Bar code printing in Windows

K. B. Swiatlowski Mgr
Micross Electronics Ltd.,
Units 4-5, Great Western Court,
Ross-on-Wye, Herefordshire.
HR9 7XP
UK
Tel. +44 1989 768080
Fax. +44 1989 768163
Email: kbs@micross.co.uk, Website: http://www.micross.co.uk

Abstract

Industrial computer systems frequently require bar codes to be included in printouts. This paper presents the construction of a single Forth word that does all the work for you, and results in an accurate and easily read printed bar code.

Introduction

Printing in Windows

Windows provides to a programmer a set of functions, called WinAPI. Among many functions there are graphic functions. We can distinguish those responsible for:

- choosing tools used for drawing like: brushes, pens, fonts, fillings, colours;
- drawing itself: characters, lines, solid shapes and pattern filled shapes;
- transformations of graphics output (resizing, rotating, etc.);
- handling output devices: initializing/finishing a display, refreshing, set-up of device's parameters;
- handling users' interaction on graphical output (screen).

All details about those functions can be obtained from Microsoft's website developer's service or any programmer's reference.

Bar code structure

Information in a bar code is presented by a sequence of ones and zeros. The logical one is agreed to be presented as a bar – a dark rectangle, zero is represented as an absence of a bar. Bar, or its absence should have a fixed width, so we could recognize a number of bars in repeatedly appearing ones or zeros.

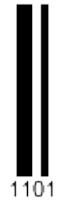

Fig. 1. Bar representation of a digit – binary 1101 (13 decimal)

A char set or a bar set is a sequence distinguishing one character (numeric, alpha-numeric, ASCII).

With software presented it is up to you how you will represent your information. You can even assign a word or a sentence to a one bar set. Encoding must be designed wisely. Zeros

and ones should interlace mutually quite often so the hardware can recognize the sequence. The Internet is full of many proposals and described standards (http://www.bar codeisland.com).

There are generally two types of bar code symbologies: discrete and continuous. A discrete symbology is one where each and every character encoded in the symbol may be interpreted individually without respect to the rest of the bar code. Such symbologies have characters that both start and end with a bar. Individual characters are separated by some amount of inter-character spacing.

A continuous symbology is one in which the individual characters of the symbology cannot be interpreted by themselves. This is due to the fact that characters start with a bar and end with a space. Continuous symbologies normally implement some kind of special termination bar or termination sequence such that the last space of the last data character is terminated by the termination bar.

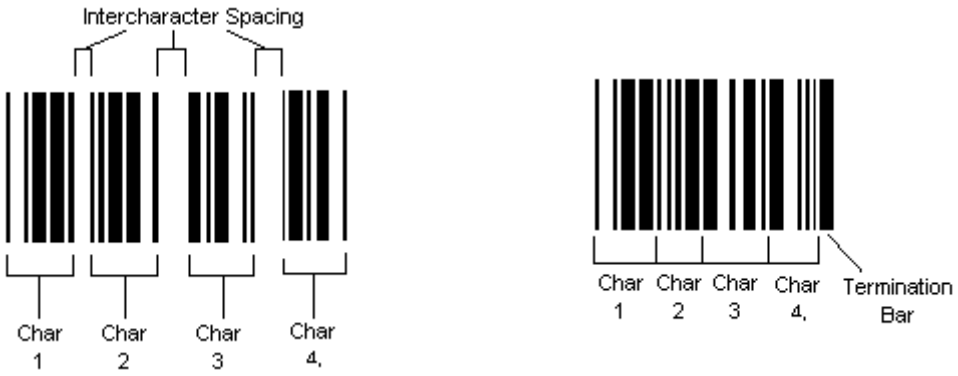

Fig. 2. Discrete and continuous bar codes

Bar codes may be either fixed or variable-length. A fixed-length symbology is one which must, by definition, encode a certain number of characters or digits (UPC-A, EAN13). A variable-length symbology is one which can carry a message of any length (Code 128), it may encode any number of characters.

Solution

We are glad to present a solution for convenient printing of a flexible bar code on a device. Words have been grouped into two files:

Barcode-print.fth deals with printing itself. It uses Windows functions and takes printing parameters.

Ean13.fth is an implementation of EAN13 standard. It deals with data generation, checksum calculations and sets bar sets' appearance flags. It is your part to design data generation routines, translation from bar code and to bar code relative to other standards or your own idea..

General printing parameters

There are two places where the appearance of the bar code can be set. One of them is a structure *BARCODE-PRINTCTL* and it is used for setting parameters of the bar code as a whole. Common parameters are e.g. a width of the bar, a font used for printing and finally

pointers to data arrays. Furthermore, appearance of a single bar set, representing one character or a digit, should be amended in an array of control flags *B-CONTROL-TBL* assigned to every single bar set of the bar code. In that array you can define height ratio, bar count and others. Note that following table indicates by italic style font values that should not be set directly. There are calculated for every call of the interface words provided.

Table 1. Structure members setting printing parameters for a whole bar code

Sizes in bytes	***Name***	***Comment***
7 CELLS 32 +	B-BARSFONT	MUST be first so you could use WinAPI's LOGFONT structure offset flags on this structure
CELL	B-HDC	Device Context (window, printer etc.)
CELL	B-DIGITS-IN-CODE	How many digits in a bar code, including checksum, a guard bar set etc
CELL	B-CHARWIDTH%	If 600[%] of B-WDTH then for a one bar set of 8 bars it will occupy 6 bars wide
CELL	B-CHARHEIGHT%	If 900[%] of B-WDTH will produce 9 B-WDTH * high char
CELL	B-HSHORTBAR%	If 80[%] then for a long bar 5 units high, a short one will be 4 units high
CELL	B-CHARDIST%	If 90[%] then chars will appear at 4.5 units of bar height
CELL	B-CONTROL-TBL	Ptr to controlling bytes for each bar set
CELL	B-STRING-TBL	Table of pointers to counted strings shown below bar code
CELL	B-CODE-TBL	Table ptr for binary representation of bar codes
CELL	B-SPECIALSTRING	Ptr to String shown above the bar code
CELL	B-RECT-ALIGN	Bar code alignment inside the rectangle
CELL	*B-LOGICWDTH*	A Bar width in device units
CELL	*B-LOGICHEIGHT*	A Bar height in device units
CELL	*B-WDTH*	A Bar width in device units
CELL	*B-HEIGHT*	A Bar height in device units
_RECTNG	*B-ONEBARSIZE*	Bar size in device points used by printing routines
_XFORM	*B-XFORM*	Transformation for rotating bar code
_XFORM	*B-XFORM-BACKUP*	Current transformation backup, will be restored after printing
CELL	*B-GRAPHICSMODE*	Graphic mode backup. If user wants to rotate the bar code it must be done in ADVANCE MODE
2 CELLS	*B-LOGICRATIO*	Ratio computed for current mapping mode. Used to calculate dev units for given distance.
2 CELLS	*B-WINORG-BACKUP*	Backup for window origin coordinates

Deferred word *BCTL* is used for accessing in functions the structure shown in the above table, so you could use your own word to access your data.

There are several words you can call to print the bar code on a chosen device. It should be activated after configuration of the *BARCODE-PRINTCTL* and filling of data arrays.

```
: BAR-CODE ( hdc x y bar-width bar-height angle -- )
```

A device context – *hdc*, can be obtained by calling a WinAPI function, and it represents a printer or a computer screen. An *angle* is an integer number which specifies the angle, in tenths of degrees counter-clockwise. Setting it to a number greater than 0 turns on the advance graphic mode.

All other parameters are defined in a chosen metrics, so it can be set in device units (pixels), in tenths of a unit eg. millimeter. Left upper coordinates are indicated by *x* and *y*, it is a hook point of rotation as well. Bar width and height follows those coordinates. There is a simple call, if bar height and width are already set, which takes only coordinates, an angle, and device context.

```
: DOBAR-CODE ( hdc x y angle -- )
```

In setting character height and width you can determine the size of the font, which will be a multiple of a bar width in thousandths of a bar width. Text below the bar sets is centered aligned. Reseting those values to zero will let you set font size by manipulating a LOGFONT structure.

The last word provided is one which uses a *Rectangle* structure to define the position and size of the whole bar code. In that case Bar width and height will be calculated to efficiently fill the given area. Because bar width multiplicity can be smaller then a rectangle width it is possible to set the alignment of the bar code inside the rectangle. To do that assign one of the font's text alignment flags to the *B-RECT-ALIGN* field.

```
: RECTBAR-CODE ( hdc rectangle angle -- )
```

Bar set printing parameters

The structure *BARCODE-PRINTCTL* hold fields pointing to each bar set controlling flags – *B-CONTROL-TBL*, arrays of strings displayed below each bar set – *B-STRING-TBL*, and array of cells for storing binary representation of each bar set *B-CODE-TBL*.

B-CODE-TBL is filled in routines declared by a user thus there might be many different encoding and algorithms. There is enough room for up to 31 bars representing one bar set.

B-STRING-TBL is an array of counted strings. It holds empty strings or multi character strings, the only restriction is a resolution of the device or a legibility of the message.

In addition you show one more string which might to be useful for displaying extra information above the bar code. The pointer to that string should be placed in *B-SPECIALSTRING* field of the previously discussed structure.

B-CONTROL-TBL holds an array of bytes. The 5 least significant bits hold the number of bars for each bar set, so you can have varying counts of bars in a bar set. I.e. in EAN13's center and band guard patterns have different counts (5 and 3) than the information bars (7). It gives you an opportunity to design flexible structure in your bar code. *B-CONTROL-TBL* takes also values of constants, shown in Table 2, except for a mask which is used in the code to retrieve bar count form the control table. The drawing routine prints the sequence of bars according to that count. That sequence is placed in *B-CODE-TBL,* the state of rightmost bar is set according to the least significant bit in that cell, and so on.

Table 2. Constants controlling appearance of a single bar set, placed in a control array

Value	*Name of a constant*	*Comment*
31	BARCOUNT-MASK	Mask for retrieve bars count for one bar set
32	PRINT-TITLE	Prints a title string above appointed bar set.
64	SHORT-BAR	Draws short bar leaving space for string below
128	PRINT-FOOT	Prints string from STIRNG-TBL below bar set

An example of EAN13

EAN-13 was implemented by the International Article Numbering Association (EAN) in Europe. As based upon the American version, it is a superset of UPC-A. This means that any software or hardware capable of reading an EAN-13 symbol will automatically be able to read an UPC-A symbol.

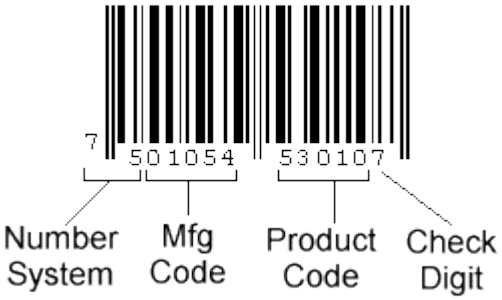

Fig. 3. The structure of data represented by EAN13 bar code

EAN-13 number system code consists of two digits ranging from 00 through 99, which is essentially a country code – an authority which assigns manufacturer codes to companies within its jurisdiction. The manufacturer code is usually five digits long, as is the product code – which are attended by a producer itself. The last digit is a checksum.

Guard bar sets (no. 1 and 15) delimit the bar code from both sides, and the bar set no. 8 is a middle guard bar set, which separates the manufacturer part from the product part. As you can see on the Figure 4, the first digit of the number system is represented as blank bar set, although it is still taken for the checksum calculations.

Fig. 4. Bars alignment and bar set's indexing

Lets follow the steps to achieve a working EAN13 bar code. The following code sample shows how to create instances of all tables and structures and how to fill them with data.

```
CREATE BCTL-INSTANCE BARCODE-PRINTCTL ALLOT
: (BCTL) ( -- addr )
  BCTL-INSTANCE
; ASSIGN (BCTL) TO-DO BCTL          \ BCTL will return an address
```

`400 BCTL B-CHARWIDTH% !`	Char width will consume 4 bars below
`700 BCTL B-CHARHEIGHT% !`	Windows will ATTEMPT to make that rectangle
`NULL BCTL B-GRAPHICSMODE !`	Clearing is a must
`NULL BCTL B-SPECIALSTRING !`	Clearing
`92 BCTL B-CHARDIST% !`	Upper band of the char will be at 92% of the height provided
`90 BCTL B-HSHORTBAR% !`	Shorter bars (2,...,7 and 9,...,14) will be 10% shorter then long ones
`16 CONSTANT #DIGITS-IN-CODE`	Number of bar sets including blank, checksums, guards etc.
`7 CONSTANT #BAR-ELEMENTS`	Max is 31 how many bars represent one digit (predominately)
`CREATE CODE-TBL #DIGITS-IN-CODE CELLS ALLOT`	Table for binary representation of bar codes
`CREATE STRING-TBL #DIGITS-IN-CODE CELLS ALLOT`	For pointers to counted strings shown below bar code
`CREATE CONTROL-TBL #DIGITS-IN-CODE ALLOT`	How many bars shall be printed for each digit
`CONTROL-TBL #DIGITS-IN-CODE #BAR-ELEMENTS SHORT-BAR PRINT-FOOT OR OR FILL`	Puts a standard control flags into all bar sets (shorter, print char below, 7 bars)
`7 PRINT-FOOT SHORT-BAR OR OR CONTROL-TBL C!`	7 bars wide and shorter for first number system, deletes previous setting
`3 CONTROL-TBL 1+ C!`	Left band control bars, new setting
`5 CONTROL-TBL 8 + C!`	Middle control bars, new setting
`3 CONTROL-TBL #DIGITS-IN-CODE 1- + C!`	Right band control bars
`10 CODE-TBL 8 CELLS+ !`	Middle control-guard bars 01010
`5 CODE-TBL 1 CELLS+ !`	Left band control-guard bars 101
`0 CODE-TBL !`	First number system bar set is blank
`5 CODE-TBL #DIGITS-IN-CODE 1- CELLS+ !`	Right band control-guard bars 101
`61 CODE-TBL 6 CELLS+ !`	61 decimal is 0111101 binary and it encodes 3 of 6th bar set (Fig.4)
`51 CODE-TBL 7 CELLS+ !`	51 is 0110011 binary, it represents bars of 7th bar set (Fig.4)

Conclusion

With our application you are given full control over the structure, size, alignment and the appearance of the bar code. Due to data feeding being separated from printing it is possible to develop a unique information encoding, and apply it with the printing routines provided.

This applied technique has a major advantage over the True Type Font bar code. It prints bars in device units, ensuring fixed widths and spaces of every bar and gap, which can not be said about scalable techniques. This yields the required accuracy for reading devices.

Appendix

Nonstandard, undeclared functions used in the code:

Forth call	*Comment*
ALSO F-PACK ALSO FORTH DEFINITIONS	Includes nonstandard dictionaries in search chain.
REALS	Switches floating point numbers routines.
SF!	Stores a floating point number at address.
SF@	Fetches a 32-bit floating point number from address.
S>F	Converts a 32-bit signed integer to a floating point number.
F/	Divides double length numbers by other.
DEG>RAD	Converts degrees to radians (both double length floating point numbers).
FSINCOS	Calculates the sine and cosine of angle in radians, returning (both double length floating point numbers).
FNEGATE	Changes the sign of the top double length floating point number.
FDUP	Duplicates the double floating point number on the stack. An analogue of 2DROP.
INTEGER?	Attempts to convert the counted string at “addr” to a number returning 0 for conversion failed, “n” and 1 for converting a single number, and “d” and 2 for converting a double number.

WinAPI functions used in the code (*.dll)

WinAPI call	*Short description*	*Forth call*
SetTextAlign	The function sets the text-alignment flags for the specified device context.	F-SETTEXTALIGN (flag hdc -- oldflag)
TextOut	The function writes a character string at the specified location, using the currently selected font.	WINTEXTOUT (hdc, xcoordinate, ycoordinate, lpString, strCount -- status)
Rectangle	The Rectangle function draws a rectangle outlined by using the current pen and filled by using the current brush.	WINRECTANGLE (hdc nLeftRect nTopRect nRightRect nBottomRect -- status)
LPtoDP	The function converts logical coordinates into device coordinates.	WINLPTODP (hdc lpPoints[2] nCount -- t/f)

WinAPI call	*Short description*	*Forth call*
SetMapMode	The function sets the mapping mode of the device context defining the unit of measure and defining the orientation of the device's x and y axes	F-SETGRAPHICSMODE (flagmode hdc -- oldmode)
GetGraphicsMode	The function retrieves the current graphics mode for the specified device context.	F-GETGRAPHICSMODE (hdc -- value)
SetGraphicsMode	The function sets the graphics mode for the specified device context.	WINSETMAPMODE (hdc flagMode -- oldMode)
GetWorldTransform	The function retrieves the current world-space to page-space transformation.	F-GETWORLDTRANSFORM (lpXform[6] hdc -- t/f)
SetWorldTransform	The function sets a two-dimensional linear transformation between world space and page space for the device context (used to rotate).	F-SETWORLDTRANSFORM(lpXform[6] hdc -- t/f)
CombineTransform	The function concatenates two world-space to page-space transformations.	F-COMBINETRANSFORM (lpxformResult[6] lpxform1[6] lpxform[6] -- t/f)
GetDeviceCaps	The function retrieves device-specific information about a specified device: number of pixels per logical inch or width, in pixels, of the screen	WINGETDEVICECAPS (hdc queryFlag -- value)
SetWindowOrgEx	The function sets the window origin of the device context by using the specified coordinates	WINSETWINDOWORGEX (hdc x-coord y-coord oldPoint [2] -- t/f)
GetStockObject	The function retrieves a handle to one of the predefined stock pens, brushes, fonts.	WINGETSTOCKOBJECT (flag -- objHndl)
SelectObject	The function selects an object into the specified device context	WINSELECTOBJECT (hdc ObjHnld -- OldHndl)
CreateFontIndirect	The function creates a logical font that has the characteristics specified in the specified structure.	WINCREATEFONTINDIREC(LOGFONT -- FontHndl)
SetBkMode	The function sets the background mix mode of the specified device context.	WINSETBKMODE (hdc mode -- oldMode)
DeleteObject	The function deletes a logical pen, brush, font etc freeing resources associated with the object.	WINDELETEOBJECT (objHndl - t/f)

Visit our website in order to download the source code (http://www.micross.co.uk)