Contents

The dBarcode Dynamic Link Library

dBarcode DLL is a Dynamic Link Library which allows barcode images, in the form of Windows Metafiles, to be created within the user's own software. The metafile may be displayed on screen, printed on a printer or passed to the Windows clipboard for incorporation into another Windows application.

This description of the dBarcode DLL assumes that you are familiar with Windows programming in C using the Microsoft SDK.

More:

Supplied files

Data Structure

Supplied files

The installation program places several files into a subdirectory of your choice. Apart from documentation and help files, these files have the following functions:

DLSBARn.DLL is the library file. This must be available to applications which you create and which need to use the library. When you distribute your application, this file should be placed either in the application's directory or in the Windows directory of the system on which your application is being installed. This file needs to be distributed with your application.

DLSBARn.LIC is the licence file. This is a traceable licence file which must be present in the same directory as DLSBARn.DLL for the library to function correctly. If this file is missing or invalid barcode images will contain a defacing line through the middle of the image, and the pattern string returned will be scrambled.

DLSBARn.H is a header file which contains the definition of the structure used to pass data between your application and the DLL library, and prototypes for the calls which can be made to the library. This file (which is small) should be copied to your compiler's INCLUDE subdirectory. This file does not need to be distributed with your application. Note that if you are using using C++ and Microsoft's 16 bit C++ compiler then DLSBARn.H must be included using:

```
extern "C" {
#include DLSBARn.h
}
```

DLSBARn.TXT contains the import section of a DEF file, showing the import ordinals you need to add to your application's DEF file before calling the library routines. The contents of this file may be pasted into your application's DEF file. This file does not need to be distributed with your application.

For Microsoft's 16 bit C compiler the import section of your application must include:

```
IMPORTS
        BarCodem=dlsbarn.2
        BarInfo=dlsbarn.3
        BarCodeh=dlsbarn.4
```

DLSBARn.LIB The LIB file created during the compilation of the DLL using Microsoft Visual C.


BarMan
File
Edit
Options
Help
Code type
required chars provided
Status
EAN13+5
17
17
OK
Code
50123456789012345
Auto checksum
12345>
5 012345 678900

Data Structure

The dBarcode DLL uses a single data structure for transferring data into the library routines, BARCODE, which is defined in DLSBARn.H along with a long pointer to the structure LPBAR.

```
typedef struct tagBARCODE
        {
        int             code;
        char            name[12];
        int             show_text;
        int             auto_parity;
        int             orientation;
        int             show_checkdigit;
        int             nominal_size;
        int             line_reduction;
        int             border_thickness;
        COLORREF        fore_color;
        COLORREF        back_color;
        int             error;
        char                    string[48];
        int             req_number;
        int             parity;
        int             alphabetic;
        int             length;
        int             height;
        int             subscript;
        int             flags;
        int             textgap;
        char            string2[48];
        int             pcfont2;
        int             spare;   } BARCODE;

typedef BARCODE far *LPBAR;
```

A BARCODE structure must exist before any library routines are called, so your program requires a declaration such as

BARCODE bc;

to appear in the program before any calls are made to the library routines.

Some members of the Barcode structure must be filled before any calls are made to the library routines. Other members of the structure are used for returning information to the calling application and so do not require filling prior to library calls.

The prototypes of the library calls are:

LPHANDLE FAR PASCAL BarCodem(HANDLE hInstance, LPRECT lpRectIn, LPLOGFONT lpLogFontIn, LPBAR bc, LPHANDLE &hmf, LPSTR szPat)

where

hInstance is the instance (Module) handle of the calling application

lpRectIn is a long pointer to a rectangle which is the destination rectangle for the barcode image (with coordinates in units of 0.01 mm).

lpLogFontIn is a long pointer to a LOGFONT structure filled with information about the font with which

any text will be displayed on the barcode. The font height will be treated as Points and treated as positive.

bc is a BARCODE structure - the LPBAR cast is required.

hmf is a HMETAFILEPICT handle which will contain the handle of the metafile image on return.

szPat if a long pointer to a string which receives a pattern of 0s and 1s representing spaces and bars respectively. This variable must be declared as

char szPat[1024];

before calling the library routines - even if you do not intend to use the pattern.

Calls to BarCodem create the barcode metafile image and the image pattern (for developers who wish to draw their own bars), returning an error code in the barcode structure in the event of an error.

An almost identical call is provided in the 32 bit version of the library:

LPHANDLE FAR PASCAL BarCodemx(HANDLE hInstance, LPRECT lpRectIn, LPLOGFONT lpLogFontIn, LPBAR bc, LPHANDLE &hmf, LPSTR szPat)

where the only difference from Barcodem is the handle hmf and the handle returned are HENHMETAFILE type (ie. Refer to Enhanced metafiles).

int FAR PASCAL BarInfo(HANDLE hInstance, int itype,LPBAR bc)

where

hInstance is the instance handle of the calling application.

itype is the barcode type index (0-27), and

bc is a long pointer to a BARCODE structure.

Calls to BarInfo with an integer value specifiying the barcode type in itype will return bc containing information about that type, including its abbreviated name, nominal length and height (if any), required number of characters (if restricted), check digit (or parity) requirements, alphabetic character support, and whether the code supports or requires a subscript (or supplementary code). This information is useful for filling a listbox with the barcode type names, and for checking whether a specific number of characters is required to generate a valid code image.

int FAR PASCAL BarCodeh(HDC hDC, HANDLE hInstance, LPRECT lpRectInL, LPLOGFONT lpLogFontIn, LPBAR bc, LPSTR szPat);

Calls to BarCodeh draw the barcode image into the specified device context, with other parameters as defined for BarCodem.

where

hDC is the target device context

hInstance is the instance handle of the calling application

lpRectIn is a long pointer to a rectangle which is the destination rectangle for the barcode image (with coordinates in units of 0.01 mm).

lpLogFontIn is a long pointer to a LOGFONT structure filled with information about the font with which any text will be displayed on the barcode. The font height will be treated as Points and treated as positive.

bc is a BARCODE structure - the LPBAR cast is required.

szPat if a long pointer to a string which receives a pattern of 0s and 1s representing spaces and bars respectively. This variable must be declared as

char szPat[1024];

before calling the library routines - even if you do not intend to use the pattern.

Calls to BarCodeh draws the barcode image on the specified device context and returns the image pattern (for developers who wish to draw their own bars or derive information about the number of elements), returning an error code in the barcode structure in the event of an error.

Distributing Programs using dBarcode Controls or Libraries

Any program using dBarcode OCX Controls must have access to the dBarcode OCX in file DBCOCX16.OCX (16 bit) and DBCOCX32.OCX (32 bit) .

Any program using dBarcode VBX Controls must have access to the dBarcode Library in file BRMK.VBX.

In addition the dBarcode Controls requires access to the DLL which contains the barcode algorithms, DLSBARn.DLL, where n is the version number, and the corresponding licence file DLSBARn.LIC.

The appropriate VBX or OCX file and the corresponding DLL would have been installed in the dBarcode directory during installation of dBarcode and must be included on any distribution disks containing programs which use the control.

If you distribute programs with an installation system you should install the appropriate VBX or OCX, DLL and LIC files in either your programs .EXE file directory or in the Windows System directory, and in the case of the OCX, ensure that the control is registered in the system registry.

If you intend to redistribute programs containing dBarcode controls and/or libraries for commercial purposes, then you must ensure that you obtain the appropriate licence from DL Technology.

A redistributable version - which does not require the licence file - is available from DL Technology

DBARCODE CONTROLS WILL NOT WORK WITHOUT ACCESS TO THE CORRESPONDING LIBRARY FILE(S).

The BARCODE structure

The dBarcode DLL uses a single data structure for transferring data into the library routines, BARCODE, which is defined in DLSBARn.H along with a long pointer to the structure LPBAR.

```
typedef struct tagBARCODE
        {
        int             code;
        char            name[12];
        int             show_text;
        int             auto_parity;
        int             orientation;
        int             show_checkdigit;
        int             nominal_size;
        int             line_reduction;
        int             border_thickness;
        COLORREF        fore_color;
        COLORREF        back_color;
        int             error;
        char                    string[48];
        int             req_number;
        int             parity;
        int             alphabetic;
        int             length;
        int             height;
        int             subscript;
        int             flags;
        int             textgap;
        char            string2[48];
        int             pcfont2;
        int             spare;   } BARCODE;
```

The components for this structure are defined below.

More:

int code

char name[12]

int show_text

int auto_parity

int orientation

int show_checkdigit

int nominal_size

int line_reduction

int border_thickness

COLORREF fore_color

COLORREF back_color

int error

char string[48]

int req_number

int parity
int alphabetic
int length
int height
int subscript;
int flags;
int textgap;
int spare;
Other variables

int code

Default value 0

The value of this variable specifies the code type as defined as code # in the barcodes table:

Allowed values:

The barcodes table has been moved to the barcodes help file/documentation appendix. This file will be updated with the DLL files as new barcode types are added. For this reason the barcodes help file may be more up to date than your printed documentation.

char name[12]

On calls to BarInfo this variable is filled with an abbreviated version of the codename for the code type specified in the *itype* parameter, as defined in the table above.

int show_text

Default value 0 This variable specifies whether a text version of the barcode should appear (usually underneath the bar image) when a call to BarCodem generates an image.

Allowed values:

0 - no text appears

1 - text appears, using the font specified in the lpLogFontIn structure.

int auto_parity

Default value 0 This variable specifies whether a checkdigit should be automatically generated when a call to BarCodem generates the barcode image.

Allowed values:

0 - no automatic calculation

1 - checkdigit is automatically calculated

Note that for barcode types which do not have check digits this parameter is ignored.

int orientation

Default value 0 This variable specifies whether the barcode image should be generated normally or with clockwise or anticlockwise rotation through 90 degrees.

Allowed values

0 - Normal orientation

1 - clockwise rotation by 90 degrees

2 anti-clockwise rotation by 90 degrees

-1 inverted (ie rotated by 180 degrees

Note that only TrueType or compatible fonts may be rotated

int show_checkdigit

Default value 0 This variable specifies whether any checkdigit should be included with the code text when a call to BarCodem generates the barcode image.

Allowed values:

0 - checkdigit is not included

1 - checkdigit is included

Note that for barcode types which do not have check digits this parameter is ignored. It is also ignored if text inclusion has not been requested.

int nominal_size

Default value 0 Some barcodes are specified by their designers to be reproduced at a nominal size (EAN and UPC codes are examples). This variable may be used to generate a barcode image which is at the nominal size rather than the size specified by the lpRectIn parameter in calls to BarCodem

Allowed values:

0 - Size specified in lpRectIn is used

1 - Nominal size is used and lpRectIn is ignored

Note that for barcode types which do not have a nominal size this parameter is ignored.

int line_reduction

Default: 0 — the line thickness of each line drawn on the barcode image is reduced by this percentage amount. This property may be used to compensate for ink spreading during wet-ink printing.

Allowed values: 0 - 50 (%)

int border_thickness

Default value 0 This variable specifies the thickness (in points) of any border required around the barcode image. Any border is created using the colour specified as the background colour.

Allowed values: 0 (no border)

to 1/3 * ImageHeight

COLORREF fore_color

Default: 0	Set the colour of the image foreground, i.e. the bars and text colour.
Allowed values:	0 (black)
to	&H00FFFFFF& (white)

COLORREF back_color

Default: &H00FFFFFF& Sets the colour of the image background.
Allowed values: 0 (black)
to &H00FFFFFF& (white)

int error

Returns a value representing the error code if a valid barcode image cannot be created on a call to BarCodem. The error codes are shown below.

code	meaning
0	No error
1	Wrong code length
2	Unrecognised code type
3	Wrong add-on code length
4	Illegal character in code
5	Error in embedded code
6	Generated line width less than 1 unit
7	font error
8	unable to create metafile
9	no code provided

char string[48]

This variable must contain the code string from which the barcode image is to be created before a call to BarCodem.

Note that in barcodes the difference between upper and lower case letters is significant. Few coding schemes support lower case letters.

int req_number

For the current barcode type (ie *itype* in a call to BarInfo), this variable returns the number of characters required for a valid code, or 0 for coding schemes which do not specify a number of characters.

int parity

For the current barcode type (ie *itype* in a call to BarInfo), this variable returns 1 if a checkdigit is required for a valid code, or 0 for coding schemes which do not require a checkdigit.

int alphabetic

For the current barcode type (ie *itype* in a call to BarInfo), this variable returns 1 for coding schemes which support alphabetic characters as well as numbers, or 0 for schemes which support only numbers.

int length

For the current barcode type (ie *itype* in a call to BarInfo), this variable returns the nominal length (in units of 0.01 mm) for codes which have a nominal size specified, or 0 for codes which may be any size.

int height

For the current barcode type (ie *itype* in a call to BarInfo), this variable returns the nominal height (in units of 0.01 mm) for codes which have a nominal size specified, or 0 for codes which may be any size.

int subscript;

For the current barcode type (ie *itype* in a call to BarInfo), this variable returns the number of supplementary characters required for a valid code, or 0 for coding schemes which do not require a supplement.

Note that this number is does not include the number of characters in the primary code; thus EAN 13 with a 5 character supplement returns 5 in this variable.

int flags;

The flags variable holds a number of bit flags which determine aspects of the formatting for some barcode types. The variable may be filled by ORing the available codes defined in DLSBARn.H:

#define DL_FLAG_MARGINS 1 specifies that EAN codes should show margin indicators (ie. < and > marks), and that UPC-A and UPC-E codes should have the Country Prefix code and the checkdigit printed in the light margin regions. Also used to show the end bars for ITF-6 and ITF-14 codes although in this case the flag is only applied if the Bearers flag is also set.

#define DL_FLAG_BEARERS 2 specifies that Bearer bars should be shown for ITF-6 and ITF-14 codes.

#define DL_FLAG_TRANSPARENT 4 specifies that the barcode image is generated with no background (ie any background colour is ignored). However, this flag should be used with caution as a background image showing through a barcode can disrupt the scanning process.

Two additional flags have been added, DL_FLAG_EXTRA1 and DL_FLAG_EXTRA2. These will normally be unused. They do, however, provide for extra functions which are specific to particular barcode types. For details see the Barcodes HELP system.

int textgap;

This variable specifies a distance (as a percentage of the Text Font Height) by which the text form of the code is raised before displaying. This variable has been included because some applications do not correctly align text when displaying metafiles.

int spare;

This variable is is used to define the spacing between characters which appear in the text under a barcode. The default value of 0 results in the text being spaced out using the default Windows spacing (ie normal text).
An alternative value of between 10 and 100% may be used to specify the percentage of the barcode width which the text will occupy. Thus a value of 100% causes the text form to be spread out to occupy the entire width of the barcode.

Other variables

```
char   string2[48];
int pcfont2;
```

These variables are not used in the present release of dBarcode DLL, but are included in the structure definition to allow for planned developments.

The Library calls

BarInfo

BarCodem

BarCodeh

BarCodeB

BarInfo

Calls to BarInfo with an integer value specifiying the barcode type in itype will return bc containing information about that type, including its abbreviated name, nominal length and height (if any), required number of characters (if restricted), check digit (or parity) requirements, alphabetic character support, and whether the code supports or requires a subscript (or supplementary code).

BarInfo return the value of itype for code types which it can process; otherwise it returns -1.

Thus to fill a list box with the names of available code types the following sample code may be used:

```
for (i=0; (BarInfo(hInstance,i,&bc)>=0); i++){
SendDlgItemMessage(hDlg,IDD_TYPE,CB_ADDSTRING,0,(LONG)(LPSTR)bc.name);
        }
```

When a specific code type has been chosen a BARCODE structure may be filled with information about the code using

```
bc.code=n;
BarInfo(hInstance,bc.code,&bc);
```

bc.req_number now contains the number of characters required for the primary code, bc.subscript contains the number required for any supplementary, bc.parity is 1 if the code type supports a check digit, etc.

int FAR PASCAL BarInfo(HANDLE hInstance, int itype,LPBAR bc)

where

hInstance is the instance handle of the calling application. (May be 0 for 16 bit applications)

itype is the barcode type index (0-27), and

bc is a long pointer to a BARCODE structure.

Users of the 32-bit version of the DLL should see the Notes on the 32-bit DLL

BarCodem

Calls to BarCodem create the barcode metafile image and the image pattern (for developers who wish to draw their own bars), returning an error code in the barcode structure in the event of an error.

Thus the following sample code shows a call to BarCodem followed by the code to display the metafile on the the device specified by hDC within the rectangle defined by lpRect (the device context in this example is in MM_ISOTROPIC mode and uses logical units of 0.01 mm with the Y value increasing DOWN the page. Your starting point may be different. If it is, for example if your Y increases UP the page, don't forget to adjust the calculation of **iht** to get a positive value - otherwise your image will be inverted!).

In this example the metafile is deleted once displayed.

```
BarCodem(0,lpRect,&logfont,(LPBAR)&bc,(LPHANDLE)&hmf,(LPSTR)szPat);
iht=lpRect->bottom-lpRect->top;
iwid=lpRect->right-lpRect->left;
CopyRect(&tRect,lpRect);
LPtoDP(hDC,(LPPOINT)&tRect,2);
i=(tRect.right-tRect.left);
j=(int)((long)i*(long)iht/(long)iwid);
if (hmf>0)
        {
        SaveDC(hDC);
        SetMapMode(hDC,MM_ANISOTROPIC);
        SetWindowOrg(hDC,0,0);
        SetViewportOrg(hDC,tRect.left,tRect.top);
        SetViewportExt(hDC,i, j);
        PlayMetaFile(hDC,hmf);
        RestoreDC(hDC,-1);
        DeleteMetaFile(hmf);

        }
```

Note that if bc.nominal_size had been set to 1, then the lpRect values would have been reset to represent the nominal size in units of 0.01 mm.

LPHANDLE FAR PASCAL BarCodem(HANDLE hInstance, LPRECT lpRectIn, LPLOGFONT lpLogFontIn, LPBAR bc, LPHANDLE &hmf, LPSTR szPat)

where

hInstance is the instance handle of the calling application. For 16 bit applications may be 0, in which case dBarcode creates its own using LoadLibrary.

lpRectIn is a long pointer to a rectangle which is the destination rectangle for the barcode image (with coordinates in units of 0.01 mm).

lpLogFontIn is a long pointer to a LOGFONT structure filled with information about the font with which any text will be displayed on the barcode. The font height will be treated as Points and treated as positive.

bc is a BARCODE structure - the LPBAR cast is required.

hmf is a HMETAFILEPICT handle which will contain the handle of the metafile image on return.

szPat if a long pointer to a string which receives a pattern of 0s and 1s representing spaces and bars respectively. This variable must be declared as

char szPat[1024];

before calling the library routines - even if you do not intend to use the pattern.

Users of the 32-bit version of the DLL should see the Notes on the 32-bit DLL

For the 32 bit DLL a call to BarCodemx is almost identical, except that the hmf handle and the handle returned are of type HENHMETAFILE and provide Enhanced metafile pictures rather than standard metafile pictures.

BarCodeh

Calls to BarCodeh draw the barcode image into the specified device context, rather than creating a metafile image. Apart from that the parameters are as defined for BarCodem.

int FAR PASCAL BarCodeh(HDC hDC, HANDLE hInstance, LPRECT lpRectInL, LPLOGFONT lpLogFontIn, LPBAR bc, LPSTR szPat);

where

hDC is the target device context

hInstance is the instance handle of the calling application (may be 0 for 16 bit applications).

lpRectIn is a long pointer to a rectangle which is the destination rectangle for the barcode image (with coordinates in units of 0.01 mm).

lpLogFontIn is a long pointer to a LOGFONT structure filled with information about the font with which any text will be displayed on the barcode. The font height will be treated as Points and treated as positive.

bc is a BARCODE structure - the LPBAR cast is required.

szPat if a long pointer to a string which receives a pattern of 0s and 1s representing spaces and bars respectively. This variable must be declared as

char szPat[1024];

before calling the library routines - even if you do not intend to use the pattern.

Users of the 32-bit version of the DLL should see the Notes on the 32-bit DLL

Calls to BarCodeh draws the barcode image on the specified device context and returns the image pattern (for developers who wish to draw their own bars or derive information about the number of elements), returning an error code in the barcode structure in the event of an error.

The following sample code shows a call to BarCodeh to display the barcode image on the the device specified by hDC within the rectangle defined by lpRect (the device context in this example is in MM_ISOTROPIC mode and uses logical units of 0.01 mm with the Y value increasing DOWN the page. Your starting point may be different.).

```
// lpRect is target rectangle in logical coords   ie 0.01 mm
CopyRect(&tRect,lpRect);
LPtoDP(hDC,(LPPOINT)&tRect,2);
SaveDC(hDC);
SetMapMode(hDC,MM_ANISOTROPIC);
SetWindowOrg(hDC,0,0);
SetViewportOrg(hDC,tRect.left,tRect.top);
SetViewportExt(hDC,tRect.right-tRect.left, tRect.bottom-tRect.top);
BarCodeh(hDC,0,lpRect,&logfont,(LPBAR)&bc,(LPSTR)szPat);
RestoreDC(hDC,-1);
```

BarCodeB

[16 bit only - for calling the DLL from Visual Basic 3]

This function allows a call to the DLL to be made from Visual Basic. The usual difficulty with calling the other functions is that Basic cannot pass the structures into the DLL, so the BarcodeB call passes only simple parameters - and only generates the pattern string (NOT the metafile picture)

The call is

int FAR PASCAL BarCodeb(int code, LPSTR string, int autoparity, int extras, LPINT perrorcode, LPSTR szPat);

where

int code is the code type
LPSTR string is the text to be converted into the barcode
int autoparity is non-zero is the DLL is to generate the check digit.
int extras carries carries Extra1 and Extra2 (and may be 0, 1 or 2)
LPINT perrorcode returns an error code, and
LPSTR szPat returns the pattern string.

The function returns the number of valid characters in the pattern string.

To call this function from Visual Basic the function must first be Declared in the Visual Basic module:

Declare Function BarcodeB Lib "DLSBAR2" (ByVal code As Integer, ByVal codestring As Any, ByVal AutoParity As Integer, ByVal extras As Integer, perrorcode As Integer, ByVal szPat As String) As Integer

The call may be made using code such as:

```
Dim code%, AutoParity%, errorcode%, extras%
Dim codestring$
code% = 8
codestring$ = "12345"
AutoParity% = 0
extras% = 0
errorcode% = 0
Pattern$ = String$(512, 0)
x% = BarcodeB(code%, codestring$, AutoParity%, extras%, errorcode%, Pattern$)
```

See the example in the DEMO program included with the dBarcode Developers Kit.

Notes on Metafiles

The picture images placed on the clipboard by dBarcode are ANISOTROPIC metafiles. This means that they can be resized within applications (usually by dragging a corner).

While the barcode bars can be resized over very wide ranges, any text included within the image may not resize as expected. In general changing the height of the image by resizing within another application will change the fontsize used to render the text. Changing the width of the image within another application may cause the position of any text under the barcode to change.

To overcome text size problems caused by resizing metafile images choose an alternative fontsize within dBarcode. The use of TrueType fonts is recommended to prevent unusual effects caused by resizing of text.

The 32 bit version of the DLL can also produce Enhanced metafile pictures using calls to BarCodemx.

Notes on the 32-bit DLL

The 32-bit version of the library uses essentially the same calls as the 16-bit version, ie. as described herein. However, in 32-bit Windows systems (both Windows NT and Windows 95) HINSTANCE parameters of the library calls have a somewhat different meaning from the corresponding parameter in 16-bit Windows, because DLLs are loaded into the virtual address space of the application which calls them. In 32-bit systems the HINSTANCE parameter is actually an HMODULE quantity and must refer to the module/instance of the DLL which is being used by the calling application.

This quantity may be obtained as the return value from a call such as

hInstance= GetModuleHandle("DLSBAR32");

or by calling the DLLs own GetHandle function using

hInstance= GhDlsbar();

If the latter option is used then the following declaration is required in the calling application:

HMODULE far PASCAL GhDlsbar(void);

Note also that the quantities passed as integers, either in library calls or via the BARCODE structure, are 32 bit integers and cannot be replaced by short integers. These quantities must be declared as **int** variables or the library calls will produce unexpected results.