Simplified Windows BMP Bitmap File Format Specification

(Last Mod: *27 November 2010 21:37:52)*

Overview

Several popular formats exist for digital image files, usually referred to by the file extension used. The more common formats include BMP, JPG, GIF, and PNG. As is typically the case, there are pros and cons associated with each format. From the standpoint of gaining familiarity with the subject of image files, the Windows BMP file offers significant advantages because of its simplicity. It also has the advantage of being highly standardized and extremely widespread. Its principal disadvantage is that it does not support effective image compression except on specific types of images. However, there is a hidden benefit to this shortcoming: since other image file formats do offer significant data compression, very few images stored in BMP format are compressed and, therefore, a very simple BMP editor that doesn't support working with compressed BMP files will seldom encounter a file it can't work with.

The information contained here is rather minimal. A much more thorough treatment of image file formats can be found in such texts as *Compressed File Formats* by John Miano. An excellent and in-depth description of the Windows BMP file format can be found here.

The Windows BMP file format dates back nearly two decades and several incompatible versions existed at one time. However, since Microsoft has full control of the format definition, these older forms are virtually never encountered and the format has been very stable since the version released with Windows 3.0 (Version 3). Certainly enhancements have been added to the format definition since then, but they have occurred in an upwardly compatible fashion; furthermore, none of those enhancements have really gained much traction so, once again, a simple editor that can only work with the basic options will seldom encounter difficulties. At the time of this writing, there were Versions 4 and 5 of the specification that support various aspects of the graphics libraries found in later versions of Windows, but virtually all editors still limit themselves to the Version 3 format.

We can get a useful idea of what options should be supported by looking at which options are supported by Paint, the simple image editor that comes with Windows. Version 5.1 (which shipped with Windows XP) supports Monochrome, 16-color, 256-color, and 24-bit color. These are the 1-bit, 4-bit, 8-bit, and 24-bit options, respectively. We will therefore describe these in detail. In addition, we will include 16-bit and 32-bit options; even though these are almost never encountered in practice, they use the concept of color bitmasks which are useful to at least know about.

Even though the file format does support a very primitive type of compression, we will not go into that here. As mentioned previously, these options are almost never used since people usually use another format such as PNG for JPG if they want compression.

The basic file structure is binary (as opposed to a text file) and is broken into the following four sections:

- The File Header (14 bytes)

    - Confirms that the file is (at least probably) a BMP file.
    - Tells exactly how large the file is.
    - Tells where the actual image data is located within the file.
- The Image Header (40 bytes in the versions of interest)
    - Tells how large the image is (rows and columns).
    - Tells what format option is used (bits per pixel).
    - Tells which type of compression, if any, is used.
    - Provides other details, all of which are seldom used.
- The Color Table (length varies and is not always present)
    - Provides the color palette for bit depths of 8 or less.
    - Provides the (optional) bit masks for bit depths of 16 and 32.
    - Not used for 24-bit images.
- The Pixel Data
    - Pixel by pixel color information
    - Row-by-row, bottom to top.
    - Rows start on double word (4-byte) boundaries and are null padded if necessary.
    - Each row is column-by-column, left to right.
    - In 24-bit images, color order is Red, Green, Blue.
    - In images less than 8-bits, the higher order bits are the left-most pixels.

Each of these four sections is discussed in detail below.

Endianness - Since this is a Microsoft format specification originally targeted for Intel processors, most multibyte values are stored in Little Endian format. This means that the bytes within the value are stored starting with the least significant byte first and the most significant byte last. Since this is how multibyte values are represented internally on an Intel processor, you will not need to worry about this convention as long as you transfer multibyte values directly between the file and memory. The only exception to this rule is for 16-bit pixel data in the WinNT version, which is stored in Big Endian. Note that ASCII character strings are not multibyte values -- they are strings of single-byte characters -- and, hence, the very concept of Endianness does not apply to them.

The File Header

The File Header has exactly fourteen bytes separated into five fields.

Field Name	Size in Bytes	Description
bfType	2	The characters "BM"
bfSize	4	The size of the file in bytes
bfReserved1	2	Unused - must be zero
bfReserved2	2	Unused - must be zero
bfOffBits	4	Offset to start of Pixel Data

bfType

The first two bytes must be the ASCII codes for the characters 'B' and 'M'. The software should check these values to confirm that the file it is reading is most probably a Windows BMP file. There are actually other options for these two characters, but they all refer to formats developed for the OS/2 operating system, which is largely extinct.

bfSize

The second field is a four-byte integer that contains the size of the file in bytes. In theory, this means that a BMP file could be as large as 4GB (4,294,967,296 bytes), however it is not safe to assume that all editors will treat this as an unsigned value and therefore a more reasonable limit is 2GB (2,147,483,648 bytes). This can accommodate a 24-bit image that contains over 715 million pixels (e.g., a picture larger than 32,000 by 23,000 pixels). It is probably worth noting that the BMP file specification actually supports storing multiple images in a single file; however, this is almost never done and very few editors support this option.

bfReserved1

bfReserved2

The third and fourth fields are each two bytes long and are reserved for future extensions to the format definition. While certainly not impossible, it is unlikely at this point that either will ever be used. The present definition requires that both of these fields be zero. A well-written reader should behave reasonably if non-zero values are encountered, but what is "reasonable"? Should the reader ignore the values regardless of what they are, or should it refuse to proceed any further on the assumption that the file conforms to a later version of the format specification and that it has no basis for assuming that it can properly interpret the contents? Good arguments can be made for either approach. Assuming that the decision is made to go ahead and read the file while ignoring these values, what should happen when the file is written back to disk? Should these values be

forced to zero, in order to conform to the present specification, or should they be preserved assuming that they have meaning to the program that originally generated the file? Again, good arguments can be made either way.

bfOffBits

The final four byte field is an integer that gives the offset of the start of the Pixel Data section relative to the start of the file.

The Image Header
There are actually several options for the Image Header, particularly when Versions 4 and 5 of the specification are considered. Limiting the discussion to Version 3, there are still two distinct options: one that was developed for the OS/2 BMP format and one that is for the Windows BMP format. The OS/2 Image Header is exactly 12 bytes long while the Windows Image Header is at least 40 bytes long. Fortunately, the first four bytes of each format (including Versions 4 and 5 and, probably, all future versions) is the length of the Image Header in bytes and therefore a simple examination of this value tells you which format is being used. While the Windows Image Header allows headers longer than 40 bytes (and this was supported by Windows 95), few applications ever adopted it and, once again, an editor that restricts itself to only working with 40-byte Image Headers will likely never encounter a file it can't work with.

The following is the format for the basic 40-byte Windows Image Header.

Field Name	Size in Bytes	Description
biSize	4	Header Size - Must be at least 40
biWidth	4	Image width in pixels
biHeight	4	Image height in pixels
biPlanes	2	Must be 1
biBitCount	2	Bits per pixel - 1, 4, 8, 16, 24, or 32
biCompression	4	Compression type (0 = uncompressed)
biSizeImage	4	Image Size - may be zero for uncompressed images
biXPelsPerMeter	4	Preferred resolution in pixels per meter
biYPelsPerMeter	4	Preferred resolution in pixels per meter
biClrUsed	4	Number Color Map entries that are actually used
biClrImportant	4	Number of significant colors

biSize

The size, in bytes, of the Image Header. Nearly all BMP files use the basic 40-byte header from the Windows Version 3 specification.

biWidth

This is the width of the image, in pixels.

biHeight

This is the height of the image, in pixels. The pixel data is ordered from bottom to top. If this value is negative, then the data is ordered from top to bottom.

biPlanes

This is the number of image planes in the image. It is always equal to 1.

biBitCount

This is the number of bits used to represent the data for each pixel. Bit Counts of 8 or less are "indexed" with the data being an index into the color pallete that is located stored in the Color Table section. Bit Counts greater than 8 are considered to be "true color" data with the complete color information being contained in each pixel's data. The overwhelming number of true color BMP files contain 24-bit images in which each pixel is represented by three bytes of data with one byte each for red, green, and blue intensities; no Color Table is present for such an image. Bit counts of 16 and 32 are color-masked bit fields and although all of the color data is present for each pixel, a set of three bit-masks, one for each primary color, is needed in order to determine which bits correspond to which colors; these bitmasks are stored in the Color Table section (with a default set of color masks being used if no Color Table is present).

biCompression

This value indicates what type of compression, if any, is used. The possible values are:

biCompression	Meaning
0	Uncompressed
1	RLE-8 (Usable only with 8-bit images)
2	RLE-4 (Usable only with 4-bit images)
3	Bitfields (Used - and required - only with 16- and 32-bit images)

biSizeImage

This indicates the size of the actual pixel data, in bytes, stored in the file. It is required for compressed image

types. For uncompressed types (including bit-fielded images) it is generally set equal to zero since the size of the data can be calculated directly from the image size and width.

biXPelsPerMeter

This is the preferred horizontal resolution of the image, in pixels per meter, when it is printed or displayed. It is only recommendation to the display device and is generally equal to zero, indicating no preference.

biYPelsPerMeter

This is the preferred vertical resolution of the image, in pixels per meter, when it is printed or displayed. It is only recommendation to the display device and is generally equal to zero, indicating no preference.

biClrUsed

This value is zero except for indexed images using fewer than the maximum number of colors, in which case it refers to the number of colors contained in the Color Table. For the indexed bit depths (i.e., Bit Counts of 8 and smaller) the Color Table contains a list of colors to which the pixel data refer. The maximum number of colors that can be used is 2^N where N is the value of Bit Count. For instance, 8-bit images can support a maximum of 256 colors. But suppose that an image only uses 47 colors. Why store 256? More to the point, why have to make up 109 colors that aren't being used? Instead, simply set this parameter to 47 and store those 47 colors and no more.

In practice it is very uncommon to run across images that have anything other than zero for this parameter. The reason is that it is a simple matter to go ahead and set all of the unused colors to all zero (black) -- or some other color -- and store all 2^N of them in the Color Table. The space penalty is negligible since the file size is dominated by the pixel data for all but the smallest images.

biClrImportant

This is the number of colors that are considered important when rendering the image. For instance, an image might contain 47 different colors but perhaps a reasonable image can be generated using just 13 of them. If the image is displayed on hardware having very limited color capability, say a cheap LCD panel supporting only 16 colors, then the reader knows that it should be able to generate an acceptable image by just using the first 16 colors in the Color Table. Needless to say, the important colors need to be stored first in the Color Table for this to work. It is left up to the application reading the file to determine how to map the "unimportant" colors into the available ones.

If this parameter is equal to zero, then all colors in the Color Table are to be considered important.

The Color Table
If we are dealing with images having a bit depth of 8 or less, then the pixel data is actually an index into a color palette. For instance, in a 4-bit image there are a maximum of 16 colors in the palette. If the data for a particular pixel is, say, 9, then the color that is used for that pixel is given by the tenth entry in the table (because the numbering starts with 0 and not 1).

In all versions of BMP files starting with Version 3 (Win3x), the color entries occupy 4 bytes each so that they can be efficiently read and written as single 32-bit values. Taken as a single value, the four bytes are ordered as follows: [ZERO][RED][GREEN][BLUE]. Due to the Little Endian format, this means that the Blue value comes first followed by the green and then the red. A fourth, unused, byte comes next which is expected to be equal to 0.

If we are dealing with 16-bit or 32-bit images, then the Color Table contains a set of bit masks used to define which bits in the pixel data to associate with each color. Note that the original Version 3 BMP format did not support these image types. Instead, these were an extension of the format developed for WindowsNT. For both the 16-bit and the 32-bit variants, the color masks are 32-bits long with the green mask being first and the blue mask being last. In both cases the bit masks start with the most significant bits, meaning that for the 16-bit images the least significant two bytes are zero. The format requires that the bits in each mask be contiguous and that the masks be non-overlapping. The most common bit masks for 16-bit images are RGB555 and RGB565 while the most common bitmasks for 32-bit images are RGB888 and RGB101010.

If we are dealing with a 24-bit image, then there is no Color Table present.

The Pixel Data
The pixel data is organized in rows from bottom to top and, within each row, from left to right. Each row is called a "scan line". If the image height is given as a negative number, then the rows are ordered from top to bottom.

In the uncompressed formats (including the color masked formats) each scan line is null-padded so as to occupy an integer number of dwords (4-byte words). In other words, the number of bytes needed to store each scan line must be an even multiple of four and, if necessary, null bytes (bytes whose values are zero) are appended to the end of the pixel data for that row in order to make this so. For instance, in the 24-bit format each pixel requires three bytes of data. If the image is 13 pixels wide then each row requires 39 bytes of data and a single null byte would be appended to bring this total to 40.

1-bit (Monochrome Bitmap)

Each byte of data represents 8 pixels. The most significant bit maps to the left most pixel in the group of eight. Any unused bits are set to zero.

4-bit (16 Color Bitmap)

Each byte of data represents 2 pixels. The most significant nibble maps to the left most pixel in the group of two. Any unused nibble is set to zero.

8-bit (256 Color Bitmap)

Each byte represents 1 pixel.

16-bit (Up to 65,536 Colors, but commonly 32,768 Colors)

Each pixel is represented by two bytes. A common representation is RGB555 which allocates 5 bits to each color allowing for 32K colors while leaving one bit unused. Since the eye is most sensitive to green, another

common representation is RGB565 which allocates this unused bit to the green component

24-bit (16,777,216 Colors)

Each pixel is represented by three bytes. The first byte gives the intensity of the red component, the second byte gives the intensity of the green component, and the third byte gives the intensity of the blue component.

32-bit (Up to 4,294,967,296 Colors, but commonly 1,073,741,824 or 16,777,216 Colors)

Each pixel is represented by four bytes. Even though this means that over four billion colors could be represented, very few display devices are capable of such resolution. However, it can be desirable to store images with such high resolution so that significant processing can be performed on the data without significant degradation building up from accumulated round-off errors. Even so, it is generally sufficient to only maintain an additional two bits of information per pixel, so the RGB101010 representation allocates 10 bits to each color. Another common representation is the RGB888 which simply uses a 32-bit format to store 24-bit images in order to leverage the ability of the processor to work with data in 32-bit chunks more efficiently.