

Les appels d'APIs en VB6

Quand on commence à faire de la programmation proche du système, donc avancée, on a souvent besoin d'API : fonctions proposées par le système d'exploitation. Ces fonctions sont stockées dans de bibliothèque de liens : les DLLs.

Il en existe trois catégories principales :

- User32.dll : toutes le fonctions en rapport avec les fenêtres
- Gdi32.dll : tout ce qui est en rapport avec les images
- Kernel32.dll : tout ce qui a rapport avec les mécanismes internes : informations système, système de fichier, mutex, sémaphores.

1. Déclaration

Les déclarations d'APIs se font comme suit :

```
{Private|Public} Declare {Function|Sub} nom_fonction [Alias "vrai_nom_fonction"] Lib "nom_dll.dll"  
(Param1 As Type1, Param2 As Type2,... ParamN As TypeN) [As TypeRetour]
```

{Private|Public} : portée de la déclaration (privée ou publique)

{Function|Sub} : fonction ou procédure, une API Windows est rarement une procédure (sauf CopyMemory)

[Alias "*vrai_nom_fonction*"] : permet de renommer la fonction si son vrai nom est, par exemple, trop long ou trop compliqué ([_fct@8](#)) ou pour importer des fonctions exportées par Index (dans ce cas l'alias sera *#index*).

Exemple :

```
Private Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (ByRef Destination As Any, ByRef Source As Any, ByVal Length As Long)
```

```
Private Declare Function GetComputerName Lib "kernel32.dll" Alias "GetComputerNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As Long
```

Pour trouver une déclaration d'API, vous pouvez utiliser l'API viewer ou tout autre liste sur Internet.

2. Les paramètres

Il existe deux types de paramètres : les paramètres par valeur et les paramètres par référence (pointeur).

Un paramètre par référence indique que l'on passe l'adresse de la variable à la fonction pour pouvoir modifier la variable à l'intérieur. Le nom d'un tel paramètre est précédé de ByRef ou rien (ByRef par défaut).

Un paramètre par valeur indique que l'on passe le contenu de la variable à la fonction. On ne peut donc pas modifier la variable à l'intérieur de la fonction. Le nom d'un tel paramètre est précédé de ByVal.

Deux types ne peuvent pas être ByVal : les tableaux et les structures.

3. Le type de retour

Si c'est une Sub, alors il n'y a pas de type de retour.

Sinon, c'est presque toujours un Long. **Tout type de retour de plus de 4 octets devrait être passé par référence en paramètre.**

4. Et si ma fonction n'est pas dans une liste

Alors la c'est un peut plus compliqué. Mais si vous êtes là, c'est que vous avez le prototype C de votre fonction.

Pour que VB puisse utiliser une fonction C comme une API, il faut :

- **Que la convention d'appel soit STDCALL (par défaut cdecl en C)**
- **Qu'elle soit exportée avec un extern « C » ou un def file**

Voici un tableau de types courants et de leurs traductions VB6 :

Type en C	Type en VB
char/BYTE	ByVal Byte (mais considéré non signé)
short	ByVal Integer
int	ByVal Long
long	ByVal Long
unsigned char	ByVal Byte
unsigned short/WORD/USHORT	ByVal Integer (mais considéré signé)
unsigned int/DWORD	ByVal Long (mais considéré signé)
unsigned long/ULONG	ByVal Long (mais considéré signé)
BOOL	ByVal Long (mais <> Boolean)
float	ByVal Single
double	ByVal Double
ULONGLONG, LONGLONG	ByVal Currency (considéré comme un flottant avec 4 décimales)
char*	ByVal String
short*	ByRef Integer
int*	ByRef Long
long*	ByRef Long
unsigned char*	ByRef Byte
unsigned short*	ByRef Integer (mais considéré signé)
unsigned int*	ByRef Long (mais considéré signé)
unsigned long*	ByRef Long (mais considéré signé)
BOOL*	ByRef Long (mais <> Boolean)
float*	ByRef Single
double*	ByRef Double
ULONGLONG*, LONGLONG*	ByRef Currency (considéré comme un flottant avec 4 décimales)
BSTR, LPWSTR*	ByVal Long et StrPtr pour le passage du parameter ou IDL (voir autre tuto)
BSTR*	ByVal Long et VarPtr pour le passage du parameter ou IDL (voir autre tuto)
<i>Interface*</i>	ByVal {Object <i>Interface</i> }
<i>Interface**</i>	ByRef {Object <i>Interface</i> }
<i>Structure</i>	<i>Voir plus bas</i>
<i>Structure*</i>	ByRef <i>Structure</i>
SAFEARRAY(<i>Type</i>)*	ByRef <i>Type</i> ()

<i>Tableau C</i>	<i>Voir plus bas</i>
Hxxx	Long (représente un handle)

Les types pointeurs

Tous les types pointeurs peuvent être remplacés par ByVal Long. Dans ce cas, on utilisera VarPtr(*la_variable*) ou StrPtr(*la_variable*) pour passer l'adresse de la variable. Voir *Travailler avec les pointeurs*.

Les types Structures

Les types structures ne peuvent pas être passé par valeur. Pour pouvoir les passer tout de même par valeur, il mettre autant de paramètres dans le Declare que de membre dans la structure.

Note sur les structures :

- L'alignement des données de VB6 dans les structures se fait sur 4 octets :

Par exemple :

```
Private Type
    B As Byte
    L As Long
End Type
```

Cette structure prendra 8 octets de mémoire avec 3 octets de vide entre B et L.

Il faut donc faire attention à l'alignement dans les structures C qui peut aller jusqu'à l'octet (voir #pragma pack).

- Les chaînes de taille fixe dans les structures (char *nom*[*taille*]) se traduisent en *nom* As String *
taille
- Les chaînes char*, WCHAR* sont remplacées par Long. On utilise ensuite CopyMemory pour copier le contenu de la variable au bout du pointeur.
- **Les pointeurs sont interdits dans les structures VB6.** On les remplace par Long et on utilise CopyMemory pour copier le contenu dans une autre variable. (CopyMemory ByVal *pointeur*, ByVal *variable*, *taille_zone_pointée*). Voir *Travailler avec les pointeurs*.
- Les tableaux de taille fixe *type nom*[*taille*] se traduisent par *nom*(*taille*) As *Type*
- Les **unions** n'existent pas : on les remplace par le membre de l'union qui a la plus grande taille en octets. Pour accéder à la donnée d'une union dans un des autres types, on utilisera CopyMemory ByVal *var_autre_type*, ByVal *membre_union*, LenB(*var_autre_type*)
- Pour les autres types :

Type C	Type VB
char/BYTE	Byte (interprété non signé)
short/USHORT/WORD	Integer
int, long/DWORD/ULONG	Long
ULONGLONG	Currency
Float	Single
double	Double
BOOL/BOOLEAN	Long
bool	Byte

Les types Tableaux

Les types tableaux ne peuvent pas être passé par valeur. On peut mais c'est très difficile et très inutile. Les tableaux VB6 ne sont pas simplement une zone mémoire de donnée et un pointeur dessus. Toute une structure les accompagne.

Passer un tableau type C

Pour passer un tableau de type C, on a deux solutions :

- *ByRef Type_des_cases_du_tableau* : dans ce cas, on passe la première case du tableau, en général, il y a un paramètre pour donner la taille du tableau que l'on passe

Par exemple :

```
Dim t(10) As Long
Res = Fct(t(0),10)
```

- *ByVal Long* : dans ce cas, on passe *VarPtr(le_tableau(0))* (si 0 est la première case).

Par exemple :

```
Dim t(10) As Long
Res = Fct(VarPtr(t(0)),10)
```

La seconde solution permet par exemple de passer plusieurs type de données à une fonction.

Passer un tampon chaîne ASCII type C

Pour passer un buffer chaîne C : *char* buffer* (int taille, suit en général), on utilise un *ByVal String*. **Il faut impérativement remplir la chaîne avec des caractères avant de la passer à la fonction.**

Par exemple :

```
Dim s as string
s = space(20)
res = Fct(s,20)
```

Passer un tampon chaîne UNICODE type C

Pour passer un buffer chaîne C : *wchar* buffer* (int taille, suit en général), on utilise un *ByVal Long*. **Il faut impérativement remplir la chaîne avec des caractères avant de la passer à la fonction.** Comme les chaînes VB6 sont UNICODE, on passe *StrPtr(le_buffer)* à la fonction.

Par exemple :

```
Dim s as string
s = space(20)
res = Fct(StrPtr(s),20)
```

Any

Ce type spécial permet de passer n'importe quel type au paramètre de la fonction. On peut utiliser les mots *ByVal* et *ByRef* dans les appels de fonctions.

Passer un pointeur de fonction

Pour passer un pointeur de fonction, on utilise un *ByVal Long* et *AddressOf* dans l'appel de fonction.

Cette fonction doit être dans un module (bas).

Par exemple :

```
res = Fct(AddressOf MaFonction)
```

Travailler avec les pointeurs

Tous les types pointeurs que vous avez remplacé par *Long* dans les structures ou les pointeurs de pointeurs (autres que *Object*) ne peuvent pas être utilisés directement par VB6. Pour accéder à la donnée pointée il faut utiliser *CopyMemory*. La méthode est la suivante :

Vous avez une adresse dans un *Long* : membre de structure ou variable...

Vous déclarez une variable du type de la donnée pointée : structure, type simple ou chaîne

Si la donnée pointée est une *STRUCTURE* ou type simple (*Long*, *Integer*, ...) :

```
Dim variable As Type
```

```
'le pointeur est pointeur As Long
```

```
CopyMemory ByRef variable, ByVal pointeur, LenB(variable)
```

```
'ou
```

```
CopyMemory ByVal VarPtr(variable), ByVal pointeur, LenB(variable)
```

Si la donnée pointée est un tableau C de STRUCTURES ou types simples (Long, Integer, ...) :

Dim *variable*() As *Type*

'le pointeur est *pointeur* As Long et *taille* est le nombre de cases du tableau

ReDim *variable*(*taille*)

CopyMemory ByVal *variable*(0), ByVal *pointeur*, LenB(*variable*) * *taille*

'ou

CopyMemory ByVal VarPtr(*variable*(0)), ByVal *pointeur*, LenB(*variable*) * *taille*

Si la donnée pointée est une chaîne ANSI :

'lstrlenA : api renvoyant la taille ANSI d'une chaîne de caractère

Private Declare Function lstrlenA Lib "kernel32.dll" (ByVal lpString As Long) As Long

Dim *variable* As String

'le pointeur est *pointeur* As Long

taille = lstrlenA(*pointeur*)

variable = Space(*taille*)

CopyMemory ByVal StrPtr(*variable*), ByVal *pointeur*, *taille* + 1

variable = StrConv(*variable*, vbUnicode)

Si la donnée pointée est une chaîne UNICODE :

'lstrlenW : api renvoyant la taille UNICODE d'une chaîne de caractère

Private Declare Function lstrlenW Lib "kernel32.dll" (ByVal lpString As Long) As Long

Dim *variable* As String

'le pointeur est *pointeur* As Long

taille = lstrlenW(*pointeur*)

variable = Space(*taille*)

CopyMemory ByVal StrPtr(*variable*), ByVal *pointeur*, *taille* + 2