In Visual Basic a String variable which has been initialsed with say Xout$=“ “ will suffice.

Flags - type Integer (or int)

The flags parameter combines three flags which allow various features of the barcode to be specified, and may be calculated using the formula:

flags = icheckdigit + 2 * iextra1 + 4 * iextra2

The three flags, icheckdigit, iextra1 and iextra2 may only have values of 0 or 1. Any other value may generate unexpected results.

The icheckdigit flag enables the user to specify whether a check digit is to be calculated automatically by the DLL (and included in the output string).

If the icheckdigit flag is 1 then the DLL will calculate the check digit. If icheckdigit is 0 it will not.

The two flags iextra1 and iextra2 may normally be set to 0. However, their functions differ for different barcode types and the effects of setting values to 1 will be mentioned under the specific code type.

The Code type parameter values are shown in the **Barcode types table** at the end of this manual.

Calling dBarFont()

To call the dBarFont function just use a section of code analogous to this:

(shown for Visual Basic)

```
Dim xin$
Dim yout$
Dim icode%, iflags%

xin$ = Form1.Text1.Text
icode% = 0 ‘ for Code 39
```

```
icheck = 1 ‘ check digit required
ix1=0: ix2=0: ‘ no other flags

iflags% = icheck + 2 * ix1 + 4 * ix2:
yout$ = String$(128, 0)

i% = dBarFont(icode%, xin$, yout$, iflags%)

Form1.Text1.Text = yout$ ‘ to display barcode
```

Of course, you do need to display the Text box (Form1.Text1 in this example) using the appropriate font!

Many more examples in different systems are included on the distribution disks or are available from our web site.

dBarFonth2() Function

The dBarFont2() function is the same as the dBarFont() function except that it returns the length of the xout string, or –1 in the case of an error.

dBarFontc() Function

VB declaration

Declare Function dBarFontc Lib "DFONT32.DLL" (ByVal icode As Integer, ByVal xin As String, ByVal flags As Integer) As Integer

C declaration

int FAR PASCAL dBarFontc(int icode, LPSTR sztin, int Flags);

where the parameters have the same meaning as for dBarFont().

The dBarFontc function generates the barcode characters and copies them to the clipboard. Note that the characters pasted from the clipboard into another application will need to have their font property set.

The function returns –1 in the case of an error, or 0 in the case of success.

dBarFonth() Function

The dBarFonth() function is the same as the dBarFont() function except that it also returns the human readable string for the barcode in the human parameter. This is useful for obtain the human readable form including check digit(s)

VB declaration

Declare Function dBarFonth Lib "DFONT32.DLL" (ByVal icode As Integer, ByVal xin As String, ByVal xout As String, ByVal flags As Integer, ByVal human As String) As Integer

C declaration

int FAR PASCAL dBarFonth(int icode, LPSTR sztin, LPSTR sztout, int Flags, LPSTR human);

dBarFonts() Funtion

VB declaration

Declare Function dBarFonts Lib "DFONT32.DLL" (ByVal icode As Integer, ByVal xin As String, ByVal flags As Integer, ByVal fout As String) As Integer

C declaration

int FAR PASCAL dBarFonts(int icode, LPSTR sztin, int Flags, LPSTR fout);

where fout contains the fully qualified filename of the text file to receive the barcode text, and the other parameters have the same meaning as for dBarFont().

The dBarFonts function saves the barcode text to a file name specified in the fout parameter.

The function return the number of bytes copied to the file, or –1 in the case of an error.

dBFAsk() function

The dFont DLL contains a function which enables the barcode type required to be selected from a popup dialog box. This is particularly useful where several barcode types are being supported. It is not necessary to call this function – the parameters used in the dBarFont() calls may be fixed – but when dBFAsk() is called the CodeType and Flags parameters it returns may be used in a following call to the dBarFont() function.

Most applications which will call this function require the function name and the name of the DLL to be declared within the application.

VB declaration

Declare Function dBFAsk Lib "DFONT32.DLL" (icode As Integer, flags As Integer) As Integer

[Note that ByVal should NOT be used in this case]

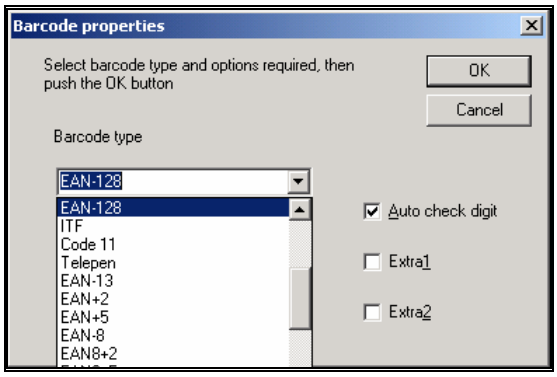
C declaration

int FAR PASCAL dBFAsk(LPINT pCodetype, LPINT pFlags);

to appear at the start of the program (or in the declarations section/header file).

Note that some systems are case sensitive and require the function name to be specified as dBFAsk - not dbfask.

When dBFAsk() is called a dialog box is displayed, showing a drop down list of the barcode types supported, and checkboxes allowing the Auto Check Digit and Extra parameters settings to be specified



The sample applications supplied with the dFont Developers Kit include source code illustrating the use of dBFAsk() calls.

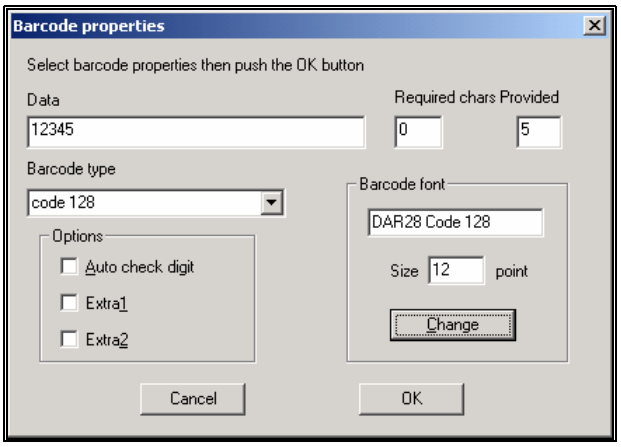
Group 2 calls

Group 2 calls generate barcode strings that use the current default barcode properties, such as code type, automatic check digit calculation, Extra 1 and Extra 2 flag settings and the barcode font, as specified using the BarAsk dialog. The default settings will continue to be valid until reset either by using the dialog or, in the case of the barcode font, by using the SetBarName function. These functions allow for the simplest operation where the barcode characteristics are constant for long periods.

BarAsk

The BarAsk function display a dialog box that allows the required barcode properties to be set and the font used to display/print the barcode to be selected. When the OK button is pushed the barcode properties (except the barcode data) selected become the default barcode properties for future barcodes generated by calls to the BarFont functions (see below).

The barcode data may be displayed in the dialog's data window by passing the data as the text parameter, and the data window may be enable for editing by setting the bEnable parameter to a non-zero value. If bEnable is non-zero then the text data displayed in the dialog may be edited and the text parameter contains the modified data if the dialog is closed by pushing its OK button.



VB declaration

Declare Function BarAsk Lib "DFONT32.DLL" (ByVal text As String, ByRef bEnable As Long,) As Long

C declaration

int FAR PASCAL BarAsk(LPSTR data, LPINT bEnable);

BarFont

The BarFont call generates a barcode string using the x parameter as the data and the current default barcode properties. On exit the y parameter contains the barcode string, fa contains the FaceName (Font.Name) of the current barcode font, and ht contain the font height (in points).

VB declaration

Declare Function BarFont Lib "DFONT32.DLL" (ByVal x As String, ByVal y As String, ByVal fa As String, ByRef ht As Long) As Long

Note that VB Strings that are to receive characters must be created before the function is called and must contain sufficient space to receive the characters (eg. y=String(255,vbNullChar))

C declaration

int WINAPI BarFont(LPSTR szin, LPSTR szout, LPSTR fname, LPINT fht);

The szout and fname pointers must point to char arrays that are large enough to receive the barcode string and facename (24 characters) respectively. To account for the largest possible szout string, szout should point to an array that is at least n+5 in length, where n is the length of the szin input string.

BarFonts

The BarFont call generates a barcode string using the x parameter as the data and the current default barcode properties. The barcode string is then saved in the file specified by the fully qualified filename specified in the name parameter.

VB declaration

Declare Function BarFonts Lib "DFONT32.DLL" (ByVal x As String, ByVal name As String) As Long

C declaration

int WINAPI BarFonts(LPSTR szin, LPSTR szfname);

BarFontc

The BarFontc call generates a barcode string using the x parameter as the data and the current default barcode properties. The barcode string is then copied to the clipboard

VB declaration

Declare Function BarFontc Lib "DFONT32.DLL" (ByVal x As String) As Long

C declaration

int WINAPI BarFontc(LPSTR szin);

GetBarDefs

The GetBarDefs function retrieves the current default values of the barcode.

VB declaration

Declare Function GetBarDefs(ByRef ty As Long, ByRef pc As Long, ByRef p1 As Long, ByRef p2 As Long, ByRef ht As Long, ByRef bl, ByRef it, ByRef fn As String) As Long

Where:

ty is an INTEGER to receive the default barcode type

pc is a BOOL to receive the default setting for AutoCheckDigit

p1 is an integer to receive the default value of Extra1

p2 is an integer to receive the default value of Extra2

ht is an integer to receive the default barcode font height (in points)

bl is a BOOL to receive the default barcode font Bold state

it is a BOOL to receive the default barcode font Italic state

fn is a String that will receive the facename of the barcode font.

Note that VB Strings that are to receive characters must be created before the function is called and must contain sufficient space to receive the characters (eg. y=String(28,vbNullChar))

C declaration

int WINAPI GetBarDefs(LPINT ptype, LPINT pchk, LPINT pex1, LPINT pex2, LPINT pfht, LPINT pbold, LPINT pital, LPSTR pfname);

Note that C char arrays that are to receive characters must be created before the function is called and must contain sufficient characters to accommodate the string, eg. Char fn[28];

SetBarDefs

The SetBarDefs function set the default values of the barcode parameters which may then be used in subsequent BarFontx calls.

VB declaration

Declare Function SetBarDefs(ByRef ty As Long, ByRef pc As Long, ByRef p1 As Long, ByRef p2 As Long, ByRef ht As Long, ByRef bl, ByRef it, ByRef fn As String) As Long

Where:

Ty is an INTEGER containing the default barcode type

pc is a BOOL containing the default setting for AutoCheckDigit

p1 is an integer containing the default value of Extra1

p2 is an integer containing the default value of Extra2

ht is an integer containing the default barcode font height (in points)

bl is a BOOL containing the default barcode font Bold state

it is a BOOL containing the default barcode font Italic state

fn is a String contianing the facename of the barcode font.

C declaration

int WINAPI SetBarDefs(LPINT ptype, LPINT pchk, LPINT pex1, LPINT pex2, LPINT pfht, LPINT pbold, LPINT pital, LPSTR pfname);

SetBarName

The SetBarName function set the barcode font name and barcode font height (in points) that become the current default settings (ie. Those returned after a BarFont() call).

VB declaration

Declare Function SetBarName Lib "DFONT32.DLL" (ByVal name As String, ByRef ht As Long) As Long

C declaration

int WINAPI SetBarName(LPSTR foname, LPINT foht);

GetBarName

The GetBarName function gets the barcode font name and barcode font height (in points) that become the current default settings

VB declaration

Declare Function GetBarName Lib "DFONT32.DLL" (ByVal name As String, ByRef ht As Long) As Long

Note that Strings that are to receive characters must be created before the function is called and must contain sufficient space to receive the characters (eg. Name=String(36,vbNullChar))

C declaration

int WINAPI GetBarName(LPSTR foname, LPINT foht);

The foname pointer must point to a char array that is large enough to receive the font facename (24 characters).

Miscellaneous calls

getLength()

Supplies any required length for barcode data (eg. EAN requires 13 digits, or 12 if the 1 checkdigit is being calculated automatically).

VB declaration

Declare Function getLength Lib "DFONT32.DLL" (ByRef icode As Integer, ByRef Autoc As Integer) As Integer

C declaration

int FAR PASCAL getLength(LPINT icode, LPINT Autoc);

where icode refers to the barcode type for which the required number of characters is desired, and Autoc is 1 if dFont will also be calculating the check digit, or 0 if the checkdigit is being provided.

The getLength function returns the number of characters required as input for the barcode to be created, or 0 is the barcode does not require a specified number of characters.

getName()

Supplies the name of a barcode type corresponding to a icode value (eg. Code 39 is icode=0 – see the table above)

VB declaration

Declare Function getName Lib "DFONT32.DLL" (ByRef icode As Long, ByVal name As String) As Long

C declaration

int FAR PASCAL getName(LPINT icode, LPSTR name);

where icode refers to the codetype whose barcode type name is required, and name is a buffer to receive the name as a string. Note that name MUST be predefined to allow for 24 characters or an error may result.

getError()

Obtains a text error message from an errorcode.

VB declaration

Declare Function getError Lib "DFONT32.DLL" (ByRef ierror As Integer, ByVal text As String) As Integer

C declaration

int FAR PASCAL getError(LPINT ierror, LPSTR text);

where ierror refers to the errorcode number whose explanation is required, and text is a buffer to receive the error message as a string. Note that text MUST be predefined to allow for 36 characters or an error may result.

Error codes

A non-zero value returned from the dBarFont() call indicates one of the following errors:

1 Invalid data length

2 Invalid code type

3 Invalid parameters

4 Illegal character in data

5 Invalid embedded code

9 Error creating barcode

The error message string may be ontained using the getError() function or the GetError() method.

The dFont Control

The DFONTOCX control is an Active-X control which simplifies the use of the many of the barcode fonts.

The DFONTOCX control may be placed on a form in most applications which support Active-X controls, such as Visual Basic, Microsoft Access, etc. The barcode properties may be specified through a series of Properties, either via programming, or by setting the properties in the control's property pages.

Placing the control on a form

Visual Basic

To add the control to a Visual Basic project select Components from the Project menu, then select DFONTOCX Active-X Control module from the list of controls displayed and push the OK button. The module's icon will appear in the Toolbox. The control may then be added to a form by clicking on the control's icon and then drawing a rectangle for the control on the required form.

If the control is to be visible then the area allowed should be large enough to hold the largest barcode required. If the control is to be hidden, then the size is irrelevant.

Once added to a form, selection of the control will show the available properties in Visual Basic's properties window.

VB.NET and C#

The control may be added to the Visual Studio.NET Toolbox by right clicking on the Toolbox and selecting Customise Toolbox from the pop-up menu displayed, then checking the control in the list presented. The control will then appear as an icon on the Toolbox.

Access

To add the control to an Access Form or Report, open the form or report in Design view and ensure that the Toolbox is visible (its on the View menu). Select the "More controls" icon on the Toolbox and then select the DFONTOCX Active-X Control module from the list of controls displayed. Draw a rectangle for the control on the from or report.

Once added to a form or report, selection of the control will show the available properties in Access's properties window. The font should be selected by right clicking on the control and choosing DFONTOCX Control Properties from the menu displayed.

Property pages

The control also has its own property pages which may be accessed by right-clicking on the control and selecting Properties from the displayed menu.

General Property page

This permits the setting of the barcode's type, data and other properties, and allows the control to be set to resize itself each time it is drawn

Font Property page

This page allows the user to select the barcode font and font size used to generate the barcode.

Colors Property page

This page allows the user to specify the foreground (the bars) and background color of the barcode generated. In general barcodes should be produced with a black foreground and a white background color.

Control Properties

The control has the following properties which may be set in the property pages or programmatically with Visual Basic.

Caption: (BSTR) the data which will be converted into a barcode

AutoCheckDigit: (Boolean) if non-zero causes the control to calculate the barcode's check digit. If 0 (False) no check digit calculation is performed, and it is assumed that the check digit(s) are in the Caption data. Note that this property is ignored for COde 128 and EAN128 barcodes, where the check digit is both required and hidden from the user

AutoSize: (Boolean) if non-zero causes the control to resize itself to contain the barcode.

BackColor: (Colorref) the background color

CapLength: (integer) normally 0, in which case the length of the data string in the Caption property is determined automatically. If non-zero it is taken to be the number of characters in the Caption property to be used as data.

CodeType: a value which specifies the type of barcode to be produced. (See the **Barcode type table** for list of types)

Errorcode: (integer) a non-zero value is returned if the data supplied in the Caption property can be converted into a valid barcode.

Font: (Font) the font in which the Ouput characters will be displayed

ForeColor: (Colorref) the color of the bars in the barcode

Extra1: (integer) A flag for setting a specific barcode-dependent characteristic (see the individual barcode type description for details). Set to 0 if not required.

Extra2: (integer) A flag for setting a specific barcode-dependent characteristic (see the individual barcode type description for details). Set to 0 if not required.

Output: (CString) the characters which when displayed in the correct font, produce the barcode. Note that this string may be much larger than the Caption string.

Human: (Cstring) the human readable form of the barcode (including check digit). This property is provided so that the human readable form may be obtained when a check digit is being caluclat4ed by the control.

In addition the control supports the normal Active-X properties, such as Visible, Height, Width, etc.

Control Methods

BarSave(filename)

Int Barsave(LPSTR filename)

Causes the control to save the barcode characters into a text file. Filename must be the fully qualified file name, complete with .TXT extension – such as that available from the Windows SaveDialog control.

The methods returns the number of bytes saved in the file, or 0 if an error occurs.

BarCopy()

Int BarCopy(void)

Copies the barcode text to the Windows clipboard, from where it may be pasted into other applications.

Note that only the text is copied; pasted text will need to be set into the appropiate font before a barcode will be seen.

GetError()

BSTR GetError(int errorcode)

Returns the error message associated with the value of errorcode. The list of error codes is shown in the Error codes table.

GetLength()

Int GetLength(int icode, int iauto)

Returns any required length for barcode data (eg. EAN requires 13 digits, or 12 if the 1 checkdigit is being calculated automatically).

where icode refers to the barcode type for which the required number of characters is desired, and iauto is 1 if dFont will also be calculating the check digit, or 0 if the checkdigit is being provided.

A return of 0 indicates that the barcode does not have a specific data length requirement.

GetTypeName()

BSTR GetName(int icode)

Return a string containing the name of the barcode type represented by icode.

If icode contains a value not supported by the control and empty string is returned.

This method may be used to enumerate the barcode types supported by the control, eg:

```
Dim i
Dim xs As String
Dim TypeList As New ArrayList()
For i = 0 To 100
   xs = AxDfontocx1.GetTypeName(i)
   If Len(xs) > 0 Then
      TypeList.Add(xs)
   Else
      Exit For
   End If
Next
codetype.DataSource = TypeList
codetype.SelectedIndex = 0
```

dFont.NET component

The dFont.NET component is DfontLib.Dfont in the DFONTLIB.DLL. The Component is a managed code component that allows barcodes to be created within the user's own .NET application. A barcode may be displayed on screen or printed on a printer, and the content may be passed to any other component.

The Component is designed to work with Visual Studio.NET and requires the .NET run-time to be installed on any computer using the components. Example code is provided with the component for users of Visual Basic .NET.

Adding a dFont.NET Component to the ToolBox

To add a dFont Component to the Visual Studio ToolBox, display the ToolBox and select the Components tab. Right click on the Components pane and select Add/Remover Items.. from the pop-up menu displayed. A dialog box is displayed listing the currently installed components. Ensure that the .NET Framework Components page is displayed.

Push the Browse button and navigate to the location where you have installed or copied your dFont.NET component and select the DFONTLIB.DLL.

Then push the Open button.

The list of installed components is now displayed, including your dFont.NET component. Ensure that the checkbox alongside the component name is checked. Now push the OK button.

The component appears as an icon on the ToolBox Components pane, with the dFont class name alongside:

Adding a dFont.NET component to a project.

With a project's form open in design mode drag the dFont component icon from the toolbox onto the form.

The component icon appears on the panel below the form – it does NOT appear on the form itself. The instance of the component will be given a default name (eg dfont1) which appears in the properties panel when the component is selected. A single Form may contain any number of dFont Components. The first to be added will be called dfont1, the second dfont2, and so on; the names may be changed by the user by modifying the Name property within the Properties box.

The properties panel also displays all other settable properties for the component, and these values will be used as defaults unless properties are changed programmatically within your project.

Setting and retrieving property values programmatically

The dFont Component may be operated entirely by setting or retrieving Property values programmatically.

Clicking on the dFont Component in the panel under the form when Visual Studio's Properties box is displayed will show the current settings for component's available properties. Most of these may be edited using the Properties box, or may have their values set from within the user's program by statements of the kind

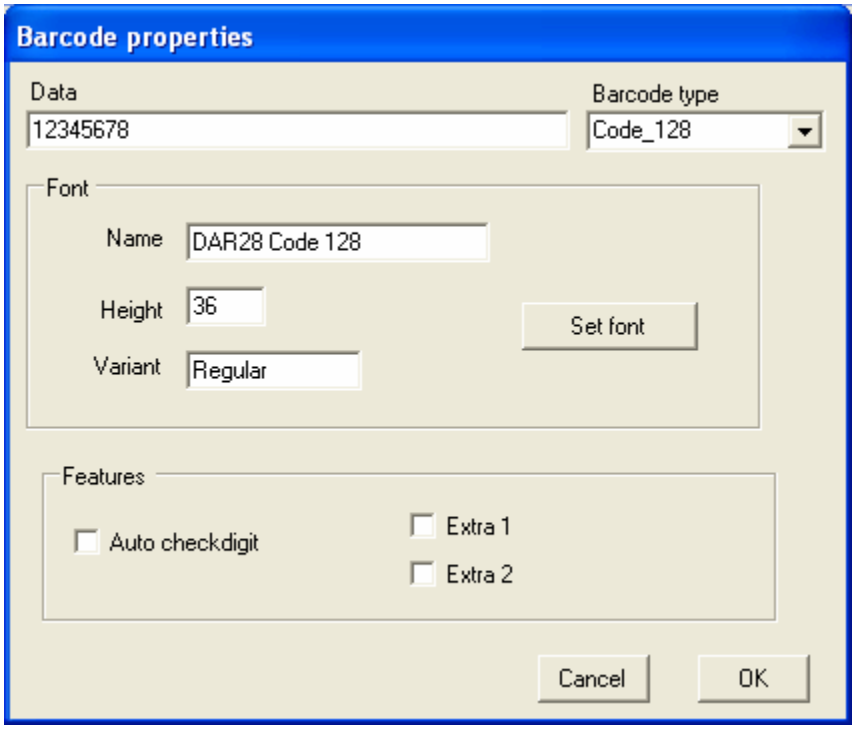
```
Dfont1.Caption="12345"              Visual Basic
Dfont1.Caption="1234";              C#
Dfont1.set_Caption("1234");                 J#
```

dFont Component properties that are set AFTER a barcode has been created may be retrieved within user's programs by statements of the kind:

```
x=Dfont1.Error                  Visual Basic
x=Dfont1.Error;                 C#
x=Dfont1.get_Error()                    J#
```

Setting properties through the Barcode properties dialog box

Using the Method Properties() casues the Barcode properties dialog to be displayed. This displays all settable properties in a convenient form and enables changes to be made by selecting from drop-down lists or entering values into edit boxes, or summoning standard Windows dialogs for the selection of the font with which the barcode should be rendered.



Displaying a barcode on a form

To display a barcode on a form a TextBox is used to hold the barcode characters.

Place a TextBox Windows Forms control on the form and add some code to your program to take the barcode characters from the dFont component and use it as the Text for the TextBox

For example:

```
private void DoBarcode()                    Visual Basic
{
      Dfont1.Caption="1234"
      TextBox1.Image=Dfont1.Barcode
```

```
}

private void DoBarcode()                    C#
{
      dfont1.Caption="1234";
      textBox1.Text=dfont1.Barcode;
}

private void DoBarcode()                    J#
{
      dfont1.set_Caption("1234");
      textBox1.set_Text(dfont1.Barcode);
}
```

Printing from VB.NET or C#

Printing from Managed code is accomplished using the PrintDocument control from the Visual Studio.NET ToolBox. Add the control to your application, then use it as illustrated below; in this example only the barcode is printed.

```
Private Sub print1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles print1.Click
     Try
          AddHandler PrintDocument1.PrintPage, AddressOf
      Me.PrintDocument1_PrintPage
      PrintDocument1.Print()
      Catch ex As Exception
            MessageBox.Show("An error occurred while printing", _
                ex.ToString())
      End Try
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

e.Graphics.PageUnit = GraphicsUnit.Document '
e.Graphics.DrawString(Dfont1.Barcode, Dfont1.Font, Brushes.Black, 300.0F,
300.0F)
' Indicate that this is the last page to print.
e.HasMorePages = False
End Sub
```

Properties and Methods

The dFont.NET component uses the following properties and methods:

AutoCheckdigit

Type: Boolean

Allowed values: True or False

When true this property ensure that the barcode is created with an automatically calculated check digit – where the barcode type supports check digits.

When false no check digit is calculated.

An automatically calculated check digit is generated when the Barcode property is accessed. Once the check digit has been calculated it is included in the String2 property.

Note: Check digits for Code 128 barcodes are not available – because they do not have a human readable form

Barcode

Type: string, (Read Only)

This property returns the characters that form the barcode when displayed in a suitable font.

Note that accessing this property causes the barcode data provided in the Caption property to be converted into a barcode. The String2 property is not valid until the Barcode property has been accessed.

Caption

Type: String

The value set into this property is the data used to form the barcode

CodeType

Type: bCode

Allowed values: see CodeType table

Sets the barcode type that will be created. The CodeType may be set as an member of the enumerated values listed in the CodeType table, or by setting the integer value of the CodeTypeValue property.

The CodeType may also be selected at design time from the drop-down list of the available CodeTypes.

CodeTypeName

Type: string, (Read Only)

Returns the CodeType in string form.

CodeTypeValue

Type: Integer

Allowed values are listed in the CodeType table.

Value of the CodeType property. Setting the CodeTypeValue to to one of the allowed values will automatically set the CodeType to the corresponding barcode type.

ErrorCode

Type: Integer (Read Only)

Returns an error code after the Barcode property is accessed. ErrorCode will contain 0 if a barcode string has been generated.

The values of ErrorCode are:

ErrorCode	Meaning
0	OK, no error
1	Illegal character in data
2	Wrong data length
3	Error in barcode data

Extra1 and Extra2

Types: Boolean

These properties have effects that are specific to barcode types. See the individual barcode types for details.

Font

Type: Font

This property may be used to hold the font properties for the font required to display/print the barcode. It also returns the Font properties selected using the dFont barcode properties dialog.

It is no necessary to use this property, as in all cases the Font used to display or print a barcode may be specified by your own code.

Status

Type: string (Read Only)

Returns a status message after the Barcode property is accessed. Status will contain “OK” if a barcode string has been generated.

The status message is a text version of the ErrorCode property.

String2

Type: string, ReadOnly

This property return the barcode data, including any automatically calculated check digit.

Note: Check digits for Code 128 barcodes are not available – because they do not have a human readable form

dFont.NET CodeType table

The CodeType shown below shows the CodeType properties returned by enumerating the bCode type. For example, to fill a dialog box with the available CodeTypes,

```
ArrayList ttype = new ArrayList();
foreach(string s in Enum.GetNames(typeof(bCode)))
      ttype.Add(s);
      dlg1.codeType.DataSource = ttype;
```

CodeTypeValue	CodeType
0	Code_39
1	Extended_39
2	Code_93
3	Extended_93
4	RM4SCC
5	MSI_Plessey
6	PostNet
7	Code_128
8	EAN_128
9	Interleaved_2_of_5
10	Code_11
11	Telepen
12	EAN_13
13	EAN_13_plus_2
14	EAN_13_plus_5
15	EAN_8
16	EAN_8_plus_2
17	EAN_8_plus_5
18	UPC_A
19	UPC_A_plus_2
20	UPC_A_plus_5
21	UPC_E
22	UPC_E_plus_2
23	UPC_E_plus_5
24	Code_B
25	Codabar
26	Telepen_A
27	Telepen_N
28	DeutschePost
29	SSCC
30	EAN_14
31	ISBN
32	ISSN
33	ISMN
34	FourState

Excel macro

A sample macro for converting data into barcode characters is provided in the Excel spreadsheet sample (dfx2k.xls is the Excel 2000 version; other versions may be included in the dFont/excel directory). The source code for the macro (Barcode) is accessible and can be modified by an experienced Excel user. The macro calls the dFont DLL, so this DLL must be accessible to Excel.

The Barcode macro is used as follows:

a) Select the column on the right of the cells containing the barcode data and format the cells to hold text – by selecting the Text item in the dialog that appears when the Format – Cells – Number menu item is chosen

b) Select a range of cells in a single column containing the barcode data

c) Select Barcode from the Tools – Macro – Macros menu

d) A dialog box appears requesting details of the barcode type required (this may be prevented and the barcode details hard-coded by editing the macro)

e) The macro takes data from each cell in turn and places the barcode text in the adjacent cell. Processing is terminated at the end of the selection or the first blank cell.

f) Select the column containing the new barcode text and set its font to the required barcode font – by selecting the font (an size) in the dialog that appears when the Format – Cells – Font menu item is chosen. Also centre the text in this column and set the column width to ensure that the barcodes are at least 5 mm clear of the gridlines. If the same font will always be used these steps may be incorporated into the macro.

g) Select all rows containing a barcode and set the Row Height to the required value.

The spreadsheet containing the barcode should now be printable.

Installation of macro in another spreadsheet

To install the macro provided in the sample spreadsheet into any other spreadsheet proceed as follows:

1) Open BOTH the sample spreadsheet and the desired target spreadsheet

2) Start the Visual Basic Editor (by selecting Visual Basic Editor from the Tools – Macro menu

3) Expand the dfx.xls item in the Project details pane, then expand the Modules section to that the Module1 name is visible

4) Expand the target spreadsheet so that the ThisWorkbook name is visible

5) Drag Module1 from the sample spreadsheet to the ThisWorkbook name of the target spreadsheet

6) From the File menu select Close and Return to Microsoft Excel, then close the sample spreadsheet

The macro is now available in the target spreadsheet and may be run by selecting Barcode from the Tools – Macro – Macros menu once the required data cells have been selected.

Crystal Reports UFL

The Crystal Reports UFL and associated sample files are placed in the installation directory of your dLSoft Barcode Font kit, and must be installed manually before it can be used with Crystal Reports.

Installation

1. Locate the file U2LDFONT.DLL in the dLSoft Barcode Font kit installation directory, which by default is C:\Program Files\dLSoft\dFont\crystal

2. Locate your Crystal Reports executable file (CRW32.EXE) in the Crystal Reports installation directory, which by default, is C:\Program Files\Seagate Software\Crystal Reports

3. Copy the file U2LDFONT.DLL to the directory that contain CRW32.EXE

That's all.

Note that if you add barcode to reports using the Testware version and subsequently upgrade to the full version, you must delete the function and recreate it - otherwise the barcodes will continue to be scrambled.

Running the sample report

The sample report included with the kit is "Order Packing List1.rpt" which will be found in the crystal subdirectory of the dFont installation directory. This is a modified version of the "Order Packing List.rpt" report included with the Crystal Reports Xtreme Mountain Bike Inc samples. If you do not have the samples installed on your system then the modified sample report will not work and you should skip to the section "Creating a barcode on a report" below. Also the report displays barcodes using the A39R Code 39 font; if you have not installed that font then the report will not display a barcode - but a collection of characters. (You can change the barcode font to one you have installed as described below and modify the code type parameter, and the barcode will then display correctly.)

The "Order Packing List1.rpt" sample may be run on a machine that contains a full Crystal Reports installation just by double clicking on the report file. A barcode is included on the report with its data taken from the Order ID filed of the data.

Switching the report to Design view, Right-clicking on the barcode and selecting Edit Field Object from the drop-down menu displays the Function (DLFontcr) and parameters used to generate the barcode. The first parameter is the barcode data (a string), the second parameter is an integer that specifies the barcode type (code type 8 is Code 39), and the final parameter (normally 0) can be used to specify extra barcode features.

Creating a barcode on a report

To create your own barcode on a report follow the steps below:

1. Open the report in Design view

2. Select Field Object from the Insert menu

3. Select the Formula Fields item in the dialog box; ensure that the item becomes highlighted.

4. Either hold down the Control key and press N, or select the New icon in the toolbar.

5. Enter a name for the formula - such as barcode1 - then push the OK button. The Formula Editor appears.

6. From the list of Functions (normally the middle list) scroll down to the Additional Functions item; the expand this item by click on the + symbol alongside the Additional Items name.

7. Select the DLFontcr function from the list, then either double click on it or press the Enter key; the function then appears in the formula box below the lists, complete with its parameter brackets and commas, and the cursor in the position of the first parameter. ie.

DLFontcr (,,)

8. The first parameter must be a string containing the barcode data. This can be a literal string (ie. data enclosed in quotation marks, such as "1234"), or field data. If field data is to be used it must be text data - so if the required field actually contains numeric data this must be converted into text data.

If a required data field contains text data just double click on the field name in the list of Report Fields, and the field name enclosed in curly brackets will be copied to the function's first parameter position. eg.

DLFontcr ({Customer.Region},,)

If a required field contains numeric data then expand the Strings item in the list of functions and the expand the ToText function; Now select the required function - which will usually be ToText(x,y,z) where the x represent the number to be used as data, y represent the number of decimal places (typically 0) and z is a character used to separate thousands from hundreds etc (which unfortunately defaults to a comma and is generally not wanted in a barcode). Double click on the required version of the ToText function and this will be copied to the first parameter position of the DLFontcr function, with the cursor now placed in the first parameter position of the ToText function, ie.

DLFontcr (ToText (, ,),,)

Now double click on the required data source field in the list of Report fields, eg.

DLFontcr (ToText (, ,),,)

and fill in the other two ToText parameters with a 0 (the number of decimal places) and a NULL character (two single quotes) respectively, ie.

DLFontcr (ToText ({Orders.Order ID},0 ,"),,)

9. Now complete the other two DLFontcr parameters with the barcode type code (see the **Barcode types table** for a complete list of these) and a 0, both enclosed in quotation marks, eg.

DLFontcr (ToText ({Orders.Order ID},0 ,"),"0","0")

The final parameter is equivalent to the Flags parameter in the dBarFont() function call of the DLL.

10. Now click on the Save and Close icon to return to the Field Explorer and the formula named by you will be present in the list. Double-click on the formula name the move the cursor to locate the object (ie. the barcode) where you want it on the report.

11. Finally select the barcode object, right click on it, choose Format Field from the drop down menu, and click on the Font tab in the displayed dialog box. From the list of fonts presented select the dLSoft barcode font you require and select a suitable size (typically 24 - 36 point, although 72 point is required for retail codes). The character spacing must be left at 0, otherwise the barcodes will look nice but won't scan. Then click on OK.

12 Return to the report's Preview display and you should have a perfect barcode. If you are using data from a database as the data source for the barcode, then the barcode will change as you navigate around the record source.

To make changes to the barcode formula

Open the report in Design view, right click on the barcode (or its' empty box) and select Edit Field Object from the drop down menu.

Follow the procedure above to replace the formula.

Note that invalid barcodes will not be visible. Some barcode types support only digits, other support only digits and upper case letters. Barcode that support only digits do not support spaces!!

Barcode types table

The following barcode types are available through the supporting software. The code type may be specified using the relevant barcode type **code** value shown in the table below.

code	Barcode type
0	"Code 39"
1	"Ext. Code 39"
2	"Code 93"
3	"Ext. Code 93"
4	"RM4SCC"
5	"MSI/Plessey"
6	"PostNet"
7	"Code 128"
8	"EAN-128"
9	"ITF"
10	"Code 11"
11	"Telepen"
12	"EAN/JAN/ISBN/ISSN"
13	"EAN+2"
14	"EAN+5"
15	"EAN 8"
16	"EAN 8+2"
17	"EAN 8+5"
18	"UPC-A"
19	"UPCA+2"
20	"UPCA+5"
21	"UPC-E"
22	"UPC-E+2"
23	"UPC-E+5"
24	“Code B”
25	“Codabar”
26	“Telepen ASCII”
28	“DeutschePost
29	“SSCC”
30	“EAN 14”
31	“ISBN”
32	“ISSN”
33	“ISMN”
34	“4-State”

Index

P

R

S

T

V