#### Appel d'API Win32 à partir de VB

Contenido

[Introduction 2](#_Toc214282287)

[Les déclarations d'API 2](#_Toc214282288)

[Les API Win32, qu'est ce que c'est ??? 2](#_Toc214282289)

[Comment déclarer l'utilisation d'une API 2](#_Toc214282290)

[ByRef, ByVal ça sert à quoi exactement ??? 3](#_Toc214282291)

[Les pointeurs en Visual Basic 4](#_Toc214282292)

[Un petit détour sur une API particulière : CopyMemory 5](#_Toc214282293)

[Implémentation de l’indirection avec CopyMemory 6](#_Toc214282294)

[Aparté sur les chaînes de caractères 7](#_Toc214282295)

[La saga des chaînes de caractères en VB 7](#_Toc214282296)

[Conversion des chaînes de caractères par VB 9](#_Toc214282297)

[Comment passer une chaîne de caractères à une API ??? 10](#_Toc214282298)

[Comment recevoir une chaîne de caractères d’une API ??? 11](#_Toc214282299)

[Non supporté mais très utile : VarPtr, StrPtr 13](#_Toc214282300)

[Et dans les autres cas … 14](#_Toc214282301)

[Pour conclure 15](#_Toc214282302)

[Referencia Bibliográfica 16](#_Toc214282303)

# Introduction

Qui parmi nous, développeurs Visual Basic (VB), ne s'est pas cassé les dents sur l'utilisation d'API Win32 (ou autres d'ailleurs). Une API Win32 qui demande un pointeur ce n'est pas rare, mais nous qui travaillons en VB nous ne les utilisons pas (ou plutôt ne savons pas que nous les utilisons). En effet, les pointeurs sont, pour nous, un héritage de cet étrange langage mi-réel, mi-ésotérique qu'est C\C++ et il faut bien avouer que nous considérons souvent, avec un certain snobisme, que VB est au-dessus de tout ça. Pis encore, certaines API nous retournent ces identifiants un peu étranges que l'on appelle les handles et qui ne correspondent à aucun type précis en VB.

Ce document a pour but de mettre en évidence qu'il est tout à fait possible d'utiliser sereinement la quasi-totalité des API Win32 en expliquant tout d'abord les quelques mécanismes de base que VB réalise à notre insu (mais pour notre plus grand bien) lorsque l'on passe un argument à une API.

Par la lecture de ce document, je nourris le secret espoir que le lecteur saura utiliser non seulement une API Win32, mais aussi tout autre API écrite en C\C++ et ce, quelque soit le type de paramètres.

Ce document n'est cependant pas un guide exhaustif d'utilisation des API Win32, car d'une part je ne les connais pas toutes, et d'autre part il existe déjà des outils et documents très complets dans le domaine.

**"Il ne savait pas que c'était impossible alors il l'a fait".**

Anonyme

# Les déclarations d'API

## Les API Win32, qu'est ce que c'est ???

Les API Win32 peuvent être définies comme une ensemble de fonctions qui fournissent un accès "programmatique" aux fonctionnalités du système d'exploitation (SE) Win9x et WinNT/2000/XP. Il n'existe pas que les API Win32 sur ces SE, cependant nous allons essentiellement parler de celles-ci car elles sont les plus courantes et seront la base d'exemples simples et facilement généralisables.

Je sais (et j'ai les noms) que certains se demandent pourquoi il est si complexe d'utiliser les API Win32 depuis VB, alors qu'il est si simple, lorsque l'on développe une DLL ActiveX (en VB) de la réutiliser en faisant simplement une référence à celle-ci (à partir du menu Projet → Références) et en utilisant tout simplement les objets.

Les raisons sont les suivantes : d'une part, ces DLL ActiveX répondent, contrairement aux API Win32, à une norme bien précise, la norme COM (Component Object Model). D'autre part, les API Win32 sont écrites pour la plupart en C\C++, qui, par nature, est un langage très différent de VB et qui met à disposition des types de données qui n'existent pas en VB.

## Comment déclarer l'utilisation d'une API

Le mot clé ***Declare*** est utilisé pour appeler une fonction externe qui réside dans une DLL. Ce mot clé peut être utilisé au niveau du module, du formulaire ou de la classe.

La forme générique d'une déclaration donne quelque chose comme cela :

- Pour les procédures :

[Public | Private] Declare Sub *NomDeLaProcedure* Lib "*NomDeLaLibrairie*" [Alias "*NomAlias*"] [([*ListeArgument*])]

- Pour les fonctions :

[Public | Private] Declare Func NomDeLaFonction Lib "NomDeLaLibrairie" [Alias "NomAlias"] [([ListeArgument])] [as type]

Comme ce n’est pas vraiment compréhensible tel quel :

* Tout d'abord ce qui est entre crochets ( [ ] ) est optionnel, et les barres verticales ( | ) indiquent qu'il faut faire un choix.
* [Public | Private] : si on choisit Public, notre déclaration sera visible dans tout le projet, si l'on choisit Private notre déclaration sera visible uniquement au niveau du module. Pour que l'on n’ait pas de choix cornélien à faire entre Private et Public, il faut savoir que dans le code associé à une forme ou dans un module de classe on ne peut utiliser que Private, par contre dans un module de code standard, on peut utiliser Public ou Private (attention, par défaut si vous ne mettez rien ce sera Public).
* *NomDeLaProcedure, NomDeLaFonction* : soit c'est le nom de la fonction ou de la procédure tel qu'il est exporté par la DLL soit, lorsque l'on utilise le mot clé Alias, c'est un nom différent du nom exporté par la DLL.
* *NomAlias* : c'est le nom de la fonction tel qu'il est exporté par la DLL.
* *ListeArgument* : c'est la liste des arguments qu'accepte la procédure ou la fonction.
* *Type* : c'est le type de l'argument retourné dans le cas d'une fonction.

C'est plus clair maintenant ? Non !!! Et bien il ne faut pas désespérer il y aura des exemples pour éclaircir tout cela.

# ByRef, ByVal ça sert à quoi exactement ???

La définition d'une fonction est quelque chose de très simple, cependant, certains auront remarqué qu'il peut apparaître devant l'argument le mot-clé ByRef, d'autres fois ByVal ou bien encore aucun de ces deux mots. Prenons quelques lignes pour se éclaircir sur ce mystère.

Prenons deux exemples :

- Premier exemple en précisant ByRef dans la déclaration de la fonction :

Private Sub Command1\_Click()

Dim j As Integer

j = 5

Debug.Print "Avant appel de la fonction j : " & j

maFonc j

Debug.Print "Après appel de la fonction j : " & j

End Sub

Private Sub MaFonc (ByRef i as integer)

i = 2

End Sub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Résultat :

Avant appel de la fonction j : 5

Après appel de la fonction j : 2

- Deuxième exemple en précisant ByVal dans la déclaration de la fonction :

Private Sub Command1\_Click()

Dim j As Integer

j = 5

Debug.Print "Avant appel de la fonction j : " & j

maFonc j

Debug.Print "Après appel de la fonction j : " & j

End Sub

Private Sub MaFonc (ByVal i as integer)

i = 2

End Sub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Résultat :

Avant appel de la fonction j : 5

Après appel de la fonction j : 5

Que constate-t'on ? Dans le premier exemple la valeur de la variable j a changé dans la fonction appelante, dans le second cas elle n'a pas changé. La raison en est simple, dans le premier cas on a demandé à VB de passer le paramètre par référence (donc de façon implicite on passe un pointeur, vous voyez c'est bien ce que je vous disais, on travaille avec des pointeurs sans le savoir…) et dans le second cas VB passe le paramètre par valeur, c'est à dire qu'il passe une copie de la valeur et non plus un pointeur sur celle-ci.

Et lorsque l'on ne met aucun mot clé devant l'argument me direz-vous, et bien par défaut VB travaille par référence.

**A RETENIR** :

**Dans tous les cas, lorsque l'on passe un argument par référence (en utilisant ByRef), on passe l'adresse de celui-ci.**

**Dans la plupart des cas, lorsque l'on passe un argument par valeur (en utilisant ByVal), on passe une copie de la valeur**

# Les pointeurs en Visual Basic

Imaginons (nous allons enfin prouver au monde qu’un informaticien a beaucoup d’imagination) une fonction appelée monAPI, exportée à partir de la DLL mesAPIs.dll dont l'implémentation en C serait la suivante :

void monAPI (short \*pVar)

{

/\* on travaille \*/

}

La question est : "Comment la déclarer en VB ?"

Nous savons maintenant que VB passe par défaut l'argument par référence (cf chapitre ci-dessus), donc un pointeur.

Un exemple de déclaration dans VB est donc le suivant :

Public Declare Sub APIVb Lib "c:\CustomAPI\mesAPIs.dll" Alias monAPI (var as integer)

Cette déclaration est strictement équivalente à la déclaration ci-dessous (qui est cependant plus explicite, car on précise le ByRef) :

Public Declare Sub APIVb Lib "c:\CustomAPI\mesAPIs.dll" Alias monAPI (ByRef var as integer)

Avec une telle déclaration on demande à VB de passer un pointeur (donc l’adresse) sur un entier, et c'est exactement ce qu'attend la fonction en C.

L'utilisation de cette déclaration dans VB est alors la suivante :

Sub maProc()

Dim j as integer

APIVb j

End Sub

Tout cela est beaucoup trop simple, donc on va compliquer un petit peu. Imaginons maintenant que l'on ait déjà dans notre code VB l'adresse sur l'entier (par exemple dans le cas ou une autre API nous a déjà retourné un pointeur sur un entier).

Nous devons alors adapter notre déclaration, sinon on ne va pas envoyer un pointeur sur un entier, mais un pointeur sur un pointeur (et là les problèmes vont commencer).

La nouvelle déclaration devient alors :

Public Declare Sub APIVb Lib "c:\CustomAPI\mesAPIs.dll" Alias monAPI (ByVal var as long)

On utilisera alors cette déclaration de la façon suivante :

Sub maProc()

Dim j as integer

Dim ptr as long

ptr = varPtr (j) ‘cf varPtr dans le chapitre Non supporté mais très

‘utile : VarPtr, StrPtr

APIVb ptr

End Sub

Dans ce cas, en précisant le mot clé ByVal devant le paramètre, on demande à VB de passer une copie de la variable, attendu que la variable est déjà un pointeur, l'API va donc bien recevoir un pointeur.

Remarque : un pointeur dans l'environnement de travail Win32 est toujours représenté sur 32 bits, ce qui explique qu’un pointeur sera toujours déclaré en tant que long.

# Un petit détour sur une API particulière : CopyMemory

La vie de CopyMemory n’a pas été une vie de tout repos, cette fonction a changé de nom au cours du temps, et elle n’est actuellement qu’un alias de la fonction RltMoveMemory. Si vous voulez en savoir un peu plus sur la vie de cette fonction, vous pouvez lire l’article : <http://support.microsoft.com/support/kb/articles/q129/9/47.asp>.

Cette fonction est très simple, elle copie le contenu d’une zone mémoire (dont on donne l’adresse source), dans une autre zone mémoire (dont on donne l’adresse de destination). Toute la subtilité de cette fonction réside dans l’utilisation de byval et de byref.

Tout d’abord regardons l’une des déclarations de CopyMemory avec un petit exemple d’utilisation :

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (ByRef pDst As Any, ByRef pSrc As Any, ByVal ByteLen As Long)

Sub maProc()

Dim lng as Long

Dim i as Integer

Dim bArray (1 To 4) as Byte

lng = &H4030201 'Attention, ce n'est pas une adresse

CopyMemory bArray(1), lng, 4

For i = 1 To 4

Debug.Print Hex(bArray(i))

Next

End Sub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Résultat :

1 2 3 4

Ce programme est très simple, les étapes principales sont les suivantes :

Allocation d’un tableau de Byte (bArray)

* Initialisation de la variable lng
* Utilisation de CopyMemory pour copier le contenu de lng dans le tableau bArray
* Affichage du tableau dans la fenêtre de déboggage

Attendu que l’on passe bArray et lng on passe donc des pointeurs (souvenez-vous c’est tout l’objet du chapitre "ByVal, ByRef à quoi ça sert ???"). Ensuite la fonction CopyMemory copie le contenu de l’adresse source dans l’adresse destination.

Cependant attention, comme toutes les API, une mauvaise utilisation de CopyMemory peut générer une faute générale de protection.

## Implémentation de l’indirection avec CopyMemory

Imaginons qu’une API quelconque (Win32 ou autre) nous retourne, dans notre code VB, un pointeur sur un long. En C, c’est facile d’avoir la valeur pointée par le pointeur, on utilise tout simplement l’opérateur d’indirection \*. En Visual Basic c’est un peu plus compliqué car il n’y a pas de tels opérateurs, mais heureusement CopyMemory est là pour nous aider.

Là encore le petit exemple va nous permettre de tout comprendre :

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (pDst As Any, pSrc As Any, ByVal ByteLen As Long)

Sub maProc()

Dim Pointeur as Long

Dim Target as Integer

Dim i as Integer

i = 5

‘Supposons que RetourneAdresse() est une mystérieuse API qui retourne

‘un pointeur. Par exemple dans ce cas retourne l’adresse de i.

Pointeur = RetourneAdresse (i)

CopyMemory Target, ByVal Pointeur, lenB (Target)

Debug.Print Target

End Sub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Résultat :

5

« Target » est passé par référence (rappelez vous que si l’on ne précise rien, par défaut on passe les paramètres par référence), la fonction reçoit donc l’adresse de la variable destination. Puisque « Pointeur » est passé par valeur, la fonction reçoit le contenu de « Pointeur » qui contient l’adresse de la mémoire source (ici c’est l’adresse de i).

Donc on copie ce qui est pointé par Pointeur, dans Target et enfin on affiche Target.

On a bien réalisé une indirection, puisqu’on a récupéré la valeur de ce qui était pointé par la variable « Pointeur » CQFD.

# Aparté sur les chaînes de caractères

Certains se demandent pourquoi il faut parler des chaînes de caractères dans un document qui présente l’appel des API Win32 depuis Visual Basic. Et bien faites moi confiance, il le faut, sinon je vous assure que je m’en serais passé.

## La saga des chaînes de caractères en VB

Tout le monde connaît le type de données *« String »* en VB, mais peu de gens savent précisément à quoi il correspond. Il pourrait être défini comme une séquence fixe de caractères, c’est pas mal comme définition, et c’est générique comme ça on a peu de chance de se tromper, le problème c’est qu’avec VB, ce n’est pas du tout ça. Il y a effectivement une suite de caractères, mais l’analogie s’arrête là.

Nous allons tenter ici de définir ce qu’est ce type de données en VB, type plus formellement connu sous le nom de Basic STRing (BSTR), d’ailleurs pour la suite nous parlerons de BSTR et non plus de type de données *« String ».*

Fort de l’adage « Un petit schéma vaut mieux qu’un long discours », voici d’abord le schéma (avec un petit exemple de code de haut niveau, on est quand même des développeurs…) :

Sub maProc()

Dim maChaine as string

maChaine = "AIDE"

End Sub

A

I

D

E

8

*Pointeur*

\0

xxxx

maChaine =

Figure 1

Et ensuite le discours …

Comme nous le voyons sur le schéma :

* La BSTR est donc un pointeur (comme tous les pointeurs Win32 il a une taille de 32 bits) sur le **premier** caractère d’un tableau de caractères codés en **unicode**. Qui dit unicode dit donc 2 octets par caractère.
* Le tableau de caractères unicode sur lequel pointe la BSTR est terminé par un champ de 2 octets contenant le caractère NULL (ANSI = UNICODE = 0).
* Le tableau de caractères unicode sur lequel pointe la BSTR est précédé par un champ de 4 octets contenant la taille en octets (et non en caractères car n’oublions pas qu’un caractère unicode fait 2 octets) du tableau SANS prendre en compte le champ terminal contenant le caractère NULL .
* Le caractère NULL peut apparaître dans le tableau de caractères ailleurs qu’en fin de chaîne, c’est pour cela que le premier champ indiquant la taille de la chaîne est particulièrement important.

**A RETENIR, le petit résumé suivant :**

**Un BSTR est un pointeur sur le PREMIER CARACTERE d’un tableau de caractères unicode. Tableau,**

* **dont le DERNIER caractère est le caractère NULL codé en unicode (je le rappelle pour la dernière fois, ça fait donc 2 octets)**
* **précédé par 4 octets représentant la taille du tableau en octets sans tenir compte du caractère NULL de fin de chaîne.**

## Conversion des chaînes de caractères par VB

Maintenant je vais vous raconter une histoire très étrange : comment VB passe un BSTR à une fonction dans une DLL externe (typiquement une API Win32).

Ce n’est plus un secret pour nous, VB utilise en interne unicode pour le codage des chaînes de caractères. Windows NT et 2000/XP font de même, et la totalité des API Win32 sur ces SE supportent unicode, donc jusque là pas de problème. Par contre pour Windows 9x la plupart des API ne connaissent même pas unicode.

Pour pallier à cela, VB convertit tout simplement le tableau de caractères unicode en un (autre) tableau de caractères ANSI et passe alors le pointeur sur ce nouveau tableau AINSI. VB réalise cette opération de façon systématique, que le système d’exploitation soit Win9x ou Win NT/2000/XP.

Pour illustrer cela prenons un exemple très simple :

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (pDst As Any, pSrc As Any, ByVal ByteLen As Long)

Dim maChaine as String

Dim monTab(19) as Byte

Dim i as Integer

Sub maProc()

maChaine = "AIDE"

CopyMemory monTab(0), ByVal maChaine, lenB (maChaine)

For i = 0 to 19

Debug.Print monTab(i)

Next

End Sub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sortie du Debug.Print

65 73 68 69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Prenons un deuxième exemple :

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (pDst As Any, pSrc As Any, ByVal ByteLen As Long)

Dim maChaine as String

Dim monTab(1 To 20) as Byte

Dim i as integer

Sub maProc()

Dim lg as long

maChaine = "AIDE"

lg = StrPtr (maChaine)

CopyMemory monTab(1), lg, lenB (maChaine)

For i = 1 to 20

Debug.Print monTab(i)

Next

End Sub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sortie du Debug.Print

65 00 73 00 68 00 69 00 00 00 00 00 00 00 00 00 00 00 00

Ces deux exemples, même s’ils ne sont pas follement excitants, sont très intéressants. En effet, on constate que le fait de passer le BSTR byVal (dans le premier exemple) fait que l’on copie dans notre tableau le code ANSI de chacun des caractères (donc codé sur un seul octet). Cela confirme bien ce qui était dit auparavant, à savoir que lorsque l’on demande à VB de passer une chaîne de caractères à une API, il la convertit en ANSI.

Dans le deuxième exemple c’est moi (ou plutôt mon code) qui récupère explicitement la valeur du BSTR et précise cette valeur dans CopyMemory, donc VB ne sait pas que cette valeur est en fait un BSTR et ne fait pas la conversion en ANSI. Le résultat : j’ai bien un codage en unicode (sur 2 octets. Par exemple pour A : 65 00). Précisons que la plupart des APIs Win32 sous Windows NT/2000/XP ont un version ANSI et un version UNICODE (Pour ce qui est de Win9x, il n'y a la plupart du temps que les API ANSI).

Donc, attention, vous travaillez avec VB en unicode, mais dès que vous passez un BSTR à une API, VB qui est très sympa avec nous (!!!), la convertit en ANSI (dans un tableau temporaire intermédiaire) avant de passer le pointeur sur cet autre tableau ANSI au système d’exploitation.

*PS. : Vous vous demandez certainement ce que fait cette mystérieuse fonction StrPtr. C’est une fonction qui n’est ni supportée, ni documentée, mais si vraiment vous en avez besoin, j’ai fait un chapitre spécial en fin de document sur cette fonction (chapitre : « Non supporté mais très utile : VarPtr, StrPtr »).*

## Comment passer une chaîne de caractères à une API ???

Après ce petit aparté (fort utile) sur les chaînes de caractères en VB, nous allons regarder comment passer une chaîne de caractères à une API depuis VB. Il est vrai que nous avons déjà largement traité cet aspect dans le chapitre « Conversion des chaînes de caractères par VB », mais nous allons donner ici quelques précisions qui me semblent importantes.

Certains auteurs affirment que ByVal se comporte différemment pour les BSTR (en fait ces auteurs parlent plutôt de chaîne de caractères entretenant ainsi un certaine ambiguité) et qu’il fait implicitement pour vous un passage par référence. Et bien je le dis haut et fort je ne suis pas d’accord.

Prenons un exemple :

ByVal str as String

Cette déclaration demande à VB de passer le contenu de la BSTR, qui est un pointeur sur le tableau de caractères. C’est exactement ce que fait ByVal, il passe le contenu de la BSTR mais comme ce contenu est un pointeur, on simule un passage par référence. ByVal reste donc fidèle à lui-même, car de façon générique ByVal passe le contenu d’une variable, et c’est exactement ce qu’il fait ici.

Par contre, si pour passer un pointeur sur un tableau de chaînes de caractères on utilise une déclaration telle que :

ByRef str as String

Alors là c’est que l’on a rien compris (alors on prend son courage à 2 mains et on relit le chapitre sur la saga des chaînes de caractères), puisque si l'on utilise cette déclaration on passe le BSTR par référence, c’est-à-dire que l’on va passer un pointeur sur ce BSTR. Conclusion, l’API ne va plus recevoir un pointeur sur un tableau de caractères, mais un pointeur de pointeur sur un tableau de caractères, et là, rien ne va plus…

## Comment recevoir une chaîne de caractères d’une API ???

Maintenant nous savons comment envoyer une BSTR à une API depuis VB, étudions le cas contraire, à savoir comment recevoir une chaîne de caractères à partir d’une API.

Pour que notre étude soit concrète, nous allons utiliser l’API GetWindowText dont la déclaration en VB est la suivante :

Declare Function GetWindowText Lib "USER32" Alias "GetWindowTextA” (\_

ByVal hWnd as Long, ByVal lpString as String, ByVal nMaxCount as Long) as Long

Cette API retourne dans lpString, le titre d’une fenêtre dont on passe l’identifiant (dans ce cas c’est un handle) par l’intermédiaire du paramètre hWnd.

Un exemple d’utilisation de cette API est le suivant :

Dim sTemp as String, sTitre as String, i as Integer

sTemp = String$ (4, "\*")

‘Généralement on utilisera String$(4,0), mais ici on utilise "\*" (car le

‘schéma qui suit sera plus clair)

c = GetWindowText (hProject, sTemp, 5)

‘hProject est un handle de fenêtre quelconque

sTitre = Left$ (sTemp, c)

Pour comprendre ce qu’il se passe voici un schema :

\*

\*

\*

\*

8

*Pointeur*

\0

xxxx

sTemp =

**1 – Initialisation de la BSTR :**

sTemp = String$(4, "\*")

**2 – VB passe une copie de la BSTR au format ANSI à la function GetWindowText :**

c = GetWindowText (hProject, sTemp, 5)

\*

\*

\*

\*

8

*Pointeur*

\0

xxxx

sTemp =

yyyy

Copie

\0

\*

\*

\*

\*

\*

\*

\*

\*

8

*Pointeur*

\0

xxxx

sTemp =

yyyy

Copie

\0

\*

\0

B

A

**3 – Windows écrit dans le tableau de caractères au format ANSI et retourne dans c la longueur réelle :**

c = GetWindowText (hProject, sTemp, 5)

A

B

\0

\*

8

*Pointeur*

\0

xxxx

sTemp =

**4 – VB convertit le tableau de caractères ANSI en Unicode et le copie dans le tableau pointé par sTemp :**

**5 – On redimensionne le tableau de caractères :**

sTitre = Left$ (sTemp, c)

A

B

4

*Pointeur*

zzzz

sTitre =

\0

# Non supporté mais très utile : VarPtr, StrPtr

Je ne vais pas rentrer dans le détail des fonctions qui retournent des pointeurs en VB car d’une part ce n’est pas supporté et normalement non documenté. Mais bon, il y a quand même un article très intéressant sur le sujet et je vous engage à aller y jeter un œil : <http://support.microsoft.com/support/kb/articles/q199/8/24.asp>.

VarPtr vous renvoie un pointeur, par exemple :

Dim i as Integer

Dim ptr as Long

ptr = VarPtr (i)

ptr contient l’adresse de i (pour ceux qui font du C c’est equivalent à &i).

StrPtr retourne un pointeur (c’est original, me direz-vous, mais VarPtr retourne aussi pointeur), mais StrPtr est utilisé uniquement sur les BSTR, et il retourne la valeur du BSTR.

Concrètement :

\0

sTitre =

zzzz

*Pointeur*

4

B

A

A

B

4

*Pointeur*

zzzz

sTitre =

\0

Si l’on prend StrPtr(sTitre) on obtient alors zzzz.

Attention, si l’on fait l’erreur d’utiliser VarPtr sur un chaîne de caractères, on retourne alors un pointeur sur le BSTR, et non pas comme attendu un pointeur sur le premier élément du tableau. On a alors un pointeur de pointeur…

# Et dans les autres cas …

Et les autres cas me direz vous ? C’est-à-dire, les cas où l’on ne travaille ni avec une chaîne de caractères, ni avec un Integer, ni avec un Long …

Il n’était pas possible, dans ce document d’aborder tous les cas, mais nous avons vu l’essentiel, et surtout nous avons acquis une certaine philosophie des passages de paramètres aux API Win32.

Cependant, dans le souci de ne pas vous laisser comme ça tout bête devant votre API Win32 (ou autre), voici un tableau qui fait une synthèse du type à utiliser en VB pour un type C défini.

|  |  |  |
| --- | --- | --- |
| Type C | Taille en Octet | Type VB |
| char | 1 | Byte |
| signed char | 1 | Byte |
| unsigned char | 1 | Byte |
| BYTE | 1 | Byte |
| wchar\_t | 2 | Integer |
| TCHAR | 1 ou 2 | Byte / Integer Depend si la DLL est compilée en UNICODE ou ANSI. Pour plus d’informations : <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mapi/html/_mapi1book_c_type_tchar.asp> |
| (signed) short | 2 | Integer |
| unsigned short | 2 | Integer |
| (signed) int | 4 | Long |
| unsigned int | 4 | Long |
| unsigned long | 4 | Long |
| float | 4 | Long |
| double | 8 | Double |
| long double | 8 | Double |
| bool | 1 | Byte |
| BOOL | 4 | Long |
| boolean | 4 | Long |
| \* (représente un pointeur) | 4 | Long |
| HANDLE | 4 | Long |
| FARPROC | 4 | Long |
| LONGLONG | 8 | Double |
| VARIANT | 16 | Variant |

J’ai repris dans ce tableau les types les plus génériques, mais attention, il ne faut pas tomber dans dans le piège du « typedef » (apparemment c’est une des fonctions préférées du développeur C) qui n’est qu’un alias sur un type de base.

Par exemple si on prend HANDLE, il est défini dans winnt.h comme :

typedef void \*HANDLE;

Un HANDLE est un pointeur, et à ce titre, il sera représenté en VB par un Long. J’ai cependant souhaité le préciser dans le tableau, car il est tellement courant qu’il est toujours pratique de l’avoir sous la main.

En règle générale tous les pseudo-types en majuscule sont issus d’un typedef, donc il faut regarder le type de base pour savoir comment les déclarer en VB.

Pour la forme, on va prendre un autre exemple UINT qui n’est pas dans le tableau :

typedef unsigned int UINT;

Si l’on se réfère au tableau, unsigned int est représenté en VB par un Long. Toute la problématique est d’obtenir cette information, c’est documenté dans MSDN ou autodocumenté dans le cas des API Win32 car ceci est défini dans les fichiers.h.

# Pour conclure

J’espère que le lecteur avide de connaissances que vous êtes, aura trouvé dans ce document matière à assouvir sa soif de savoir. Je pense que le lecteur sera en mesure, après avoir lu ce document, de comprendre certaines problématiques liées au passage de paramètres à une API, grâce à une meilleure connaissance du mécanisme utilisé par VB, notamment, lors du passage d’une chaîne de caractères (en fait un BSTR, mais il faut recommencer à parler comme tout le monde). Et si vraiment vous avez un doute sur le type à utiliser en VB dans le cas des API Win32, il y a toujours l’API Viewer fourni avec VB ou encore l’excellent site <http://www.allapi.net/> qui est certainement le meilleur dans le domaine et qui propose des exemples de code pour de très nombreuses API.

**"Aime la vérité, mais pardonne l’erreur".**

Voltaire

# Referencia Bibliográfica

Documento Obtenido de la red el día 12 de Noviembre de 2008

<download.microsoft.com/download/5/f/8/5f862469-98d3-4936-a9d4-a7f83d779493/apiwin32.doc>

Joaquín Medina Serrano mailto:joaquin@medina.name