Les API en Visual Basic

Par J-M RABILLOUD

Préambule

On entend par API (Application Programming Interface), l'ensemble des fonctions systèmes de l'OS qui peuvent être appelées à partir du code Visual Basic. Elles se trouvent dans des fichiers DLL du répertoire système. Par extension, on peut de la même façon utiliser des fonctions se trouvant dans des DLL non-système.
Le présent article n'a pas pour but de présenter une liste d'API ainsi que des exemples d'utilisation, mais bien de comprendre comment les utiliser, les avantages et les risques liés à leur utilisation.
N.B : Pour écrire ce texte je me suis appuyé sur le chapitre « Accès aux DLL et à l'API de Windows » du MSDN Microsoft →Visual Studio ™ 6.0

Des API pour faire quoi ?

En fait je distinguerai trois cas où l'usage des API est intéressant :

La programmation système

Elles sont principalement utilisées dans ce cas. Il y a un certain nombre de fonctionnalités que VB ne peut pas traiter et qui oblige à passer par l'utilisation d'API (par exemple changer l'imprimante par défaut)

L'accélération du code

Visual Basic étant un langage reconnu plus lent que les autres. Il peut être donc avantageux d'utiliser des API dans certaines phases pour augmenter les performances du programme.

L'allégement du programme

Pour des raisons de taille aussi il peut être préférable d'utiliser des API plutôt que des contrôles ActiveX.

Les risques

Les principales réticences à l'utilisation des API sont dues :

- Au risque inhérent à la programmation système
- A la faiblesse de la documentation sur ces fonctions.

Cette idée de risque est due à mon avis au fait que l'on ne sait pas comment fonctionnent ces API de manière interne. Croyez moi, elles ne présentent pas plus de risques que n'importe quelle autre fonction, du moment où l'on sait ce que l'on fait. Pour la documentation, il existe de plus en plus de site Internet abordant ce sujet. Certains sont très bien fait. Mais je ne saurais trop vous mettre en garde contre le copier/coller de code, sans chercher à comprendre ce que celui-ci fait. C'est à ce moment là que débute les problèmes.
Mais trêve de philosophie, regardons en détail l'utilisation des API.
Attention, il faut se méfier du fait que certaines de ces API sont propres à un système. Dans certains cas, une application tournant sur Win95 provoquera une erreur sur WinNT.

La déclaration de la fonction

Les fonctions API s'utilisent comme les fonctions VB à ceci près qu'elles **doivent** être déclarées avant d'être appelées.
Reprenons la syntaxe générale de déclaration
[Private | Public] Declare Function | Sub *publicname* **Lib** "*libname*" **[Alias** " *aliasname*"**] [([ByVal]** *variable* **[As** *type***] [,[ByRef]** *variable* **[As** *type***]]...])] [As Type]**
L'instruction **Private** ou **Public** définit comme pour une fonction VB la visibilité de la fonction. La déclaration ne peut être publique que dans un module, dans ce cas elle est visible dans toute l'application.
L'instruction **Declare** est obligatoire. Elle annonce un appel à une procédure externe.
Function ou Sub comme en VB, Sub si la procédure ne renvoie rien. L'un des deux doit être indiqué.
Publicname est le nom de la procédure appelée. Ce doit être un nom Visual Basic valide. Il est sensible à la casse (majuscule/minuscule) si *aliasname n'est pas défini.*

Lib est obligatoire pour signaler la procédure externe
Libname est le nom de la DLL. Il doit être forcément entre guillemets. Il est à noter que pour les DLL du répertoire système, ni le chemin ni l'extension ne sont obligatoire (par ex : Lib "User32" est équivalent à Lib "c:\windows\system\user32.dll"), par contre pour les autres il faut le nom complet.
Alias " *aliasname* " Le mot clé Alias sert à préciser que la procédure dans la DLL porte un autre nom que celui spécifier dans publicname. Cela peut servir si la fonction porte un nom réservé VB ou pour donner un nom plus explicite à la fonction. Aliasname est donc le vrai nom de la procédure dans la DLL. Il est toujours entre guillemets. Il est évidemment sensible à la casse.
As Type est le type de la donnée renvoyée si c'est une fonction et se situe en fin de déclaration.

Passage de paramètres et pointeurs

C'est là que se trouve le problème principal. Bien que ces notions soit importantes en Visual Basic, beaucoup de programmeurs sont passés à travers et ne savent pas trop comment ils passent leurs paramètres. Ces concepts sont encore plus importants pour les API qui sont des fonctions C.
Le mot clé **ByVal** annonce que l'on fait une copie de la variable, et c'est cette copie qui est passée comme paramètre. Par contre **ByRef** passe l'adresse de la variable comme paramètre. Bien évidemment ce n'est pas la même chose.
Dans un passage par valeur, si la fonction modifie le paramètre, la variable d'origine ne sera pas modifiée, alors que dans un passage par référence (adresse), la variable est modifiable. En fait le passage par référence est un pointeur sur la variable. Or en Visual Basic, le mode de passage par défaut (c.a.d si le mot clé est omis) est ByRef, ce qui fait que la plupart du temps la variable est modifiable par la fonction. Si lorsqu'on écrit une fonction ce n'est pas trop grave (pour peu qu'on sache ce que l'on fait), dans un appel à une DLL cela peut être redoutable. Il convient donc de faire très attention à respecter le mode de passage.
Il est à noter que les pointeurs (adresse) sont toujours des **Long**.
Comme ce sujet est fondamental, je vous conseille de lire le chapitre 4 du cours de C++ de Christian Casteyde que vous trouverez là ➔ http://www.developpez.com/c/megacours/c1474.html
Je vous conseille aussi d'utiliser la visionneuse API fournit avec VB6, qui permet au moins de ne pas faire de faute dans la déclaration.
Pour finir sur ce point, il faut noter que la possibilité de modifier un paramètre est beaucoup utiliser par les API, et que c'est très souvent ces paramètres qui contiennent l'information à récupérer. Mais j'y reviendrai au cours des exemples.

Les chaînes

Je ne vais pas me lancer dans de longues explications sur le type string en Visual Basic, il vous suffit de savoir que les variables de type string se passent par valeur. Pour ceux qui sont intéressés par plus de détails reportez-vous au chapitre " Passage de chaînes à une procédure DLL" de l'aide MSDN.
Les fonctions DLL attendent des chaînes dimensionnées pour qu'elles puissent écrire dedans. Donc, avant de passer la chaîne ont fait soit *space$(nbCarac)* soit *string$(nbCarac,vbNullChar)*. Il est impératif que la taille donnée soit suffisante pour que la fonction puisse écrire toute la chaîne, sinon soit la fonction ne renvoie rien (ce qui est ennuyeux) soit elle écrira en mémoire la chaîne quitte à écraser d'autres données (ce qui peut être désastreux).

Les tableaux

Le passage de tableaux va dépendre de la fonction appelée. Si elle supporte automation, on passe le tableau comme en VB, sinon on passe le premier élément du tableau **par référence**. Je reviendrai plus loin sur ce cas avec un exemple.

Les types utilisateurs

Dans les appels API les types utilisateurs sont souvent employés. Comme ils apparaissent comme types de paramètres, on peut les trouver avec la visionneuse API. Ils sont presque toujours passer par référence.

Les constantes

On les trouvent toutes dans la visionneuse API mais pour savoir quelle fonction utilise quelle constante, il faut regarder dans la documentation. C'est à mon avis, le principal défaut de la visionneuse.

Les paramètres particuliers

Certaines API attendent des paramètres que l'on n'utilise pas souvent. Par exemple, le pointeur de fonction renvoie l'adresse d'une fonction (Visual Basic) que l'on a écrite. Là encore il suffit de savoir que le mot clé AddressOf renvoie ce pointeur. Dans les exemples de la deuxième partie je montrerai des exemples de ces paramètres.

Les descripteurs Hdc et hwnd

Bien que l'on puisse écrire de nombreuses applications sans jamais utiliser les descripteurs, ils sont indispensables à la programmation des API. Sachez simplement qu'ils servent au système d'exploitation pour identifier de manière unique des objets ou des contextes de périphérique.
N.B : ne mettez jamais la propriété hdc dans une variable car la valeur peut changer pendant l'exécution.

L'appel de la fonction

Une fois déclarées, ces fonctions s'utilisent comme des fonctions Visual Basic.

La documentation

C'est le défaut principal des API. Il existe des sites Internet bien fait qui traitent de ce sujet. Mais principalement, on trouve sur le site Microsoft un fichier Win32.hlp qui contient toute l'aide de programmation des API. Ils a été écrit pour les développeurs C++, mais si les concepts exposés ci-dessus ont été saisis, la traduction est très simple. Pour ma part je me sers principalement de ce fichier (et de l'aide du SDK), ainsi que du tableau "Conversion en Visual Basic de déclarations en langage C" du MSDN.

Enfin des exemples

Comme nous voilà farci de théorie, nous allons maintenant parcourir un certain nombre d'exemples, pour regarder de plus près tous ces concepts.
La plus simple des API est une procédure qui attend un paramètre. Elle sert à stopper l'exécution du thread courant pendant n millisecondes

```
Private Declare Sub Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
```

Il est à noter que c'est une procédure et non une fonction ce qui est assez rare.
Dans le code, pour faire une pause de 300 millisecondes :

```
Sleep 30
```

Pour continuer dans les procédures nous allons regarder un cas beaucoup plus intéressant, la procédure GetSystemInfo. Cet exemple va nous permettre de bien saisir le passage par référence.
Dans ma visionneuse je cherche ma fonction, et je trouve :

```
Private Declare Sub GetSystemInfo Lib "kernel32" Alias "GetSystemInfo" (lpSystemInfo As SYSTEM_INFO)
```

Là, je passe donc un paramètre de type SYSTEM_INFO. Comme je ne connais pas ce type, je le cherche dans les types de la visionneuse et j'obtiens :

```
Private Type SYSTEM_INFO
    dwOemID As Long
    dwPageSize As Long
    lpMinimumApplicationAddress As Long
    lpMaximumApplicationAddress As Long
    dwActiveProcessorMask As Long
    dwNumberOrfProcessors As Long
    dwProcessorType As Long
    dwAllocationGranularity As Long
    dwReserved As Long
End Type
```

L'appel dans le code est le suivant :

```
Dim RecupInfo as SYSTEM_INFO
GetSystemInfo RecupInfo
Debug.print RecupInfo. DwProcessorType
```

Comme on le voit, il s'agit d'une procédure, et pourtant maintenant ma variable RecupInfo contient l'ensemble des valeurs désirées. Cela est dû au passage par référence qui permet à la fonction de modifier la variable.
Regardons maintenant le passage d'une chaîne. La fonction GetUserNameA renvoie le nom de l'utilisateur.
Dans la visionneuse on voit qu'il y aussi une fonction GetUserNameW. Le A en fin de chaîne signifie ANSI, le W unicode. Comme VB fait la conversion en ANSI automatiquement, nous utiliserons les fonctions A. Donc,

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
```

Comme nous l'avons vu précédemment, la chaîne est bien passée par valeur, elle va pourtant être modifiée.
La fonction s'utilise ainsi
Dim strTampon As String, Taille As Long, Retour as Long, UserName as String
strTampon = Space$(255)
Taille = 256
Retour= GetUserName(strTampon, Taille)
UserName=Left$(strTampon,Taille-1) ou
UserName = Mid$(strTampon, 1, InStr(1, strTampon, Chr$(0)) - 1)
Comme vous le voyez, j'ai dimensionné ma chaîne au préalable. La variable Retour contiendra 0 si la fonction a échoué). Je vous donne en fin de fonction deux façons de récupérer la partie de la chaîne qui nous intéresse.

Se méfier de la visionneuse

Nous allons prendre un exemple un peu plus compliqué. Je vais utiliser la fonction Polygon qui permet de tracer un polygone sur un objet possédant un **contexte de périphérique** (hdc).
Je vais donc dans la visionneuse API et je récupère :
Private Declare Function Polygon Lib "gdi32" Alias "Polygon" (ByVal hdc As Long, lpPoint As POINTAPI, ByVal nCount As Long) As Long
Je vais aussi récupérer le type POINTAPI.

```
Private Type POINTAPI
        x As Long
        y As Long
End Type
```

Donc si je regarde mon code, je vois que la structure POINTAPI contient les coordonnées d'un point. Mais si je regarde la fonction, je suis en droit de me demander comment on trace un polygone avec un point. Si par curiosité je vais voir dans mon fichier win32.hlp la même fonction, je trouve

```
BOOL Polygon(
   HDC  hdc,       // handle of device context
   CONST POINT  *lpPoints,       // address of polygon's vertices
   int  nCount     // count of polygon's vertices
  );
```

Parameters

Hdc	Identifies the device context.
LpPoints	Points to an array of POINT structures that specify the vertices of the polygon.
NCount	Specifies the number of vertices in the array. This value must be greater than or equal to 2.

Donc la fonction attend un pointeur sur un **tableau** de structure POINTAPI, le nombre d'éléments de ce tableau étant nCount.
Mon code sera alors :
Dim Coords(1 To 5) As POINTAPI, NbPoints As Long
Là je définirai mes coordonnées, je fixerai NbPoints = 5
Et l'appel de la fonction sera
Polygon Me.hdc, Coords (1), NbPoints
Nous noterons que je lui passe Me.hdc qui est le contexte de périphérique de la fenêtre. De même comme nous l'avons vu précédemment, pour passer un tableau en paramètre, je lui passe le premier élément par référence.

Trouver les constantes

Je vais prendre un exemple simple que j'ai déjà donné sur le forum, comment désactiver la croix de fermeture des fenêtres. Comme la croix est un menu système je vais utiliser deux fonctions GetSystemMenu et RemoveMenu.
Private Declare Function GetSystemMenu Lib "user32" Alias "GetSystemMenu" (ByVal hwnd As Long, ByVal bRevert As Long) As Long
Private Declare Function RemoveMenu Lib "user32" Alias "RemoveMenu" (ByVal hMenu As Long, ByVal nPosition As Long, ByVal wFlags As Long) As Long
Ces fonctions n'utilisent pas de type utilisateur. Comme je suis un programmeur prudent je vais voir dans le Win32.hlp. Dans la fonction GetSystemMenu, l'aide me dit (en anglais) :

"The System menu initially contains items with various identifier values, such as SC_CLOSE, SC_MOVE, and SC_SIZE."

Pour la fonction RemoveMenu :

```
BOOL RemoveMenu(
        HMENU  hmenu,           // handle of menu
        UINT  uItem,     // menu item identifier or position
        UINT  fuFlags)   // menu item flag
```

Parameters

Hmenu Identifies the menu to be changed.

UItem Specifies the menu item to be deleted, as determined by the fuFlags parameter.

FuFlags Specifies how the uItem parameter is interpreted. This parameter must be one of the following values:

Value	Meaning
MF_BYCOMMAND	Indicates that uItem gives the identifier of the menu item. If neither the MF_BYCOMMAND nor MF_BYPOSITION flag is specified, the MF_BYCOMMAND flag is the default flag.
MF_BYPOSITION	Indicates that uItem gives the zero-based relative position of the menu item.

Comme je parle couramment anglais, je vois qu'il me faut aller chercher deux commandes SC_CLOSE et MF_BYPOSITON. J'utilise la visionneuse et j'obtiens :

```
Private Const SC_CLOSE = &HF060&
Private Const MF_BYCOMMAND = &H0&
```

Dans le load de ma feuille je n'aurais plus qu'à écrire

```
Dim hSysMenu As Long
hSysMenu = GetSystemMenu(Me.hwnd, False)
RemoveMenu hSysMenu, SC_CLOSE, MF_BYCOMMAND
```

Pointeur NULL

Pour certaines fonctions il arrive qu'il soit nécessaire de passer un pointeur NULL (pointeur qui ne pointe sur rien).

La fonction FindWindow donne le handle d'une fenêtre dont on possède soit le nom de classe soit le titre.

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
```

Dans mon exemple je vais récupérer le handle de la fenêtre Excel.

```
Titre ="Microsoft Excel"
HandleExcel=FindWindow(vbNullString,Titre)
```

J'utilise donc vbNullString qui est un pointeur Null de type chaîne. Pour les autres types de pointeur je passerai le paramètre ByVal 0&. Arrêtons-nous sur ce paramètre. Le caractère & signifie que c'est un long. Mais pourquoi passer ByVal un paramètre déclarer ByRef dans la fonction?

La raison est simple, si je passe comme paramètre par référence 0& la fonction va recevoir un pointeur sur une variable de valeur 0. Or je veux que ce soit l'adresse qui soit nulle. Donc je passe comme paramètre "ByVal 0&". C'est la difficulté de l'utilisation des pointeurs. Si on ne fait pas attention au paramètre, on a vite fait de passer un pointeur de pointeur plutôt qu'un pointeur de variable. Mais je m'éloigne du sujet.

Maintenant que nous maîtrisons les API comme un apache son tomahawk, nous allons aborder un cas un peu plus complexe mais très souvent utilisés.

Enumération (AddressOf)

Pour de nombreuses fonctions d'énumération il faut écrire une fonction dont on passera l'adresse à la fonction API. En C, cela s'appelle un CallBack. Je vais prendre la plus connue de ces fonctions "EnumWindows". Comme j'en ai pris désormais l'habitude je vais voir dans mon fichier Win32.hlp et je trouve :

```
BOOL EnumWindows(
   WNDENUMPROC  lpEnumFunc,          // address of callback function
   LPARAM  lParam        // application-defined value
  );
```

lpEnumFunc Points to an application-defined callback function. For more information, see the EnumWindowsProc callback function.

Je vais donc voir ce qu'il y a dans EnumWindowsProc et je trouve

```
BOOL CALLBACK EnumWindowsProc(
  HWND  hwnd,           // handle of parent window
  LPARAM  lParam        // application-defined value
);
```

Parameters
Hwnd Identifies a top-level window.
LParam Specifies the application-defined value given in EnumWindows.

Return Value
To continue enumeration, the callback function must return TRUE; to stop enumeration, it must return FALSE.

Pour traduire, il me dis que la fonction EnumWindow nécessite une fonction EnumWindowsProc qui attends deux paramètres un handle, et le deuxième paramètre que j'aurais passé à la fonction EnumWindows.
Je vais dans cet exemple créer un jeu de fonctions qui minimise toutes les fenêtres ouvertes.
Je déclare :

```
Private Declare Function EnumWindows Lib "user32" (ByVal lpEnumFunc As Long, ByVal lParam As Long) As Long
Private Declare Function CloseWindow Lib "user32" Alias "CloseWindow" (ByVal hwnd As Long) As Long
```

La fonction CloseWindow minimise la fenêtre dont on passe le handle.

```
Public Sub MiniFenetre()
        Dim Retour as long
        Retour=EnumWindows(AddressOf EnumWindowsProc, 0)
End Sub

Public Function EnumWindowsProc(ByVal lgHwnd As Long, ByVal lgParam As Long) As Long
        Call CloseWindow(lgHwnd)
        EnumWindowsProc = 1
End Function
```

Quelques remarques sont nécessaires.
La fonction dont on passe l'adresse **doit** :

- Etre dans un module (.bas)
- Etre dans le projet ou on déclare la DLL

Enfin bien que je puisse vous garantir que cette fonction fonctionne parfaitement, je vous déconseille de la tester.

Conclusion

Comme nous l'avons vu, la maîtrise des API ne pose pas de véritables problèmes. Elle demande juste de la rigueur, et de faire un effort de conversion du C vers Visual Basic. Je vous recommande encore une fois de télécharger le fichier Win32.hlp, et de regarder le tableau "Conversion en Visual Basic de déclarations en langage C" de l'aide.
Le prochain article abordera une programmation plus poussée des API avec des notions de sur classement, de hook (crochet) et de sous classement.