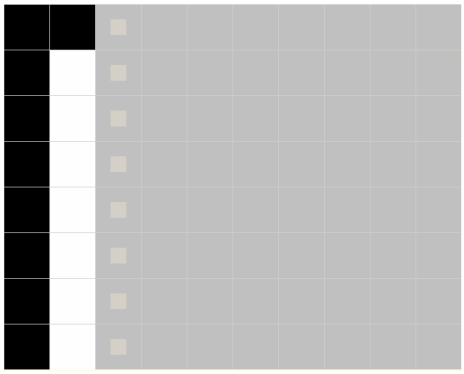
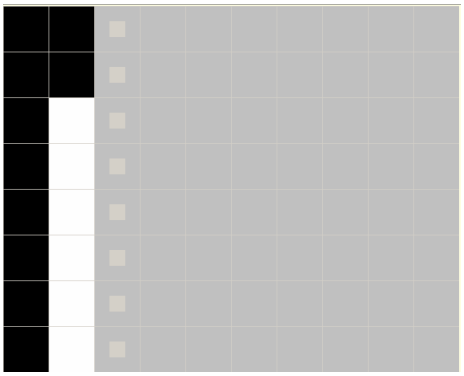
fontcreator test



```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x0A, 0x03, 0x08, 0x01, 0x00, 0x00,
   0x02, //buchstabenbreite
   0xFF, 0x01 //pixeldaten
};
```



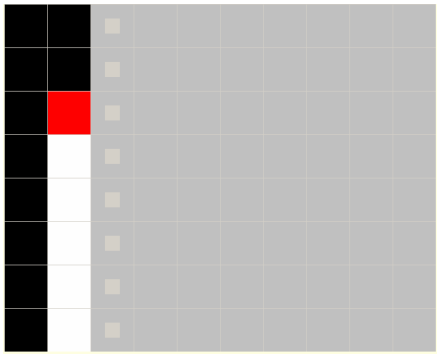
```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x0A, 0x03, 0x08, 0x01, 0x00, 0x00,
   0x02,
   0xFF, 0x03 //pixeldaten
};
```

Bit per pixel (msb =0->uncompressed)

```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x10, 0x03, 0x08, 0x04, 0x00, 0x00,
   0x02,
   0x11, 0x11, 0x11, 0x11, 0x11, 0x08, 0x00, 0x00
};
```

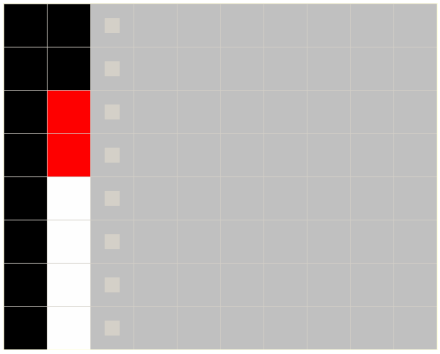
im 4 bit mode ist schwarz 0x1

rot ist 0x8

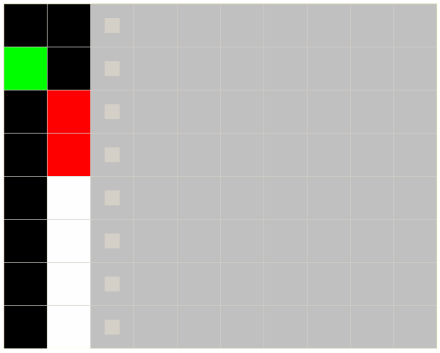


```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x10, 0x03, 0x08, 0x04, 0x00, 0x00,
   0x02,
   0x11, 0x11, 0x11, 0x11, 0x11, 0x08, 0x00, 0x00 //pixeldaten
};
```

Farben

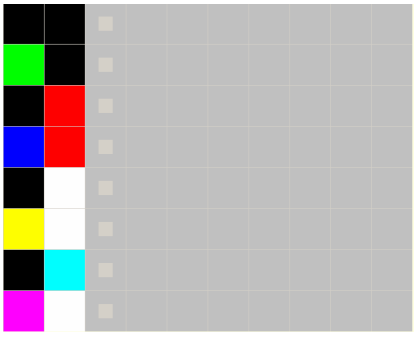


```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x10, 0x03, 0x08, 0x04, 0x00, 0x00,
   0x02,
   0x11, 0x11, 0x11, 0x11, 0x11, 0x88, 0x00, 0x00 //pixeldaten
};
```



4bit mode
schwarz 0x1
rot 0x8
grün 0x9

```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x10, 0x03, 0x08, 0x04, 0x00, 0x00,
   0x02,
   0x91, 0x11, 0x11, 0x11, 0x11, 0x88, 0x00, 0x00 //pixeldaten
};
```



4bit mode
schwarz 0x1
rot 0x8
grün 0x9
blau 0xA
gelb 0xB
magenta 0xC
türkis 0xD

```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x10, 0x03, 0x08, 0x04, 0x00, 0x00,
   0x02,
   0x91, 0xA1, 0xB1, 0xC1, 0x11, 0x88, 0x00, 0x0D //pixeldaten
};
```

Farbauftiefe

die Farbtiefe wird entsprechend der verwendeten Farben gewählt. wählt man beim zeichnen nur Farben aus der Palette unter den ersten 16, dann ist die Auflösung 4 bit (2hoch4). welche Farbe das Tool anzeigt ist in der font.ini Datei festgelegt. Man muss selbst für die passende Zuordnung/Interpretation in der eigenen Software sorgen, die die pixeldaten auswertet.

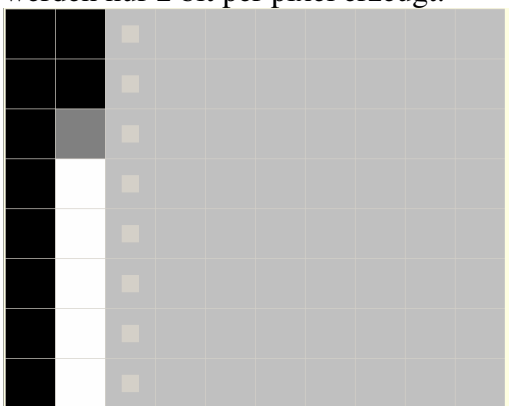
festlegung in: Font.ini datei:

[Colors]
0=$0080FFFF
1=$00000000
2=$00808080
3=$00FFFFFF
4=$0080FF80
5=$00FFFF80
6=$00800000
7=$00008080
8=$000000FF
9=$0000FF00
10=$00FF0000
11=$0000FFFF
12=$00FF00FF
13=$00FFFF00
14=$00808000
15=$00800080

Farbauswahl:

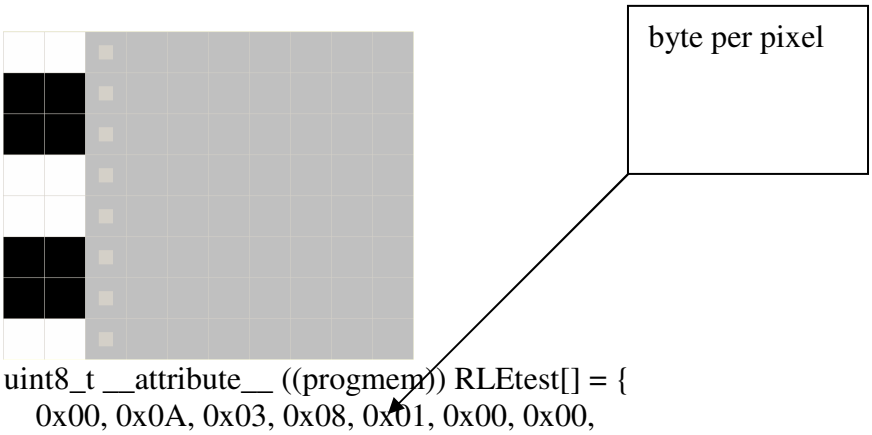
0 1 2 3 4 5 6 7 8 9 a b c d e f (im 4 bit modus)

nutzt man nur die farben weiß,schwarz und grau (können auch anders definiert werden). werden nur 2 bit per pixel erzeugt.

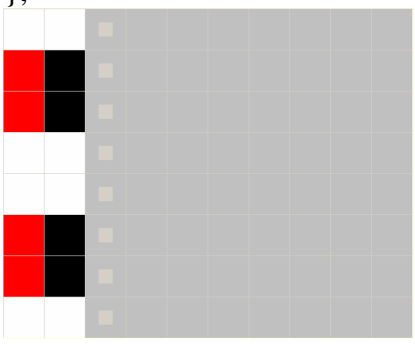


```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x0C, 0x03, 0x08, 0x02, 0x00, 0x00,
   0x02,
   0x55, 0x55, 0x25, 0xFF //pixeldaten
};
```

Orientierungs test




```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x0A, 0x03, 0x08, 0x01, 0x00, 0x00,
   0x02,
   0x66, 0x66 ——————————————————————————————→ 0x6=0110
};
```



```
uint8_t __attribute__ ((progmem)) RLEtest[] = {
   0x00, 0x10, 0x03, 0x08, 0x04, 0x00, 0x00,
   0x02,
   0x80, 0x08, 0x80, 0x08, 0x10, 0x01, 0x10, 0x01      4bit mode   0x6 wird zu 0x10, 0x01
};
```

Ein Buchstabe anhand „arial_bold_14.h“ (die datei beinhaltete noch keine bit per pixel byte)

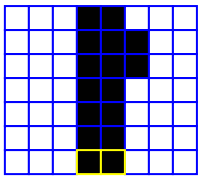
font infos:

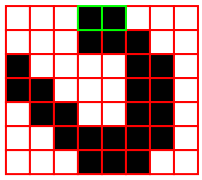
0x0A, // width d10
0x0E, // height d14

0x07 char width d7

Anhand der ‚**2**’
die Daten wurden einem File, erzeugt mit font.exe entnommen

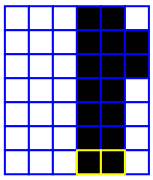
0x18, 0x1C, 0x86, 0xC6, 0x66, 0x3E, 0x1C, 0x18, 0x1C, 0x1C, 0x18, 0x18, 0x18, 0x18, // 50

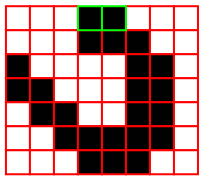


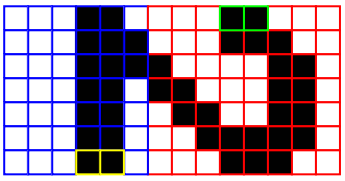


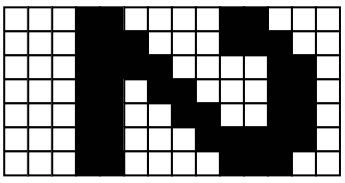
0x18,
0x1C,
0x1C,
0x18,
0x18,
0x18,
0x18

0x18,
0x1C,
0x86,
0xC6,
0x66,
0x3E,
0x1C









Struktur des fonts

```
struct _FONT_ {
    // common shared fields
      uint16_t   font_Size_in_Bytes_over_all_included_Size_it_self;
      uint8_t    font_Width_in_Pixel_for_fixed_drawing;
      uint8_t    font_Height_in_Pixel_for_all_Characters;
      uint8_t    font_Bits_per_Pixels;
                 // if MSB are set then font is a compressed font
      uint8_t    font_First_Char;
      uint8_t    font_Last_Char;
      uint8_t    font_Char_Widths[font_Last_Char - font_First_Char +1];
                 // for each character the separate width in pixels,
                 // characters < 128 have an implicit virtual right empty row
                 // characters with font_Char_Widths[] == 0 are undefined

    // if compressed font then additional fields
      uint8_t    font_Byte_Padding;
                 // each Char in the table are aligned in size to this value
      uint8_t    font_RLE_Table[3];
                 // Run Length Encoding Table for compression
      uint8_t    font_Char_Size_in_Bytes[font_Last_Char - font_First_Char +1];
                 // for each char the size in (bytes / font_Byte_Padding) are stored,
                 // this get us the table to seek to the right beginning of each char
                 // in the font_data[].

    // for compressed and uncompressed fonts
      uint8_t    font_data[];
                 // bit field of all characters
    }
```