BitMap

COLLABORATORS			
	TITLE : BitMap		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 19, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

BitMap

1.1 BitMap V1.00

```
PureBasic - BitMap library V1.00

  Un 'BitMap' est une zone de mémoire utilisée pour stocker
  et/ou pour afficher des informations graphiques. Sur Amiga,
  le système natif d'affichage (OCS/ECS/AGA) utilise ce mode
  d'affichage dit par plans ('Planar') car chaque pixel est
  codé sur plusieurs plans (Si l'ecran a plus de 2 couleurs).
  Dans un 'BitMap', on peut trouver 1 ou plusieurs 'BitPlanes'.
  Chaque 'BitPlane' a une surface equivalente à la taille
  de l'écran (example: 320*200) et peut contenir pour chaque
  point, soit 0, soit 1 (un bit est un booléen, ne pouvant
  prendre que les valeurs 0 ou 1). Si on superpose plusieurs
  'BitPlanes' les uns sur les autres, on peut avoir plus de
  possibilités pour chaque point: Pour 4 'BitPlanes' on aura
  2^4 possibilité soit 16 valeurs differentes par points
  (dans notre cas, 16 couleurs differentes).

Commandes disponibles:

   AllocateBitMap
   AllocateLinearBitMap
   BitMapID
   BitMapRastPort
   FreeBitMap
   InitBitMap
   UseBitMap
   ShowBitMap

Exemple:

  Double buffering
```

1.2 allocatebitmap

```
  Syntaxe
```

```
BitMapID.l = AllocateBitMap(#BitMap, Width, Height, Depth)

  Résumé
Alloue une zone mémoire definie par la largeur, la hauteur
et la profondeur du BitMap. Cette fonction est compatible
avec les cartes graphiques (CyberGFX et Picasso 96).
Si la valeur retournée par cette function est NULLE, alors
le BitMap n'a pas pu être créé et par conséquent aucune
opération ne doit être effectué par la suite sur ce BitMap.

#BitMap: Identifiant numérique qui servira à reconnaitre ce
bitmap ultérieurement

Width : Largeur du BitMap en pixels

Height: Hauteur du BitMap en pixels

Depth : Profondeur du BitMap (nombre de BitPlanes). Cela definit
        le nombre de couleur que pourra supporter ce BitMap.
        Le calcul est le suivant: Nombre de couleurs = 2^Depth
        Donc un BitMap de Depth=5 pourra gérer 32 couleurs.
```

1.3 allocatelinearbitmap

```
  Syntaxe
BitMapID.l = AllocateLinearBitMap(#BitMap, Width, Height, Depth)

  Résumé
Créé un nouveau bitmap en accord avec les paramètres donnés. Ce
bimap est un peu spécial, car il est composé que d'un seul bloc
de mémoire. Il est alloué obligatoirement en mémoire CHIP et
ne devrait servir que pour les conversions Chunky/Planar. C'est
d'ailleur dans ce seul but que cette fonction a été créée. Ce bitmap
n'est pas compatible avec les cartes graphiques, donc si vous ne
faites pas de chunky/planar utilisez la fonction  AllocateBitMap() .

Si le résultat est NULL, il n'y a plus assez de mémoire pour allouer
le bitmap.


#BitMap: Identifiant numérique qui servira à reconnaitre ce
bitmap ultérieurement

Width : Largeur du BitMap en pixels

Height: Hauteur du BitMap en pixels

Depth : Profondeur du BitMap (nombre de BitPlanes). Cela definit
        le nombre de couleur que pourra supporter ce BitMap.
        Le calcul est le suivant: Nombre de couleurs = 2^Depth
        Donc un BitMap de Depth=5 pourra gérer 32 couleurs.
```

1.4 freebitmap

```
  Syntaxe
FreeBitMap(#BitMap)

  Résumé
Libère l'espace memoire occupé par le BitMap. Aucune référence sur ce
bitmap ne peut être faite ultérieurement.

#BitMap: Identifiant numérique representant le BitMap précédement
créé par la fonction  AllocateBitMap()
```

1.5 initbitmap

```
  Syntaxe
Resultat.l = InitBitMap(#NumBitMapMax)

  Résumé
Initialise l'environnement nécessaire pour la gestion du nombre
de BitMap passé en paramètre. Si la valeur retournée est NULLE,
alors l'environnement ne peut pas être créé et vous ne devez
en aucun cas utiliser une des functions de la bibliothèque 'BitMap'.

#NumBitMapMax: Nombre maximum de BitMap à gérer.
```

1.6 bitmapid

```
  Syntaxe
BitMapID.l = BitMapID()

  Résumé
Retourne l'addresse mémoire du BitMap courant, qui sera parfois
nécessaire pour les autre fonctions du PureBasic.


Note pour les programmeurs expérimentés: ce pointeur permet d'accéder
aux informations sur le BitMap grâce à la structure 'BitMap' qui est
definie dans les RKM du Rom3.0+.
Exemple: *MyBitMap.BitMap = BitMapID()
```

1.7 bitmaprastport

```
  Syntaxe
RastPort.l = BitMapRastPort()

  Résumé
Retourne l'adresse du 'RastPort' du BitMap courant. Cette adresse est
nécessaire pour certaines autre fonctions du PureBasic comme par exemple
les fonctions de la bibliothèque '2D Drawing'. Ne vous inquiétez pas si
```

vous ne savez pas ce qu'est réellement un RastPort, ca n'a que peu
d'importance. Il faut juste retenir que c'est un identifiant permettant
de savoir où les graphiques vont être dessinés. Utilisez cette fonctions
dès que vous avez besoin d'un 'RastPort'.

1.8 usebitmap

Syntaxe
UseBitMap(#BitMap)

Résumé
Change le BitMap courant par le nouveau BitMap passé en paramètre.

#BitMap: Identifiant numérique représentant le nouveau BitMap à
définir comme courant.

1.9 showbitmap

Syntaxe
ShowBitMap(#BitMap, ScreenID, x, y)

Résumé
Affiche le contenu du BitMap (#BitMap) sur l'écran (ScreenID) à la
position x, y. Cette fonction est respecte le système Amiga et
permet de faire un 'double buffering' très rapide. Un 'Vertical
Wait' (attente que le spot de balayage ait atteint le haut de l'écran)
est automatiquement executé pour un maximum de synchronization.

#BitMap: Identifiant du BitMap à afficher.

ScreenID: Identifiant de l'écran qui doit afficher le BitMap. Cette valeur
peut être obtenue rapidement et simplement par la fonction ScreenID().

x,y : Position à laquelle doit être affiché le BitMap par rapport à l'écran
(en pixels). La valeur 0,0 se situe en haut à gauche de l'écran.