

# VARIABLE VERS BLOB

Certaines commandes 4D — méconnues ou dont on ne soupçonne pas toute la richesse — méritent un coup de projecteur. Dans chaque numéro, Planète 4D vous aide à les (re)découvrir.

**C**ETTE COMMANDE 4D DE HAUT NIVEAU DISSIMULE, sous une apparence simplicité, une grande puissance puisqu'elle permet de stocker n'importe quoi (ou presque) dans un BLOB. Grâce à elle, on peut concaténer des tableaux et des images, des entiers et des chaînes, des dates et des... BLOBs et les récupérer sans se soucier de la plate-forme. Son utilisation n'a d'autre limite que la taille du BLOB, la mémoire disponible et l'imagination. Cet article dévoile la richesse de la commande et propose des méthodes génériques qui élargissent encore ses possibilités.

## Description de la commande

### ■ Syntaxe :

**VARIABLE VERS BLOB**(MaVariable;MonBlob) stocke la variable MaVariable dans le BLOB MonBlob. Dans ce cas, seule la variable est dans le BLOB. En d'autres termes, si le BLOB contenait autre chose, il est redimensionné pour stocker la variable et le contenu précédent est perdu.

**VARIABLE VERS BLOB**(MaVariable;MonBlob;\*) stocke la variable MaVariable à la fin de MonBlob et la taille du BLOB est redimensionnée en conséquence. L'astérisque, en troisième paramètre, permet de stocker les unes derrière les autres autant de variables que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

**VARIABLE VERS BLOB**(MaVariable;MonBlob;UnOffset) stocke la variable MaVariable à l'offset UnOffset (décalage en nombre d'octets par rapport à 0) dans le BLOB dont la taille est redimensionnée en conséquence. Après l'appel à la commande, unOffset contient unOffset + taille écrite. Cette fonctionnalité est disponible depuis la version 6.5

### ■ Détection des erreurs :

C'est la variable système OK qui renseigne sur l'exécution correcte ou non de la commande. Si la variable a été correctement stockée, OK prend la valeur 1, si l'opération n'a pas pu être effectuée à cause d'un manque de mémoire, la variable OK prend la valeur 0.

### ■ Limites :

MaVariable ne peut pas être un champ ni un pointeur, un tableau de pointeurs ou un tableau à deux dimensions (il est toutefois possible de stocker la deuxième dimension d'un tableau en autant de tableaux à une dimension).

Si MaVariable est un Entier long qui est une référence à une liste hiérarchique, ce n'est pas la liste qui est stockée mais sa référence. Pour stocker des listes hiérarchiques dans un BLOB, il faut utiliser la commande **LISTE VERS BLOB**.

### ■ Remarque :

Les variables stockées dans un BLOB à l'aide de cette commande le sont dans un format interne à 4D (voir encadré) : C'est la commande **BLOB VERS VARIABLE** qui permet de les récupérer. 4D gère l'indépendance de plate-forme en assurant la conversion des données d'une plate-forme à l'autre (conversion des octets ou « byte-swapping »). Ainsi des variables stockées dans un BLOB avec cette commande sous MacOS seront récupérées avec la commande **BLOB VERS VARIABLE**, sans manipulation, sous Windows et vice-versa.

L'utilisation de la concaténation de variables dans un BLOB nous semble particulièrement utile pour :

- Restituer rapidement à l'écran des éléments informatifs qui ne feront pas l'objet de recherche ou de tri et n'ont donc pas besoin d'être enregistrés dans des champs. Par exemple un historique des modifications apportées à l'enregistrement (dans 3 tableaux : tDateModif, tTypeModif et tUtilisateur...), une liste de mots-clés gérés par ailleurs dans des « clusters », plusieurs résolutions d'une image... Cette approche est d'autant plus intéressante que le BLOB peut alors être stocké dans une autre table, limitant le nombre de champs dans la table principale et la taille de l'enregistrement, qui sont des éléments pénalisants en client-serveur lors du chargement de l'enregistrement.

- Retrouver rapidement des variables calculées à partir des champs de la table et pour lequel on veut éviter de refaire le calcul à chaque fois que l'on affiche ou imprime un formulaire.

- Sauvegarder tous les champs d'un enregistrement qui va être supprimé, offrant ainsi la possibilité de récupérer *a posteriori* des enregistrements supprimés.

Pour cette dernière utilisation, le moyen permettant de contourner la limitation de **VARIABLE VERS BLOB** (qui n'accepte pas les champs comme paramètre) sera présenté plus loin. Nous traitons d'abord le stockage dans un BLOB et la récupération de variables depuis un BLOB avec les méthodes projets Variables\_vers\_BLOB et BLOB\_vers\_variables. Dans une troisième partie, enfin, nous étudierons la possibilité d'avoir un accès indexé aux BLOBs construits avec cette commande.

## ■ exemple 1. Stocker et récupérer des variables dans un BLOB

La méthode Variables\_vers\_BLOB concatène les variables pointées par les paramètres \$2 à \$n dans le BLOB pointé par \$1. Elle contrôle les limites de la commande et gère la reconnaissance des listes hiérarchiques.

```

`Méthode : Variables_vers_BLOB
`OK:=Variables_vers_BLOB (->BLOB;->var1;...;->varN)
`Stocke les variables pointées par les paramètres $2 à $n
`dans le BLOB pointé par $1
`Retourne 1 si l'opération a été menée à bien, sinon 0

```

```

` Le BLOB Récepteur
C_POINTEUR($1)
`Les variables à stocker
C_POINTEUR($2)
`Code de contrôle d'exécution
C_ENTIER($0)
C_ENTIER LONG($nParam;$i;$TypeVariable;$NoTable;$NoChamp)
`BLOB tampon
C_BLOB($BLOB)
C_POINTEUR($VariablePtr)
C_ALPHA(44;$nomVar)
`Initialisations
$0:=0

```

\$nParam:=**Nombre de parametres**

`Au moins une variable à stocker

OK:=**Num**(\$nParam>1)

**Si** (OK=1)

`Boucle sur le nombre de variables à mettre dans le BLOB

**Boucle** (\$i;2;\$nParam)

`Récupération du paramètre de rang \$i dans une locale

\$VariablePtr:=\${\$i}

`Vérifier que le pointeur est correctement initialisé

OK:=**Num**(**Non**(**Nil**(\$VariablePtr)))

**Si** (OK=1)

`Vérifier qu'il s'agit bien d'une variable

**RESOUDRE POINTEUR**(\$VariablePtr;\$nomVar;\$NoTable;\$NoChamp)

OK:=**Num**(\$NoTable=-1)

**Fin de si**

**Si** (OK=1)

`Vérifier que la variable peut bien être stockée dans un BLOB

\$TypeVariable:=**Type**(\$VariablePtr->)

OK:=**Num**(\$TypeVariable#**Est un pointeur**) & (\$TypeVariable#**Est un tableau pointeur**) & (\$TypeVariable#**Est un tableau 2D**)

**Fin de si**

**Si** (OK=1)

`Attention aux entiers longs...

**Si** (\$TypeVariable=**Est un entier long**)

`...qui peuvent être une référence à une liste hiérarchique :

**Si** (**Liste existante**(\$VariablePtr->))

`Oui : stocker la Liste hiérarchique à la fin du BLOB ;

**LISTE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

**Sinon**

`Non : Stocker l'entier long à la fin du BLOB.

**VARIABLE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

**Fin de si**

**Sinon**

`Stocker la variable à la fin du BLOB

**VARIABLE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

**Fin de si**

**Fin de si**

`La mémoire est elle saturée ?

**Si** (OK=0)

`Sortir de la boucle

\$i:= \$nParam+1

**Fin de si**

**Fin de boucle**

**Si** (OK=1)

`Répondre que tout est OK.

\$0:=1

`Gagner de la place si possible

**COMPRESSER BLOB**(\$BLOB)

`Changer le BLOB récepteur

\$1->:=\$BLOB

**Fin de si**

**Fin de si**

La méthode retourne 1 si l'exécution s'est déroulée correctement. En cas d'erreur (mémoire insuffisante ou erreur dans l'appel de la méthode) elle retourne 0 et le BLOB pointé par \$1 n'est pas modifié (les variables sont d'abord stockées dans un BLOB local avant que celui-ci ne soit recopié dans le BLOB pointé). C'est donc à la méthode appelante, si elle récupère un code d'erreur, de décider s'il convient d'effacer le BLOB ou de le conserver.

Selon vos habitudes ou par souci d'économiser de la mémoire vive au maximum, ce code pourrait être facilement modifié en utilisant \$1- > à la place du \$BLOB tampon. Vous pouvez aussi systématiquement effacer le BLOB avant l'appel de la méthode ou en ajoutant, en début de méthode, l'instruction **FIXER TAILLE BLOB** (\$1- >; 0)

Lorsque cette méthode est ajoutée à votre application, vous pouvez écrire :

```
OK:= Variables_vers_BLOB(->vBLOB;->vImage;->vTab1D_1;  
-> vTab1D_2;->vListe1;->vBLOB2)
```

La méthode inverse (BLOB\_vers\_variables) récupère les variables pointées par les paramètres \$2 à \$n dans le BLOB pointé par \$1. Lorsque cette méthode est ajoutée à votre application, vous pouvez écrire :

**C\_IMAGE**(vImage)

**C\_ENTIER LONG**(vListe1)

**C\_BLOB**(vBLOB2)

`Il n'est pas possible de créer une liste avec BLOB vers liste :

`Il faut impérativement la créer avant de la récupérer.

`De même, si vous souhaitez réceptionner la liste blobée dans une liste existante, il faut la supprimer au préalable.

**Si** (**Liste existante**(vListe1))

**SUPPRIMER LISTE**(vListe;\*)

**Fin de si**

vListe1:=**Nouvelle liste**

```
OK:=BLOB_vers_variables (->vBLOB;->vImage;->vTab1D_1;-  
>vTab1D_2;->vListe1;->vBLOB2)
```

`Méthode : BLOB\_vers\_variables

`OK:=BLOB\_vers\_variables (->BLOB;->var1;...;->varN)

`Récupère les variables pointées par les paramètres \$2 à \$n

`dans le BLOB pointé par \$1

`Le BLOB Réceptacle

**C\_POINTEUR**(\$1)

`Les variables à récupérer

**C\_POINTEUR**(\$2)

`Code de contrôle d'exécution

**C\_ENTIER**(\$0)

**C\_ENTIER LONG**(\$nParam;\$i;\$TailleBLOB;\$Offset;\$compressé)

**C\_BLOB**(\$BLOB)

**C\_POINTEUR**(\$VariablePtr)

`Initialisations

\$nParam:=**Nombre de parametres**

`Au moins une variable à récupérer

OK:=**Num**(\$nParam>1)

**Si** (OK=1)

`Il est préférable de dépointer les BLOBs (JPR)

\$BLOB:= \$1->

`Décompression éventuelle du BLOB...

**LIRE PROPRIETES BLOB**(\$BLOB;\$compressé)

`...s'il est compressé.

**Si** (\$Compressé#**Non compressé**)

**DECOMPRESSER BLOB**(\$BLOB)

**Fin de si**

**Si** (OK=1)

\$Offset:=0

\$TailleBLOB:=**Taille BLOB**(\$BLOB)

`Boucle sur le nombre de variables à récupérer

```

Boucle ($i;2;$nParam)
OK:=Num($Offset<$TailleBLOB)
Si (OK=1)
$VariablePtr:=${$i}
Si (Type($VariablePtr->)=Est un entier long)
`L'entier long est-il une référence à une liste hiérarchique ?
Si (Liste existante($VariablePtr->))
`Récupérer la Liste hiérarchique
$VariablePtr->:=BLOB vers liste($BLOB;$Offset)
Sinon
`Récupérer l'entier long
BLOB VERS VARIABLE($BLOB;$VariablePtr->,$Offset)
Fin de si
Sinon
`Récupérer la variable
BLOB VERS VARIABLE($BLOB;$VariablePtr->,$Offset)
Fin de si
`La mémoire est elle saturée ?
Si (OK=0)
`Sortir de la boucle
$i:=$nParam+1
Fin de si
Sinon
`Sortir de la boucle
$i:=$nParam+1
Fin de si
Fin de boucle
Fin de si
Fin de si
$0:=OK

```

## ■ exemple 2. Utiliser Variable\_vers\_BLOB avec des pointeurs de champs

Pour pouvoir passer des pointeurs de champs comme paramètres à la méthode Variables\_vers\_BLOB, il faut enrichir cette dernière (pour contourner la limitation de **VARIABLE VERS BLOB** qui n'accepte pas les champs comme paramètres) en plaçant les champs dans des variables génériques au moment du « blobage » :

```

.../... [Code de la méthode Variables_vers_BLOB]
Si (OK=1)
`Vérifier qu'il s'agit bien d'une variable
RESOUDRE POINTEUR($VariablePtr;$nomVar;$NoTable;$NoChamp)
OK:=Num($NoTable=-1)
Si (OK=0)
`Pointe vers un champ : traduire dans une variable
$VariablePtr:=Ptr_Champ_vers_variable($VariablePtr)
`Vérifier qu'il n'y a pas eu d'erreur
OK:=Non(Nil($VariablePtr))
Fin de si
Fin de si
.../...

```

Avec méthode **Ptr\_Champ\_vers\_variable** :

```

`Méthode : Ptr_Champ_vers_variable
`Place le champ pointé par $1 dans une variable process
`et retourne le pointeur sur cette variable
`Pointeur vers un champ
C_POINTEUR($1)
`Pointeur vers la variable ou a été traduit le champ. Nil si erreur
C_POINTEUR($0)
`C_BLOB(Gen_BLOB)

```

```

C_ENTIER(Gen_Entier)
C_ENTIER LONG(Gen_Entier_long)
C_REEL(Gen_Réel)
C_TEXTE(Gen_Texte)
C_ALPHA(255;Gen_Alpha255)
C_IMAGE(Gen_Image)
C_BOOLEAN(Gen_Boolean)
C_DATE(Gen_Date)
C_HEURE(Gen_Heure)
`
C_ENTIER($TypeObjet)
`
$TypeObjet:=Type($1->)
`

```

### Au cas ou

```

:($TypeObjet=Est un BLOB)
$0:=->Gen_BLOB
:($TypeObjet=Est un entier)
$0:=->Gen_Entier
:($TypeObjet=Est un entier long)
$0:=->Gen_Entier_long
:($TypeObjet=Est un numérique)
$0:=->Gen_Réel
:($TypeObjet=Est un texte)
$0:=->Gen_Texte
:($TypeObjet=Est un champ alpha)
$0:=->Gen_Alpha255
:($TypeObjet=Est une image)
$0:=->Gen_Image
:($TypeObjet=Est un booléen)
$0:=->Gen_Boolean
:($TypeObjet=Est une date)
$0:=->Gen_Date
:($TypeObjet=Est une heure)
$0:=->Gen_Heure

```

### Sinon

**ALERTE**("Appel de la méthode avec un objet non géré!")

### TRACE

### Fin de cas

**Si** (**Non(Nil(\$0))**)

\$0->:=\$1->

### Fin de si

La méthode, ainsi modifiée, pourrait s'appeler VariablesOuChamps\_vers\_BLOB! et être appelée dans votre code pour stocker des champs dans un BLOB :

```

OK:=VariablesOuChamps_vers_BLOB(->vBLOB;->[Table]Champ1;
...;-> [Table] ChampN)

```

De la même façon, la méthode de « déblobage », **BLOB\_vers\_VariablesOuChamps**, devra s'appuyer sur ce principe de récupération de la valeur des champs dans des variables process « tampon » pour accepter des pointeurs vers des champs comme paramètres.

## ■ exemple 3. BLOBs « indexés »

Malheureusement, le format des BLOBs créés avec les exemples précédents, ne permet d'accéder aux variables que de façon séquentielle. En d'autres termes, pour récupérer la cinquième variable d'un BLOB qui en contient dix il faut déjà extraire les quatre premières... Pire pour n'obtenir que la dernière variable d'un BLOB de vingt, il faut passer par les dix-neuf premières. Il est souhaitable dans certains cas d'avoir un accès indexé aux variables stockées dans le BLOB. La solution consiste à conca-

téner les variables dans le BLOB et à sauvegarder leurs offsets dans un tableau, ce dernier étant stocké à la fin du BLOB. Ainsi, lors du « déblobage indexé » il suffira d'extraire le tableau d'offsets et d'y lire l'adresse du *i*ème élément.

La modification de la méthode `Variable_vers_BLOB` en `Variables_vers_BLOB_Indexé` permet de construire un tel BLOB :

```

`Méthode : Variables_vers_BLOB_Indexé
`OK:=Variables_vers_BLOB_Indexé (->BLOB;->var1;...;->varN)
`Stocke les variables pointées par les paramètres $2 à $n
`dans le BLOB pointé par $1
`Retourne 1 si l'opération a été menée à bien, sinon 0
`
` Le BLOB Récepteur
C_POINTEUR($1)
`Les variables à stocker
C_POINTEUR($2)
`Code de contrôle d'exécution
C_ENTIER($0)
C_ENTIER
LONG($nParam;$i;$TypeVariable;$numTable;$numChamp;$Adresse)
C_BLOB($BLOB)
C_POINTEUR($VariablePtr)
C_ALPHA(44;$nomVar)
`
`Initialisations
$0:=0
$nParam:=Nombre de parametres
`Au moins une variable à stocker
OK:=Num($nParam>1)
Si (OK=1)
`Préparer un tableau pour recevoir les adresses des variables
TABLEAU ENTIER LONG($tBLOBIndex;$nParam-1)
` Boucle sur le nombre de variables à mettre dans le BLOB
Boucle ($i;2;$nParam)
` Récupération du paramètre de rang $i dans une locale
$VariablePtr:=${$i}
`Vérifier que le pointeur est correctement initialisé
OK:=Num(Non(Nil($VariablePtr)))
Si (OK=1)
`Vérifier qu'il s'agit bien d'une variable
RESOUDRE POINTEUR($VariablePtr;$nomVar;$NoTable;$NoChamp)
OK:=Num($NoTable=-1)
Fin de si
`
Si (OK=1)
`Vérifier que la variable peut bien être stockée dans un BLOB
$TypeVariable:=Type($VariablePtr->)
OK:=Num($TypeVariable#Est un pointeur) & ($TypeVariable#Est un tableau pointeur) & ($TypeVariable#Est un tableau 2D))
Fin de si
`
Si (OK=1)
`L'offset où la variable est sauvegardée
$tBLOBIndex{$i-1}:=Taille BLOB($BLOB)
`
`Attention aux entiers longs...
Si ($TypeVariable=Est un entier long)
`...qui peuvent être une référence à une liste hiérarchique :
Si (Liste existante($VariablePtr->))
`Oui : stocker la Liste hiérarchique à la fin du BLOB ;
LISTE VERS BLOB($VariablePtr->,$BLOB;*)
Sinon
`Non : stocker l'entier long à la fin du BLOB.
VARIABLE VERS BLOB($VariablePtr->,$BLOB;*)

```

## Fin de si

### Sinon

` Stocker la variable à la fin du BLOB

**VARIABLE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

### Fin de si

### Fin de si

`La mémoire est-elle saturée ?

### Si

`Sortir de la boucle

\$i:=\$nParam+1

### Fin de si

### Fin de boucle

### Si

`Sauvegarder le tableau d'index :

`L'offset où va être sauvegardé le tableau

\$Adresse:=**Taille BLOB**(\$BLOB)

`Stocker le tableau à la fin du BLOB

**VARIABLE VERS BLOB**(\$tBLOBIndex,\$BLOB;\*)

### Si

`Stocker la position du tableau à la fin du BLOB

`en forçant l'ordre des octets pour que l'entier long

`soit toujours stocké/restitué dans le même

`ordre de byte-swapping, ce qui garantit sa validité

**ENTIER LONG VERS BLOB**(\$Adresse,\$BLOB;Ordre octets Macintosh;\*)

### Fin de si

### Si

`Répondre que tout est OK.

\$0:=1

`Gagner de la place si possible

**COMPRESSER BLOB**(\$BLOB)

`Changer le BLOB récepteur

\$1->:=\$BLOB

### Fin de si

### Fin de si

### Fin de si

La méthode `BLOB_indexé_vers_variable` permet alors de récupérer une image stockée en 10<sup>e</sup> position dans le BLOB en écrivant :

### C\_IMAGE

OK := **BLOB\_indexé\_vers\_variable** (->vBLOB;->vImage ;10)

`Méthode : `BLOB_indexé_vers_variable`

`OK:=**BLOB\_indexé\_vers\_variable** (->BLOB;->Var;\$Position)

`Récupère la variable pointée par \$2

`stockée à la 3<sup>e</sup> position dans le BLOB pointé par \$1

`Le BLOB Réceptacle

**C\_POINTEUR**(\$1)

`La variable à récupérer

**C\_POINTEUR**(\$2)

`L'index (position) de la variable dans le BLOB

**C\_ENTIER LONG**(\$3)

`Code de contrôle d'exécution

**C\_ENTIER**(\$0)

**C\_ENTIER LONG**(\$nParam;\$Position;\$Offset;\$Adresse;

\$NoTable;\$NoChamp;\$Compressé)

**C\_BLOB**(\$BLOB)

**C\_POINTEUR**(\$VariablePtr)

**C\_ALPHA**(44;\$NomVar)

```

` Initialisations
$0:=0
$NParam:=Nombre de parametres
` Nombre de paramètres attendus
OK:=Num($NParam=3)
`
Si (OK=1)
` Il est préférable de dépointer les BLOBs (JPR)
$BLOB:=$1->
$VariablePtr:=$2
$Position:=$3
` Vérifier que le pointeur est correctement initialisé
OK:=Num(Non(Nil($VariablePtr)))
Si (OK=1)
` Vérifier qu'il s'agit bien d'une variable
RESOUDRE POINTEUR($VariablePtr;$NomVar;$NoTable;$NoChamp)
OK:=Num($NoTable=-1)
Fin de si
`
Si (OK=1)
` Décompression éventuelle du BLOB...
LIRE PROPRIETES BLOB($BLOB;$Compressé)
` ...s'il est compressé.
Si ($Compressé#Non compressé)
DECOMPRESSER BLOB($BLOB)
Fin de si
Fin de si
`
Si (OK=1)
` Récupérer la position du tableau des offsets (1 entier long = 4 octets)
$Offset:=Taille BLOB($BLOB)-4
$Adresse:=BLOB vers entier long($BLOB;Ordre octets Macintosh;$Offset)
` Vérifier que l'adresse a été récupérée correctement
OK:=Num($Adresse>0)
`
Si (OK=1)
` Récupérer le tableau d'offsets
TABLEAU ENTIER LONG($tBLOBIndex;0)
BLOB VERS VARIABLE($BLOB;$tBLOBIndex;$Adresse)
`
Si (OK=1)
` Vérifier que $3 correspond à un indice valide du tableau
OK:=Num(Taille tableau($tBLOBIndex)>=$Position)
`
Si (OK=1)
` Récupérer l'offset de la variable...
$Offset:=$tBLOBIndex{$Position}
` ... et la variable.
BLOB VERS VARIABLE($BLOB;$VariablePtr->;$Offset)
Fin de si
Fin de si
Fin de si
Fin de si
`
$0:=OK

```

Il manque enfin une méthode permettant de modifier une variable dans un BLOB indexé, c'est le but de `variable_dans_BLOB_indexé`. Après l'appel de cette méthode, l'index est mis à jour et le BLOB est éventuellement redimensionné si la variable modifiée n'occupe pas la même place que la variable d'origine. La modification d'une variable image en 3e position dans un BLOB se fera en écrivant :

```

OK := Variable_dans_BLOB_indexé (->vBLOB;->vNouvelleImage;3)

` Méthode : Variable_dans_BLOB_indexé
` OK:=Variable_dans_BLOB_indexé (->BLOB;->var;$Position)
` Stocke la variable pointée par $2
` à la $3e position dans le BLOB pointé par $1
`
` Le BLOB
C_POINTEUR($1)
` La variable à stocker
C_POINTEUR($2)
` L'index (position) de la variable dans le BLOB
C_ENTIER LONG($3)
` Code de contrôle d'exécution
C_ENTIER($0)
`
C_ENTIER LONG($NParam;$Offset;$Adresse;$Index;$i;$nbVar)
C_BLOB($BLOB;$BLOB_2)
C_POINTEUR($VariablePtr)
`
$0:=0
$NParam:=Nombre de parametres
`
OK:=Num($NParam=3)
`
Si (OK=1)
$BLOB:=$1->
$VariablePtr:=$2
$Index:=$3
` Décompression éventuelle du BLOB
LIRE PROPRIETES BLOB($BLOB;$Compressé)
` ...s'il est compressé.
Si ($Compressé#Non compressé)
DECOMPRESSER BLOB($BLOB)
Fin de si
`
Si (OK=1)
` Récupérer la position du tableau des offsets (1 entier long = 4 octets)
$Offset:=Taille BLOB($BLOB)-4
$Adresse:=BLOB vers entier long($BLOB;Ordre octets Macintosh;$Offset)
` Vérifier que l'adresse a été récupérée correctement
OK:=Num($Adresse>0)
`
Si (OK=1)
` Récupérer le tableau d'offsets...
TABLEAU ENTIER LONG($tBLOBIndex;0)
` Utiliser $Offset pour conserver $Adresse qui est réutilisée plus loin
$Offset:=$Adresse
BLOB VERS VARIABLE($BLOB;$tBLOBIndex;$Offset)
`
Si (OK=1)
` ... et le supprimer du BLOB.
SUPPRIMER DANS BLOB($BLOB;$Adresse;
Taille BLOB($BLOB)-$Adresse)
`
Si (OK=1)
` Nombre variables stockées dans le BLOB
$nbVar:=Taille tableau($tBLOBIndex)
OK:=Num($nbVar>0)
Si (OK=1)
Si ($Index<$nbVar)
` Ce n'est pas la dernière variable du BLOB
` Récupérer l'Offset de la variable suivante
$Offset:=$tBLOBIndex{$Index+1}
` Ne garder que la fin du BLOB
COPIER BLOB($BLOB;$BLOB_2;$Offset;0;Taille BLOB($BLOB)-$Offset)

```

## Fin de si

`Offset de la variable à stocker

\$Offset:=StBLOBIndex{\$Index}

`Garder le début du BLOB...

**SUPPRIMER DANS BLOB**(\$BLOB;\$Offset;**Taille BLOB**(\$BLOB)-\$Offset)

**Si** (OK=1)

`...ajouter la variable modifiée...

`...attention aux entier longs...

**Si** (**Type**(\$VariablePtr->)=Est un entier long)

`...qui peuvent être une référence à une liste hiérarchique :

**Si** (**Liste existante**(\$VariablePtr->))

`Oui : stocker la Liste hiérarchique à la fin du BLOB ;

**LISTE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

**Sinon**

`Non : Stocker l'entier long à la fin du BLOB.

**VARIABLE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

**Fin de si**

**Sinon**

`Stocker la variable à la fin du BLOB

**VARIABLE VERS BLOB**(\$VariablePtr->,\$BLOB;\*)

**Fin de si**

**Si** (OK=1)

\$Offset:=**Taille BLOB**(\$BLOB)

`...et, si ce n'était pas la dernière variable...

**Si** (\$Index<\$nbVar)

`...la fin du BLOB.

**COPIER BLOB**(\$BLOB\_2,\$BLOB;0;\$Offset;**Taille BLOB**(\$BLOB\_2))

**Si** (OK=1)

`Corriger le décalage d'offset dans le tableau

\$Décalage:=\$Offset-\$tBLOBIndex{\$Index+1}

**Boucle** (\$i;\$Index+1;\$nbVar)

\$tBLOBIndex{\$i}:=\$tBLOBIndex{\$i}+\$décalage

**Fin de boucle**

**Fin de si**

**Fin de si**

**Si** (OK=1)

`Sauvegarder le tableau d'index :

`L'offset où va être sauvegardé le tableau

\$Adresse:=**Taille BLOB**(\$BLOB)

`Stocker le tableau à la fin du BLOB

**VARIABLE VERS BLOB**(\$tBLOBIndex,\$BLOB;\*)

**Si** (OK=1)

`Stocker la position du tableau à la fin du BLOB

**ENTIER LONG VERS BLOB**(\$Adresse,\$BLOB;**Ordre octets** Macintosh;\*)

**Fin de si**

**Si** (OK=1)

`Répondre que tout est OK.

\$0:=1

`Gagner de la place si possible

**COMPRESSER BLOB**(\$BLOB)

`Changer le BLOB récepteur

\$1->:=\$BLOB

**Fin de si**

[ajouter 9 fin de si supplémentaires]

## Pour aller plus loin

Le format interne de stockage d'une variable dans un BLOB :

### Une partie constante :

- 4 octets, indiquant qu'une variable est enregistrée dans le BLOB au format 4D. Ces 4 caractères sont «BLVR» si la commande a été appelée depuis un système MacOS et «RVLB» si l'enregistrement de la variable dans le BLOB a été effectué depuis un système

Windows («BLLS» ou «SLLB» dans le cas de Liste vers BLOB).

*Note* : L'ordre de « byte-swapping » est déduit de ces 4 premiers octets. Si vous devez manipuler directement «l'intérieur» de tels BLOBs, vous effectuez la conversion d'octets nécessaire.

- Un octet pour le type de la variable (même valeur que les constantes de type de variable en sachant que Entier et Entier long ne sont pas différenciés et sont référencés sous le type 9)

### Une partie variable suivant le type, pour une variable :

- **Image** : Un entier long (4 octets) pour sa taille puis l'image (les 8 premiers octets, le rectangle, doivent être convertis).
- **Texte** : Un entier (2 octets) pour la longueur puis le texte.
- **Booléen** : Un octet pour sa valeur (0 ou 1 pour Faux et Vrai).
- **Entier** ou **entier long** : L'entier ou l'entier long.
- **Alpha** : Un entier pour la longueur déclarée de la variable ; un octet pour la longueur réelle de la variable ; la chaîne.
- **Date** : Un entier pour le jour ; un entier pour le mois ; un entier pour l'année (sur 4 chiffres).
- **Heure** : Un entier long pour le nombre de secondes écoulées depuis minuit.
- **Tableau** : Un entier long pour le nombre d'éléments du tableau (tel que retourné par la fonction **Taille tableau**), un entier long pour l'élément couramment sélectionné, puis le tableau.
- **BLOB** : Un entier long pour la taille du BLOB, puis le BLOB.

Connaissant cette structure, il est possible, par exemple, de créer la méthode PICT\_VERS\_BLOB :

**VARIABLE VERS BLOB**(MonImage;MonBLOB;\*)

`Supprimer 9 octets : 5 pour la partie constante + 4 pour la taille

**SUPPRIMER DANS BLOB**(MonBLOB;0;9)

Il ne reste plus alors, dans le BLOB, qu'une PICT débarrassée des informations 4D. Vous pouvez maintenant, si vous maîtrisez le format PICT, manipuler cette image. Attention toutefois à la conversion d'octets du rectangle enregistré dans les 8 premiers octets. La restitution de l'image se fera avec la méthode BLOB\_VERS\_PICT :

`Reconstruire les 9 premiers octets :

`En tête en fonction de la plate-forme

**Si** (*IsWindows*)

\$EnTete := "RVLB"

**Sinon**

\$EnTete := "BLVR"

**Fin de si**

**TEXTE VERS BLOB**(\$EnTete,\$BLOB;Texte sans longueur;\*)

`Type image =10

**INSERER DANS BLOB**(\$BLOB;4;1;10)

`Taille de l'image = taille du BLOB

**ENTIER LONG VERS BLOB**(**Taille BLOB**(MonBLOB);\$BLOB;0;\*)

`Insérer 9 octets avant la PICT...

**INSERER DANS BLOB**(MonBLOB;0;9)

`...y mettre les informations d'entête de type et de taille.

**COPIER BLOB**(\$BLOB;MonBLOB;0;0;9)

`Récupérer l'image

**BLOB VERS VARIABLE**(MonBLOB;MonImage;0)

Vincent de Lachaux

*Vincent de Lachaux (vincent.delachaux@wanadoo.fr) est médecin hémobio-  
logiste à Paris. Entre deux globules rouges, il décortique la documen-  
tation 4D et en extrait la substantifique moelle. Une devise : « Quand tu as  
un problème, pense à 4D. Quand tu as un problème avec 4D, pense aux BLOBs ».*