CHAPTER 1

DIB Format Extensions for Microsoft Windows

The DIB format extensions for Microsoft Windows add the capabilities to handle new compression formats, custom compression formats, and inverted DIBs. The extensions also include an escape message to let applications interrogate display drivers to determine their capabilities. This chapter includes the following topics related to these extensions:

- 16 and 32 bit extensions to the BI_RGB compression format
- 16 and 32 bit BI_BITFIELDS compression format extensions
- Extensions for custom compression formats
- Determining display driver capabilities
- Inverted DIBs

Windows Compression Formats

Compression flags for a bitmap are specified in the BITMAPINFOHEADER data structure defined by Windows. This structure has the following fields:

```
typedef struct tagBITMAPINFOHEADER {
   DWORD biSize;
   LONG  biWidth;
   LONG  biHeight;
   WORD  biPlanes;
   WORD  biBitCount;
   DWORD biCompression;
   DWORD biSizeImage;
   LONG  biXPelsPerMeter;
   LONG  biYPelsPerMeter;
   DWORD biClrUsed;
   DWORD biClrImportant;
} BITMAPINFOHEADER;
```

Information about the compression format is specified in the **biCompression** and **biBitCount** fields. The **biCompression** field specifies the type of compression used or requested. Both existing and new compression formats use this field.

The **biBitCount** field specifies the number of bits per pixel. Some compression formats need this information to properly decode the colors in the pixel.

When the value in the **biBitCount** field is set to less than or equal to eight, video drivers can assume the bitmap uses a palette or color table defined in the BITMAPINFO data structure. This data structure has the following fields:

```
typedef struct tagBITMAPINFO {
   BITMAPINFOHEADER bmiHeader;
   RGBQUAD bmiColors[1]
} BITMAPINFO;
```

When the value in the **biBitCount** field is set to greater than eight, video drivers can assume bitmaps are true color and they do not use a color table. For more information on these data structures, see the *Microsoft Windows Programmer's Reference*.

Existing Formats

Windows defines the following compression formats:

BI_RGB	Specifies the bitmap is not compressed. (Valid for **biBitCount** set to 1, 4, 8, 16, 24, or 32.)
BI_RLE8	Specifies a run-length encoded format for bitmaps with 8 bits per pixel. (Valid for **biBitCount** set to 8.)
BI_RLE4	Specifies a run-length encoded format for bitmaps with 4 bits per pixel. (Valid for **biBitCount** set to 4.)

For more information on these formats, see the *Microsoft Windows Programmer's Reference*.

Extensions to the BI_RGB Format

Extensions to the BI_RGB format include 16 and 32 bits per pixel bitmap formats. These formats do not use a color table. They embed the colors in the WORD or DWORD representing each pixel.

The 16 bit BI_RGB format is identified by setting **biCompression** to BI_RGB and **biBitCount** to 16. For this format, each pixel in the bitmap is represented by a 16 bit RGB color value. The high-bit of this value is zero. The remaining bits are divided into three groups of 5-bits to represent the red, green, and blue color values. The group containing the five most significant bits represents red. The group containing the five least significant bits represents blue. (This format is also referred to as the RGB555 format.

This format supports 32K colors.) The following illustration shows the bit organization of the RGB555 format:

16 bit BI_RGB format.

The 32 bit BI_RGB format is identified by setting **biCompression** to BI_RGB and **biBitCount** to 32. For this format, each pixel is represented by a 32 bit (4 byte) RGB color value. The first byte is zero. The second byte represents red, the third byte represents green, and the last byte represents blue. The following illustration shows the bit organization of this format:

32 bit BI_RGB format.

Display drivers must support the BI_RGB format for 1, 4, 8, and 24 bits per pixel bitmaps. If practical, they should also support this format for 16 and 32 bits per pixel bitmaps.

Formats Using BI_BITFIELDS and Color Masks

In addition to the 16 and 32 bits per pixel BI_RGB format, the BI_BITFIELDS flag has been defined for 16 and 32 bit bitmaps. This flag is recognized only by enhanced display drivers and does not need to be supported by most display drivers. The BI_BITFIELDS flag has the following definition:

BI_BITFIELDS	Specifies the bitmap is not compressed and a color mask is defined in the **bmiColors** field of the BITMAPINFO data structure. (Valid for **biBitCount** set to 16 or 32.)

Setting the **biCompression** field to BI_BITFIELDS indicates the **bmiColors** field contains three DWORDS used to mask each pixel in the bitmap. The masks are used to obtain the RGB color values of the pixel. The first DWORD contains the red mask, the second DWORD contains the green mask, and the third DWORD contains the blue mask. The image bits follow the three DWORDs. The color masks have the following characteristics:

- The bits in a mask must not overlap any bits in another mask.
- The set of bits defined for each mask must be contiguous.

These characteristics do not restrict any one mask to a particular location in a DWORD. For example, the red mask can occupy the least significant, most significant, or central position of the ORed combination of all three masks. The position of each mask corresponds to the color position defined for the appropriate RGB component of each pixel. This implies for a 16 bit image, the color masks will reside in the low-ordered word of the DWORD. (For 16 bit images, set the **biBitCount** field of the BITMAPINFOHEADER data structure to 16; for 32 bit images set it to 32.)

Additionally, you need to set the bits in a mask only for the bit positions in a pixel that represent color. Because unused bits in a pixel will always be masked, you can set the unused bits in a pixel to either zero or one.

For example, color masks can be used to decode the colors of a 16 bit pixel divided into three unequal groups of bits to represent the red, green, and blue color values. The group containing the five most significant bits represents red. The group containing the five least significant bits represents blue. The group containing the middle six bits represents green. (This format is also referred to as the RGB565 format.) The following illustration shows the definitions of the color masks and the bit organization of a pixel with the RGB565 format:

RGB565 format using BI_BITFIELDS.

Drivers obtain the RGB values for a pixel by masking the pixel with the DWORD corresponding to each color mask and then they map the colors to the appropriate registers for display. (If an application needs to retrieve the individual color values for a pixel, it can use the color masks to separate the color components and then right shift each color component by the number of least significant zeros in the mask.)

Custom Formats

Your driver can define custom compression and bitmap formats by assigning a four-character code to the **biCompression** field in place of the standard constants. When you define a custom format, you must specify the number of bytes in the image in the **biSizeImage** field.

The compression type four-character code must be unique. If you want to create a new four-character code for a compression type, register it with Microsoft to set up a standard definition of it and avoid any conflicts with other compression codes that might be defined. To register a code for a compression type, request a *Multimedia Developer Registration Kit* from the following group:

Microsoft Corporation
Multimedia Systems Group
Product Marketing
One Microsoft Way
Redmond, WA 98052-6399

The following is a list of the currently reserved compression types:

Four-character Code	biBitCount	Compression Method	Registered by
CRAM Cram	8, 16	Video compression	Microsoft
JPEG	24	JPEG format for images	Microsoft
YUV9	24, 16	411 YUV format for images	Microsoft
TYUV	8, 16	YUV	Microsoft
RYUV	8	Delta YUV	Microsoft

For more information on four character codes, see the *Microsoft Windows Multimedia Programmer's Guide*, *Microsoft Windows Multimedia Programmer's Reference*, and Chapter 10, "Video Compression and Decompression Drivers.".

Determining Display Driver Capabilities

You can determine if a display driver can handle a DIB with the QUERYDIBSUPPORT escape. The following syntax statement illustrates the use of this escape:

short **Escape**(*hdc*, QUERYDIBSUPPORT, *nSize*, *lpbi*, *lpFlags*)

The following parameter descriptions apply to the QUERYDIBSUPPORT escape:

Parameter	Data Type	Description
hdc	HDC	Identifies the device context.
nSize	int	Specifies the size of the BITMAPINFO data structure passed.
lpbi	LPBITMAPINFO	Points to a BITMAPINFO data structure containing the characteristics of the bitmap.

(Continued)

Parameter	Data Type	Description
lpFlags	LPINT	Points to an integer containing the return flags. Drivers set these flags to indicate which capabilities they support. The following flags are defined:

Flag	Description
QDI_SETDIBITS	Device can convert

	DIB to bitmap
QDI_GETDIBITS	Device can convert bitmap to DIB
QDI_DIBTOSCREEN	Device can draw DIB
QDI_STRETCHDIB	Device can stretch DIB

When a display driver gets this escape it should examine the BITMAPINFO structure indicated by *lpbi* and determine if it supports the DIB. The driver checks **biBitCount** for the proper bit depth, **biCompression** for the proper compression type, and **biHeight** for a positive or negative value (negative values indicate an inverted DIB). If **biCompression** is set to BI_BITFIELDS, the driver also checks the bit masks in the color table.

A display driver will set flags for *lpFlags* if it provides either partial or complete functionality corresponding to the flag. For example, a driver sets the QDI_STRETCHDIB flag if can stretch a DIB by integer amounts (partial functionality) or if it can stretch a DIB by both integer and non-integer values (complete functionality).

Inverted DIBs

Video drivers incorporating the DIB format extensions will let you specify negative values for **biHeight**. If **biHeight** is negative then the origin of the bitmap is the upper-left corner and the height is the absolute value of **biHeight**.

Applications determine if a driver supports inverted DIBs by sending the QUERYDIBSUPPORT flag with **biHeight** set to a negative value. Drivers return the QDI_DIBTOSCREEN flag in response to this if they support inverted DIBs.

Definition of the Flags and Escape

The flags, constants, and escape values described in this chapter are defined in MMREG.H. Use this header file until these flags, escapes, and constants are added to the header files distributed with Microsoft Windows.