# Graphics File Formats

This topic describes the graphics-file formats used by the Microsoft Windows operating system. Graphics files include bitmap files, icon-resource files, and cursor-resource files.

## Bitmap-File Formats

Windows bitmap files are stored in a device-independent bitmap (DIB) format that allows Windows to display the bitmap on any type of display device. The term "device independent" means that the bitmap specifies pixel color in a form independent of the method used by a display to represent color. The default filename extension of a Windows DIB file is .BMP.

## Bitmap-File Structures

Each bitmap file contains a bitmap-file header, a bitmap-information header, a color table, and an array of bytes that defines the bitmap bits. The file has the following form:

```
BITMAPFILEHEADER bmfh;
BITMAPINFOHEADER bmih;
RGBQUAD          aColors[];
BYTE             aBitmapBits[];
```

The bitmap-file header contains information about the type, size, and layout of a device-independent bitmap file. The header is defined as a BITMAPFILEHEADER structure.

The bitmap-information header, defined as a BITMAPINFOHEADER structure, specifies the dimensions, compression type, and color format for the bitmap.

The color table, defined as an array of RGBQUAD structures, contains as many elements as there are colors in the bitmap. The color table is not present for bitmaps with 24 color bits because each pixel is represented by 24-bit red-green-blue (RGB) values in the actual bitmap data area. The colors in the table should appear in order of importance. This helps a display driver render a bitmap on a device that cannot display as many colors as there are in the bitmap. If the DIB is in Windows version 3.0 or later format, the driver can use the biClrImportant member of the BITMAPINFOHEADER structure to determine which colors are important.

The BITMAPINFO structure can be used to represent a combined bitmap-information header and color table.  The bitmap bits, immediately following the color table, consist of an array of BYTE values representing consecutive rows, or "scan lines," of the bitmap. Each scan line consists of consecutive bytes representing the pixels in the scan line, in left-to-right order. The number of bytes representing a scan line depends on the color format and the width, in pixels, of the bitmap. If necessary, a scan line must be zero-padded to end on a 32-bit boundary. However, segment boundaries can appear anywhere in the bitmap. The scan lines in the bitmap are stored from bottom up. This means that the first byte in the array represents the pixels in the lower-left corner of the bitmap and the last byte represents the pixels in the upper-right corner.

The biBitCount member of the BITMAPINFOHEADER structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap. These members can have any of the following values:

| Value | Meaning |
|---|---|
| 1 | Bitmap is monochrome and the color table contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the color table. If the bit is set, the pixel has the color of the second entry in the table. |
| 4 | Bitmap has a maximum of 16 colors. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry. |
| 8 | Bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by a 1-byte index into the color table. For example, if the first byte in the bitmap is 0x1F, the first pixel has the color of the thirty-second table entry. |

24      Bitmap has a maximum of 2^24 colors. The bmiColors (or bmciColors) member is NULL, and each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, for a pixel.

The biClrUsed member of the BITMAPINFOHEADER structure specifies the number of color indexes in the color table actually used by the bitmap. If the biClrUsed member is set to zero, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member.  An alternative form of bitmap file uses the BITMAPCOREINFO, BITMAPCOREHEADER, and RGBTRIPLE structures.

Bitmap Compression

Windows versions 3.0 and later support run-length encoded (RLE) formats for compressing bitmaps that use 4 bits per pixel and 8 bits per pixel. Compression reduces the disk and memory storage required for a bitmap.

Compression of 8-Bits-per-Pixel Bitmaps

When the biCompression member of the BITMAPINFOHEADER structure is set to BI_RLE8, the DIB is compressed using a run-length encoded format for a 256-color bitmap. This format uses two modes: encoded mode and absolute mode. Both modes can occur anywhere throughout a single bitmap.

Encoded Mode

A unit of information in encoded mode consists of two bytes. The first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte.  The first byte of the pair can be set to zero to indicate an escape that denotes the end of a line, the end of the bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair, which must be in the range 0x00 through 0x02. Following are the meanings of the escape values that can be used in the second byte:

| Second byte | Meaning |
|---|---|
| 0 | End of line. |
| 1 | End of bitmap. |
| 2 | Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offsets of the next pixel from the current position. |

Absolute Mode

Absolute mode is signaled by the first byte in the pair being set to zero and the second byte to a value between 0x03 and 0xFF. The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. Each run must be aligned on a word boundary.  Following is an example of an 8-bit RLE bitmap (the two-digit hexadecimal values in the second column represent a color index for a single pixel):

| Compressed data | Expanded data |
|---|---|
| 03 04 | 04 04 04 |
| 05 06 | 06 06 06 06 06 |
| 00 03 45 56 67 00 | 45 56 67 |
| 02 78 | 78 78 |
| 00 02 05 01 | Move 5 right and 1 down |
| 02 78 | 78 78 |
| 00 00 | End of line |
| 09 1E | 1E 1E 1E 1E 1E 1E 1E 1E 1E |
| 00 01 | End of RLE bitmap |

Compression of 4-Bits-per-Pixel Bitmaps

When the biCompression member of the BITMAPINFOHEADER structure is set to BI_RLE4, the DIB is compressed using a run-length encoded format for a 16-color bitmap. This format uses two modes: encoded mode and absolute mode.

Encoded Mode

A unit of information in encoded mode consists of two bytes. The first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte.

The second byte contains two color indexes, one in its high-order nibble
(that is, its low-order 4 bits) and one in its low-order nibble.

The first pixel is drawn using the color specified by the high-order nibble,
the second is drawn using the color in the low-order nibble, the third is
drawn with the color in the high-order nibble, and so on, until all the
pixels specified by the first byte have been drawn.

The first byte of the pair can be set to zero to indicate an escape that
denotes the end of a line, the end of the bitmap, or a delta. The
interpretation of the escape depends on the value of the second byte of the
pair. In encoded mode, the second byte has a value in the range 0x00 through
0x02. The meaning of these values is the same as for a DIB with 8 bits per
pixel.

Absolute Mode

In absolute mode, the first byte contains zero, the second byte contains the
number of color indexes that follow, and subsequent bytes contain color
indexes in their high- and low-order nibbles, one color index for each pixel.
Each run must be aligned on a word boundary.

Following is an example of a 4-bit RLE bitmap (the one-digit hexadecimal
values in the second column represent a color index for a single pixel):

| Compressed data | Expanded data |
|---|---|
| 03 04 | 0 4 0 |
| 05 06 | 0 6 0 6 0 |
| 00 06 45 56 67 00 | 4 5 5 6 6 7 |
| 04 78 | 7 8 7 8 |
| 00 02 05 01 | Move 5 right and 1 down |
| 04 78 | 7 8 7 8 |
| 00 00 | End of line |
| 09 1E | 1 E 1 E 1 E 1 E 1 |
| 00 01 | End of RLE bitmap |

Bitmap Example

The following example is a text dump of a 16-color bitmap (4 bits per pixel):

```
Win3DIBFile
              BitmapFileHeader
                  Type       19778
                  Size       3118
                  Reserved1  0
                  Reserved2  0
                  OffsetBits 118
              BitmapInfoHeader
                  Size            40
                  Width           80
                  Height          75
                  Planes          1
                  BitCount        4
                  Compression     0
                  SizeImage       3000

                  XPelsPerMeter   0
                  YPelsPerMeter   0
                  ColorsUsed      16
                  ColorsImportant 16
              Win3ColorTable
                  Blue  Green  Red  Unused
[00000000]        84    252    84   0
[00000001]        252   252    84   0
[00000002]        84    84     252  0
[00000003]        252   84     252  0
[00000004]        84    252    252  0
[00000005]        252   252    252  0
[00000006]        0     0      0    0
[00000007]        168   0      0    0
[00000008]        0     168    0    0
[00000009]        168   168    0    0
[0000000A]        0     0      168  0
[0000000B]        168   0      168  0
```

```
[0000000C]        0     168    168  0
[0000000D]        168   168    168  0
[0000000E]        84    84     84   0
[0000000F]        252   84     84   0
              Image
    .
    .                                         Bitmap data
    .
```

## Icon-Resource File Format

An icon-resource file contains image data for icons used by Windows applications. The file consists of an icon directory identifying the number and types of icon images in the file, plus one or more icon images. The default filename extension for an icon-resource file is .ICO.

### Icon Directory

Each icon-resource file starts with an icon directory. The icon directory, defined as an ICONDIR structure, specifies the number of icons in the resource and the dimensions and color format of each icon image. The ICONDIR structure has the following form:

```
typedef struct ICONDIR {
    WORD          idReserved;
    WORD          idType;
    WORD          idCount;
    ICONDIRENTRY  idEntries[1];
} ICONHEADER;
```

Following are the members in the ICONDIR structure:

idReserved      Reserved; must be zero.
idType          Specifies the resource type. This member is set to 1.
idCount         Specifies the number of entries in the directory.
idEntries       Specifies an array of ICONDIRENTRY structures containing information about individual icons. The idCount member specifies the number of structures in the array.

The ICONDIRENTRY structure specifies the dimensions and color format for an icon. The structure has the following form:

```
struct IconDirectoryEntry {
    BYTE  bWidth;
    BYTE  bHeight;
    BYTE  bColorCount;
    BYTE  bReserved;
    WORD  wPlanes;
    WORD  wBitCount;
    DWORD dwBytesInRes;
    DWORD dwImageOffset;
};
```

Following are the members in the ICONDIRENTRY structure:

bWidth          Specifies the width of the icon, in pixels. Acceptable values are 16, 32, and 64.

bHeight         Specifies the height of the icon, in pixels. Acceptable values are 16, 32, and 64.

bColorCount     Specifies the number of colors in the icon. Acceptable values are 2, 8, and 16.

bReserved       Reserved; must be zero.
wPlanes         Specifies the number of color planes in the icon bitmap.
wBitCount       Specifies the number of bits in the icon bitmap.
dwBytesInRes    Specifies the size of the resource, in bytes.
dwImageOffset   Specifies the offset, in bytes, from the beginning of the file to the icon image.

### Icon Image

Each icon-resource file contains one icon image for each image identified in the icon directory. An icon image consists of an icon-image header, a color table, an XOR mask, and an AND mask. The icon image has the following form:

```
BITMAPINFOHEADER    icHeader;
RGBQUAD             icColors[];
BYTE                icXOR[];
BYTE                icAND[];
```

The icon-image header, defined as a BITMAPINFOHEADER structure, specifies the dimensions and color format of the icon bitmap. Only the biSize through biBitCount members and the biSizeImage member are used. All other members (such as biCompression and biClrImportant) must be set to zero.

The color table, defined as an array of RGBQUAD structures, specifies the colors used in the XOR mask. As with the color table in a bitmap file, the biBitCount member in the icon-image header determines the number of elements in the array. For more information about the color table, see Section 1.1, "Bitmap-File Formats."

The XOR mask, immediately following the color table, is an array of BYTE values representing consecutive rows of a bitmap. The bitmap defines the basic shape and color of the icon image. As with the bitmap bits in a bitmap file, the bitmap data in an icon-resource file is organized in scan lines, with each byte representing one or more pixels, as defined by the color format. For more information about these bitmap bits, see Section 1.1, "Bitmap-File Formats."

The AND mask, immediately following the XOR mask, is an array of BYTE values, representing a monochrome bitmap with the same width and height as the XOR mask. The array is organized in scan lines, with each byte representing 8 pixels.

When Windows draws an icon, it uses the AND and XOR masks to combine the icon image with the pixels already on the display surface. Windows first applies the AND mask by using a bitwise AND operation; this preserves or removes existing pixel color.  Windows then applies the XOR mask by using a bitwise XOR operation. This sets the final color for each pixel.

The following illustration shows the XOR and AND masks that create a monochrome icon (measuring 8 pixels by 8 pixels) in the form of an uppercase K:

## Windows Icon Selection

Windows detects the resolution of the current display and matches it against the width and height specified for each version of the icon image. If Windows determines that there is an exact match between an icon image and the current device, it uses the matching image. Otherwise, it selects the closest match and stretches the image to the proper size.

If an icon-resource file contains more than one image for a particular resolution, Windows uses the icon image that most closely matches the color capabilities of the current display. If no image matches the device capabilities exactly, Windows selects the image that has the greatest number of colors without exceeding the number of display colors. If all images exceed the color capabilities of the current display, Windows uses the icon image with the least number of colors.

## Cursor-Resource File Format

A cursor-resource file contains image data for cursors used by Windows applications. The file consists of a cursor directory identifying the number and types of cursor images in the file, plus one or more cursor images. The default filename extension for a cursor-resource file is .CUR.

### Cursor Directory

Each cursor-resource file starts with a cursor directory. The cursor directory, defined as a CURSORDIR structure, specifies the number of cursors in the file and the dimensions and color format of each cursor image. The

CURSORDIR structure has the following form:

```
typedef struct _CURSORDIR {
    WORD           cdReserved;
    WORD           cdType;
    WORD           cdCount;
    CURSORDIRENTRY cdEntries[];
} CURSORDIR;
```

Following are the members in the CURSORDIR structure:

| | |
|---|---|
| cdReserved | Reserved; must be zero. |
| cdType | Specifies the resource type. This member must be set to 2. |
| cdCount | Specifies the number of cursors in the file. |
| cdEntries | Specifies an array of CURSORDIRENTRY structures containing information about individual cursors. The cdCount member specifies the number of structures in the array. |

A CURSORDIRENTRY structure specifies the dimensions and color format of a cursor image. The structure has the following form:

```
typedef struct _CURSORDIRENTRY {
    BYTE  bWidth;
    BYTE  bHeight;
    BYTE  bColorCount;
    BYTE  bReserved;
    WORD  wXHotspot;
    WORD  wYHotspot;
    DWORD lBytesInRes;
    DWORD dwImageOffset;
} CURSORDIRENTRY;
```

Following are the members in the CURSORDIRENTRY structure:

| | |
|---|---|
| bWidth | Specifies the width of the cursor, in pixels. |
| bHeight | Specifies the height of the cursor, in pixels. |
| bColorCount | Reserved; must be zero. |
| bReserved | Reserved; must be zero. |
| wXHotspot | Specifies the x-coordinate, in pixels, of the hot spot. |
| wYHotspot | Specifies the y-coordinate, in pixels, of the hot spot. |
| lBytesInRes | Specifies the size of the resource, in bytes. |
| dwImageOffset | Specifies the offset, in bytes, from the start of the file to the cursor image. |

Cursor Image

Each cursor-resource file contains one cursor image for each image identified in the cursor directory. A cursor image consists of a cursor-image header, a color table, an XOR mask, and an AND mask. The cursor image has the following form:

```
BITMAPINFOHEADER    crHeader;
RGBQUAD             crColors[];
BYTE                crXOR[];
BYTE                crAND[];
```

The cursor hot spot is a single pixel in the cursor bitmap that Windows uses to track the cursor. The crXHotspot and crYHotspot members specify the x- and y-coordinates of the cursor hot spot. These coordinates are 16-bit integers.

The cursor-image header, defined as a BITMAPINFOHEADER structure, specifies the dimensions and color format of the cursor bitmap. Only the biSize through biBitCount members and the biSizeImage member are used. The biHeight member specifies the combined height of the XOR and AND masks for the cursor. This value is twice the height of the XOR mask. The biPlanes and biBitCount members must be 1. All other members (such as biCompression and biClrImportant) must be set to zero.

The color table, defined as an array of RGBQUAD structures, specifies the colors used in the XOR mask. For a cursor image, the table contains exactly two structures, since the biBitCount member in the cursor-image header is always 1.

The XOR mask, immediately following the color table, is an array of BYTE values representing consecutive rows of a bitmap. The bitmap defines the basic shape and color of the cursor image. As with the bitmap bits in a bitmap file, the bitmap data in a cursor-resource file is organized in scan lines, with each byte representing one or more pixels, as defined by the color format. For more information about these bitmap bits, see Section 1.1, "Bitmap-File Formats."

The AND mask, immediately following the XOR mask, is an array of BYTE values representing a monochrome bitmap with the same width and height as the XOR mask. The array is organized in scan lines, with each byte representing 8 pixels.

When Windows draws a cursor, it uses the AND and XOR masks to combine the cursor image with the pixels already on the display surface. Windows first applies the AND mask by using a bitwise AND operation; this preserves or removes existing pixel color.  Window then applies the XOR mask by using a bitwise XOR operation. This sets the final color for each pixel.

The following illustration shows the XOR and the AND masks that create a cursor (measuring 8 pixels by 8 pixels) in the form of an arrow:

Following are the bit-mask values necessary to produce black, white, inverted, and transparent results:

| Pixel result | AND mask | XOR mask |
|---|---|---|
| Black | 0 | 0 |
| White | 0 | 1 |
| Transparent | 1 | 0 |
| Inverted1 | 1 | |

Windows Cursor Selection

If a cursor-resource file contains more than one cursor image, Windows determines the best match for a particular display by examining the width and height of the cursor images.

==============================================================================

BITMAPFILEHEADER (3.0)

```
typedef struct tagBITMAPFILEHEADER {    /* bmfh */
    UINT    bfType;
    DWORD   bfSize;
    UINT    bfReserved1;
    UINT    bfReserved2;
    DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

The BITMAPFILEHEADER structure contains information about the type, size, and layout of a device-independent bitmap (DIB) file.

| Member | Description |
|---|---|
| bfType | Specifies the type of file. This member must be BM. |
| bfSize | Specifies the size of the file, in bytes. |
| bfReserved1 | Reserved; must be set to zero. |
| bfReserved2 | Reserved; must be set to zero. |
| bfOffBits | Specifies the byte offset from the BITMAPFILEHEADER structure to the actual bitmap data in the file. |

Comments

A BITMAPINFO or BITMAPCOREINFO structure immediately follows the BITMAPFILEHEADER structure in the DIB file.

See Also

BITMAPCOREINFO, BITMAPINFO

==============================================================================
## BITMAPINFO (3.0)

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

The BITMAPINFO structure fully defines the dimensions and color information for a Windows 3.0 or later device-independent bitmap (DIB).

| Member | Description |
|---|---|
| bmiHeader | Specifies a BITMAPINFOHEADER structure that contains information about the dimensions and color format of a DIB. |
| bmiColors | Specifies an array of RGBQUAD structures that define the colors in the bitmap. |

Comments

A Windows 3.0 or later DIB consists of two distinct parts: a BITMAPINFO structure, which describes the dimensions and colors of the bitmap, and an array of bytes defining the pixels of the bitmap. The bits in the array are packed together, but each scan line must be zero-padded to end on a LONG boundary. Segment boundaries, however, can appear anywhere in the bitmap. The origin of the bitmap is the lower-left corner.

The biBitCount member of the BITMAPINFOHEADER structure determines the number of bits which define each pixel and the maximum number of colors in the bitmap. This member may be set to any of the following values:

| Value | Meaning |
|---|---|
| 1 | The bitmap is monochrome, and the bmciColors member must contain two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the bmciColors table. If the bit is set, the pixel has the color of the second entry in the table. |
| 4 | The bitmap has a maximum of 16 colors, and the bmciColors member contains 16 entries. Each pixel in the bitmap is represented by a four-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry. |
| 8 | The bitmap has a maximum of 256 colors, and the bmciColors member contains 256 entries. In this case, each byte in the array represents a single pixel. |
| 24 | The bitmap has a maximum of 2^24 colors. The bmciColors member is NULL, and each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, of a pixel. |

The biClrUsed member of the BITMAPINFOHEADER structure specifies the number of color indexes in the color table actually used by the bitmap. If the biClrUsed member is set to zero, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member.

The colors in the bmiColors table should appear in order of importance. Alternatively, for functions that use DIBs, the bmiColors member can be an array of 16-bit unsigned integers that specify an index into the currently realized logical palette instead of explicit RGB values. In this case, an application using the bitmap must call DIB functions with the wUsage parameter set to DIB_PAL_COLORS.

Note:   The bmiColors member should not contain palette indexes if the bitmap is to be stored in a file or transferred to another application. Unless the application uses the bitmap exclusively and under its complete control, the bitmap color table should contain explicit RGB values.

See Also

BITMAPINFOHEADER, RGBQUAD

==============================================================================
BITMAPINFOHEADER (3.0)

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

The BITMAPINFOHEADER structure contains information about the dimensions and color format of a Windows 3.0 or later device-independent bitmap (DIB).

Member          Description

biSize          Specifies the number of bytes required by the BITMAPINFOHEADER structure.

biWidth         Specifies the width of the bitmap, in pixels.
biHeightSpecifies the height of the bitmap, in pixels.

biPlanesSpecifies the number of planes for the target device. This member must be set to 1.

biBitCount      Specifies the number of bits per pixel. This value must be 1, 4, 8, or 24.

biCompression   Specifies the type of compression for a compressed bitmap. It can be one of the following values:

Value           Meaning

BI_RGB          Specifies that the bitmap is not compressed.

BI_RLE8         Specifies a run-length encoded format for bitmaps with 8 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by a byte containing a color index.  For more information, see the following Comments section.

BI_RLE4         Specifies a run-length encoded format for bitmaps with 4 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes.  For more information, see the following Comments section.

biSizeImage     Specifies the size, in bytes, of the image. It is valid to set this member to zero if the bitmap is in the BI_RGB format.

biXPelsPerMeter Specifies the horizontal resolution, in pixels per meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

biYPelsPerMeter Specifies the vertical resolution, in pixels per meter, of the target device for the bitmap.

biClrUsed       Specifies the number of color indexes in the color table actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member. For more information on the maximum sizes of the color table, see the description of the BITMAPINFO structure earlier in this topic.

If the biClrUsed member is nonzero, it specifies the actual number of colors that the graphics engine or device driver will access if the biBitCount member is less than 24. If biBitCount is set to 24, biClrUsed specifies the size of the reference color table used to optimize performance of Windows color palettes.  If the bitmap is a packed bitmap (that is, a bitmap in which the bitmap array immediately follows the BITMAPINFO header and which is referenced by a single pointer), the biClrUsed member must be set to zero or to the actual size of the color table.

biClrImportant  Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is zero, all colors are important.

Comments

The BITMAPINFO structure combines the BITMAPINFOHEADER structure and a color table to provide a complete definition of the dimensions and colors of a Windows 3.0 or later DIB. For more information about specifying a Windows 3.0 DIB, see the description of the BITMAPINFO structure.

An application should use the information stored in the biSize member to locate the color table in a BITMAPINFO structure as follows:

```
pColor = ((LPSTR) pBitmapInfo + (WORD) (pBitmapInfo->bmiHeader.biSize))
```

Windows supports formats for compressing bitmaps that define their colors with 8 bits per pixel and with 4 bits per pixel. Compression reduces the disk and memory storage required for the bitmap. The following paragraphs describe these formats.

BI_RLE8

When the biCompression member is set to BI_RLE8, the bitmap is compressed using a run-length encoding format for an 8-bit bitmap. This format may be compressed in either of two modes: encoded and absolute. Both modes can occur anywhere throughout a single bitmap.

Encoded mode consists of two bytes: the first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. In addition, the first byte of the pair can be set to zero to indicate an escape that denotes an end of line, end of bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. The following list shows the meaning of the second byte:

| Value | Meaning |
|---|---|
| 0 | End of line. |
| 1 | End of bitmap. |
| 2 | Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offset of the next pixel from the current position. |

Absolute mode is signaled by the first byte set to zero and the second byte set to a value between 0x03 and 0xFF. In absolute mode, the second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. When the second byte is set to 2 or less, the escape has the same meaning as in encoded mode. In absolute mode, each run must be aligned on a word boundary.  The following example shows the hexadecimal values of an 8-bit compressed bitmap:

```
03 04 05 06 00 03 45 56 67 00 02 78 00 02 05 01
02 78 00 00 09 1E 00 01
```

This bitmap would expand as follows (two-digit values represent a color index for a single pixel):

```
04 04 04
06 06 06 06 06
45 56 67
78 78
move current position 5 right and 1 down
78 78
end of line
```

```
1E 1E 1E 1E 1E 1E 1E 1E 1E
end of RLE bitmap
```

BI_RLE4

When the biCompression member is set to BI_RLE4, the bitmap is compressed using a run-length encoding (RLE) format for a 4-bit bitmap, which also uses encoded and absolute modes. In encoded mode, the first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte. The second byte contains two color indexes, one in its high-order nibble (that is, its low-order four bits) and one in its low-order nibble. The first of the pixels is drawn using the color specified by the high-order nibble, the second is drawn using the color in the low-order nibble, the third is drawn with the color in the high-order nibble, and so on, until all the pixels specified by the first byte have been drawn. In absolute mode, the first byte contains zero, the second byte contains the number of color indexes that follow, and subsequent bytes contain color indexes in their high- and low-order nibbles, one color index for each pixel. In absolute mode, each run must be aligned on a word boundary. The end-of-line, end-of-bitmap, and delta escapes also apply to BI_RLE4.

The following example shows the hexadecimal values of a 4-bit compressed bitmap:

```
03 04 05 06 00 06 45 56 67 00 04 78 00 02 05 01
04 78 00 00 09 1E 00 01
```

This bitmap would expand as follows (single-digit values represent a color index for a single pixel):

```
0 4 0
0 6 0 6 0
4 5 5 6 6 7
7 8 7 8
move current position 5 right and 1 down
7 8 7 8
end of line
1 E 1 E 1 E 1 E 1
end of RLE bitmap
```

See Also

BITMAPINFO

==============================================================================

RGBQUAD (3.0)

```
typedef struct tagRGBQUAD {     /* rgbq */
    BYTE    rgbBlue;
    BYTE    rgbGreen;
    BYTE    rgbRed;
    BYTE    rgbReserved;
} RGBQUAD;
```

The RGBQUAD structure describes a color consisting of relative intensities of red, green, and blue. The bmiColors member of the BITMAPINFO structure consists of an array of RGBQUAD structures.

| Member | Description |
|---|---|
| rgbBlue | Specifies the intensity of blue in the color. |
| rgbGreen | Specifies the intensity of green in the color. |
| rgbRed | Specifies the intensity of red in the color. |
| rgbReserved | Not used; must be set to zero. |

See Also

BITMAPINFO

==============================================================================

```
RGB (2.x)

COLORREF RGB(cRed, cGreen, cBlue)

BYTE cRed;      /* red component of color       */
BYTE cGreen;    /* green component of color     */
BYTE cBlue;     /* blue component of color      */
```

The RGB macro selects an RGB color based on the parameters supplied and the color capabilities of the output device.

| Parameter | Description |
|---|---|
| cRed | Specifies the intensity of the red color field. |
| cGreen | Specifies the intensity of the green color field. |
| cBlue | Specifies the intensity of the blue color field. |

Returns

The return value specifies the resultant RGB color.

Comments

The intensity for each argument can range from 0 through 255. If all three intensities are specified as zero, the result is black. If all three intensities are specified as 255, the result is white.

Comments

The RGB macro is defined in WINDOWS.H as follows:

```
#define RGB(r,g,b)   ((COLORREF)(((BYTE)(r)|((WORD)(g)<<8))| \
    (((DWORD)(BYTE)(b))<<16)))
```

See Also

GetBValue, GetGValue, GetRValue, PALETTEINDEX, PALETTERGB

==============================================================================
BITMAPCOREINFO (3.0)

```
typedef struct tagBITMAPCOREINFO {  /* bmci */
    BITMAPCOREHEADER bmciHeader;
    RGBTRIPLE        bmciColors[1];
} BITMAPCOREINFO;
```

The BITMAPCOREINFO structure fully defines the dimensions and color information for a device-independent bitmap (DIB).  Windows applications should use the BITMAPINFO structure instead of BITMAPCOREINFO whenever possible.

| Member | Description |
|---|---|
| bmciHeader | Specifies a BITMAPCOREHEADER structure that contains information about the dimensions and color format of a DIB. |
| bmciColors | Specifies an array of RGBTRIPLE structures that define the colors in the bitmap. |

Comments

The BITMAPCOREINFO structure describes the dimensions and colors of a bitmap. It is followed immediately in memory by an array of bytes which define the pixels of the bitmap. The bits in the array are packed together, but each scan line must be zero-padded to end on a LONG boundary. Segment boundaries, however, can appear anywhere in the bitmap. The origin of the bitmap is the lower-left corner.

The bcBitCount member of the BITMAPCOREHEADER structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap. This member may be set to any of the following values:

| Value | Meaning |
|---|---|
| 1 | The bitmap is monochrome, and the bmciColors member must contain two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the bmciColors table. If the bit is set, the pixel has the color of the second entry in the table. |
| 4 | The bitmap has a maximum of 16 colors, and the bmciColors member contains 16 entries. Each pixel in the bitmap is represented by a four-bit index into the color table. |

For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.

| Value | Meaning |
|---|---|
| 8 | The bitmap has a maximum of 256 colors, and the bmciColors member contains 256 entries. In this case, each byte in the array represents a single pixel. |
| 24 | The bitmap has a maximum of 2^24 colors. The bmciColors member is NULL, and each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, of a pixel. |

The colors in the bmciColors table should appear in order of importance. Alternatively, for functions that use DIBs, the bmciColors member can be an array of 16-bit unsigned integers that specify an index into the currently realized logical palette instead of explicit RGB values. In this case, an application using the bitmap must call DIB functions with the wUsage parameter set to DIB_PAL_COLORS.

Note:   The bmciColors member should not contain palette indexes if the bitmap is to be stored in a file or transferred to another application. Unless the application uses the bitmap exclusively and under its complete control, the bitmap color table should contain explicit RGB values.

See Also

BITMAPINFO, BITMAPCOREHEADER, RGBTRIPLE

==============================================================================

BITMAPCOREHEADER (3.0)

```
typedef struct tagBITMAPCOREHEADER {    /* bmch */
    DWORD   bcSize;
    short   bcWidth;
    short   bcHeight;
    WORD    bcPlanes;
    WORD    bcBitCount;
} BITMAPCOREHEADER;
```

The BITMAPCOREHEADER structure contains information about the dimensions and color format of a device-independent bitmap (DIB). Windows applications should use the BITMAPINFOHEADER structure instead of BITMAPCOREHEADER whenever possible.

| Member | Description |
|---|---|
| bcSize | Specifies the number of bytes required by the BITMAPCOREHEADER structure. |
| bcWidth | Specifies the width of the bitmap, in pixels. |
| bcHeight | Specifies the height of the bitmap, in pixels. |
| bcPlanes | Specifies the number of planes for the target device. This member must be set to 1. |
| bcBitCount | Specifies the number of bits per pixel. This value must be 1, 4, 8, or 24. |

Comments

The BITMAPCOREINFO structure combines the BITMAPCOREHEADER structure and a color table to provide a complete definition of the dimensions and colors of a DIB. See the description of the BITMAPCOREINFO structure for more information about specifying a DIB.

An application should use the information stored in the bcSize member to locate the color table in a BITMAPCOREINFO structure with a method such as the following:

```
lpColor = ((LPSTR) pBitmapCoreInfo + (UINT) (pBitmapCoreInfo->bcSize))
```

See Also

BITMAPCOREINFO, BITMAPINFOHEADER, BITMAPINFOHEADER

==============================================================================
RGBTRIPLE (3.0)

```
typedef struct tagRGBTRIPLE {   /* rgbt */
    BYTE    rgbtBlue;
    BYTE    rgbtGreen;
    BYTE    rgbtRed;
} RGBTRIPLE;
```

The RGBTRIPLE structure describes a color consisting of relative intensities of red, green, and blue. The bmciColors member of the BITMAPCOREINFO structure consists of an array of RGBTRIPLE structures.  Windows applications should use the BITMAPINFO structure instead of BITMAPCOREINFO whenever possible. The BITMAPINFO structure uses an RGBQUAD structure instead of the RGBTRIPLE structure.

| Member | Description |
|---|---|
| rgbtBlue | Specifies the intensity of blue in the color. |
| rgbtGreen | Specifies the intensity of green in the color. |
| rgbtRed | Specifies the intensity of red in the color. |

See Also

BITMAPCOREINFO, BITMAPINFO, RGBQUAD

==============================================================================