Contents

The dBarcode Dynamic Link Library

dBarcode-2D DLL is a Dynamic Link Library which allows 2D-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-2D 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:

DLS2D.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 System directory of the system on which your application is being installed. This file needs to be distributed with your application.
DLS2D.LIC	is the licence file. This is a traceable licence file which must be present in the same directory as DLS2D.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.
DLS2D.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.
DLS2D.LIB	The LIB file created during the compilation of the DLL using Microsoft Visual C.

Data structure

The dBarcode-2D DLL uses a single data structure for transferring data into the library routines, BARCODE2D, which is defined in DLS2D.H along with a long pointer to the structure (LPBAR2D).

```
typedef struct tagBARCODE2D
    {
    int                 code;
    char                name[12];
    int                 xunit;
    int                  yunit;
    int                 start_mode;
    int                 security_level;
    float               aspect_ratio;
    int                 orientation;
    int                 nominal_size;
    int         line_reduction;
    int         border_thickness;
    COLORREF            fore_color;
    COLORREF            back_color;
    int                 error;
    char                string[4096];
    int                 req_number;
    int                 parity;
    int         length;
    int         height;
    int         flags;
    int         spare;
    } BARCODE2D;
```

typedef BARCODE2D* LPBAR2D;

A BARCODE2D structure must exist before any library routines are called, so your program requires a declaration such as

BARCODE2D bc;

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

Some members of the Barcode2D 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 Bar2Dm(HANDLE hModule, LPRECT lpRectIn, LPBAR2D bc, LPHANDLE &hmf, LPSTR szPat)

where

hModule is the 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).

bc is a BARCODE2D structure - the LPBAR2D 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[4096];

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

Calls to Bar2Dm create the barcode metafile image and the image pattern (for developers who wish to draw their own bars), returning an error code in the barcode2d structure in the event of an error.

int FAR PASCAL Bar2DInf(HANDLE hModule, int itype,LPBAR2D bc)

where

hModule is the Module 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 Bar2DInf 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 Bar2Dh(HDC hDC, HANDLE hModule, LPRECT lpRectInL, LPBAR2D bc, LPSTR szPat);

Calls to Bar2Dh draw the barcode image into the specified device context, with other parameters as defined for Bar2Dm.

where

hDC is the target device context

hModule is the 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).

bc is a BARCODE2D structure - the LPBAR2D 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[4096];

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

Calls to Bar2Dh 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 barcode2D structure in the event of an error.

HMODULE WINAPI GhDlsbar()

Calls to GhDlsbar return the HMODULE of the DLL and provide a convenient way of obtaining the module handle required for other library calls.

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), DBCOCX32.OCX (32 bit) or DB2DOCX.OCX (32 bit two dimensional) .

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 Two dimensional DLL is DLS2D.DLL and the corresponding licence file is DLS2D.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 BARCODE2D structure

The dBarcode-2D DLL uses a single data structure for transferring data into the library routines, BARCODE2D, which is defined in DLS2D.H along with a long pointer to the structure (LPBAR2D).

```
typedef struct tagBARCODE2D
	{
	int			code;
	char			name[12];
	int			xunit;
	int			 yunit;
	int			start_mode;
	int			security_level;
	float			aspect_ratio;
	int			orientation;
	int			nominal_size;
	int		line_reduction;
	int		border_thickness;
	COLORREF		fore_color;
	COLORREF		back_color;
	int			error;
	char			string[4096];
	int			req_number;
	int			parity;
	int		length;
	int		height;
	int		flags;
	int		spare;
	} BARCODE2D;
```

typedef BARCODE2D* LPBAR2D;

A BARCODE2D structure must exist before any library routines are called, so your program requires a declaration such as

BARCODE2D bc;

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

More:

int code

char name[12]

int xunit

int yunit

int start_mode

int security_level

float aspect_ratio

int orientation

int nominal_size

int line_reduction

int border_thickness

COLORREF fore_color

COLORREF back_color

int error

char string[4096]

int req_number

int parity

int length

int height

int flags

int spare

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 Bar2DInf 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 xunit

The nominal size of an X-unit (ie the width of the smallest bar) in units of 0.01 mm. As the metafile images are fully resizeable this parameter serves only to set the default size at which the metafile picture would be drawn within an application.

int yunit

The nominal size of an Y-unit (ie the height of a bar) in units of 0.01 mm. As the metafile images are fully resizeable this parameter serves only to set the default size at which the metafile picture would be drawn within an application.

int start_mode

Some 2D barcode types support different modes of encoding data. This parameter allows the initial mode to be specified, according to the table below.

Code 16k (see 2D Barcodes Help for details

start_mode parameter	**Start mode**
0	Automatic
1	Mode 0
2	Mode 1
3	Mode 2
4	Mode 3
5	Mode 4
6	Mode 5
7	Mode 6

PDF417 (see 2D Barcodes Help for details)

start_mode parameter	**Start mode**
0	EXC Alpha
1	EXC Lower
2	EXC Mixed
3	EXC Punctuation.
4	Binary/ASCII Plus
5	Numeric mode

For most applications this parameter should be 0 for PDF417, the algorithm within the DLL will adjust to the most appropriate mode.

int security_level

This parameter specifies the security level for a PDF417 code - i.e. the amount of redundancy built in to the barcode image to allow for errors to be corrected. The greater the security level the larger the barcode image. See 2D-Barcodes Help for details.

Allowed values are 0 - 8 for PDF417. For other barcode types this parameter should be 0.

float aspect_ratio

This parameter allows the user to specify the aspect ratio (rows/codewords per row) for PDF417 barcodes. While the allowed range is 0.1 - 10.0 users should bear in mind that PDF417 requires a minimum of three rows of symbols, and that ratios which result in a wide code may be difficult to scan.

This parameter is ignored for other barcode types.

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

int nominal_size

Not used

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 mm) 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 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 Bar2Dm. 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[4096]

This variable must contain the code string from which the barcode image is to be created before a call to Bar2Dm.

Note that in barcodes the difference between upper and lower case letters is significant.

int   req_number

Not used in this release

int parity

Not used in this release

int length

Not used in this release

int height

Not used in this release

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 DLS2D.H:

#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 spare

Not used in this release

The Library calls

Bar2Dinf

Bar2Dm

Bar2Dh

Bar2Dinf

Calls to Bar2DInf with an integer value specifying the barcode type in itype will return bc the abbreviated name of that type

Bar2DInf returns 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; (Bar2DInf(hModule,i,&bc)>=0); i++){
SendDlgItemMessage(hDlg,IDD_TYPE,CB_ADDSTRING,0,(LONG)(LPSTR)bc.name);
    }
```

int FAR PASCAL BarInfo(HANDLE hModule, int itype,LPBAR2D bc)

where

hModule is the Module handle of the calling application.

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

bc is a long pointer to a BARCODE2D structure.

Bar2Dm

Calls to Bar2Dm 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 Bar2Dm followed by the code to display the metafile on 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.

```
lph=(HMETAFILE*)Bar2Dm(GhDlsbar(),lpRect,(LPBAR2D)&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);

        }
```

LPHANDLE FAR PASCAL Bar2Dm(HANDLE hModule, LPRECT lpRectIn, LPBAR2D bc, LPHANDLE &hmf, LPSTR szPat)

where

hModule is the 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).

bc is a BARCODE2D structure - the LPBAR2D 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[4096];

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

Bar2Dh

Calls to Bar2Dh draw the barcode image into the specified device context, rather than creating a metafile image. Apart from that the parameters are as defined for Bar2Dm.

int FAR PASCAL Bar2Dh(HDC hDC, HANDLE hModule, LPRECT lpRectInL, LPBAR2D bc, LPSTR szPat);

where

hDC is the target device context

hModule is the 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).

bc is a BARCODE2D structure - the LPBAR2D 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[4096];

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

Calls to Bar2Dh 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 barcode2d structure in the event of an error.

The following sample code shows a call to Bar2Dh to display the barcode image on 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,,(LPBAR2D)&bc,(LPSTR)szPat);
RestoreDC(hDC,-1);
```