


<http://www.elet.polimi.it/upload/martucci/index.html>

- Lucidi del Corso di Informatica C Aerospaziali
- AA 2002-03
- 1° Modulo
- Es. [rappresentazione dell'informazione](#)

Fondamenti di Informatica C - AA 2002-03 - Microsoft Internet Explorer

File Modifica Visualizza Preferiti Strumenti ?

Indietro > >> Indirizzo h Vai

 **Politecnico di Milano**  
**Informatica C**

Facoltà di Ingegneria - Milano Bovisa  
Corso di Laurea Ingegneria Aerospaziale [E-O]  
Anno Accademico 2002-03 - secondo semestre

Proff. *R. Martucci* - L. Mazzei - M. Mussini

Ultimo aggiornamento domenica 23 febbraio 2003

<a href="#">Orario delle lezioni ed aule</a>	◆ sarà allocata una seconda aula di laboratorio e si provvederà alla divisione in 4 squadre (due per aula)
<a href="#">Programma del Corso</a>	◆ la articolazione nelle unità didattiche sarà contenuta nel piano delle lezioni definitivo
<a href="#">Piano delle Lezioni</a>	◆ Attenzione al calendario di lezioni, esercitazioni e laboratorio (giorni di calendario e N. di ore) - nel caso di ore perse per qualunque motivo, i recuperi verranno comunicati sul sito

Internet

# Standard Image Format

- **Definition:** A standard image format is one that is cross-platform compatible and supported by the majority of graphics applications.
- The most common standard bitmap-based formats are TIFF, JPG, GIF, and PNG. On the Windows platform BMP is a standard format and PICT is a standard bitmap format on the Macintosh. Photoshop's PSD format, though proprietary, is supported to some degree by most graphics applications, but be aware that transferring PSD between non-Adobe applications may give unexpected results.
- When you are sending files over the Internet or transporting them between graphics applications, it is important to use one of these standard formats, or you may not get the results you expect. When sending graphic images via email and the Web, it is best to use JPEG or GIF format, which can be displayed by any Web browser on any computer. When in doubt, ask the recipient of your files which image formats they can accept.



# The .bmp file format

## Introduction:

The .bmp file format (sometimes also saved as .dib) is the standard for a Windows 3.0 or later [DIB\(device independent bitmap\)](#) file. It may use compression (though I never came across a compressed .bmp-file) and is (by itself) not capable of storing animation. However, you can animate a bitmap using different methods but you have to write the code which performs the animation. There are different ways to compress a .bmp-file, but I won't explain them here because they are so rarely used. The image data itself can either contain pointers to entries in a color table or literal RGB values (this is explained later).

## Basic structure:

A .bmp file contains of the following data structures:

```

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

*bmfh* contains some information about the bitmap file (about the file, not about the bitmap itself). *bmih* contains information about the bitmap such as size, colors,... The *aColors* array contains a color table. The rest is the image data, which format is specified by the *bmih* structure.

## Exact structure:

The following tables give exact information about the data structures and also contain the settings for a bitmap with the following dimensions: size 100x100, 256 colors, no compression. The *start*-value is the position of the byte in the file at which the explained data element of the structure starts, the *size*-value contains the number of bytes used by this data element, the *name*-value is the name assigned to this data element by the Microsoft API documentation. *Stdvalue* stands for standard value. There actually is no such a thing as a standard value but this is the value Paint assigns to the data element if using the bitmap dimensions specified above (100x100x256). The *meaning*-column gives a short explanation of the purpose of this data element.

### The BITMAPFILEHEADER:

start	size	name	stdvalue	purpose
1	2	bfType	19778	must always be set to 'BM' to declare that this is a .bmp-file.
3	4	bfSize	??	specifies the size of the file in bytes.
7	2	bfReserved1	0	must always be set to zero.
9	2	bfReserved2	0	must always be set to zero.
11	4	bfOffBits	1078	specifies the offset from the beginning of the file to the bitmap data.

### The BITMAPINFOHEADER:

start	size	name	stdvalue	purpose
15	4	biSize	40	specifies the size of the BITMAPINFOHEADER structure, in bytes.
19	4	biWidth	100	specifies the width of the image, in pixels.
23	4	biHeight	100	specifies the height of the image, in pixels.
27	2	biPlanes	1	specifies the number of planes of the target device, must be set to zero.
29	2	biBitCount	8	specifies the number of bits per pixel.

31	4	biCompression	0	Specifies the type of compression, usually set to zero (no compression).
35	4	biSizeImage	0	specifies the size of the image data, in bytes. If there is no compression, it is valid to set this member to zero.
39	4	biXPelsPerMeter	0	specifies the the horizontal pixels per meter on the designated target device, usually set to zero.
43	4	biYPelsPerMeter	0	specifies the the vertical pixels per meter on the designated target device, usually set to zero.
47	4	biClrUsed	0	specifies the number of colors used in the bitmap, if set to zero the number of colors is calculated using the biBitCount member.
51	4	biClrImportant	0	specifies the number of color that are 'important' for the bitmap, if set to zero, all colors are important.

Note that *biBitCount* actually specifies the color resolution of the bitmap. The possible values are: 1 (black/white); 4 (16 colors); 8 (256 colors); 24 (16.7 million colors). The *biBitCount* data element also decides if there is a color table in the file and how it looks like. In 1-bit mode the color table has to contain 2 entries (usually white and black). If a bit in the image data is clear, it points to the first palette entry. If the bit is set, it points to the second. In 4-bit mode the color table must contain 16 colors. Every byte in the image data represents two pixels. The byte is split into the higher 4 bits and the lower 4 bits and each value of them points to a palette entry. There are also standard colors for 16 colors mode (16 out of Windows 20 [reserved colors](#) (without the entries 8, 9, 246, 247)). Note that you do not need to use this standard colors if the bitmap is to be displayed on a screen which support 256 colors or more, however (nearly) every 4-bit image uses this standard colors. In 8-bit mode every byte represents a pixel. The value points to an entry in the color table which contains 256 entries (for details see [Palettes in Windows](#). In 24-bit mode three bytes represent one pixel. The first byte represents the red part, the second the green and the third the blue part. There is no need for a palette because every pixel contains a literal RGB-value, so the palette is omitted.

### The RGBQUAD array:

The following table shows a single RGBQUAD structure:

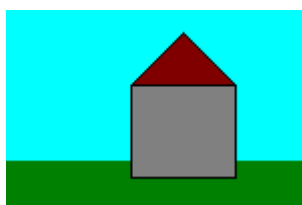
start	size	name	stdvalue	purpose
1	1	rgbBlue	-	specifies the blue part of the color.
2	1	rgbGreen	-	specifies the green part of the color.
3	1	rgbRed	-	specifies the red part of the color.
4	1	rgbReserved	-	must always be set to zero.

Note that the term *palette* does not refer to a RGBQUAD array, which is called *color table* instead. Also note that, in a color table (RGBQUAD), the specification for a color starts with the blue byte. In a palette a color always starts with the red byte. There is no simple way to map the whole color table into a LOGPALETTE structure, which you will need to display the bitmap. You will have to write a function that copies byte after byte.

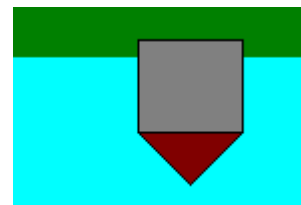
### The pixel data:

It depends on the BITMAPINFOHEADER structure how the pixel data is to be interpreted ([see above](#)).

It is important to know that the rows of a DIB are stored upside down. That means that the uppermost row which appears on the screen actually is the lowest row stored in the bitmap, a short example:



pixels displayed on the screen



pixels stored in .bmp-file

You do not need to turn around the rows manually. The API functions which also display the bitmap will do that for you automatically.

Another important thing is that the number of bytes in one row must always be adjusted to fit into the border of a multiple of four. You simply append zero bytes until the number of bytes in a row reaches a multiple of four, an example:

```
6 bytes that represent a row in the bitmap: A0 37 F2 8B 31 C4
                                         must be saved as: A0 37 F2 8B 31 C4 00 00
```

to reach the multiple of four which is the next higher after six (eight). If you keep these few rules in mind while working with .bmp files it should be easy for you, to master it.

[Back to the main page](#)

---

Copyright 1998 Stefan Hetzl. If you have questions or comments or have discovered an error, [send mail](#) to [hetzl@teleweb.at](mailto:hetzl@teleweb.at). You may forward this document or publish it on your webpage as long as you don't change it and leave this notice at the end.



---

[Previous Table of Contents Next](#)

## ***Part III***

### ***Uncompressed Files***

## **Chapter 6**

### **The Windows BMP Format**

#### **The BMP Format**

One of the nice things about the Windows operating system is that it supplies a standard bitmapped image file format, the BMP, as well as a standard vector graphics file format, the Windows MetaFile (WMF). Also, Windows has several API calls that directly manipulate and display these file formats. While the WMF format is not part of the scope of this book, the BMP format is the foundation of all further development. This chapter details the structure and usage of the BMP format.

As its name implies, the Windows bitmap format is a direct bitmap representation of an image. In essence, a BMP has three main sections: the header, the palette, if necessary, and the image bitmap itself. Each of these sections contains some of the information required to display the image. These sections are covered in order of appearance within a BMP file.

The header consists of two distinct parts, which may take one of two different forms. The form the header takes depends on which version of Windows created the BMP file. If the BMP file was created by a version of Windows earlier than Windows 3.0, or by OS/2, the two parts are formed from the BitmapFileHeader and the BitmapCoreHeader structures. If the BMP file was created under Windows 3.0 or newer, the two parts used are the BitmapFileHeader and the BitmapInfoHeader structures. These structures take the form:

#### **BitmapFileHeader**

---

Field	Size	Purpose
Type	Word	Hold characters "B" and "M"
Size	LongInt	Size of file in bytes
Reserved 1	Word	Reserved by Microsoft
Reserved 2	Word	Reserved by Microsoft
Offset	LongInt	Offset to start of bitmap in bytes

---

#### **BitmapCoreHeader**

---

Field	Size	Purpose
Size	LongInt	Size of this header structure
Width	Integer	Width of the BMP in pixels
Height	Integer	Height of the BMP in pixels
Planes	Word	Number of color planes--always 1
BitCount	Word	Bits per pixel

---

#### **BitmapInfoHeader**

---

Field	Size	Purpose
-------	------	---------

---

Size	LongInt	Size of this header structure
Width	LongInt	Width of BMP in pixels
Height	LongInt	Height of BMP in pixels
Planes	Word	Number of color planes--always 1
BitCount	Word	Bits per pixel
Compression	LongInt	Compression flag
SizeImage	LongInt	Image size in bytes
XPelsPerMeter	LongInt	Horizontal resolution
YPelsPerMeter	LongInt	Vertical resolution
ColorUsed	LongInt	Number of colors in color table
ColorImportant	LongInt	Number of important colors

If the BMP is an older version file, the color palette information follows the header in this format:

### RGBTriple

Field	Size	Purpose
Blue	Byte	Holds the blue intensity, 0..255
Green	Byte	Holds the green intensity, 0..255
Red	Byte	Holds the red intensity, 0..255

If the BMP uses the newer format, the color palette information following the header is in the following format:

### RGBQuad

Field	Size	Purpose
Blue	Byte	Holds the blue intensity, 0..255
Green	Byte	Holds the green intensity, 0..255
Red	Byte	Holds the red intensity, 0..255
Reserved	Byte	Holds flags relating to new format

If present in the BMP file, either of these structures repeats for the number of colors needed by the image's color palette. Thus, an old-style 16-color BMP file has sixteen repetitions of the RGBTriple structure after the BitmapCoreHeader section, each containing one of the sets of red, green, and blue values present in the 16-color palette. A Windows 3.x-compatible BMP file for a 256-color image has 256 RGBQuad structures immediately after the BitmapInfoHeader section. On the other hand, a 24-bit image does not need any color palette information. In such a situation, the image's bitmap data appears directly after the BitmapInfoHeader section. The final section in all BMP files is the bitmapped image itself. Depending upon the number of bits used per pixel, the image may be stored in one, two, or eight pixels per byte, with the possibility of three bytes per pixel, if the image is a 24-bit per pixel "true-color" image. These pixels are stored in scan line order, with the last image scan line being the first row in the BMP file. Each row holds as many bytes as necessary to hold the pixels in a scan line, padded to round the byte count to a number evenly divisible by four. Thus, an image based upon a 256-color palette with a size of 57 pixels by 64 rows would be stored as 60 pixels by 64 scan rows, with the extra three bytes per row initialized to a zero value, similar to the table shown below. When this image is loaded into a picture object, the extra padding bytes are stripped and do not show.

```

Pixels:  0  1  2  3  4  5  6  7  8.....54 55 56 57 58 59
Rows: 63 01 24 A4 39 7F BC 57 00 83.....2E 89 D0 00 00 00
      62 A2 B3 31 49 F7 CB 59 03 84.....F1 65 0F 00 00 00
      ...
      0 54 CA E2 74 82 05 B3 21 CF.....AA 47 92 00 00 00

```