Microsoft Windows Bitmap File Format Summary

Also Known As:

BMP, DIB, Windows BMP, Windows DIB, Compatible Bitmap

Type	Bitmap
Colors	1-, 4-, 8-, 16-, 24-, and 32-bits
Compression	RLE, uncompressed
Maximum Image Size	32Kx32K and 2Gx2G pixels
Multiple Images Per File	No
Numerical Format	Little-endian
Originator	Microsoft Corporation
Platform	Intel machines running Microsoft Windows, Windows NT, Windows 95, OS/2, and MS-DOS

Usage
Used as the standard bitmap storage format in the Microsoft Windows environment. Although it is based on Windows internal bitmap data structures, it is supported by many non-Windows and non-PC applications.

Comments
A well-defined format for programmers having access to the Microsoft Developer's Network Knowledge Base and Software Development Kits (SDKs). Its simple RLE compression scheme is rather inefficient for complex images. Its many variations and differences from the OS/2 BMP format can be confusing.

The Microsoft Windows Bitmap (BMP) file format is one of several graphics file formats supported by the Microsoft Windows operating environment. BMP is the native bitmap format of Windows and is used to store virtually any type of bitmap data. Most graphics and imaging applications running under Microsoft Windows support the creation and display of BMP format files. BMP is also very popular in the MS-DOS operating system. It is also the native bitmap file format of OS/2.

The original bitmap format created for Windows 1.0 was very simple. It had a fixed color palette, did not support bitmap data compression, and was designed to support the most popular IBM PC graphics cards in use at the time (CGA, EGA, Hercules, and others). This defunct format is now often referred to as the original Windows device-dependent bitmap (DDB).

As Windows 2.0 was being developed, support for a programmable color palette was added to both Windows and BMP, allowing user-definable color data to be stored along with the bitmap data that used it. When stored in memory, this collection of information is known as a Microsoft Device Independent Bitmap, or DIB. When this information is written out to a file, it is known as the Microsoft Bitmap Format, or BMP. When you hear references to the DIB file format, it is BMP that is actually being referred to.

During the development of BMP, Microsoft shared responsibility with IBM for the development of early versions of IBM's OS/2 operating system. When Presentation Manager, the OS/2 graphical user interface, required a bitmap format, the Windows BMP format was used. Thus, the Windows 2.*x* and OS/2 1.*x* BMP formats are identical.

The BMP format modified for Windows 3.0 differs only slightly from the OS/2 Presentation Manager bitmap format that preceded it. Note that later revisions designed to support IBM OS/2 Presentation Manager 2.*x* have resulted in further divergence between the Microsoft Windows and IBM OS/2 BMP file formats. The current version of BMP for Windows 4.0 (Windows 95) contains all of the features and history of the Windows 2.*x*, 3.*x*, and Windows NT BMP formats.

The structure of BMP format files is closely tied to the API of both Windows and OS/2. In this regard, BMP was never meant to be a portable format or used for bitmap data interchange between different operating systems. As each of these operating system APIs has changed, the BMP format has changed along with it.

There are currently three versions of BMP under Windows (2.*x*, 3.*x*, and 4.*x* [Windows 95]), two versions under OS/2 (1.*x* and 2.*x*, with six possible variations), and a single version for Windows NT. This article details the three versions used under Microsoft Windows, as well as the Windows NT version. The original Microsoft device-dependent bitmap format is also discussed. For a discussion of the OS/2 BMP format versions and variants, see the article about the OS/2 BMP format.

All of the BMP versions originated on Intel-based machines and thus share a common little-endian heritage. The current BMP format is otherwise hardware-independent and can accommodate images with up to 32-bit color. Its basic design makes it a good general purpose format that can be used for color or black-and-white image storage if file size is not a factor. Its main virtues are its simplicity and widespread support in the PC marketplace.

The compression method used is a type of run-length encoding (RLE), although most BMP files to date have been stored uncompressed. A notable exception is the Microsoft Windows 3.1 sign-on screen shipped with all copies of the product. Although the BMP RLE scheme is lossless and easily and quickly decompressed, it is not considered a superior compression method.

Although the BMP format is well-defined, there is no actual format specification document published by Microsoft. Information about structure and data encoding methods is contained in a number of programmer's references, manuals, online help facilities, and include files associated with the Microsoft Windows Software Development Kits (SDKs) and Microsoft Developers Network Knowledge Base.

File Organization

Windows 1.*x* DDB files contain two sections: a file header and the bitmap data. There is no provision for a color palette or any other features that would make this format device-independent. Support for compression of the bitmap data is also lacking.

File Header
Bitmap Data

Windows 2.*x*, 3.*x*, and 4.*x* BMP files contain four sections: a file header, a bitmap information header, a color palette, and the bitmap data. Of these four sections, only the palette information may be optional, depending on the bit depth of the bitmap data. The BMP file header is 14 bytes in length and is nearly identical to the 1.*x* DDB header. The file header is followed by a second header (called the bitmap header), a variable-sized palette, and the bitmap data.

File Header
Bitmap Header
Color Palette
Bitmap Data

File Details

This section describes the original Windows DDB format and BMP format versions 2*x*, 3*x*, and 4*x* in greater detail.

Version 1 Device-Dependent Bitmap (Microsoft Windows 1.x)

DDB files contain only a file header followed by uncompressed bitmap data. The following shows the structure of the 10-byte DDB file header:

```
typedef struct _Win1xHeader
{
    WORD Type;           /* File type identifier (always 0) */
    WORD Width;          /* Width of the bitmap in pixels */
    WORD Height;         /* Height of the bitmap in scan lines */
    WORD ByteWidth;      /* Width of bitmap in bytes */
    BYTE Planes;         /* Number of color planes */
    BYTE BitsPerPixel;   /* Number of bits per pixel */
} WIN1XHEADER;
```

Type indicates the file type; for v1.*x* headers, it is always 0.

Width and Height represent the size of the bitmap in pixels and in scan lines, respectively.

ByteWidth shows the width of the bitmap in bytes. It is assumed that this value will include the size of any scan line padding that is present.

Planes is the number of color planes used to store the bitmap. This value is always 1.

BitsPerPixel is the size of each pixel in bits. This value is typically 1, 4, or 8.

The image data immediately follows the header and is stored in an uncompressed format. Each pixel stores an index value into the fixed system colormap used by Windows 1.0. The presence of scan line padding may be determined by comparing the calculated width of a line in bytes with the actual width of the line in bytes stored as the value of the ByteWidth field.

BMP Version 2 (Microsoft Windows 2.x)

All versions of BMP format files begin with the following 14-byte header:

```
typedef struct _WinBMPFileHeader
{
    WORD   FileType;     /* File type, always 4D42h ("BM") */
    DWORD  FileSize;     /* Size of the file in bytes */
    WORD   Reserved1;    /* Always 0 */
    WORD   Reserved2;    /* Always 0 */
    DWORD  BitmapOffset; /* Starting position of image data in bytes */
} WINBMPFILEHEADER;
```

FileType holds a 2-byte magic value used to identify the file type; it is always 4D42h or "BM" in ASCII. If your application reads Windows bitmap files, make sure to always check this field before attempting to use any of the data read from the file.

FileSize is the total size of the BMP file in bytes and should agree with the file size reported by the filesystem. This field only stores a useful value when the bitmap data is compressed, and this value is usually zero in uncompressed BMP files.

Reserved1 and Reserved2 do not contain useful data and are usually set to zero in a BMP header written to disk. These fields are instead used by an application when the header is read into memory.

BitmapOffset is the starting offset of the bitmap data from the beginning of the file in bytes.

Following the file header in v2.*x* BMP files is a second header called the bitmap header. This header contains information specific to the bitmap data. This header is 12 bytes in length and has the following format:

```
typedef struct _Win2xBitmapHeader
{
    DWORD Size;            /* Size of this header in bytes */
    SHORT Width;           /* Image width in pixels */
    SHORT Height;          /* Image height in pixels */
    WORD  Planes;          /* Number of color planes */
    WORD  BitsPerPixel;    /* Number of bits per pixel */
} WIN2XBITMAPHEADER;
```

Size is the size of the header in bytes. For Windows 2.*x* BMP files, this value is always 12.

Width and Height are the width and height of the image in pixels, respectively. If Height is a positive number, then the image is a "bottom-up" bitmap with the origin in the lower-left corner. If Height is a negative number, then the image is a "top-down" bitmap with the origin in the upper-left corner. Width does not include any scan-line boundary padding.

Planes is the number of color planes used to represent the bitmap data. BMP files contain only one color plane, so this value is always 1.

BitsPerPixel is the number of bits per pixel in each plane. This value will be in the range 1 to 24; the values 1, 4, 8, and 24 are the only values considered legal by the Windows 2.*x* API.

The Windows 2.*x* bitmap header is identical to the OS/2 1.*x* bitmap header except that the Width and Height fields are signed values in Windows BMP files.

Following the header is the color palette data. A color palette is always present in a BMP file if the bitmap data contains 1-, 4-, or 8-bit data. Twenty-four-bit bitmap data never uses a color palette (nor does it ever need to). Each element of the palette is three bytes in length and has the following structure:

```
typedef struct _Win2xPaletteElement
{
    BYTE Blue;       /* Blue component */
    BYTE Green;      /* Green component */
    BYTE Red;        /* Red component */
} WIN2XPALETTEELEMENT;
```

Blue, Green, and Red hold the color component values for a pixel; each is in the range 0 to 255.

The size of the color palette is calculated from the BitsPerPixel value. The color palette has 2, 16, 256, or 0 entries for a BitsPerPixel of 1, 4, 8, and 24, respectively. The number of color palette entries is calculated as follows:

```
NumberOfEntries = 1 << BitsPerPixel;
```

To detect the presence of a color palette in a BMP file (rather than just assuming that a color palette does exist), you can calculate the number of bytes between the bitmap header and the bitmap data and divide this number by the size of a single palette element. Assuming that your code is compiled using 1-byte structure element alignment, the calculation is:

```
NumberOfEntries = (BitmapOffset - sizeof(WINBMPFILEHEADER) -
         sizeof(WIN2XBITMAPHEADER)) / sizeof(WIN2XPALETTEELEMENT);
```

If NumberOfEntries is zero, there is no palette data; otherwise, you have the number of elements in the color palette.

BMP Version 3 (Microsoft Windows 3.x)

Version 3.*x* BMP files begin with the same 14-byte header as v2.*x* BMP files. The file header is also followed by a bitmap header, which is an expanded version of the v2.*x* bitmap header. It is 40 bytes in size and contains six additional fields:

```
typedef struct _Win3xBitmapHeader
{
    DWORD Size;            /* Size of this header in bytes */
    LONG  Width;           /* Image width in pixels */
    LONG  Height;          /* Image height in pixels */
    WORD  Planes;          /* Number of color planes */
    WORD  BitsPerPixel;    /* Number of bits per pixel */
    /* Fields added for Windows 3.x follow this line */
    DWORD Compression;     /* Compression methods used */
    DWORD SizeOfBitmap;    /* Size of bitmap in bytes */
    LONG  HorzResolution;  /* Horizontal resolution in pixels per meter */
    LONG  VertResolution;  /* Vertical resolution in pixels per meter */
    DWORD ColorsUsed;      /* Number of colors in the image */
    DWORD ColorsImportant; /* Minimum number of important colors */
} WIN3XBITMAPHEADER;
```

Size is the size of the header in bytes. For Windows 3.*x* BMP files, this value is always 40.

Width and Height are the width and height of the image in pixels, respectively. If Height is a positive number, then the image is a "bottom-up" bitmap with the origin in the lower-left corner. If Height is a negative number, then the image is a "top-down" bitmap with the origin in the upper-left corner. Width does not include any scan-line boundary padding.

Planes is the number of color planes used to represent the bitmap data. BMP files contain only one color plane, so this value is always 1.

BitsPerPixel is the number of bits in each pixel. This value is in the range 1 to 24; the values 1, 4, 8, and 24 are the only values considered legal by the Windows 3.*x* API.

Compression indicates the type of encoding method used to compress the bitmap data. 0 indicates that the data is uncompressed; 1 indicates that the 8-bit RLE algorithm was used; 2 indicates that the 4-bit RLE algorithm was used. (See the section called "Image Data and Compression" below for more information on BMP RLE encoding.)

SizeOfBitmap is the size of the stored bitmap in bytes. This value is typically zero when the bitmap data is uncompressed; in this case, the decoder computes the size from the image dimensions.

HorzResolution and VertResolution are the horizontal and vertical resolutions of the bitmap in pixels per meter. These values are used to help a BMP reader choose a proper resolution when printing or displaying a BMP file.

ColorsUsed is the number of colors present in the palette. If this value is zero, and the value of BitsPerPixel is less than 16, then the number of entries is equal to the maximum size possible for the colormap. BMP files with a BitsPerPixel value of 16 or greater will not have a color palette. This value is calculated by using the value of the BitsPerPixel field:

```
ColorsUsed = 1 << BitsPerPixel;
```

ColorsImportant is the number of significant colors in the palette, determined by their frequency of appearance in the bitmap data; the more frequent the occurrence of a color, the more important it is. This field is used to provide as accurate a display as possible when using graphics hardware supporting fewer colors than are defined in the image. For example, an 8-bit image with 142 colors might only have a dozen or so colors making up the bulk of the image. If these colors could be identified, a display adapter with only 16-color capability would be able to display the image more accurately using the 16 most frequently occurring colors in the image. The most important colors are always stored first in the palette; ColorsImportant is 0 if all of the colors in the palette are to be considered important.

The color palette that may follow the bitmap header is basically the same as the v2.*x* palette but adds an extra byte of padding to increase its size to four bytes. This allows palette entries to be read as 4-byte values, making these values more efficient to read in memory and easier to see in a hex dump or debugger.

```
typedef struct _Win3xPaletteElement
{
     BYTE Blue;       /* Blue component */
     BYTE Green;      /* Green component */
     BYTE Red;        /* Red component */
     BYTE Reserved;   /* Padding (always 0) */
} WIN3XPALETTEELEMENT;
```

Blue, Green, and Red hold the color component values for a pixel; each is in the range 0 to 255.

Reserved pads the structure to end on an even-byte boundary and is always zero.

BMP Version 3 (Microsoft Windows NT)

Windows NT uses a variation of the Windows 3.*x* BMP format to store 16- and 32-bit data in a BMP file. This variation adds three additional fields that follow the bitmap header in place of a color palette. The bitmap header is 40 bytes in length and has the following format:

```
typedef struct _WinNtBitmapHeader
{
     DWORD Size;             /* Size of this header in bytes */
     LONG  Width;            /* Image width in pixels */
     LONG  Height;           /* Image height in pixels */
     WORD  Planes;           /* Number of color planes */
     WORD  BitsPerPixel;     /* Number of bits per pixel */
     DWORD Compression;      /* Compression methods used */
     DWORD SizeOfBitmap;     /* Size of bitmap in bytes */
     LONG  HorzResolution;   /* Horizontal resolution in pixels per meter */
     LONG  VertResolution;   /* Vertical resolution in pixels per meter */
     DWORD ColorsUsed;       /* Number of colors in the image */
     DWORD ColorsImportant;  /* Minimum number of important colors */
} WINNTBITMAPHEADER;
```

All fields are the same as in the v3.*x* BMP format, except for the Compression field.

Compression indicates the type of encoding method used to compress the bitmap data. 0 indicates that the data is uncompressed; 1 indicates that the 8-bit RLE algorithm was used; 2 indicates that the 4-bit RLE algorithm was used; and 3 indicates that bitfields encoding was used. If the bitmap contains 16 or 32 bits per pixel, then only a Compression value of 3 is supported and the RedMask, GreenMask, and BlueMask fields will be present following the header in place of a color palette. If Compression is a value other than 3, then the file is identical to a Windows 3.*x* BMP file.

```
typedef _WinNtBitfieldsMasks
{
     DWORD RedMask;          /* Mask identifying bits of red component */
     DWORD GreenMask;        /* Mask identifying bits of green component */
     DWORD BlueMask;         /* Mask identifying bits of blue component */
} WINNTBITFIELDSMASKS;
```

RedMask, GreenMask, and BlueMask specify which bits in a pixel value correspond to a specific color in 16- and 32-bit bitmaps. The bits in these mask values must be contiguous and must not contain overlapping fields. The bits in the pixel are ordered from most significant to least significant bits. For 16-bit bitmaps, the RGB565 format is often used to specify five bits each of red and blue values, and six bits of green:

```
RedMask   = 0xF8000000;     /* 1111 1000 0000 0000 0000 0000 0000 0000 */
GreenMask = 0x07E00000;     /* 0000 0111 1110 0000 0000 0000 0000 0000 */
BlueMask  = 0x001F0000;     /* 0000 0000 0001 1111 0000 0000 0000 0000 */
```

For 32-bit bitmaps, the RGB101010 format can be used to specify 10 bits each of red, green, and blue:

```
RedMask   = 0xFFC00000;     /* 1111 1111 1100 0000 0000 0000 0000 0000 */
GreenMask = 0x003FF000;     /* 0000 0000 0011 1111 1111 0000 0000 0000 */
BlueMask  = 0x00000FFC;     /* 0000 0000 0000 0000 0000 1111 1111 1100 */
```

BMP Version 4 (Microsoft Windows 95)

Version 4.*x* BMP files begin with the same 14-byte header as v2.*x* and v3.*x* BMP files. The file header is also followed by a bitmap header, which is an expanded version of the v3.*x* bitmap header, incorporating the mask fields of the NT BMP format. This v4.*x* bitmap header is 108-bytes in size and contains 17 additional fields:

```
typedef struct _Win4xBitmapHeader
{
     DWORD Size;             /* Size of this header in bytes */
     LONG  Width;            /* Image width in pixels */
     LONG  Height;           /* Image height in pixels */
     WORD  Planes;           /* Number of color planes */
     WORD  BitsPerPixel;     /* Number of bits per pixel */
     DWORD Compression;      /* Compression methods used */
     DWORD SizeOfBitmap;     /* Size of bitmap in bytes */
     LONG  HorzResolution;   /* Horizontal resolution in pixels per meter */
     LONG  VertResolution;   /* Vertical resolution in pixels per meter */
     DWORD ColorsUsed;       /* Number of colors in the image */
     DWORD ColorsImportant;  /* Minimum number of important colors */
     /* Fields added for Windows 4.x follow this line */

     DWORD RedMask;        /* Mask identifying bits of red component */
     DWORD GreenMask;      /* Mask identifying bits of green component */
     DWORD BlueMask;       /* Mask identifying bits of blue component */
     DWORD AlphaMask;      /* Mask identifying bits of alpha component */
     DWORD CSType;         /* Color space type */
     LONG  RedX;           /* X coordinate of red endpoint */
     LONG  RedY;           /* Y coordinate of red endpoint */
     LONG  RedZ;           /* Z coordinate of red endpoint */
     LONG  GreenX;         /* X coordinate of green endpoint */
     LONG  GreenY;         /* Y coordinate of green endpoint */
     LONG  GreenZ;         /* Z coordinate of green endpoint */
```

```
    LONG  BlueX;          /* X coordinate of blue endpoint */
    LONG  BlueY;          /* Y coordinate of blue endpoint */
    LONG  BlueZ;          /* Z coordinate of blue endpoint */
    DWORD GammaRed;       /* Gamma red coordinate scale value */
    DWORD GammaGreen;     /* Gamma green coordinate scale value */
    DWORD GammaBlue;      /* Gamma blue coordinate scale value */
} WIN4XBITMAPHEADER;
```

Size is the size of the header in bytes. For Windows 4.*x* BMP files, this value is always 108.

Width and Height are the width and height of the image in pixels, respectively. If Height is a positive number, then the image is a "bottom-up" bitmap with the origin in the lower-left corner. If Height is a negative number, then the image is a "top-down" bitmap with the origin in the upper-left corner. Width does not include any scan-line boundary padding.

Planes is the number of color planes used to represent the bitmap data. BMP files contain only one color plane, so this value is always 1.

BitsPerPixel is the number of bits in each pixel. This value is in the range 1 to 24; the values 1, 4, 8, 16, 24, and 32 are the only values considered legal by the Windows 4.*x* API.

Compression indicates the type of encoding method used to compress the bitmap data. 0 indicates that the data is uncompressed; 1 indicates that the 8-bit RLE algorithm was used; 2 indicates that the 4-bit RLE algorithm was used; and 3 indicates that bitfields encoding was used. If the bitmap contains a 16- or 32-bit bitmap, then only a compression value of 3 is supported.

SizeOfBitmap is the size of the stored bitmap in bytes. This value is typically zero when the bitmap data is uncompressed (including bitfields-encoded bitmaps); in this case, the decoder computes the size from the image dimensions.

HorzResolution and VertResolution are the horizontal and vertical resolutions of the bitmap in pixels per meter. These values are used to help a BMP reader choose a proper resolution when printing or displaying a BMP file.

ColorsUsed is the number of colors present in the palette. If this value is zero and the BMP file contains a color palette, then the number of entries is equal to the maximum size possible for the color palette. If the bitmap has a pixel depth of 16 or greater, there is never a color palette, and this value will be zero.

ColorsImportant is the number of significant colors in the palette, determined by their frequency of appearance in the bitmap data; the more frequent the occurrence of a color, the more important it is. See the explanation of this field for the Windows 3.*x* bitmap header for more information.

RedMask, GreenMask, BlueMask, and AlphaMask specify which bits in a pixel value correspond to a specific color or alpha channel in 16- and 32-bit bitmaps. The bits in these mask values must be contiguous and must not contain overlapping fields. The bits in the pixel are ordered from most significant to least significant bits. For example, a 16-bit bitmap using the RGB555 format would specify five bits each of red, green, blue, and alpha as follows:

```
AlphaMask = 0xF8000000;     /* 1111 1000 0000 0000 0000 0000 0000 0000 */
RedMask   = 0x07C00000;     /* 0000 0111 1100 0000 0000 0000 0000 0000 */
GreenMask = 0x003E0000;     /* 0000 0000 0011 1110 0000 0000 0000 0000 */
BlueMask  = 0x0001F000;     /* 0000 0000 0000 0001 1111 0000 0000 0000 */
```

A 32-bit bitmap using the RGB888 format would specify eight bits each of red, green, and blue using the mask values as follows:

```
AlphaMask = 0xFF000000;     /* 1111 1111 0000 0000 0000 0000 0000 0000 */
RedMask   = 0x00FF0000;     /* 0000 0000 1111 1111 0000 0000 0000 0000 */
GreenMask = 0x0000FF00;     /* 0000 0000 0000 0000 1111 1111 0000 0000 */
BlueMask  = 0x000000FF;     /* 0000 0000 0000 0000 0000 0000 1111 1111 */
```

Note that Windows 95 only supports the RGB555 and RGB565 masks for 16-bit BMP bitmaps and RGB888 for 32-bit BMP bitmaps.

CSType is the color space type used by the bitmap data. Values for this field include 00h (calibrated RGB), 01h (device-dependent RGB), and 02h (device-dependent CMYK). Device-dependent RGB is the default color space. Calibrated RGB is defined by the 1931 CIE XYZ standard.

RedX, RedY, and RedZ specify the CIE X, Y, and Z coordinates, respectively, for the endpoint of the red component of a specified logical color space. These fields are used only when CSType is 00h (calibrated RGB).

GreenX, GreenY, and GreenZ specify the CIE X, Y, and Z coordinates, respectively, for the endpoint of the green component of a specified logical color space. These fields are used only when CSType is 00h (calibrated RGB).

BlueX, BlueY, and BlueZ specify the CIE X, Y, and Z coordinates, respectively, for the endpoint of the blue component of a specified logical color space. These fields are used only when CSType is 00h (calibrated RGB).

GammaRed, GammaGreen, and GammaBlue are the red, green, and blue gamma coordinate scale values, respectively, for this bitmap.

All of the additional fields added to the Windows 4.*x* bitmap header are used to support 16- and 32-bit bitmaps and color matching and color characterization of the bitmap data. Color processing may be performed on an image and the ICM (Image Color Matching) information stored in the BMP file. This data is used to provide color matching processing when the bitmap is printed or displayed.

Color Palette

As we have seen, a BMP color palette is an array of structures that specify the red, green, and blue intensity values of each color in a display device's color palette. Each pixel in the bitmap data stores a single value used as an index into the color palette. The color information stored in the element at that index specifies the color of that pixel.

One-, 4-, and 8-bit BMP files are expected to always contain a color palette. Sixteen-, 24-, and 32-bit BMP files never contain color palettes. Sixteen- and 32-bit BMP files contain bitfields mask values in place of the color palette.

You must be sure to check the Size field of the bitmap header to know if you are reading 3-byte or 4-byte color palette elements. A Size value of 12 indicates a Windows 2.*x* (or possibly an OS/2 1.*x*) BMP file with 3-byte elements. Larger numbers (such as 40 and 108) indicate later versions of BMP, which all use 4-byte color palette elements.

Windows BMP File Types

Each new version of BMP has added new information to the bitmap header. In some cases, the newer versions have changed the size of the color palette and added features to the format itself. Unfortunately, a field wasn't included in the header to easily indicate the version of the file's format or the type of operating system that created the BMP file. If we add Windows' four versions of BMP to OS/2's two versions--each with four possible variations--we find that as many as twelve different related file formats all have the file extension ".BMP".

It is clear that you cannot know the internal format of a BMP file based on the file extension alone. But, fortunately, you can use a short algorithm to determine the internal format of BMP files.

The FileType field of the file header is where we start. If these two byte values are 424Dh ("BM"), then you have a single-image BMP file that may have been created under Windows or OS/2. If FileType is the value 4142h ("BA"), then you have an OS/2 bitmap array file. Other OS/2 BMP variations have the file extensions .ICO and .PTR.

If your file type is "BM", then you must now read the Size field of the bitmap header to determine the version of the file. Size will be 12 for Windows 2.*x* BMP and OS/2 1.*x* BMP, 40 for Windows 3.*x* and Windows NT BMP, 12 to 64 for OS/2 2.*x* BMP, and 108 for Windows 4.*x* BMP. A Windows NT BMP file will always have a Compression value of 3; otherwise, it is read as a Windows 3.*x* BMP file.

Note that the only difference between Windows 2.*x* BMP and OS/2 1.*x* BMP is the data type of the Width and Height fields. For Windows 2.*x*, they are signed shorts and for OS/2 1.*x*, they are unsigned shorts. Windows 3.*x*, Windows NT, and OS/2 2.*x* BMP files only vary in the number of fields in the bitmap header and in the interpretation of the Compression field.

Image Data and Compression

Uncompressed data is a series of values representing either color palette indices or actual RGB color values. Pixels are packed into bytes and arranged as scan lines. Each scan line must end on a 4-byte boundary, so one, two, or three bytes of padding may follow each scan line.

Scan lines are stored from the bottom up if the value of the Height field in the bitmap header is a positive value; they are stored from the top down if the Height field value is negative. The bottom-up configuration is the most common, because scan lines stored from the top down cannot be compressed.

Monochrome bitmaps contain one bit per pixel, eight pixels per byte (with the most significant bit being the leftmost pixel), and have a 2-element color palette. If a BMP reader chooses to ignore the color palette, all "one" bits are set to the display's foreground color and all "zero" bits are set to the background color.

Four-bit pixels are packed two per byte with the most significant nibble being the leftmost pixel. Eight-bit pixels are stored one per byte. Both 4- and 8-bit pixel values are indices into color palettes 16 and 256 elements in size respectively.

Sixteen-bit pixels in the Windows NT format are two bytes in size and are stored in big-endian order. In other words, on little-endian machines these bytes must be read and flipped into little-endian order before they are used. The organization of the bit fields in the 16-bit pixels is defined by the values of the RedMask, GreenMask, and BlueMask fields in the header. The most common masks are RGB555 and RGB565. The Compression field must always be a value of 3 (bitfields encoding) when a file stores 16-bit data.

In the v4.*x* BMP format, 16- and 32-bit pixels are stored as little-endian 4-byte RGB values. Common masks for 32-bit data include RGB888 and RGB101010. These bit depths also require bitfields encoding and the mask fields in the bitmap header to define their pixel format. 24-bit bitmap data is always stored as 3-byte RGB values.

The Windows BMP format supports a simple run-length encoded (RLE) compression scheme for compressing 4-bit and 8-bit bitmap data. Since this is a byte-wise RLE scheme, 1-, 16-, 24-, and 32-bit bitmaps cannot be compressed using it, due to the typical lack of long runs of bytes with identical values in these types of data.

BMP uses a two-value RLE scheme. The first value contains a count of the number of pixels in the run, and the second value contains the value of the pixel. Runs of up to 255 identical pixel values may be encoded as only two bytes of data. Actually, it's a bit more complex than this. In addition to encoded runs, there are unencoded runs, delta markers, end-of-scan-line markers, and an end-of-RLE-data marker.

The 8-bit RLE algorithm (RLE8) stores repeating pixel values as encoded runs. The first byte of an encoded run will be in the range of 1 to 255. The second byte is the value of the 8-bit pixels in the run. For example, an encoded run of 05 18 would decode into five pixels each with the value 18, or 18 18 18 18 18.

When a scan line does not contain enough pixel runs to achieve a significant amount of compression, contiguous pixel values may be stored as literal or unencoded runs. An unencoded run may contain from 3 to 255 pixel values. The first byte of an unencoded run is always zero. This makes it possible to tell the difference between the start of an encoded and the start of an unencoded run. The second byte value is the number of unencoded pixel values that follow. If the number of pixels is odd, then a 00 padding value also follows. This padding value is not part of the original pixel data and should not be written to the decoded data stream. Here are some examples of encoded and unencoded data streams:

Encoded Bytes	Decoded Bytes
05 10	10 10 10 10 10
00 05 23 65 34 56 45 00	23 65 34 56 45
0A 0A	0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
00 04 46 57 68 79	46 57 68 79

Three marker values may also be found in the RLE data. Each of these markers also begins with a zero-byte value. The second byte value indicates the type of marker. These markers specify positional information relating to the decoded bitmap data and do not generate any data themselves.

The first marker is the end-of-scan-line marker and is identified by two byte values 00 and 00. This marker is an indication that the end of data for the current scan line has been reached. Encoded data occurring after this marker is decoded starting at the beginning of the next scan line. If an end-of-scan-line marker is not present in the encoded data, then the pixels will automatically wrap from the end of one scan line to the start of the next.

This marker is only used when you want to force the decoding of a scan line to end at a particular place. If the end-of-scan-line marker occurs in the middle of a scan line, all remaining pixels in the decoded bitmap for the line are ignored. This "short scan line" technique is used to omit unneeded portions of scan lines. Most often, it is found in icon and pointer BMP files.

The next marker is the end of RLE data marker. It is identified by the two byte values 00 and 01. This marker occurs only as the last two bytes of the RLE data. This marker is an indication that the reader should stop decoding data.

The last marker is the run offset marker, also called a *delta* or *vector* code. This marker is four bytes in size, with the first two bytes being the values 00 and 02, and the last two values specifying a pixel address using unsigned X and Y values as an offset from the current bitmap cursor position. The X value is the number of pixels across the scan line, and the Y value is the number of rows forward in the bitmap.

This run offset marker indicates the location in the bitmap where the next decoded run of pixels should be written. For example, a run offset marker value of 00 02 05 03 would indicate that the offset of the bitmap cursor should move five pixels down the scan line, three rows forward, and write out the next run. The cursor then continues writing decoded data from its new position moving forward.

Run offset markers are used when a bitmap may contain a large amount of "don't care" pixels. For example, if the BMP file holds a bitmap used as a mask (such as those used with icons and pointers), many of the pixels in the rectangular bitmap may not be used. Rather than store these unused pixels in the BMP file, only the significant pixels are stored, and the delta markers are used as "jumps" to skip over the parts of the bitmap not actually used in the mask.

The following are the BMP RLE markers:

```
00 00           End of scan line
00 01           End of bitmap data
00 02 XX YY     Run offset marker
```

Here is an example of decoding an 8-bit data stream. Each of the values is an 8-bit index value into the color palette and not an actual color value.

Encoded Bytes	**Decoding Description**	**Decoded Bytes**
04 16	Four bytes of value 16	16 16 16 16
08 45	Eight bytes of value 45	45 45 45 45 45 45 45 45
00 00	End of scan line	None
00 02 04 02	Move to offset four pixels forward and two rows up	None
03 E4	Three bytes of value E4	E4 E4 E4
00 03 12 A4 46 00	Three bytes of unencoded data	12 A4 46
00 00	End of scan line	None
00 01	End of RLE data	None

The 4-bit RLE algorithm (RLE4) stores repeating pixel values in a very similar manner to RLE8. All of the markers are the same. The only real difference is that two pixel values are packed per byte, and these pixel values alternate when decoded. For example, an RLE4-encoded data stream of 07 48 would decode to seven pixels, alternating in value as 04 08 04 08 04 08 04.

If this looks bizarre, it's because you rarely see alternating runs of pixel values in bitmaps eight bits or greater in depth. Four-bit (16-color) bitmaps, however, usually contains a lot of dithering. Most dithering algorithms will yield relatively large runs of alternating pixels. Runs of repeating sequences of three and four pixels are also fairly common output from many dithering algorithms. But the ability to efficiently encode these types of pixel runs is not currently supported in the BMP RLE scheme.

In case you are thinking that runs of identical pixel values cannot be encoded by RLE4, you are incorrect. For example, a run of twelve pixels all of the value 9 would be RLE4-encoded as 0C 99 and would decode to the run 09 09 09 09 09 09 09 09 09 09 09 09.

Padding is added to unencoded pixel runs that are an odd number of bytes, rather than pixels, in length. And an unused final nibble in odd-length runs is set to zero. For example, the six pixel values 1 3 5 7 9 0 would be stored as the unencoded run 00 06 13 57 90 00, while the five pixel values 1 3 5 7 9 would be stored as the unencoded run 00 05 13 57 90 00.

Following is an example of decoding a 4-bit data stream. Each of the values is a 4-bit index value into the color palette and not an actual color value.

Encoded Bytes	**Decoding Description**	**Decoded Bytes**
04 16	Four values alternating 1 and 6	1 6 1 6
08 44	Eight values alternating 4 and 4	4 4 4 4 4 4 4 4
00 00	End of scan line	None
00 02 04 02	Move to offset four pixels forward and two rows up	None
03 E4	Three values alternating E and 4	E 4 E
00 06 12 A4 46 00	Six values of unencoded data	1 2 A 4 4 6
00 00	End of scan line	None
00 01	End of RLE data	None

Here is a summary of Windows BMP data characteristics:

Pixel Depth	**Pixel Size**	**Compression**	**Color Palette**	**Color Masks**
1 bit	1 bit	0	Yes	No
4 bits	4 bits	0,2	Yes	No
8 bits	1 byte	0,1	Yes	No
16 bits	4 bytes	3	No	Yes
24 bits	3 bytes	0	No	No
32 bits	4 bytes	3	No	Yes