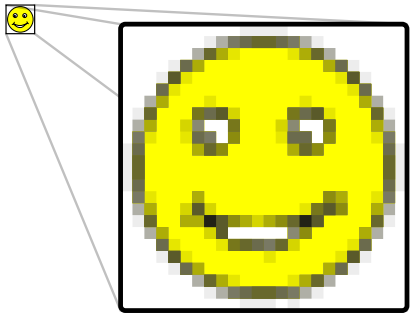
# *Binäre Dateiformate*

## *Bakera*

Alles wichtigen Links sind unter `go.bakera.de/bindat` verfügbar.

Ein Binärformat, das sich leicht analysieren und verstehen lässt, ist das Bitmap-Bildformat von Windows. Hierbei handelt es sich um ein Rasterformat, das Bilder Pixel für Pixel aufbaut. So wie der folgende Smiley.



Wie ist ein solches Bild eigentlich aufgebaut? Betrachten wir das Bildformat BMP im Detail und fangen mit dem Testbild auf der rechten Seite aus 3 x 2 Pixeln an. Zum Nachvollziehen kannst du es mit Paint nachmalen und als 24 Bit RGB-Bild abspeichern.

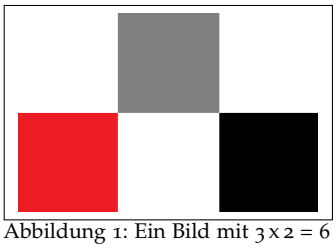


Abbildung 1: Ein Bild mit 3 x 2 = 6 Pixeln

**Auftrag 1 (Hexdump)** *Erstelle ein Programm, das eine BMP-Datei einliest und den Inhalt als Hexadezimalwert auf der Konsole ausgibt. Es sollte eine Ausgabe ähnlich der unten abgedruckten produzieren.*

Wir betrachten das Bild nun Byte für Byte. Schauen wir uns die markierten Stellen genauer an.

```
42 4d 4e 00 00 00 00 00   00 00 36 00 00 00 28 00
00 00 03 00 00 00 02 00   00 00 01 00 18 00 00 00
00 00 18 00 00 00 c4 0e   00 00 c4 0e 00 00 00 00
00 00 00 00 00 00 24 1c   ed ff ff ff 00 00 00 00
00 00 ff ff ff 7f 7f 7f   ff ff ff 00 00 00
```

| Hexwert | Bedeutung |
|---|---|
| 42 4d | „BM“ |
| 4e | Dateigröße: 78 Bytes |
| 36 | Start der Bilddaten (Offset) |
| 03 | Auflösung (Breite) |
| 02 | Auflösung (Höhe) |

Die ersten Bytes einer Binärdatei werden auch häufig als „magic number“ („magische Zahl“) bezeichnet, da man anhand dieser Zahlen häufig auf das Dateiformat schließen kann.

| Hex | ASCII | Dateityp |
|---|---|---|
| 424d | BM | BMP |
| cafebabe | – | Java Classfile |
| 4d5a | MZ | EXE |
| 5a4d | ZM | |

Abbildung 2: Interessante „magic numbers“

Die ersten beiden Bytes mit dem Inhalt $0x42$ und $0x4d$ stehen für die Buchstaben „B“ und „M“ als ASCII-Wert - also Bitmap. Durch das voran gestellte „$0x$“ wird kenntlich gemacht, dass der Hexwert 42 gemeint ist und nicht die dezimale Zahl 42. Es folgt ein Byte ($0x4e$), welches die Größe (hier dezimal 78) in Bytes angibt. Eigentlich beschreiben 4 Bytes die Dateigröße, nämlich die Stellen `4e 00 00 00`. Bei dem zehnten Byte, also an Position $0x0a$, steht eine $0x36$, die den Offset bis zum Beginn der Bilddaten angibt.

| Rot (drei Bytes) | Endianess |
|---|---|
| `00 00 ff` | little |
| `ff 00 00` | big |

*Big-Endian* beschreibt die gewöhnliche Reihenfolge, bei der die höherwertigen Stellen links stehen.

Bei *Little-Endian* werden zuerst die niederwertigen und anschließend die höherwertigen Bytes dargestellt – also „falsch herum“.

Ab dem Byte $0x36$ folgen die Bilddaten. Hierbei sind pro Pixel drei Byte abgelegt – für jede Farbe Rot, Grün und Blau jeweils ein Byte. Die Daten sind im *Little-Endian-Format* kodiert. Daher werden die Werte nicht als RGB-, sondern als BGR-Werte gespeichert.

## *Ein weiteres Beispiel*

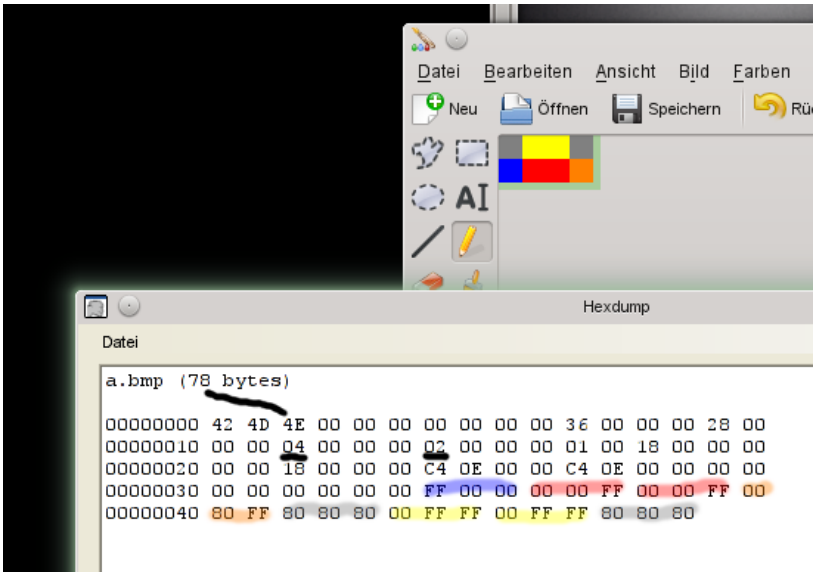



Abbildung 3: Analyse der Bildinformationen an einem weiteren Beispiel.

In Abb. 3 sind die Zusammenhänge noch einmal für ein anderes Bild zusammengefasst. Diesmal mit einer Auflösung von 4 x 2 Pixeln, da die Bildzeilen im Bitmap-Format immer mit Nullen aufgefüllt werden, bis sie ein Vielfaches von 4 ergeben. Dadurch sind bei dieser Auflösung keine „sinnlosen“ Informationen mehr im Bild enthalten.

In dem Hexdump sind Stellen, die Farbinformationen enthalten, in der entsprechenden Farbe hervorgehoben. Man erkennt, wie die Zeilen von unten nach oben und von links nach rechts aufgebaut werden.

Im dritten Byte finden wir wieder die Dateigröße – in diesem Fall den Wert $0x4E$, was wieder einer dezimalen 78 entspricht. Die Auflösung finden wir in den Bytes an den Positionen $0x12$ (der Wert $0x4$) und $0x16$ (der Wert $0x2$).

**Auftrag 2 (BMP Info)** *Erstelle ein Programm, welches eine BMP-Datei als Eingabe erhält und auf der Konsole die Metadaten der Datei und Farbinformationen für rot, grün und blau ausgibt. Eine mögliche Beispielausgabe ist rechts abgebildet.*

```
Dateigröße: 78 Bytes
Auflösung: 2 x 3
Pixel 0 RGB: 0 0 0
Pixel 1 RGB: 127 127 127
Pixel 2 RGB: 255 255 255
...
```

Abbildung 4: Informationen zu einer BMP-Datei.

**Auftrag 3 (RGB-Splitter)** *Erstelle ein Programm, das eine BMP-Datei als Eingabe erhält und daraus drei Dateien erzeugt: eine Datei für jede Farbinformation rot, grün und blau – vgl. Abb. 5*

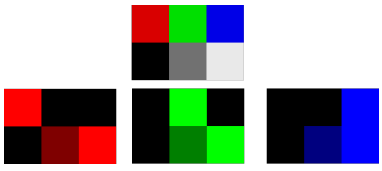


Abbildung 5: Oben das Original, unten die einzelnen Farbkanäle rot, grün und blau.

## *Übersicht*

Die folgende Abbildung fasst die unterschiedlichen Informationen zusammen. Zuerst wird das Dateiformat mit „BM“ gekennzeichnet. Es folgen die Dateigröße und ein Hinweis auf die Stelle, ab der die Bilddaten beginnen. Dann folgen Informationen zur Auflösung und Details über den Aufbau der Bilddaten. Schließlich werden die einzelnen Pixeldaten gelistet und bei Bedarf mit Nullen aufgefüllt.

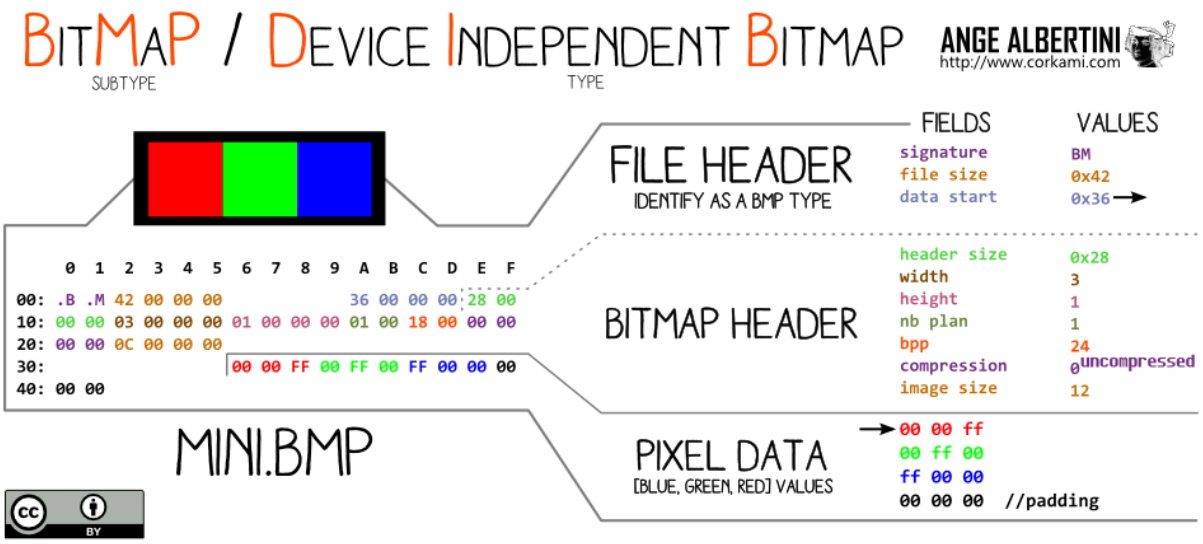



Abbildung 6: Übersicht über das BMP-Format.

## *Anwendung: Steganographie*

Es wäre nun ein Leichtes, die Bildinformationen einzelner Pixel zu ändern oder geheime Botschaften in der Bilddatei zu verstecken – wie in einem Steganogramm.[1] Hierzu wird z.B. bei jedem Farbwert das letzte Bit auf eine 0 oder 1 gesetzt, um eine Botschaft binär in den Bilddaten zu verstecken. Da der Farbwert nur um maximal eins verändert wird, fällt dies bei Betrachtung eines Bildes nicht auf.

[1] Wikipedia: „Computergestützte Steganographie“ (`go.bakera.de/bindat`)