



## FICHE TECHNIQUE N° V

### - Chapitre : CONCEPT D'OPTIMISATION –

## SECURISATION DES DONNEES

En Visual Basic, plusieurs techniques permettent de sécuriser l'entrée des données. Le but étant de sécuriser la saisie des informations. Ces différentes techniques sont :

- L'utilisation de certains événements,
- L'utilisation de certaines propriétés des contrôles,
- L'utilisation de l'InputBox,
- L'utilisation des fonctions intégrées,

### Utilisation de certains événements

Certains événements peuvent permettre de mettre au point un modèle de sécurité.

**Change** : les instructions qui sont écrites dans cet événement sont exécutées lorsque l'on modifie le contenu d'un contrôle TextBox, ou ComboBox.

Exemple : On a deux TextBox : "PrenomTxt" et "NomTxt". La TextBox "PrenomTxt" n'est pas accessible tant que la TextBox "NomTxt" n'a pas été renseignée. Lors du développement, on affecte donc la valeur False à la propriété "Enabled" du contrôle "PrenomTxt" dans la fenêtre des propriétés. Le contrôle "PrenomTxt" est donc inaccessible par défaut lors de l'exécution du programme. On place donc l'instruction "**PrenomTxt.enabled = True**" dans l'événement "**change**" du contrôle NomTxt. De cette manière lorsque l'utilisateur écrit quelque chose dans le contrôle "NomTxt", il rend le contrôle "PrenomTxt" accessible.

**KeyPress** : les instructions qui sont écrites dans cet événement sont exécutées lorsque l'on presse une touche qui est définie par son code ascii. La procédure est la suivante :

**Private Sub** NomContrôle\_KeyPress(**KeyAscii As Integer**)

NomContrôle est le nom du contrôle auquel est affecté l'événement

Il faut procéder au renseignement du paramètre KeyAscii qui est le code ascii de la touche opérant le déclenchement.

Exemple : Touche "entrée" = 13

L'instruction "KeyAscii = 13" doit donc être placée dans la procédure NomContrôle\_KeyPress(**KeyAscii As Integer**). Cela doit être la première instruction de la procédure.

**KeyUp** : les instructions qui sont écrites dans cet événement sont exécutées lorsque l'utilisateur relâche une touche.



**KeyDown** : les instructions qui sont écrites dans cet événement sont exécutées lorsque l'utilisateur enfonce une touche.

## Utilisation de certaines propriétés des contrôles

### ***Propriétés du contrôle "TextBox" :***

Le contrôle TextBox est le plus utilisé pour la saisie des informations. C'est pour cela qu'il dispose de certaines propriétés permettant de sécuriser les entrées.

**Enabled** : Propriété qui accepte les valeurs "True" (vrai) ou "False" (faux). Détermine si le contrôle est accessible (.Enabled = True) ou non (.Enabled = False). Si il n'est pas accessible, l'utilisateur ne peut rien saisir dans la zone de texte, cette dernière est grisée sur l'interface graphique.

Exemple : On a deux TextBox : "PrenomTxt" et "NomTxt". La TextBox "PrenomTxt" n'est pas accessible tant que la TextBox "NomTxt" n'a pas été renseignée. Lors du développement, on affecte donc la valeur "False" à la propriété "Enabled" du contrôle "PrenomTxt" dans la fenêtre des propriétés. Le contrôle "PrenomTxt" est donc inaccessible par défaut lors de l'exécution du programme. On place donc l'instruction "**PrenomTxt.enabled = True**" dans l'événement "**change**" du contrôle NomTxt. De cette manière lorsque l'utilisateur écrit quelque chose dans le contrôle "NomTxt", il rend le contrôle "PrenomTxt" accessible.

**Locked** : Propriété qui accepte les valeurs "True" (vrai) ou "False" (faux). Détermine si le contrôle est verrouillé (.Enabled = True) ou non (.Enabled = False). S'il est verrouillé, l'utilisateur peut accéder à la TextBox, mais il ne peut pas modifier son contenu, il peut juste déplacer son curseur. Le contrôle n'est pas grisé.

**Maxlength** : Nombre de lettres maximales pouvant être saisies dans la TextBox. Utile par exemple pour sécuriser la saisie d'une année (AnneeTxt.Maxlength = 4).

**PasswordChar** : Propriété qui permet de spécifier quel caractère est affiché dans la TextBox à la place de chaque caractère tapé. Cependant cette propriété ne concerne que l'affichage, la valeur tapée est conservée.

Exemple : On a une TextBox "PasswordTxt" qui permet de saisir un mot de passe. Dans la propriété "PasswordChar" de la TextBox, on tape une étoile (\*). Lorsque l'utilisateur saisira son mot de passe, par exemple "babar", il apparaîtra "\*\*\*\*\*". Cependant, si après on demande d'afficher ce qui a été saisi, c'est le mot de passe "babar" qui sera affiché et non les étoiles.

**Visible** : Propriété qui accepte les valeurs "True" (vrai) ou "False" (faux). Détermine si le contrôle est visible (.Enabled = True) ou non (.Enabled = False). S'il n'est pas visible, l'utilisateur ne le voit pas sur l'interface graphique.

Exemple : On a deux TextBox : "PrenomTxt" et "NomTxt". La TextBox "PrenomTxt" n'est pas visible tant que la TextBox "NomTxt" n'a pas été renseignée. Lors du développement, on affecte donc la valeur "False" à la propriété "Visible" du contrôle "PrenomTxt" dans la fenêtre des propriétés. Le contrôle "PrenomTxt" est donc invisible par défaut lors de l'exécution du programme. On place donc l'instruction "**PrenomTxt.visible = True**" dans l'événement "**change**" du contrôle NomTxt. De cette manière lorsque l'utilisateur écrit quelque chose dans le contrôle "NomTxt", il rend le contrôle "PrenomTxt" visible.

Remarques : toutes ces propriétés sont modifiables dans le code ou dans la fenêtre des propriétés.

### ***Propriétés du contrôle "ComboBox" :***

Les propriétés Enabled, Locked, et Visible expliquées ci dessus sont aussi disponibles avec le contrôle ComboBox.

Remarque : la liste d'un contrôle ComboBox ou ListBox est saisissable en mode création à l'aide de la fenêtre des propriétés (on remplit la propriété List) ou en mode exécution à l'aide de la méthode "Additem" qui permet d'ajouter un élément à la liste.

Exemple :

```
MaCombo.Additem "Bonjour"      'Ajoute Bonjour à la liste  
MaCombo est le nom du contrôle
```

**Style** : permet de définir le style de liste modifiable. Les différentes valeurs sont :

DropDown : L'utilisateur peut choisir une valeur dans la liste ou alors en saisir une qui n'est pas dedans par défaut.

DropDown List : L'utilisateur peut choisir, ou taper, une valeur qui est contenu dans la liste. Les valeurs étrangères à la liste ne sont pas acceptées.

Simple List : Comme pour la DropDown, l'utilisateur peut choisir une valeur dans la liste ou alors en saisir une qui n'est pas dedans par défaut. La seule différence réside dans le fait que la liste n'est pas déroulante, si on veut qu'elle soit affichée, il faut agrandir le contrôle sur l'interface graphique en mode création.

Remarques : toutes ces propriétés sont modifiables dans le code ou dans la fenêtre des propriétés.

### ***Propriétés du contrôle "ListBox" :***

Les propriétés Enabled et Visible expliquées ci dessus sont aussi disponibles avec la ListBox.

Remarque : la liste d'un contrôle ComboBox ou ListBox est saisissable en mode création à l'aide de la fenêtre des propriétés (on remplit la propriété List) ou en mode exécution à l'aide de la méthode "Additem" qui permet d'ajouter un élément à la liste.

Exemple :

```
Maliste.Additem "Bonjour"      'Ajoute Bonjour à la liste  
MaListe est le nom du contrôle
```

**Multiselect** : Permet de définir si on peut sélectionner plusieurs éléments ou non dans la liste. Les valeurs possibles sont :

None : multi-sélection non acceptée,

Simple : multi-sélection acceptée, en cliquant avec la souris, on sélectionne ou désélectionne un élément

Extended : appuyer sur MAJ et cliquer la souris de l'élément précédemment sélectionné jusqu'à l'élément en cours. L'inconvénient de ce type de sélection est que tous les éléments à sélectionner doivent être placés l'un en dessous de l'autre dans la liste.





## LA GESTION D'ERREURS

Lors de l'exécution d'un programme Visual Basic, il est possible qu'un certain nombre d'erreurs surviennent : ce sont les erreurs dites d'exécution. Dans un souci d'optimiser les programmes, il est indispensable de mettre au point une gestion d'erreur afin d'expliquer à l'utilisateur pourquoi le programme ne peut se poursuivre, ou mieux encore, faire en sorte que le programme contourne tout seul l'erreur et poursuive son exécution. Pour cela, Visual Basic propose un certain nombre d'instructions visant à exploiter les erreurs retournées par le compilateur.

### L'objet "Err"

"Err" est un objet Visual Basic permet de récupérer le code erreur ainsi que la description d'une erreur qui vient de se produire. La récupération du code erreur s'effectue grâce à la propriété (".Number") et la récupération de la description de l'erreur s'effectue grâce à la propriété (".Description"). Il est intéressant d'afficher ces deux informations dans des MsgBox afin que l'utilisateur puisse en prendre connaissance. Pour plus d'informations sur l'utilisation des MsgBox, consultez la rubrique "utilisation des fonctions graphiques".

Exemple :    MsgBox Err.Number    '**Affichage du code erreur dans une boîte de message**  
MsgBox Err.Description '**Affichage de la description de l'erreur dans une boîte de message**

Remarque : l'objet Err renvoie les informations de la dernière erreur produite, si il y a plusieurs erreurs successives dans le programme, il faut stocker les informations relatives à l'erreur (comme le numéro et la description) dans une variable de type tableau ou les afficher au fur et à mesure qu'elles surviennent.

### Les instructions de gestion d'erreurs

Il existe deux instructions différentes pour gérer les erreurs retournées :

- On Error Resume Next
- On Error Goto "NomEtiquette"

L'instruction "**On Error Resume Next**" permet de spécifier que si le programme rencontre une erreur, il faut sauter l'instruction causant l'erreur et poursuivre à la ligne suivante.

Exemple : On demande l'ouverture d'un programme exécutable qui n'existe pas.

On Error Resume Next                   '**En cas d'erreur, on saute l'instruction**  
Shell ("G:\\_Pjt.exe")               '**Demande l'exécution d'un programme inexistant**  
MsgBox ("Fin du programme")       '**Affichage d'un message**

Remarque : Dans ce cas l'erreur est ignorée et on affiche quand même le message. Ce code n'est donc pas optimisé car même si le programme n'est pas lancé, le message est tous de même affiché.

En effet, si une instruction nécessite que l'instruction précédente ai été correctement exécutée, le "On Error Resume Next" ne sert à rien car le programme ne pourra pas fonctionner



correctement. Exemple : si une base de données ne peut être ouverte parce qu'elle est inexistante, on ne pourra pas non plus ouvrir une table (ou Recordset) à la ligne suivante. Les erreurs ne seront pas indiquées mais le programme ne fera plus rien. Dans ce cas, il faut prévoir une suite d'instructions permettant de contourner l'erreur, par exemple, on ouvre une autre base de données et une autre table.

Cela est possible grâce à l'instruction "**On Error Goto "NomEtiquette"**" où NomEtiquette est le nom de l'étiquette créée. On écrit une série d'instructions qui seront exécutées en cas d'erreur pour résoudre cette erreur. On choisit un nom d'étiquette (qui est en sorte une balise) que l'on positionnera devant les instructions à exécuter en cas d'erreur. Dès que le programme rencontrera une erreur, il exécutera les instructions positionnées immédiatement derrière l'étiquette indiquée par la commande "**On Error Goto**". Attention, l'inconvénient de l'étiquette est qu'elle est quand même exécutée si il n'y a pas d'erreur, car le compilateur ne la saute pas automatiquement. Il faut donc utiliser l'instruction "Exit Sub" pour sortir de la procédure si il n'y a pas d'erreurs.

Exemple : On lance un programme. Une fois lancé, on affiche "programme lancé" puis on sort de la procédure grâce à l'instruction "Exit Sub". Sinon, le programme se poursuit et traite également les instructions derrière l'étiquette. En cas d'erreur, par exemple si le programme n'existe pas, une erreur est retournée. Là le programme se rend à l'étiquette "Message" et affiche "Programme non lancé".

Private Sub Exemple

```
On Error GoTo Message      'En cas d'erreur, on exécute les instructions placées derrière _
                             l'étiquette "Message"
Shell ("C:\Windows\calc.exe") 'Demande l'exécution d'un programme inexistant
MsgBox ("Programme lancé") 'Affichage du message
Exit Sub                    'Sortie anticipée de la procédure pour ne pas exécuter les
                             instructions _
                             relatives au traitement de l'erreur
Message:                    'Suite d'instructions à exécuter en cas d'erreur
MsgBox ("Programme non lancé") 'Affichage du message
End Sub
```

Remarques : Ici le traitement de l'erreur est optimal car dans le cas où il y a une erreur, on exécute un code source adapté. De plus, ce code source adapté est ignoré dans les conditions normales d'utilisation.

## UTILISATION DE MICROSOFT WORD DANS VISUAL BASIC

Visual Basic dispose d'un certain nombre de bibliothèques permettant d'utiliser la plupart des logiciels Microsoft. Parmi ces logiciels, on peut citer Microsoft Word. Il faut donc ajouter la bibliothèque correspondante. Pour cela, cliquez sur "Références" dans le menu contextuel "Projet". Cochez la bibliothèque d'objets "Microsoft Word 9.0 Object Library".

### Création d'un document Word

```
Dim MonWord As New Word.Application      'Déclaration et ouverture d'un objet Word que  
l'on nomme "MonWord"  
MonWord.Visible = True                  'Affichage de l'objet Word  
MonWord.Documents.Add                   'Création d'une nouvelle feuille Word
```

### Ouverture d'un document Word

On ouvre un document à l'aide de la méthode "Open" de la propriété "Documents" de l'objet Word.

```
MonWord.Documents.Open ("C:\MonDocument.doc")  'Ouverture du document Word  
spécifié  
MonWord.Visible = True                        'Affichage de l'objet Word
```

### Masquer l'objet Word

```
MonWord.Visible = False                      'Masquer l'objet Word
```

### Ecrire dans un document Word

```
MonWord.Selection = "Bonjour toi"           'Inscription d'un texte dans le document Word
```

### La propriété "ActiveDocument"

La propriété "ActiveDocument" fait référence au document actif dans Microsoft Word. Lorsque plusieurs documents sont ouverts dans Microsoft Word, le document actif est celui qui a le focus.

### Enregistrer un document Word

```
On peut enregistrer le document actif de l'objet Word grâce à la méthode "Save"  
MonWord.ActiveDocument.Save                'Enregistrer le document actif
```



### "Enregistrer Sous" un document Word

On peut aussi enregistrer le document actif de l'objet Word grâce à la méthode "Save as" `MonWord.ActiveDocument.SaveAs ("C:\MonDocument.doc")` 'Enregistrer le document actif sous le chemin \_et le nom "C:\MonDocument.doc"

### Fermeture de l'objet Word

`MonWord.Quit` 'Fermeture de l'objet Word

### Utilisation du correcteur d'orthographe Word

Dans la section "Notion de Client/Serveur", on a fait référence à l'utilisation de Microsoft Word comme serveur d'application. Exemple : Un utilisateur saisit une chaîne de caractères dans une interface graphique Visual Basic. On veut sécuriser la saisie en vérifiant l'orthographe de la chaîne de caractères. Pour cela, le programme Visual Basic transmet la chaîne de caractères à un logiciel de traitement de texte, comme Microsoft Word, afin d'en utiliser le correcteur orthographique. De cette manière, le client (l'application Visual Basic) demande l'exécution d'une tâche au serveur (le logiciel Word). Celui-ci exécute la tâche et renvoie le résultat, c'est à dire la chaîne corrigée, au client (l'application Visual Basic). C'est ce que l'on appelle le client / serveur d'applications. Client et serveur sont tous deux des composants logiciel. Ce type de client / serveur est utile lors des développements car il permet de ne pas re-développer des ressources déjà existantes mais de les exploiter.

La méthode ".CheckSpelling" permet d'appeler le correcteur orthographique sur le document actif. Si une ou plusieurs fautes sont détectées, la fenêtre de correction apparaît. Sinon, si il n'y a pas de fautes, rien ne se passe.

`MonWord.ActiveDocument.CheckSpelling` 'Utilisation du correcteur orthographique sur le document actif

Exemple : 'Utilisation du correcteur orthographique et affichage du texte corrigé dans une boîte de dialogue.

```
MonWord.ActiveDocument.CheckSpelling
'Affichage du texte corrigé dans une MsgBox
MsgBox (MonWord.Selection)
```

Remarque : on peut également utiliser la méthode ".CheckGrammar" pour vérifier la grammaire.



## UTILISATION DE MICROSOFT EXCEL DANS VISUAL BASIC

Visual Basic dispose d'un certain nombre de bibliothèques permettant d'utiliser la plupart des logiciels Microsoft. Parmi ces logiciels, on peut citer Microsoft Excel. Il faut donc ajouter la bibliothèque correspondante. Pour cela, cliquez sur "Références" dans le menu contextuel "Projet". Cochez la bibliothèque d'objets "Microsoft Excel 9.0 Object Library".

### Création d'un classeur Excel

```
Dim MonExcel As New Excel.Application    'Déclaration et ouverture d'un objet Excel que  
l'on nomme "MonExcel"  
MonExcel.Visible = True                 'Affichage de l'objet Excel  
MonExcel.Workbooks.Add                   'Création d'une nouveau classeur Excel
```

### Ouverture d'un document Excel

On ouvre un document à l'aide de la méthode "Open" de la propriété "Workbooks" de l'objet Excel.

```
MonExcel.Workbooks.Open ("C:\MonDocument.doc")    'Ouverture du classeur Excel  
spécifié  
MonExcel.Visible = True                           'Affichage de l'objet Excel
```

### Masquer l'objet Excel

```
MonExcel.Visible = False                         'Masquer l'objet Excel
```

### Créer une nouvelle feuille dans un classeur Excel

```
MonExcel.Worksheets.Add                         'Ajouter une feuille  
MonExcel.Sheets("Feuil4").Name = "MonTableau"    'Renommer une feuille
```

### La propriété "ActiveWorksheets"

La propriété "ActiveWorksheets" fait référence à la feuille active dans Microsoft Excel. Lorsqu'il y a plusieurs feuilles dans un classeur Microsoft Excel, la feuille active est celle qui a le focus.

### La propriété "Range"

La propriété "Range" permet de définir sur quelle cellule on va réaliser la commande suivante. Par exemple, elle est utile pour définir dans quelle cellule on va écrire.



### Ecrire dans une feuille Excel

MonExcel.Sheets("feuille2").Select 'Selection d'une feuille  
MonExcel.Range("B2") = "Bonjour" 'Ecriture d'un texte dans la cellule "B2"  
MonExcel.Range("B3") = "=2+2" 'Ecriture d'une formule de calcul dans la cellule "B3"

### Utilisation du "Copier / coller"

MonExcel.Range("B2").Copy 'Copier une cellule  
MonExcel.Range("C2").PasteSpecial 'Coller

### Enregistrer un classeur Excel

On peut enregistrer le classeur actif de l'objet Excel grâce à la méthode "Save"  
MonExcel.ActiveWorkbook.Save 'Enregistrer le classeur actif

### "Enregistrer Sous" un document Excel

On peut aussi enregistrer le document actif de l'objet Excel grâce à la méthode "Save as"  
MonExcel.ActiveWorkbook.SaveAs ("C:\MonDocument.doc") 'Enregistrer le classeur actif  
sous le chemin et nom  
\_ "C:\MonDocument.xls"

### Fermeture de l'objet Excel

MonExcel.Quit 'Fermeture de l'objet Excel

## Création d'un graphique Excel

Avec Visual Basic, il est possible de créer un graphique dans un classeur. Pour cela, il faut sélectionner une plage de valeur qui comportera les données sources nécessaires à la création du graphique grâce à la méthode ".Select" puis demander la création du graphique correspondant à cette plage de valeur grâce à la méthode ".Add" de la propriété ".Charts" de l'objet "Excel".

### Exemple :

```
-  
MonExcel.ActiveWorkbook.Charts.Add           'Ajout d'un graphique  
MonExcel.Worksheets.Add 'Ajouter une feuille  'Ajout d'une feuille  
MonExcel.Sheets("Feuil4").Name = "MonTableau" 'Renommer la feuille  
  
'Saisie des valeurs du graphique  
MonExcel.Range("A1") = "ANNEE"  
MonExcel.Range("B1") = "2001"  
MonExcel.Range("C1") = "2002"  
MonExcel.Range("A2") = "CA"  
MonExcel.Range("B2") = "15000"  
MonExcel.Range("C2") = "17500"  
MonExcel.Range("B1", "C2")                   'Selection des valeurs du graphique  
MonExcel.Charts.Add                           'Création du graphique
```



## UTILISATION D'INTERNET AU TRAVERS DE VISUAL BASIC

L'utilisation d'Internet au travers de Visual Basic est possible grâce à l'utilisation du contrôle "WebBrowser". En effet ce contrôle permet d'afficher des pages web. Pour l'utiliser, il faut sélectionner le composant "Microsoft Internet Controls" dans le menu "Composants" du menu contextuel "Projet".

Le contrôle "WebBrowser" apparaît donc dans la barre d'outils sous la forme d'une planète. Ce contrôle est en sorte une fenêtre que vous devez placer sur votre feuille ("Form") aux dimensions que vous souhaitez de manière à que la navigation soit la plus conviviale possible.

La spécification de la page web à afficher dans le contrôle se fait grâce à la méthode ".Navigation" du contrôle.

Exemple : WebBrowser1.Navigate "<http://www.ifrance.com/vbgenerator/>"

WebBrowser1 est le nom du contrôle WebBrowser.

## UTILISATION DE L'API WINDOWS

### Définition de l'API Windows

L'API Windows est composée de trois DLL principales :

- User32 (gestion des tâches relatives aux fenêtres, menus, contrôles et boîtes de dialogues,
- GDI32 (gestion des graphismes),
- Kernel32 (gestion des tâches de windows).

### Définition d'une DLL

Une DLL (Dynamic Link Library) est une Bibliothèque de Liaison Dynamique. Il s'agit de procédures externes à l'application (ici externes à l'application VB) mais qui peuvent être appelées par cette application afin d'exécuter une tâche. Une procédure contenue dans une DLL est liée au programme au moment de l'exécution alors qu'une procédure écrite dans le code source de Visual Basic est chargée à la compilation.

### Les avantages d'utiliser les DLL sont :

- Elles permettent d'utiliser des fonctionnalités non disponibles à l'aide du code Visual Basic,
- L'exécution des DLL est souvent plus rapide que l'exécution du code Visual Basic,
- Leurs mises à jour est indépendante : l'évolutivité des programmes est donc simplifiée, car la modification des DLL utilisées permet de faire évoluer les programmes sans modifier le code source Visual Basic.

### Exemples de procédures contenues dans les DLL composantes l'API Windows :

GetWindowsDirectory (membre de Kernel32.dll) : permet de récupérer le chemin du dossier de Windows,

FlashWindow (membre de User32.dll) : permet de faire clignoter une fenêtre,

ExitWindowsEx (membre de User32.dll) : permet d'éteindre ou de redémarrer l'ordinateur.

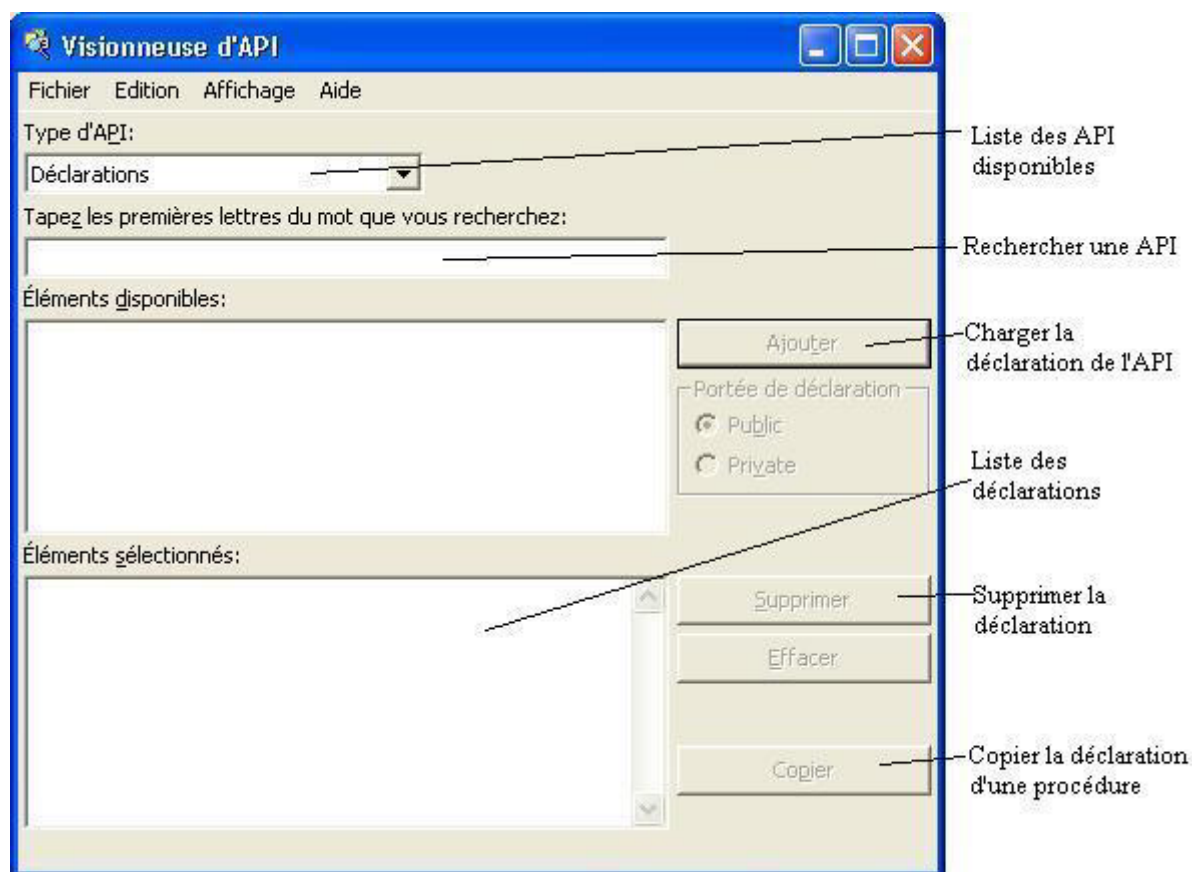
### Utilisation de l'API Windows

Pour connaître les API et leurs fonctionnalités, le seul moyen est de se documenter à l'aide des livres et aussi (mais surtout) grâce au Web qui contient de nombreux sites détaillant les noms et les fonctionnalités des API.

Pour utiliser l'API Windows, vous devez renseigner la déclaration de la procédure Windows à utiliser dans votre application Visual Basic. Comme il est difficile de les connaître, Visual Basic intègre un outil permettant de donner la déclaration des procédures Windows : "la visionneuse d'API". Elle se trouve dans le menu "Microsoft Visual Basic" du menu "Programmes" du menu "Démarrer".

### La visionneuse d'API

Voici à quoi ressemble la visionneuse d'API :





Pour utiliser la visionneuse d'API, il faut d'abord charger le fichier texte contenant toutes les déclarations des procédures Windows. Pour cela, dans le menu contextuel "Fichier", cliquez sur "Charger le fichier texte". Trois fichiers sont alors disponibles (Apiload, Mapi32, et Win32api). Il faut sélectionner le fichier "Win32api" qui contient les déclarations des procédures Windows. Dans la ComboBox "Type d'API", sélectionner "Déclaration". La liste de toutes les déclarations est alors chargée.

Puis rechercher l'API que vous souhaitez utiliser à l'aide de la barre de défilement verticale ou cliquez sur "Rechercher" et saisissez le nom de votre API dans l'InputBox. Une fois la bonne procédure sélectionnée, il vous reste à charger la déclaration correspondante en cliquant sur le bouton "Ajouter". La déclaration apparaît alors dans la liste "Eléments sélectionnés". Il ne reste plus qu'à cliquer sur "Copier" pour copier la déclaration dans le presse papier et la coller dans votre application "Visual Basic".

Ensuite, coller la déclaration dans un module standard que vous attacherez à votre application. Pour plus de renseignements sur les modules standards, reportez vous à la section "Les modules". Dans votre programme principal, appeler votre procédure à l'aide des instructions suivantes :

```
Dim Action as Variant 'Déclaration d'une variable de type Variant qui recevra la déclaration
Action = NomModule.NomProcédureAPI(paramètre1, paramètre2, ..., paramètreX)
```

Remarques : il est indispensable d'affecter l'appel du module à une variable de type "variant", NomModule est le nom de votre module, NomProcédureAPI est le nom de votre API déclarée dans votre module, si la procédure nécessite des paramètres, il faut les renseigner obligatoirement. Pour les paramètres, il faut se documenter afin de savoir les valeurs qu'ils peuvent prendre (là encore je vous conseille les nombreux sites web).

#### *Exemple d'utilisation d'API :*

Dans l'exemple suivant, le clique sur un bouton de commande (nommé "EteindreMachineCmd" permet d'éteindre l'ordinateur. Pour cela, on utilise la procédure "ExitWindowsEx" contenue dans la DLL "User32". On crée un module qui va contenir la déclaration de la DLL que l'on nomme "ModuleRedémarre" puis on l'enregistre.

#### *Code du module :*

```
'Déclaration de la procédure permettant de fermer Windows. Cette ligne de déclaration a été
obtenue en intégralité grâce à la visionneuse d'API
Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As Long, ByVal dwReserved
As Long) As Long
```



*Code de l'application VB :*

Remarque : dans l'application VB, dans la procédure événementielle "Click" du bouton "EteindreMachineCmd", on place le code suivant :

'Appel du module

Action = ModuleRedemarre.ExitWindowsEx(1, 0) 'La machine s'éteint

Remarque : Les paramètres 1 et 0 permettent d'éteindre la machine. D'autres paramètres permettent de redémarrer la machine, d'autres encore permettent de redémarrer en mode "MS-DOS". Quoi qu'il en soit, il faut obligatoirement renseigner les paramètres