# Windows Bitmap aus Wikipedia 9/2015

| Windows Bitmap | |
|---|---|
| Dateiendung: | `.bmp, .dib` |
| MIME-Type: | `image/x-ms-bmp, image/x-bmp, image/bmp` |
| Magische Zahl: | `42 4D` hex<br>`BM` |
| Entwickelt von: | Microsoft |
| Aktuelle Version: | 5 |
| Art: | Rastergrafik |

**Windows Bitmap** („**BMP**“) oder **device-independent bitmap** (DIB) ist ein zweidimensionales Rastergrafikformat, das für die Betriebssysteme Microsoft Windows und OS/2 entwickelt und mit Microsoft Windows 3.0 eingeführt wurde, welches 1990 erschien. Die Dateiendung ist *.bmp*, seltener *.dib*.

## Inhaltsverzeichnis

## Merkmale

BMPs gibt es in drei verschiedenen Versionen. Die meisten BMP-Dateien liegen in der Version 3 vor; es gibt keine früheren Versionen. Die späteren Versionen 4 und 5 sind höchst selten anzutreffen.

Windows-Bitmaps (der Version 3) erlauben Farbtiefen von 1, 4, 8, 16, 24 oder 32 bpp (bpp = bits per pixel, Bits je Bildpunkt), wobei bei 16 und 32 bpp nicht alle Bits tatsächlich genutzt werden müssen. Alphakanäle, Farbkorrektur und Metadaten werden nicht unterstützt. Windows-Bitmaps werden entweder unkomprimiert oder verlustfrei mit RLE-Komprimierung (Lauflängenkodierung) gespeichert. Dies ist ein eher schwaches Verfahren, sodass BMP-Dateien wesentlich größer sind als andere Formate wie PNG und kaum für das Internet genutzt werden. Dafür ist das BMP-Format relativ einfach aufgebaut. BMPs sind vor allem im Windows-Umfeld weit verbreitet; gängige Grafiksoftware unterstützt das Format problemlos (mit Ausnahme der eher exotischen Farbtiefen 16 und 32 bpp).

Das Bildformat ist auf 32767×32767 Pixel begrenzt.

## Dateiformat (Version 3)


BMP-Dateien bestehen aus drei Teilen: dem Dateikopf, dem Informationsblock und den Bilddaten (siehe Schema rechts).

Im Folgenden bezeichnet `WORD` einen 16-Bit-vorzeichenlosen Integer, `DWORD` einen 32-Bit-vorzeichenlosen Integer und `LONG` einen im Zweierkomplement kodierten 32-Bit-Integer. BMP verwendet die Little-Endian-Konvention.

## Dateikopf

**`BITMAPFILEHEADER` (Größe: 14 Byte)**

| Offset (Byte) | | Datentyp | | Größe | Name | Inhalt |
|---|---|---|---|---|---|---|
| Dez | Hex | Windows-Style | C-Style | | | |
| 0 | 0 | `WORD` | `uint16_t` | 2 Byte | bfType | ASCII-Zeichenkette `"BM"` (Hex: 0x42 0x4D Dez: 19778). |
| 2 | 2 | `DWORD` | `uint32_t` | 4 Byte | bfSize | Größe der BMP-Datei in Byte. (unzuverlässig) |
| 6 | 6 | `DWORD` | `uint32_t` | 4 Byte | bfReserved | Reserviert, von der Software abhängig, standardmäßig 0 |
| 10 | A | `DWORD` | `uint32_t` | 4 Byte | bfOffBits | Offset der Bilddaten in Byte vom Beginn der Datei an, bei Echtfarben fast immer 54 (manche Software ignoriert diese Angabe daher fehlerhafterweise). |

[1]

## Informationsblock

### Bitmap-Eigenschaften

Der Informationsblock beginnt mit folgender Struktur, die die Bitmap-Eigenschaften enthält[2][3]:

**`BITMAPINFOHEADER` (Größe: 40 Byte)**

| Offset (Byte) | | Datentyp | | Größe | Name | Inhalt |
|---|---|---|---|---|---|---|
| Dez | Hex | Windows-Style | C-Style | | | |
| 14 | E | `DWORD` | `uint32_t` | 4 Byte | biSize | 40 (Größe der BITMAPINFOHEADER-Struktur in Byte) |
| 18 | 12 | `LONG` | `int32_t` | 4 Byte | biWidth | Breite der Bitmap in Pixel. |
| 22 | 16 | `LONG` | `int32_t` | 4 Byte | biHeight | Der Betrag gibt die Höhe der Bitmap in Pixel an.<br>• Ist der Wert positiv, so ist die Bitmap eine sogenannte "bottom-up"-Bitmap (die Bilddaten beginnen mit der untersten und enden mit der obersten Bildzeile). Dies ist die gebräuchlichste Variante.<br>• Ist der Wert negativ, so ist die Bitmap eine "top-down"-Bitmap (die Bilddaten beginnen mit der obersten und enden mit der untersten Bildzeile). |
| 26 | 1A | `WORD` | `uint16_t` | 2 Byte | biPlanes | 1 (Stand in einigen älteren Formaten wie PCX für die Anzahl der Farbebenen, wird aber für BMP nicht verwendet) |
| 28 | 1C | `WORD` | `uint16_t` | 2 Byte | biBitCount | Gibt die Farbtiefe der Bitmap in bpp an; muss einer der folgenden Werte sein: 1, 4, 8, 16, 24 oder 32. Bei 1, 4 und 8 bpp sind die Farben indiziert. |
| 30 | 1E | `DWORD` | `uint32_t` | 4 Byte | biCompression | Einer der folgenden Werte:<br>• 0 (`BI_RGB`): Bilddaten sind unkomprimiert.<br>• 1 (`BI_RLE8`): Bilddaten sind lauflängenkodiert für 8 bpp. Nur erlaubt wenn biBitCount=8 und biHeight positiv.<br>• 2 (`BI_RLE4`): Bilddaten sind lauflängenkodiert für 4 bpp. Nur erlaubt wenn biBitCount=4 und biHeight positiv.<br>• 3 (`BI_BITFIELDS`): Bilddaten sind unkomprimiert und benutzerdefiniert (mittels Farbmasken) kodiert. Nur erlaubt wenn biBitCount=16 oder 32. |
| 34 | 22 | `DWORD` | `uint32_t` | 4 Byte | biSizeImage | • *Wenn biCompression=`BI_RGB`:* Entweder 0 oder die Größe der Bilddaten in Byte.<br>• *Ansonsten:* Größe der Bilddaten |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | in Byte. |
| 38 | 26 | `LONG` | `int32_t` | 4 Byte | biXPelsPerMeter | Horizontale Auflösung des Zielausgabegerätes in Pixel pro Meter; wird aber für BMP-Dateien meistens auf 0 gesetzt. |
| 42 | 2A | `LONG` | `int32_t` | 4 Byte | biYPelsPerMeter | Vertikale Auflösung des Zielausgabegerätes in Pixel pro Meter; wird aber für BMP-Dateien meistens auf 0 gesetzt. |
| 46 | 2E | `DWORD` | `uint32_t` | 4 Byte | biClrUsed | • *Wenn biBitCount=1:* 0.<br>• *Wenn biBitCount=4 oder 8:* die Anzahl der Einträge der Farbtabelle; 0 bedeutet die maximale Anzahl (2, 16 oder 256).<br>• *Ansonsten:* Die Anzahl der Einträge der Farbtabelle (0=keine Farbtabelle). Auch wenn sie in diesem Fall nicht notwendig ist, kann dennoch eine für die Farbquantisierung empfohlene Farbtabelle angegeben werden. |
| 50 | 32 | `DWORD` | `uint32_t` | 4 Byte | biClrImportant | • *Wenn biBitCount=1, 4 oder 8:* Die Anzahl sämtlicher im Bild verwendeten Farben; 0 bedeutet alle Farben der Farbtabelle.<br>• *Ansonsten:*<br>  • *Wenn eine Farbtabelle vorhanden ist und diese sämtliche im Bild verwendeten Farben enthält:* deren Anzahl.<br>  • *Ansonsten:* 0. |

### Farbmasken

Wenn biCompression=`BI_BITFIELDS`, dann folgen 3 `DWORD`s, die Bitmasken für die Rot-, Grün- und Blauwerte enthalten. Gesetzte Bits bedeuten, dass in den Daten eines Pixels jenes Bit für den jeweiligen Farbkanal verwendet wird. Dabei müssen folgende Bedingungen erfüllt sein:

- gesetzte Bits müssen direkt nacheinander folgen;
- für 16 bpp müssen sich die gesetzten Bits in den beiden niederwertigen Bytes befinden;
- die Bitmasken der einzelnen Farbkanäle dürfen einander nicht überlappen.

### Farbtabelle

- *Wenn biClrUsed=0:*
  - *Wenn biBitCount=1, 4 oder 8:* Es folgt eine Farbtabelle mit $2^{\text{biBitCount}}$ Einträgen.
  - *Ansonsten:* Es folgt keine Farbtabelle.
- *Ansonsten:* Es folgt eine Farbtabelle mit biClrUsed Einträgen.

Jeder Eintrag der Farbtabelle ist 4 Byte groß und enthält jeweils ein Byte für den Blau-, Grün- und Rotanteil, sowie ein auf 0 gesetztes Byte (in dieser Reihenfolge!).

## Bilddaten

Die Bilddaten beginnen am Offset bfOffBits. Die Größe der Bilddaten beträgt näherungsweise (gilt nur für durch 4 teilbare Bildbreiten) biWidth×biHeight×biBitCount/8 wenn biCompression=`BI_RGB`, ansonsten biSizeImage.

Die Bilddaten werden Zeile für Zeile gespeichert. Wenn biHeight positiv ist, beginnen die Bilddaten mit der letzten und enden mit der ersten Bildzeile, ansonsten ist es umgekehrt. Bei **`BI_BITFIELDS`** und bei **`BI_RGB`** ist die Länge jeder Zeile ein Vielfaches von 4 Bytes und wird, falls erforderlich, mit Nullbytes aufgefüllt.

Das weitere Format der Bilddaten hängt vom Wert des biCompression-Felds ab:

- **`BI_BITFIELDS`**

  Jede Bildzeile ist durch rechtsseitiges Auffüllen mit Nullen auf ein ganzzahliges Vielfaches von 4 Bytes ausgerichtet. Das Format der Pixel ist in den Farbmasken definiert. Bei 16 bpp werden nur die beiden niederwertigen Bytes der Farbmasken berücksichtigt.

- **`BI_RGB`**

  Jede Bildzeile ist durch rechtsseitiges Auffüllen mit Nullen auf ein ganzzahliges Vielfaches von 4 Bytes ausgerichtet.

  **1, 4 oder 8 bpp:**
  Die Daten jedes Pixels bestehen aus einem 0-basierten Index auf den Eintrag in der Farbtabelle.

  **16 bpp:**
  Das Format ist wie bei `BI_BITFIELDS`, wenn folgende Farbmasken verwendet würden:
  `0x00007C00` für den Rot-Kanal
  `0x000003E0` für den Grün-Kanal
  `0x0000001F` für den Blau-Kanal
  Jeder Farbkanal ist 5 Bit pro Pixel groß; insgesamt ergeben sich 32.768 mögliche Farben (ein Bit ist ungenutzt).

  **24 bpp:**
  Die Daten jedes Pixels bestehen aus jeweils einem Byte für den Blau-, Grün- und Rot-Kanal (in dieser Reihenfolge!).

  **32 bpp:**
  Das Format ist wie bei `BI_BITFIELDS`, wenn folgende Farbmasken verwendet würden:
  `0x00FF0000` für den Rot-Kanal
  `0x0000FF00` für den Grün-Kanal
  `0x000000FF` für den Blau-Kanal
  Jeder Farbkanal ist 8 Bit pro Pixel groß; insgesamt ergeben sich 16.777.216 mögliche Farben (8 Bit sind ungenutzt). Einige Programme wie etwa Adobe Photoshop interpretieren die verbleibenden 8 Bits (`0xFF000000`) als Alphakanal mit 256 möglichen Transparenzstufen. Dies ist jedoch von der Spezifikation nicht vorgesehen.

- **`BI_RLE8` und `BI_RLE4`**

  Jeweils zwei aufeinanderfolgende Bytes bilden einen Datensatz. Hat das erste Byte einen anderen Wert als 0, so wird das zweite Byte so oft (bei `BI_RLE4`: die nächsten 2 Nibbles insgesamt, Beispiel: 05 67 → 6 7 6 7 6) wiederholt, wie das erste Byte angibt. Hat das erste Byte hingegen den Wert 0, so hängt die Bedeutung vom zweiten Byte ab:

  | | |
  |---|---|
  | **0:** | Ende der Bildzeile. |
  | **1:** | Ende der Bitmap. |
  | **2:** | Verschiebung der aktuellen Pixelposition. Die beiden nächsten Bytes geben die Verschiebung nach rechts und nach unten an. |
  | ***n*=3-255:** | Die folgenden *n* Bytes (bei `BI_RLE4`: die folgenden *n* Nibbles) werden direkt übernommen; der nächste Datensatz findet sich am darauffolgenden geraden Offset (vom Start der Bilddaten aus gezählt). |

  Das Resultat wird wie im unkomprimierten Fall interpretiert.

## Vor- und Nachteile

Vorteile von Bitmaps sind unter anderem:[4]

- Die einfache Erstellung aus bereits im Arbeitsspeicher des Computers vorhandenen Pixeldaten.
- Der effiziente und simple Zugriff auf die Bilddaten auf Grund ihrer rasterartigen Anordnung.
- Eine Änderung der Farbinformation durch Änderung einer eventuell vorhandenen Palette ist möglich, ohne die Bilddaten selber zu ändern.
- Die einfache Ausgabe auf rasterbasierte Ausgabegeräte wie Monitore oder Drucker.

Ein Nachteil ist die große Dateigröße im Vergleich zu komprimierten Formaten.