BMP file format

Windows Bitmap

Filename extension	`.bmp`
Internet media type	`image/bmp`,[1] `image/x-bmp`
Type code	`'BMP '` `'BMPf'` `'BMPp'`
Uniform Type Identifier (UTI)	com.microsoft.bmp
Type of format	Raster graphics
Open format?	OSP for WMF

The **BMP file format**, also known as **bitmap image file** or **device independent bitmap (DIB) file format** or simply a **bitmap**, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems.

The BMP file format is capable of storing 2D digital images of arbitrary width, height, and resolution, both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles. The Windows Metafile (WMF) specification covers the BMP file format. Among others `wingdi.h` defines BMP constants and structures.

Device-independent bitmaps and the BMP file format

Microsoft has defined a particular representation of color bitmaps of different color depths, as an aid to exchanging bitmaps between devices and applications with a variety of internal representations. They called these device-independent bitmaps or DIBs, and the file format for them is called DIB file format or BMP image file format.

According to Microsoft support:

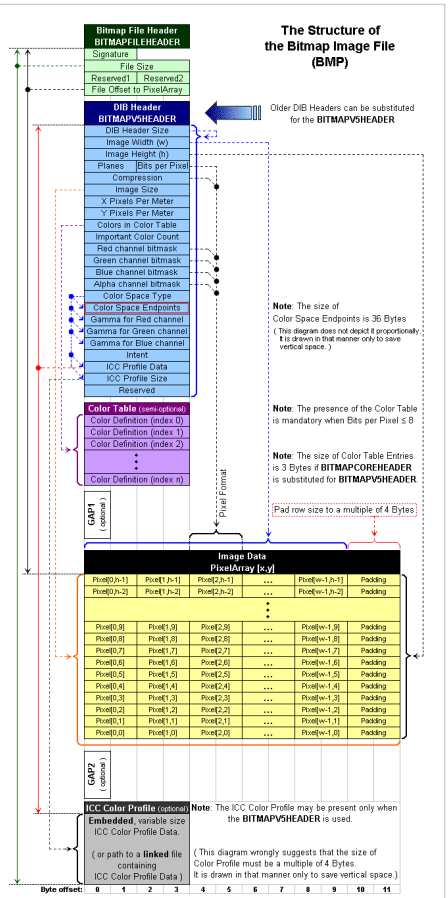
> A device-independent bitmap (DIB) is a format used to define device-independent bitmaps in various color resolutions. The main purpose of DIBs is to allow bitmaps to be moved from one device to another (hence, the device-independent part of the name). A DIB is an external format, in contrast to a device-dependent bitmap, which appears in the system as a bitmap object (created by an application...). A DIB is normally transported in metafiles (usually using the StretchDIBits() function), BMP files, and the Clipboard (CF_DIB data format).

The following sections discuss the data stored in the BMP file or DIB in detail. This is the standard BMP file format. Some applications create bitmap image files which are not compliant with the current Microsoft documentation. Also, not all fields are used; a value of 0 will be found in these unused fields.

File structure

The bitmap image file consists of fixed-size structures (headers) as well as variable-size structures appearing in a predetermined sequence. Many different versions of some of these structures can appear in the file, due to the long evolution of this file format.

Referring to the diagram 1, the bitmap file is composed of structures in the following order:



Diag. 1 – The structure of the bitmap image file

Structure Name	Optional	Size	Purpose	Comments
Bitmap File Header	No	14 Bytes	To store general information about the Bitmap Image File	Not needed after the file is loaded in memory
DIB Header	No	Fixed-size (however 7 different versions exist)	To store detailed information about the bitmap image and define the pixel format	Immediately follows the Bitmap File Header
Extra bit masks	Yes	3 or 4 DWORDs (12 or 16 Bytes)	To define the pixel format	Present only in case the DIB Header is the BITMAPINFOHEADER
Color Table	Semi-optional	Variable-size	To define colors used by the bitmap image data (Pixel Array)	Mandatory for color depths <= 8
Gap1	Yes	Variable-size	Structure alignment	An artifact of the File Offset to PixelArray in the Bitmap File Header
Pixel Array	No	Variable-size	To define the actual values of the pixels	The pixel format is defined by the DIB Header or Extra bit masks. Each row in the Pixel Array is padded to a multiple of 4 bytes in size
Gap2	Yes	Variable-size	Structure alignment	An artifact of the ICC Profile Data offset field in the DIB Header

ICC Color Profile	Yes	Variable-size	To define the color profile for color management	Can also contain a path to an external file containing the color profile. When loaded in memory as "non-packed DIB", it is located between the color table and gap1.[2]

DIBs in memory

A bitmap image file loaded into memory becomes a DIB data structure – an important component of the Windows GDI API. The in-memory DIB data structure is almost the same as the BMP file format, but it does not contain the 14-byte bitmap file header and begins with the DIB header. For DIBs loaded in memory, the color table can also consist of 16 bit entries, that constitute indexes to the currently realized palette[3] (an additional level of indirection), instead of explicit RGB color definitions. In all cases, the pixel array must begin at a memory address that is a multiple of 4 bytes. In non-packed DIBs loaded in memory, the optional color profile data should be located immediately after the color table and before the gap1 and pixel array (unlike in diag. 1).

When the size of gap1 and gap2 is zero, the in-memory DIB data structure is customarily referred to as "packed DIB" and can be referred to by a single pointer pointing to the beginning of the DIB header. In all cases, the pixel array must begin at a memory address that is a multiple of 4 bytes. In some cases it may be necessary to adjust the number of entries in the color table in order to force the memory address of the pixel array to a multiple of 4 bytes. For "packed DIBs" loaded in memory, the optional color profile data should immediately follow the pixel array, as depicted in diag. 1 (with gap1=0 and gap2=0).
"Packed DIBs" are required by Windows clipboard API functions as well as by some Windows patterned brush and resource functions.[4]

Bitmap file header

This block of bytes is at the start of the file and is used to identify the file. A typical application reads this block first to ensure that the file is actually a BMP file and that it is not damaged. The first two bytes of the BMP file format are the character 'B' then the character 'M' in 1-byte ASCII encoding. All of the integer values are stored in little-endian format (i.e. least-significant byte first).

Offset#	Size	Purpose
0	2 bytes	the header field used to identify the BMP & DIB file is 0x42 0x4D in hexadecimal, same as BM in ASCII. The following entries are possible: • **BM –** Windows 3.1x, 95, NT, ... etc. • **BA –** OS/2 struct Bitmap Array • **CI –** OS/2 struct Color Icon • **CP –** OS/2 const Color Pointer • **IC –** OS/2 struct Icon • **PT –** OS/2 Pointer
2	4 bytes	the size of the BMP file in bytes
6	2 bytes	reserved; actual value depends on the application that creates the image
8	2 bytes	reserved; actual value depends on the application that creates the image
10	4 bytes	the offset, i.e. starting address, of the byte where the bitmap image data (pixel array) can be found.

The size value occupies 4 bytes by default. However, with the use of the 4 reserved bytes, this value can occupy 8 bytes (64 bits) while still conforming to the BMPfile header format.

DIB header (bitmap information header)

This block of bytes tells the application detailed information about the image, which will be used to display the image on the screen. The block also matches the header used internally by Windows and OS/2 and has several different variants. All of them contain a dword (32 bit) field, specifying their size, so that an application can easily determine which header is used in the image. The reason that there are different headers is that Microsoft extended the DIB format several times. The new extended headers can be used with some GDI functions instead of the older ones, providing more functionality. Since the GDI supports a function for loading bitmap files, typical Windows applications use that functionality. One consequence of this is that for such applications, the BMP formats that they support match the formats supported by the Windows version being run. See the table below for more information.

Windows and OS/2 Bitmap headers

Size	Header Name	OS support	Features	Written by
12	BITMAPCOREHEADER OS21XBITMAPHEADER	Windows 2.0 or later OS/2 1.x		
64	OS22XBITMAPHEADER	OS/2 BITMAPCOREHEADER2	Adds halftoning. Adds RLE and Huffman 1D compression.	
40	BITMAPINFOHEADER	Windows NT & 3.1x or later	Adds 16bpp and 32bpp formats. Adds RLE compression.	
52	BITMAPV2INFOHEADER	Undocumented.	Adds RGB bit masks.	Adobe Photoshop
56	BITMAPV3INFOHEADER	Not officially documented, but I found this documentation on Adobe's forums, posted by an actual Adobe employee. [5]	Adds alpha channel bit mask.	Adobe Photoshop
108	BITMAPV4HEADER	Windows NT 4.0 & 95 or later	Adds color space type and gamma correction	
124	BITMAPV5HEADER	Windows NT 5.0 & 98 or later	Adds ICC color profiles	

Offset	Size	OS/2 1.x BITMAPCOREHEADER
14	4	the size of this header (12 bytes)
18	2	the bitmap width in pixels (unsigned 16bit)
20	2	the bitmap height in pixels (unsigned 16bit)
22	2	the number of color planes must be 1
24	2	the number of bits per pixel
OS/2 1.x bitmaps are uncompressed and cannot be 16 or 32bpp.		

Versions after BITMAPCOREHEADER only add fields to the end of the header of the previous version. For example: BITMAPV2INFOHEADER adds fields to BITMAPINFOHEADER and BITMAPV3INFOHEADER adds fields to BITMAPV2INFOHEADER.

An integrated alpha channel has been introduced with the undocumented BITMAPV3INFOHEADER and with the documented BITMAPV4HEADER (since Windows 95) and is used within Windows XP logon and theme system as well as Microsoft Office (since v2000); it is supported by some image editing software, such as Adobe Photoshop since version 7 and Adobe Flash since version MX 2004 (then known as Macromedia Flash). It is also supported by GIMP, Google Chrome, Microsoft PowerPoint and Microsoft Word.

For compatibility reasons, most applications use the older DIB headers for saving files. With OS/2 not more supported after Windows 2000, for now the common Windows format is the BITMAPINFOHEADER header. See

next table for its description. All values are stored as unsigned integers, unless explicitly noted.

Offset	Size	Windows BITMAPINFOHEADER
14	4	the size of this header (40 bytes)
18	4	the bitmap width in pixels (signed integer)
22	4	the bitmap height in pixels (signed integer)
26	2	the number of color planes must be 1
28	2	the number of bits per pixel, which is the color depth of the image. Typical values are 1, 4, 8, 16, 24 and 32.
30	4	the compression method being used. See the next table for a list of possible values
34	4	the image size. This is the size of the raw bitmap data; a dummy 0 can be given for BI_RGB bitmaps.
38	4	the horizontal resolution of the image. (pixel per meter, signed integer)
42	4	the vertical resolution of the image. (pixel per meter, signed integer)
46	4	the number of colors in the color palette, or 0 to default to 2^n
50	4	the number of important colors used, or 0 when every color is important; generally ignored

An OS/2 2.x OS22XBITMAPHEADER aka BITMAPCOREHEADER2 contains 24 additional bytes not yet explained here. The compression method (offset 30) can be:

Value	Identified by	Compression method	Comments
0	BI_RGB	none	Most common
1	BI_RLE8	RLE 8-bit/pixel	Can be used only with 8-bit/pixel bitmaps
2	BI_RLE4	RLE 4-bit/pixel	Can be used only with 4-bit/pixel bitmaps
3	BI_BITFIELDS	OS22XBITMAPHEADER: Huffman 1D	BITMAPV2INFOHEADER: RGB bit field masks, BITMAPV3INFOHEADER+: RGBA
4	BI_JPEG	OS22XBITMAPHEADER: RLE-24	BITMAPV4INFOHEADER+: JPEG image for printing
5	BI_PNG		BITMAPV4INFOHEADER+: PNG image for printing
6	BI_ALPHABITFIELDS	RGBA bit field masks	only Windows CE 5.0 with .NET 4.0 or later
11	BI_CMYK	none	only Windows Metafile CMYK
12	BI_CMYKRLE8	RLE-8	only Windows Metafile CMYK
13	BI_CMYKTLE4	RLE-4	only Windows Metafile CMYK

Color table

The color table (palette) occurs in the BMP image file directly after the BMP file header, the DIB header (and after optional three red, green and blue bitmasks if the BITMAPINFOHEADER header with BI_BITFIELDS option is used). Therefore, its offset is the size of the BITMAPFILEHEADER plus the size of the DIB header (plus optional 12 bytes for the three bit masks).

Note: On Windows CE the BITMAPINFOHEADER header can be used with the BI_ALPHABITFIELDS[] *option in the biCompression member.*

The number of entries in the palette is either 2^n or a smaller number specified in the header (in the OS/2 BITMAPCOREHEADER header format, only the full-size palette is supported). In most cases, each entry in the color table occupies 4 bytes, in the order blue, green, red, 0x00 (see below for exceptions). This is indexed in the BITMAPINFOHEADER under the function biBitCount.

The color table is a block of bytes (a table) listing the colors used by the image. Each pixel in an indexed color image is described by a number of bits (1, 4, or 8) which is an index of a single color described by this table. The purpose of the color palette in indexed color bitmaps is to inform the application about the actual color that each of these index values corresponds to. The purpose of the color table in non-indexed (non-palettized) bitmaps is to list the colors used by the bitmap for the purposes of optimization on devices with limited color display capability and to facilitate future conversion to different pixel formats and paletization.

The colors in the color table are usually specified in the 4-byte per entry RGBA32 format. The color table used with the OS/2 BITMAPCOREHEADER uses the 3-byte per entry RGB24 format. For DIBs loaded in memory, the color table can optionally consist of 2-byte entries - these entries constitute indexes to the currently realized palette instead of explicit RGB color definitions.

Microsoft does not disallow the presence of a valid alpha channel bit mask[6] in BITMAPV4HEADER and BITMAPV5HEADER for 1bpp, 4bpp and 8bpp indexed color images, which indicates that the color table entries can also specify an alpha component using the 8.8.8.[0-8].[0-8] format via the RGBQUAD.rgbReserved[7] member. However, some versions of Microsoft's documentation disallow this feature by stating that the RGBQUAD.rgbReserved member "must be zero".

As mentioned above, the color table is normally not used when the pixels are in the 16-bit per pixel (16bpp) format (and higher); there are normally no color table entries in those bitmap image files. However, the Microsoft documentation (on the MSDN web site as of Nov. 16, 2010[8]) specifies that for 16bpp (and higher), the color table can be present to store a list of colors intended for optimization on devices with limited color display capability, while it also specifies, that in such cases, no indexed palette entries are present in this Color Table. This may seem like a contradiction if no distinction is made between the mandatory palette entries and the optional color list.

Pixel storage

The bits representing the bitmap pixels are packed in rows. The size of each row is rounded up to a multiple of 4 bytes (a 32-bit DWORD) by padding.
For images with height > 1, multiple padded rows are stored consecutively, forming a Pixel Array.

The total number of bytes necessary to store one row of pixels can be calculated as:

$$\mathrm{RowSize} = \left\lfloor \frac{\mathrm{BitsPerPixel} \cdot \mathrm{ImageWidth} + 31}{32} \right\rfloor \cdot 4,$$

ImageWidth is expressed in pixels.

The total amount of bytes necessary to store an array of pixels in an n bits per pixel (bpp) image, with 2^n colors, can be calculated by accounting for the effect of rounding up the size of each row to a multiple of a 4 bytes, as follows:

$$\mathrm{PixelArraySize} = \mathrm{RowSize} \cdot |\mathrm{ImageHeight}|$$

ImageHeight is expressed in pixels. The absolute value is necessary because ImageHeight can be negative

Pixel array (bitmap data)

The pixel array is a block of 32-bit DWORDs, that describes the image pixel by pixel. Normally pixels are stored "upside-down" with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. Unless BITMAPCOREHEADER is used, uncompressed Windows bitmaps also can be stored from the top to bottom, when the Image Height value is negative.

In the original OS/2 DIB, the only four legal values of color depth were 1, 4, 8, and 24 bits per pixel (bpp). Contemporary DIB Headers allow pixel formats with 1, 2, 4, 8, 16, 24 and 32 bits per pixel (bpp).[9] GDI+ also permits 64 bits per pixel.

Padding bytes (not necessarily 0) must be appended to the end of the rows in order to bring up the length of the rows to a multiple of four bytes. When the pixel array is loaded into memory, each row must begin at a memory address that is a multiple of 4. This address/offset restriction is mandatory only for Pixel Arrays loaded in memory. For file storage purposes, only the size of each row must be a multiple of 4 bytes while the file offset can be arbitrary. A 24-bit bitmap with Width=1, would have 3 bytes of data per row (blue, green, red) and 1 byte of padding, while Width=2 would have 2 bytes of padding, Width=3 would have 3 bytes of padding, and Width=4 would not have any padding at all.

Compression

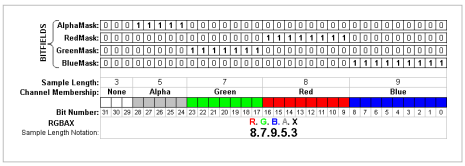
- Indexed color images may be compressed with 4-bit or 8-bit RLE or Huffman 1D algorithm.
- OS/2 BITMAPCOREHEADER2 24bpp images may be compressed with the 24-bit RLE algorithm.
- The 16bpp and 32bpp images are *always* stored uncompressed.
- Note that images in all color depths can be stored without compression if so desired.

Pixel format

In a bitmap image file on a disk or a bitmap image in memory, the pixels can be defined by a varying number of bits.

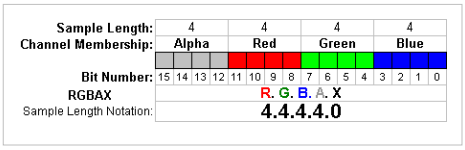
- The 1-bit per pixel (1bpp) format supports 2 distinct colors, (for example: black and white). The pixel values are stored in each bit, with the first (left-most) pixel in the most-significant bit of the first byte. Each bit is an index into a table of 2 colors. An unset bit will refer to the first color table entry, and a set bit will refer to the last (second) color table entry.
- The 2-bit per pixel (2bpp) format supports 4 distinct colors and stores 4 pixels per 1 byte, the left-most pixel being in the two most significant bits (Windows CE only:[10]). Each pixel value is a 2-bit index into a table of up to 4 colors.
- The 4-bit per pixel (4bpp) format supports 16 distinct colors and stores 2 pixels per 1 byte, the left-most pixel being in the more significant nibble. Each pixel value is a 4-bit index into a table of up to 16 colors.
- The 8-bit per pixel (8bpp) format supports 256 distinct colors and stores 1 pixel per 1 byte. Each byte is an index into a table of up to 256 colors.
- The 16-bit per pixel (16bpp) format supports 65536 distinct colors and stores 1 pixel per 2 byte WORD. Each WORD can define the alpha, red, green and blue samples of the pixel.
- The 24-bit pixel (24bpp) format supports 16,777,216 distinct colors and stores 1 pixel value per 3 bytes. Each pixel value defines the red, green and blue samples of the pixel (8.8.8.0.0 in RGBAX notation). Specifically in the order (blue, green and red, 8-bits per each sample).
- The 32-bit per pixel (32bpp) format supports 4,294,967,296 distinct colors and stores 1 pixel per 4 byte DWORD. Each DWORD can define the Alpha, Red, Green and Blue samples of the pixel.

In order to resolve the ambiguity of which bits define which samples, the DIB Headers provide certain defaults as well as specific BITFIELDS which are bit masks that define the membership of particular group of bits in a pixel to a particular channel. The following diagram defines this mechanism:

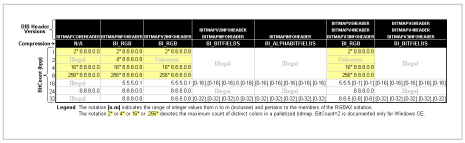


Diag. 2 – The BITFIELDS mechanism for a 32 bit pixel depicted in RGBAX sample length notation

The sample fields defined by the BITFIELDS bit masks have to be contiguous and non-overlapping but the order of the sample fields is arbitrary. The most ubiquitous field order is: Alpha, Blue, Green, Red (MSB to LSB). The red, green and blue bit masks are valid only when the Compression member of the DIB header is set to BI_BITFIELDS. The alpha bit mask is valid whenever it is present in the DIB header or when the Compression member of the DIB header is set to BI_ALPHABITFIELDS (Windows CE only).



Diag. 3 – The pixel format with an alpha channel in a 16 bit pixel (in RGBAX sample Length notation) actually generated by Adobe Photoshop[10]



RGB video subtypes

The BITFIELD mechanism described above allows for the definition of tens of thousands different pixel formats, however only several of them are used in practice, all palettized formats RGB8, RGB4, and RGB1 (marked in yellow in the table above, `dshow.h` MEDIASUBTYPE names) and:

Uncompressed RGB Video Subtypes

R.G.B.A.X	RGB subtype	R.G.B.A.X	ARGB subtype
8.8.8.0.8	RGB32	8.8.8.8.0	ARGB32
		10.10.10.2.0	A2R10G10B10
8.8.8.0.0	RGB24	10.10.10.2.0	A2B10G10R10
5.6.5.0.0	RGB565	4.4.4.4.0	ARGB4444
5.5.5.0.1	RGB555	5.5.5.1.0	ARGB1555

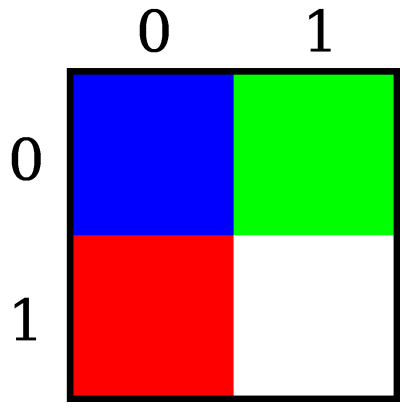
Bit fields for ten RGB bits

Bit field	Offset	Bits A2R10G10B10			Bits A2B10G10R10		
Red	36h	00 00 F0 3F	BE: 3FF00000	20...29	FF 03 00 00	BE: 000003FF	0... 9
Green	3Ah	00 FC 0F 00	BE: 000FFC00	10...19	00 FC 0F 00	BE: 000FFC00	10...19
Blue	3Eh	FF 03 00 00	BE: 000003FF	0... 9	00 00 F0 3F	BE: 3FF00000	20...29
Alpha	42h	00 00 00 C0	BE: C0000000	30...31	00 00 00 C0	BE: C0000000	30...31

In version 2.1.4 FFmpeg supported (in its own terminology) the BMP pixel formats *bgra, bgr24, rgb565le, rgb555le, rgb444le, rgb8, bgr8, rgb4_byte, bgr4_byte, gray, pal8,* and *monob*; i.e., *bgra* was the only supported pixel format with transparency.

Example 1

Following is an example of a 2×2 pixel, 24-bit bitmap (Windows DIB header BITMAPINFOHEADER) with pixel format RGB24.

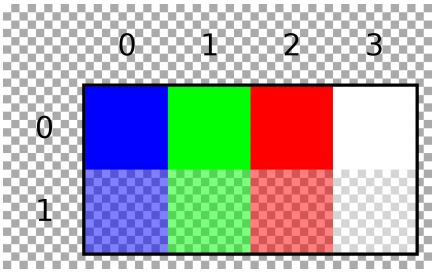


Example 1 of a 2×2 pixel bitmap, with 24 bits/pixel encoding

Offset	Size	Hex Value	Value	Description
BMP Header				
0h	2	42 4D	"BM"	ID field (42h, 4Dh)
2h	4	46 00 00 00	70 bytes (54+16)	Size of the BMP file
6h	2	00 00	Unused	Application specific
8h	2	00 00	Unused	Application specific
Ah	4	36 00 00 00	54 bytes (14+40)	Offset where the pixel array (bitmap data) can be found
DIB Header				
Eh	4	28 00 00 00	40 bytes	Number of bytes in the DIB header (from this point)
12h	4	02 00 00 00	2 pixels (left to right order)	Width of the bitmap in pixels
16h	4	02 00 00 00	2 pixels (bottom to top order)	Height of the bitmap in pixels. Positive for bottom to top pixel order.
1Ah	2	01 00	1 plane	Number of color planes being used
1Ch	2	18 00	24 bits	Number of bits per pixel
1Eh	4	00 00 00 00	0	BI_RGB, no pixel array compression used
22h	4	10 00 00 00	16 bytes	Size of the raw bitmap data (including padding)
26h	4	13 0B 00 00	2835 pixels/meter horizontal	Print resolution of the image, 72 DPI × 39.3701 inches per meter yields 2834.6472
2Ah	4	13 0B 00 00	2835 pixels/meter vertical	
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	0 means all colors are important
Start of pixel array (bitmap data)				
36h	3	00 00 FF	0 0 255	Red, Pixel (0,1)
39h	3	FF FF FF	255 255 255	White, Pixel (1,1)
3Ch	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)
3Eh	3	FF 00 00	255 0 0	Blue, Pixel (0,0)
41h	3	00 FF 00	0 255 0	Green, Pixel (1,0)
44h	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)

Example 2

Following is an example of a 4×2 pixel, 32-bit bitmap with opacity values in the alpha channel (Windows DIB Header BITMAPV4HEADER) with pixel format ARGB32.



Example 2 of a 4×2 pixel bitmap, with 32 bits/pixel encoding

Offset	Size	Hex Value	Value	Description
BMP Header				
0h	2	42 4D	"BM"	Magic number (unsigned integer 66, 77)
2h	4	9A 00 00 00	154 bytes (122+32)	Size of the BMP file
6h	2	00 00	Unused	Application specific
8h	2	00 00	Unused	Application specific
Ah	4	7A 00 00 00	122 bytes (14+108)	Offset where the pixel array (bitmap data) can be found
DIB Header				
Eh	4	6C 00 00 00	108 bytes	Number of bytes in the DIB header (from this point)
12h	4	04 00 00 00	4 pixels (left to right order)	Width of the bitmap in pixels
16h	4	02 00 00 00	2 pixels (bottom to top order)	Height of the bitmap in pixels
1Ah	2	01 00	1 plane	Number of color planes being used
1Ch	2	20 00	32 bits	Number of bits per pixel
1Eh	4	03 00 00 00	3	BI_BITFIELDS, no pixel array compression used
22h	4	20 00 00 00	32 bytes	Size of the raw bitmap data (including padding)
26h	4	13 0B 00 00	2835 pixels/meter horizontal	Print resolution of the image, 72 DPI × 39.3701 inches per meter yields 2834.6472
2Ah	4	13 0B 00 00	2835 pixels/meter vertical	
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	0 means all colors are important
36h	4	00 00 FF 00	00FF0000 in big-endian	Red channel bit mask (valid because BI_BITFIELDS is specified)
3Ah	4	00 FF 00 00	0000FF00 in big-endian	Green channel bit mask (valid because BI_BITFIELDS is specified)
3Eh	4	FF 00 00 00	000000FF in big-endian	Blue channel bit mask (valid because BI_BITFIELDS is specified)
42h	4	00 00 00 FF	FF000000 in big-endian	Alpha channel bit mask
46h	4	20 6E 69 57	little-endian "Win "	LCS_WINDOWS_COLOR_SPACE
4Ah	24h	24* 00...00	CIEXYZTRIPLE Color Space endpoints	Unused for LCS "Win " or "sRGB"
6Eh	4	00 00 00 00	0 Red Gamma	Unused for LCS "Win " or "sRGB"
72h	4	00 00 00 00	0 Green Gamma	Unused for LCS "Win " or "sRGB"
76h	4	00 00 00 00	0 Blue Gamma	Unused for LCS "Win " or "sRGB"
Start of the Pixel Array (the bitmap Data)				

7Ah	4	`FF 00 00 7F`	255 0 0 127	Blue (Alpha: 127), Pixel (0,1)
7Eh	4	`00 FF 00 7F`	0 255 0 127	Green (Alpha: 127), Pixel (1,1)
82h	4	`00 00 FF 7F`	0 0 255 127	Red (Alpha: 127), Pixel (2,1)
86h	4	`FF FF FF 7F`	255 255 255 127	White (Alpha: 127), Pixel (3,1)
8Ah	4	`FF 00 00 FF`	255 0 0 255	Blue (Alpha: 255), Pixel (0,0)
8Eh	4	`00 FF 00 FF`	0 255 0 255	Green (Alpha: 255), Pixel (1,0)
92h	4	`00 00 FF FF`	0 0 255 255	Red (Alpha: 255), Pixel (2,0)
96h	4	`FF FF FF FF`	255 255 255 255	White (Alpha: 255), Pixel (3,0)

Note that the bitmap data starts with the lower left hand corner of the image.

Usage of BMP format

The simplicity of the BMP file format, and its widespread familiarity in Windows and elsewhere, as well as the fact that this format is relatively well documented and free of patents, makes it a very common format that image processing programs from many operating systems can read and write. ICO and CUR files contain bitmaps starting with a BITMAPINFOHEADER.

Many older graphical user interfaces used bitmaps in their built-in graphics subsystems; for example, the Microsoft Windows and OS/2 platforms' GDI subsystem, where the specific format used is the *Windows and OS/2 bitmap file format*, usually named with the file extension of `.BMP`.

While most BMP files have a relatively large file size due to lack of any compression (or generally low-ratio run-length encoding on palletized images), many BMP files can be considerably compressed with lossless data compression algorithms such as ZIP because they contain redundant data. Some formats, such as RAR, even include routines specifically targeted at efficient compression of such data.

Related formats

Main article: Image file formats

The X Window System uses a similar XBM format for black-and-white images, and XPM (*pixelmap*) for color images. There are also a variety of "raw" formats, which saves raw data with no other information. The Portable Pixmap (PPM) and Truevision TGA formats also exist, but are less often used – or only for special purposes; for example, TGA can contain transparency information. Other bitmap file formats are in existence.

References

[1] .bmp MIME type not registered (http://www.iana.org/assignments/media-types/media-types.xhtml#image) at IANA
[2] MSDN Bitmap Header Types (http://msdn.microsoft.com/en-us/library/dd183386(VS.85).aspx)
[3] MSDN BITMAPINFO Structure (http://msdn.microsoft.com/en-us/library/dd183375(VS.85).aspx)
[4] Feng Yuan - Windows graphics programming: Win32 GDI and DirectDraw: Packed Device-Independent Bitmap (CreateDIBPatternBrush, CreateDIBPatternBrushPt, FindResource, LoadResource, LockResource) (http://books.google.com/books?id=-O92IIF1Bj4C&pg=PA595)
[5] https://forums.adobe.com/message/3272950#3272950
[6] MSDN - BITMAPV4HEADER: The member bV4AlphaMask (http://msdn.microsoft.com/en-us/library/dd183380(VS.85).aspx)
[7] MSDN - RGBQUAD: rgbReserved member (http://msdn.microsoft.com/en-us/library/dd162938(VS.85).aspx)
[8] see note under biClrUsed MSDN BITMAPINFOHEADER (http://msdn.microsoft.com/en-us/library/windows/desktop/dd183376(v=vs.85).aspx)
[9] MSDN - BITMAPINFOHEADER: The member biBitCount (http://msdn.microsoft.com/en-us/library/dd183376(VS.85).aspx)
[10] MSDN: Windows CE - BITMAPINFOHEADER Structure (http://msdn.microsoft.com/en-us/library/ms959648.aspx)

External links

- Bitmap File Structure (http://www.digicamsoft.com/bmp/bmp.html), at digicamsoft.com
- An introduction to DIBs (Device Independent Bitmaps) (http://www.herdsoft.com/ti/davincie/imex3j8i.htm), at herdsoft.com
- BMP test images (http://vaxa.wvnet.edu/vmswww/bmp.html), at wvnet.edu
- A simple bitmap loader C++ class (http://www.kalytta.com/bitmap.h), at kalytta.com (A2R10G10B10 not yet (http://en.wikipedia.org/w/index.php?title=BMP_file_format&action=edit) supported)
- The BMP File Format, Part 1 By David Charlap (http://drdobbs.com/architecture-and-design/184409517) at Dr. Dobb's journal of software tools (drdobbs.com), March 1995

Article Sources and Contributors

BMP file format *Source*: http://en.wikipedia.org/w/index.php?oldid=613782722 *Contributors*: (, 100DashSix, 1234r00t, 16@r, 28421u2232nfenfcenc, AJCham, Aanderson@amherst.edu, Acdx, Adys, Aeconley, Aeons, Akhristov, Alerante, Alfio, AlistairMcMillan, Andros 1337, Anonymous anonymous, Aragorn2, Arakunem, Ardonik, ArglebargleIV, Atlant, Baszoetekouw, Be..anyone, Beetstra, Benhut1, Berserkerz Crit, BigrTex, Blackjack II, Bobblehead, Burntsauce, CambridgeBayWeather, Chrisdolan, Ciphers, Cmarquezu, Cntras, Cquan, Craig t moore, Cwolfsheep, CyberSkull, DanielPharos, DataWraith, David D Allen, Dcouzin, Dee Earley, Dicklyon, DieBuche, Djmckee1, DmitriyV, DopefishJustin, Dreslough, Duckbill, Dylan anglada, ESkog, Eidako, Emilng, Evercat, FTPlus, Flipnitro, Frap, Fred Gandt, Freeboson, G0dsweed, GDallimore, GRAHAMUK, Gioto, GoingBatty, GovernmentMan, GreenReaper, Gurch, Guy Harris, Handige Harrie, Harkathmaker, Hatster301, Hiroe, Imnotminkus, Incnis Mrsi, Ippopotamus, Itamzbr, Jacobolus, Jengelh, JeromeJerome, Jill-Jênn, Jkl, Jleedev, JoeKearney, JohnnyMrNinja, Josh Parris, Jpgordon, Jschnur, Kanhef, Katimawan2005, Keenan Pepper, Kerfuffle090, Kevinrich47, Kierkkadon, Kimse, Kjlewis, Koavf, Konman72, Kotarou3, Kristian Vange, Kubanczyk, L Kensington, Larry V, LarsB, LeaveSleaves, Lee.Sailer, Lefter, LobStoR, Looris, Lordeaswar, LrdChaos, Magioladitis, Marc Mongenet, Marco255, Materialscientist, Math Champion, Mattcha, Maximaximax, Mdanh2002, Mellonj123, Meters, Michelin106, Minghong, MrOllie, Mrt3366, Mydatasaver, Naaman Brown, Nabla, Nestea Zen, Newsaholic, Nk, Nsaa, Nurg, Nuttycoconut, Olafdruemmer, Oli Filth, Oneirist, Ootachi, OverlordQ, Paranoid.android, Pavel Vozenilek, PeterJanRoes, Phillipsacp, Phlip, Pinethicket, Pkkao, Plugwash, Poccil, Quota, Qutezuce, RHaworth, Radiojon, Raijinili, Rbakker99, RenniePet, Rfl, Rich Farmbrough, RobertG, Rocastelo, Rodrigouf, Rsrikanth05, Rwxrwxrwx, Savirr, Serinde, Sikon, Silvonen, Skyfiler, SmartGuy Old, Softtest123, Soumyasch, Splintercellguy, Ssd, Stacie Croquet, Steevm, Stevenmc, Stewpeas, Sun Creator, Superjoe30, Svick, SwanQ, Tamariki, Technologeist, Teo64x, Teppy001, Terrorants, TheManOnTheWiki, Theinfo, Throwaway85, Thumperward, Tomchiukc, Trent Arms, Unyoyega, VBNetDude, VMS Mosaic, Vaxquis, Verdy p, Verpies, Vice8641, Viory, Wackelpudding, Warren, Wayne Slam, Weisebar, Wiki alf, WikiLaurent, Wikijpp, Xpclient, Yoderj, Yudiweb, Yuhong, Zero2ninE, Zundark, Zzarch, 405 anonymous edits

Image Sources, Licenses and Contributors

File:BMPfileFormat.png *Source*: http://en.wikipedia.org/w/index.php?title=File:BMPfileFormat.png *License*: Public Domain *Contributors*: Verpies

Image:BitfieldsSLN.png *Source*: http://en.wikipedia.org/w/index.php?title=File:BitfieldsSLN.png *License*: Public Domain *Contributors*: Verpies

Image:SLNotation44440.png *Source*: http://en.wikipedia.org/w/index.php?title=File:SLNotation44440.png *License*: Public Domain *Contributors*: Verpies

Image:AllBMPformats.png *Source*: http://en.wikipedia.org/w/index.php?title=File:AllBMPformats.png *License*: Public Domain *Contributors*: Verpies

Image:Bmp format.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Bmp_format.svg *License*: Public Domain *Contributors*: DoktorMandrake

File:Bmp_format2.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Bmp_format2.svg *License*: Public Domain *Contributors*: kotarou3

License