

Les fichiers images BMP

Les Images au format *.bmp sont connu pour être les premières images supportées par Windows. BMP est la contraction de BitMap (carte de bit). C'est un format d'image sans destruction, c'est à dire que chaque pixel est indépendant des autres. C'est pour cela qu'une image bmp peut vite avoir une taille énorme.

Nous allons voir comment est composée une image bmp.

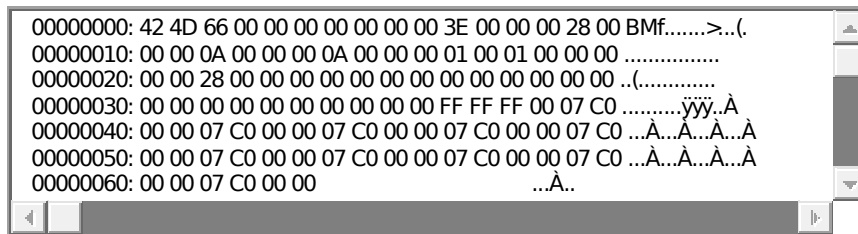
Pour cela, il nous faut :

- o Un éditeur hexadécimal (comme Free Hex editor)
- o Plusieurs images bmp (Vive Paint)

Pour commencer notre étude nous allons analyser cette image :

C'est une petite image monochrome, de taille 102 octet et de 10 fois 10 pixels.

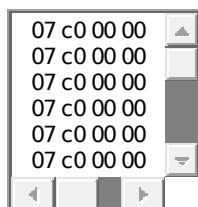
Ouvrons là avec un éditeur hexadécimal :



Que peut-on voir ?

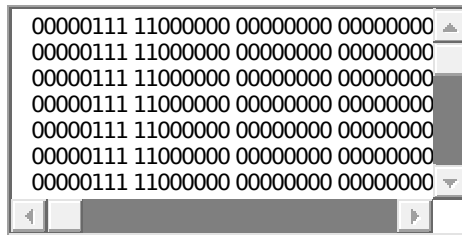
Les deux premiers octet "42 4D" sont la signature de ce type de fichier "BM" en Ascii. Il signifie que l'on a une image au format BMP. Le "66" est la taille de l'image (66 en hexadécimal donne 102 en décimal). Les deux "0A" sur la ligne 00000010 sont les dimension de l'image (0A en hexadecimal donne 10 en décimal). Le "01" qui suit est le nombre de plan (toujours à 1). le "01" suivant indique le nombre de bit par pixel, ici 1 signifie monochrome (2 couleurs). Le "28" de la ligne 00000020 est la taille de l'image en octet (28 en hexadécimal donne 40), on en déduit que la carte de l'image fait seulement 40 octets sur 102. De ce fait, la carte de l'image sera les 40 derniers octets. L'entête d'un fichier bmp est de 54 octet. Il nous reste donc $102 - (54 + 40) = 8$ octet. Ces 8 octets sont : 00 00 00 00 FF FF FF 00 . C'est la palette de couleur. On a 2 couleurs noir #000000 et blanc #FFFFFF.

La carte de l'image est :



On devrait avoir 100 pixels, (1 bit par pixel) mais on obtient 40 octets (320 pixels) qui sont décomposable en 10 séquences de 07 C0 00 00


[En binaire cela donne :](#)



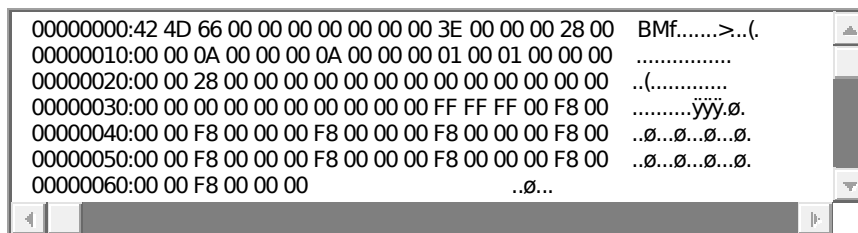
[Comment peut-on expliquer cela?](#)

Cela vient du fait que chaque ligne de l'image doit être un multiple de 4 octets. De ce fait on a 10 lignes fois 4 octets, mais seulement 10 bits servent à la carte de l'image (les 10 premiers de chaque ligne). Les autres bits sont tous à zéro.

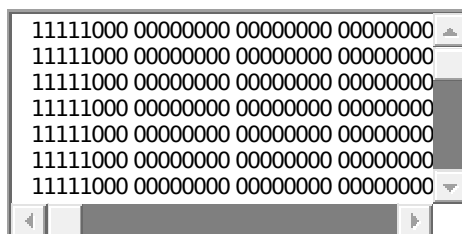
Notre carte (réel) est donc 10 lignes de 0000011111, 5 zéros et 5 uns. 5 zéros pour 5 pixels noir et 5 uns pour 5 pixels blanc.

Vérifions cela avec cette image :  qui est la même que précédemment mais avec une inversion de couleur.

[On obtient ce résultat avec l'éditeur hexadécimal :](#)



On a bien la même entête de 54 octets. Et la carte en binaire nous donne :



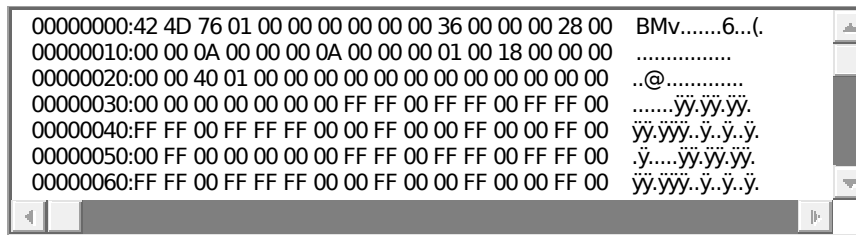
On obtient effectivement l'inversion des positions des uns. 5 uns pour 5 pixels blanc et 5 zéros pour 5 pixels noir.

Exercice :

Maintenant que l'on a vu comment était fait la carte de bit de l'image. Je vous propose de modifier l'image avec votre éditeur hexadécimal pour obtenir un damier (1 pixels noir en alternance avec 1 pixels blanc).

Maintenant passons à la couleur avec cette image :

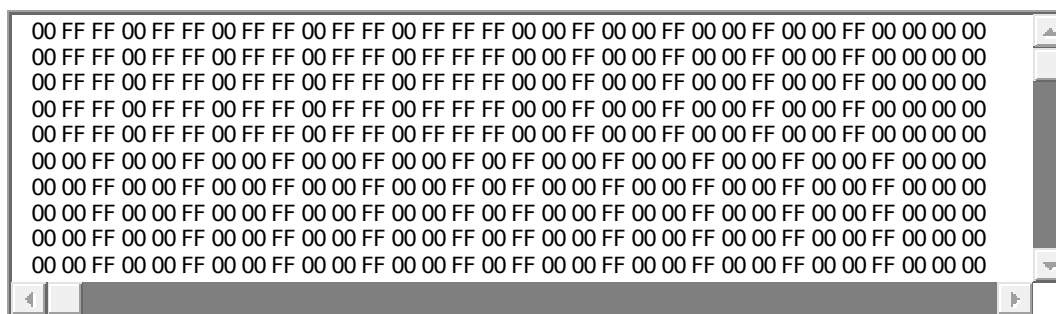
[Qu'obtient-on alors ? Voyons ce que cela donne avec l'éditeur hexadécimal :](#)



La première chose qui saute aux yeux c'est la différence de taille, on passe de 102 octets à 374. La taille de l'image en pixels est la même pourtant... Qu'est ce qui a changé ?

On a toujours notre signature "42 4D". La taille de l'image en octet est de "76 01" lire 176 (176 en hexadécimal donne 374) donc 374 octets. On retrouve nos 2 "0A" pour les dimensions de notre image (10 fois 10). On a aussi notre 01 pour le nombre de plan. La différence se trouve à ce point on a "18" au lieu de "01" ce qui veut dire que l'on a 24 bits par pixel (18 en hexadécimal donne 24 en décimal). La différence suivante est la taille en octet de la carte de bit de l'image qui passe de "28" à "40 01" (lire 140) donc 320 octets. Remarquez que l'on a $320 + 54 = 374$, ce qui signifie que cette fois-ci il n'y a pas de palette. En effet, du fait que chaque pixel est codé sur 24 bits (3 octets) cela suffit pour coder la couleur sur le pixel.

[Regardons à présent la carte de bit :](#)



Nous avons 4 groupes de séquence différents (4 couleurs dans l'image) de 3 octets : 00 FF FF pour jaune (lire "#FFFF00") FF 00 00 pour bleu (lire "#0000FF") 00 00 FF pour rouge (lire "#FF0000") et 00 FF 00 pour vert. On peut aussi constater l'ajout de 2 octets par ligne pour être multiple de 4 (3 fois 10 plus 2 égale 32, 32 est multiple de 4). Vous pouvez voir l'image est construite de bas en haut (chose que l'on a pas constatée en monochrome) en effet la première ligne nous donne du jaune et du bleu et la dernière ligne nous donne du rouge et du vert.

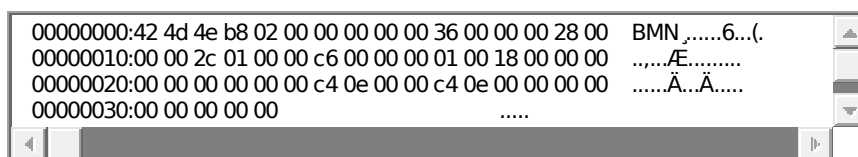
Comme vous l'avez constaté toutes les informations sont codées de droite à gauche et non l'inverse.

Exercice :

Modifier l'image avec l'éditeur hexadécimal pour obtenir une image de 14 fois 14 pixels constituée de 4 carrés identiques de couleur différentes.

Voyons maintenant ce que cela donne avec une image plus grande :

Résultat avec un éditeur hexadécimal :



Ici seul l'entête du fichier a été copier... L'image est trop grande et ce n'est pas la carte de bit qui est intéressante. On a notre signature. La taille du fichier est "4e b8 02" lire 2B84E (178254 octets). Les dimensions de l'image "2c 01" et "c6"(lire 12c et c6) on a donc une image de 300 fois 198 pixels (on peut en conclure que le premier est la dimension "colonne" et le deuxième "ligne"). On retrouve notre "01" pour notre seul plan et notre "18" pour 24 bits par pixel (donc 3 octets par pixel). Cette fois ci, on a pas d'information sur la taille en octet de la carte de bit... Mais on peut la calculer facilement (178254 - 54 (entête) = 178200 octet), on a deux nouvelles informations "c4 0e" théoriquement c'est les résolutions horizontales et verticales en pixel par mètre ici on doit avoir 3780 pixels par mètre.

Maintenant on va essayer de savoir combien on a d'octet par ligne de l'image. On sait que l'image comporte 300 pixels en "colonne" donc 300 pixels par ligne. Les couleurs sont codées sur 3 octet. Ce qui nous donne 900 octets par ligne. 900 étant un multiple de 4, on aura pas d'ajout d'octet en principe. Vérifions cela, 900 octet par ligne et on a 198 ligne, ce qui nous donne 178200 octet. C'est bien ce que l'on vient de dire, pas d'octet en plus par ligne.

Maintenant que notre étude se termine voici comment on peut décomposer un fichier image BMP :

54 octets pour l'entête de l'image
Palette de couleur si elle sont codé en moins de 24 bits
Carte de bit (ou carte des pixels)

Et voici comment est décomposé l'entête de l'image :

2 octets pour la signature "BM"
4 octets pour la taille de l'image (pas toujours fiable)
4 octets réservés
4 octets pour l'offset du début de l'image (en prenant en compte la taille de la palette)
4 octets pour la taille de la seconde entête (généralement 40 octets >> "28")
4 octets pour la largeur de l'image (colonne)
4 octets pour la longueur de l'image (ligne)
2 octets pour le nombre de plan (toujours à 1)

2 octets pour le nombre de bit par pixel
4 octets pour le type de compression (0=aucune, 1=RLE-8, 2=RLE-4)
4 octets pour la taille de la carte de bit en octet
4 octets pour résolution horizontale en pixel par mètre (pas toujours fiable)
4 octets pour résolution verticale en pixel par mètre (pas toujours fiable)
4 octets pour le nombre de couleurs dans l'image (ou 0 pour une image RVB)
4 octets pour le nombre de couleurs i