Bien implémenter un projet avec Visual Basic 6.0

par Cécile Muno

Date de publication : 23/12/2004

Dernière mise à jour :

Un petit tour d'horizon non exhaustif des possibilités offertes par Visual Basic 6.0 pour réaliser un projet le plus générique possible.

I - Introduction

Ce cours a été conçu pour vous aider à construire un projet le plus indépendant possible au niveau des données changeantes telles que l'emplacement des fichiers, bases de données, # ainsi qu'au niveau de la répétition des tâches de l'application comme le changement d'un curseur sur tous les contrôles de même type, un fond de feuille qui s'adapte à la grandeur de celle-ci ou encore, une action répétée sur des données.

Cette formation Visual Basic n'a pas la prétention de répondre à toutes vos questions dans les domaines approchés ni d'étudier en détail toutes les possibilités offertes. Il s'agit plus simplement de vous faire connaître l'éventail des choix possibles pour des actions déterminées. A vous d'adapter ces exemples dans votre situation de programmation. Rien n'est imposé, juste suggéré.

L'exemple en téléchargement est réalisé sous Visual Basic 6.0 Edition Professionnelle et testé sous Windows 2000 et Windows XP.

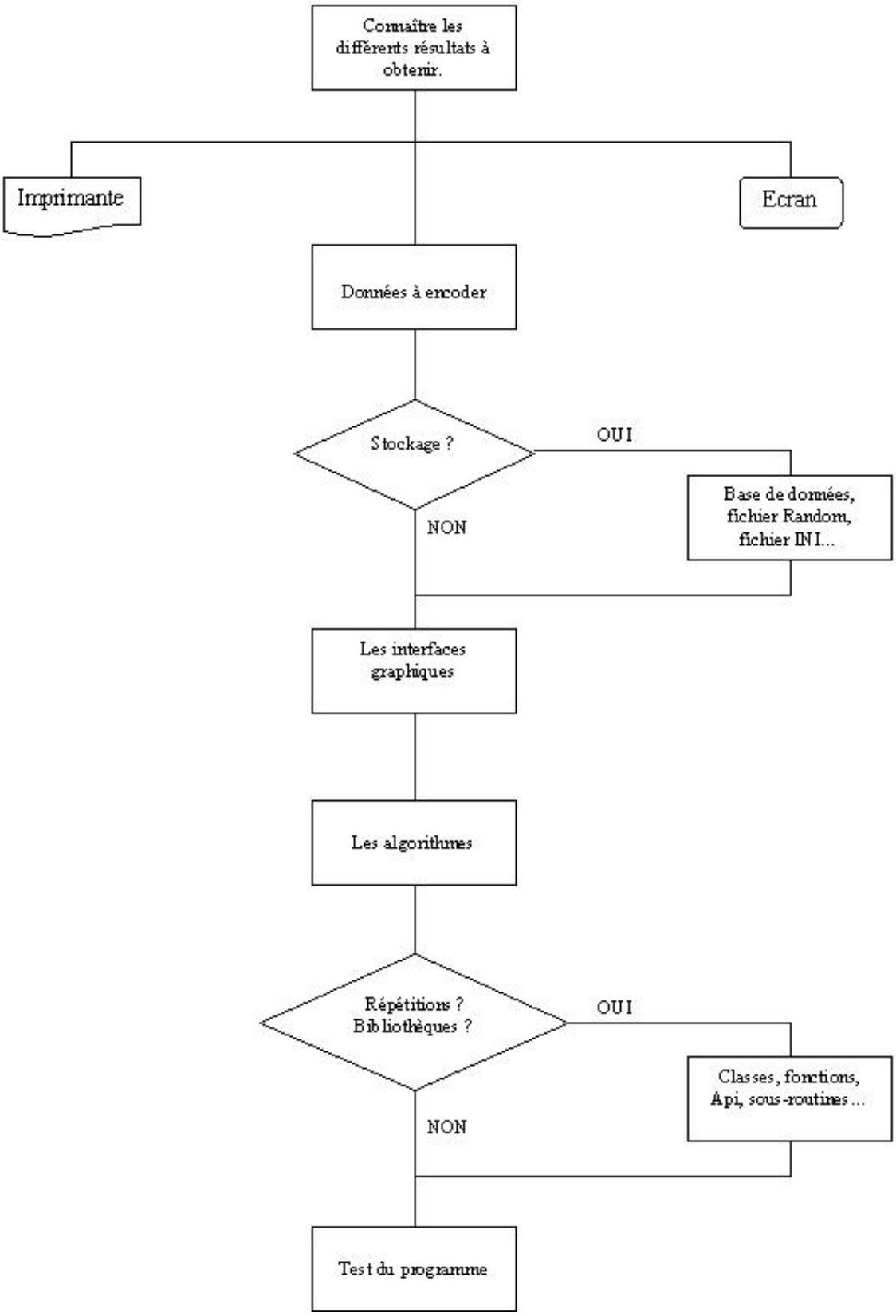
II - Débuter un projet

a - Analyse

Toute action d'envergure, même dans la vie courante, demande une analyse. Vous ne partez pas en vacances sans rien prévoir, rien préparer sinon vous allez au devant de bien des petits (et même gros) ennuis qui vous les gâcherons sûrement, vos belles vacances !

Il en est de même avec un projet de programmation. Vous devez prendre le temps de schématiser les tenants et aboutissants de ce que vous désirez réaliser.

Le schéma est assez simple en lui même mais, plus vous le complétez et l'annotez, plus facile en sera la programmation et la mise en place des éléments nécessaires.



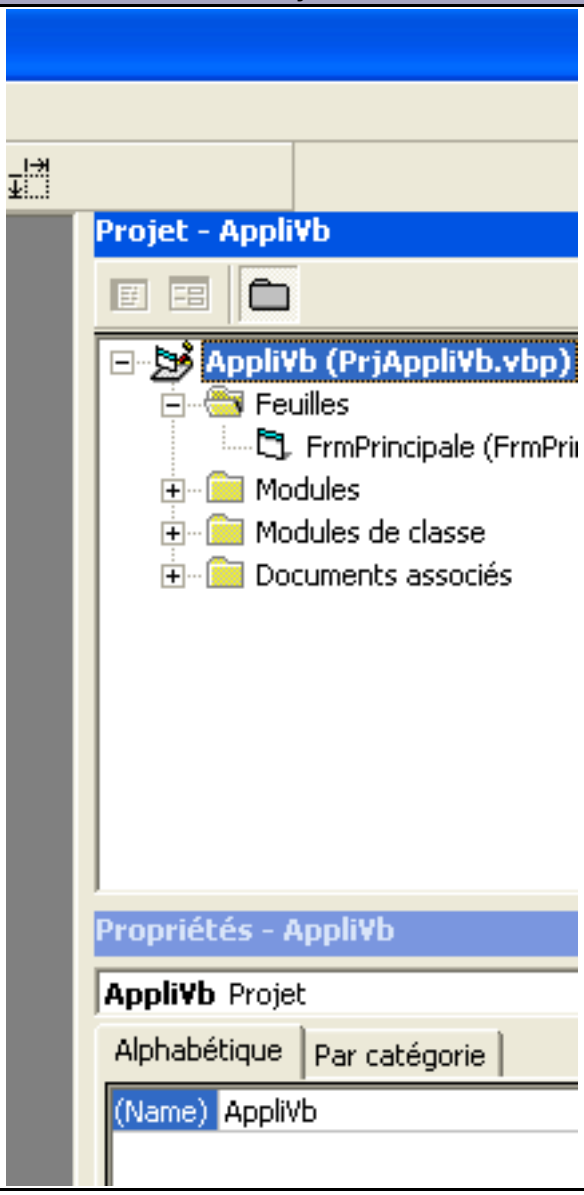
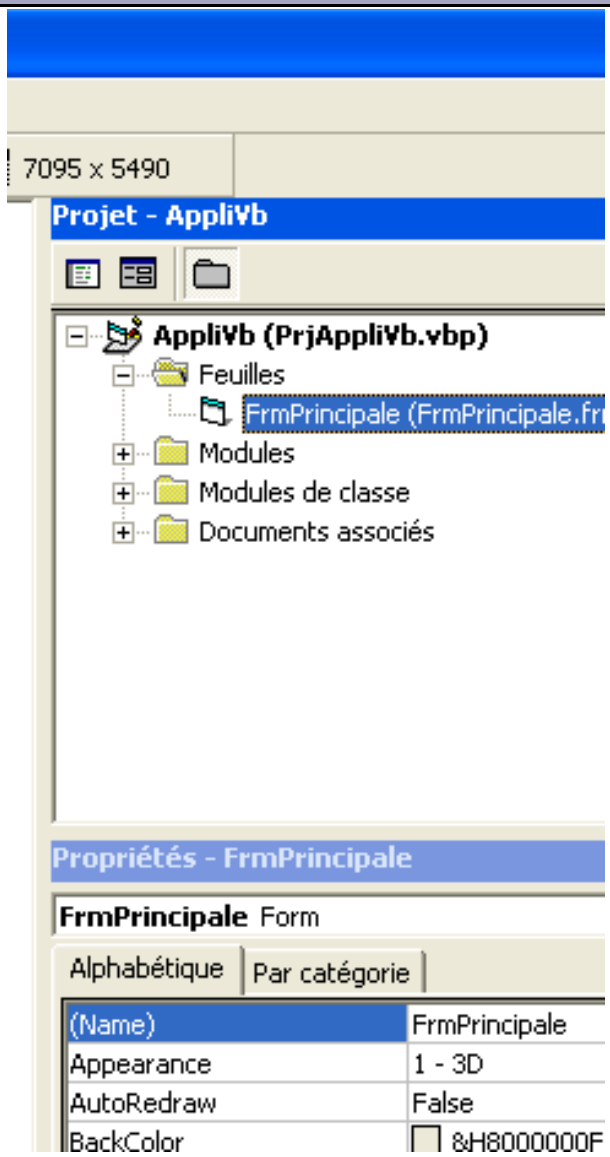
b - Création du nom et de l'emplacement du projet

Dès que vous ouvrez Visual Basic et que vous avez choisi de créer un nouveau projet quel qu'il soit, il est IMPERATIF de le nommer et de le sauvegarder dans un dossier précis.

En effet, il est tentant de se dire "Tiens, je vais vite sauver mon projet, prudence oblige !" et de cliquer sur l'icône  . Je vous prédis bien des déboires en utilisant cette méthode. Pourquoi, allez-vous me dire ? Tout simplement parce que le système est conçu pour sauver votre projet dans un répertoire par défaut : *Program Files/Microsoft Visual Basic/VB98*. Dès lors, si vous créez un autre projet et réalisez la même man#uvre, vous allez écraser purement et simplement le projet précédent. Même avec un message système, une inattention est vite arrivée !

Nommer le projet et créer un répertoire ad hoc est donc indispensable. Je suis persuadée que certains ont déjà eu la très mauvaise blague de transporter (pas déployer) un projet sur un autre poste, de vouloir l'ouvrir et de se trouver avec des messages d'erreur dès l'ouverture. Messages du style : *Impossible d'ouvrir la feuille UnTel, voulez-vous continuer à charger le projet ?* Enrageant mais normal si vous n'avez pas sauvegardé TOUS les éléments de votre application dès leur création (pour ne pas en oublier) dans un dossier spécifique.

Nommez votre projet, votre(vos) feuille(s), module(s), etc grâce à la propriété *Name* :

Projet	Feuille
	

puis, sauvegardez-le :

Fichier → *Enregistrer le projet sous*

choisissez le répertoire de votre application et enregistrez le projet, feuille(s), module(s), etc

Ainsi, vous n'aurez jamais de soucis de test de votre application sur un autre PC.

J'insiste particulièrement sur ce point de sauvegarde car il est toujours énervant de devoir recommencer tout ou une partie d'un projet pour une telle futilité.

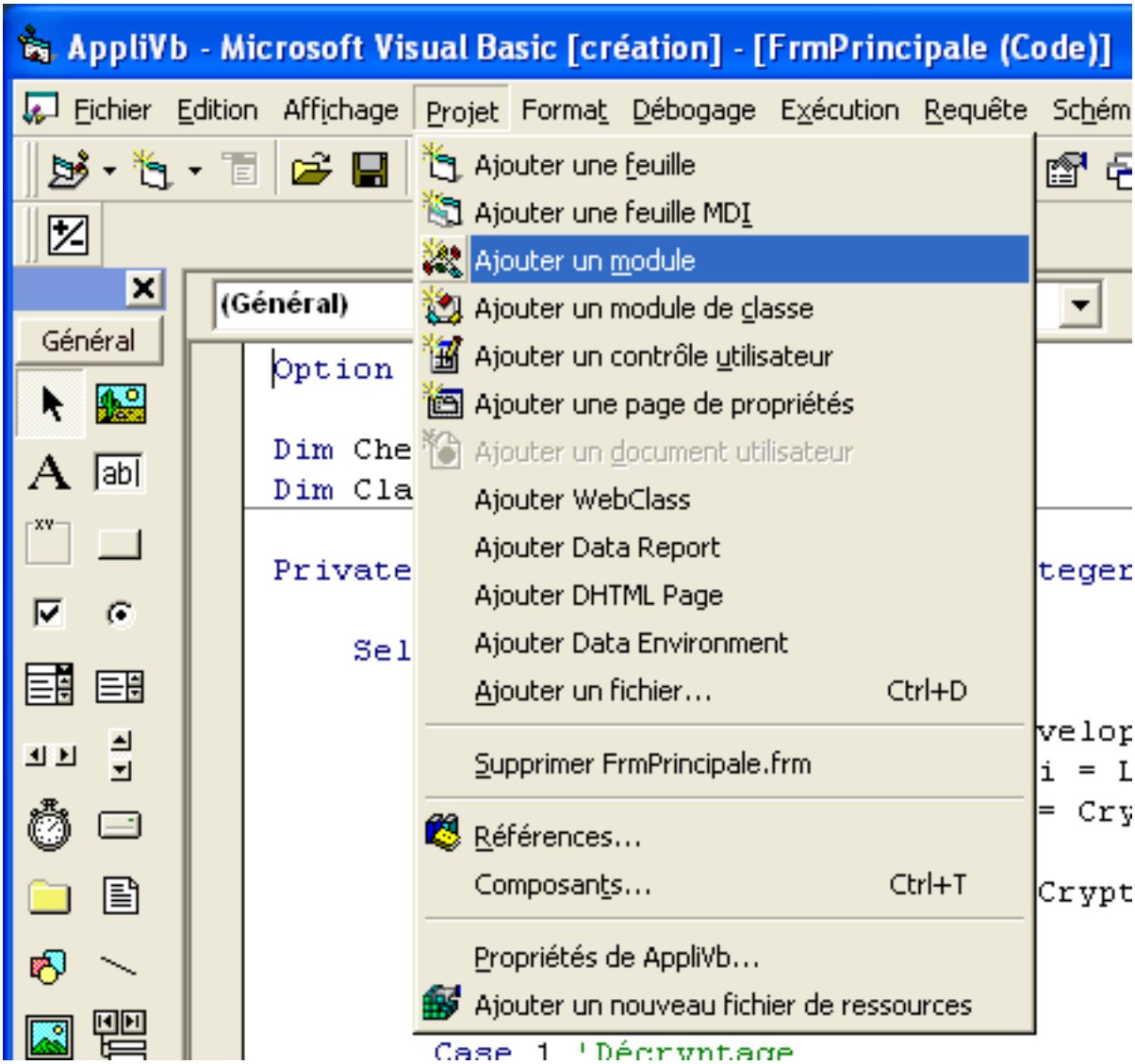
c - Ajouter des feuilles/modules/concepteurs au projet

1 - Ajouter un module BAS

Voyons comment ajouter, par exemple, un module BAS à notre projet.

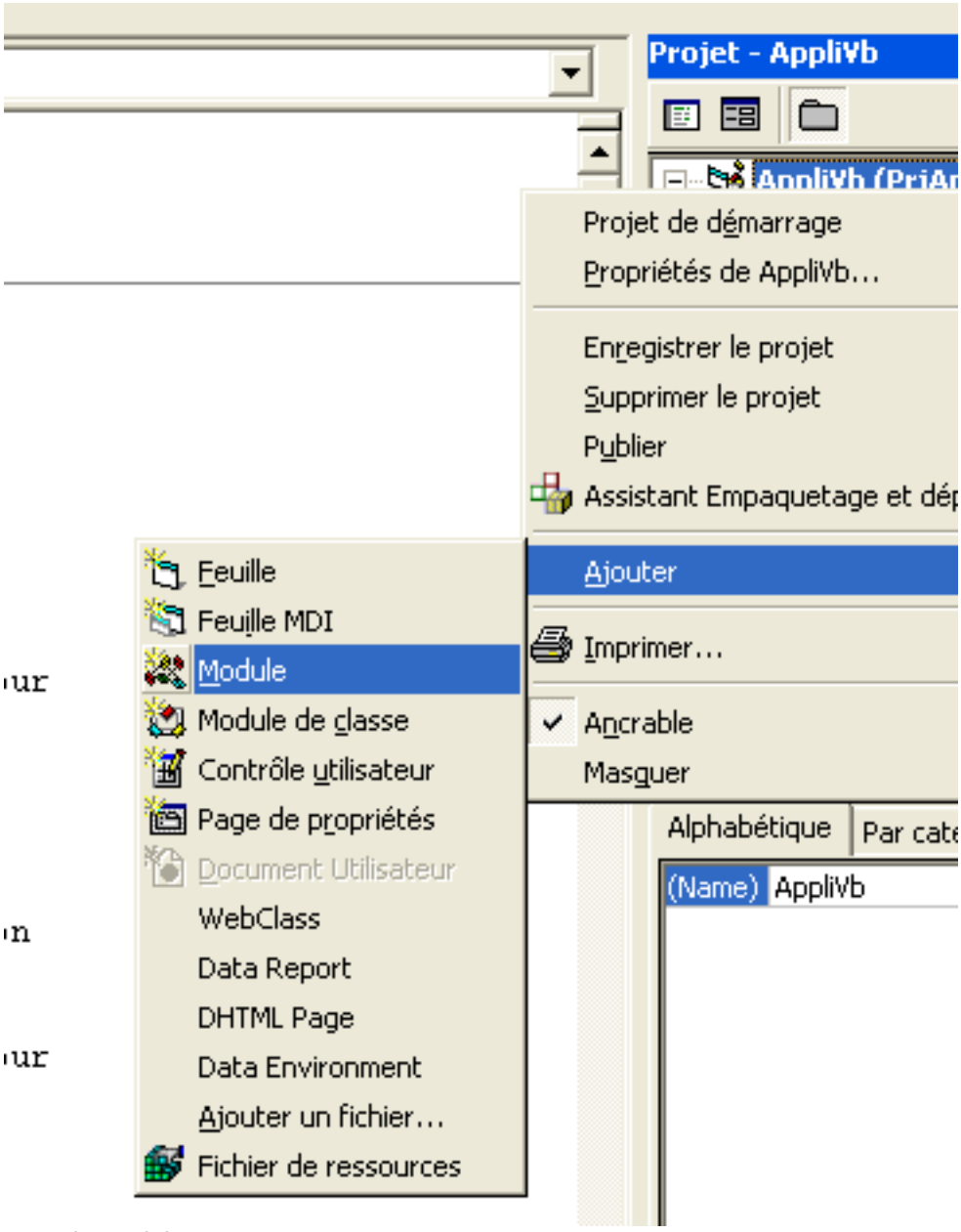
Les modules (extension BAS) permettent d'initialiser l'application au niveau Path (chemin), routines générales, fichier INI, # bref, tout ce qui peut permettre l'implémentation dans un autre environnement ainsi que tout ce qui est répétitif sans calcul dans votre programme. Il s'ajoute dans votre projet de la manière suivante :

Barre de Menu



Explorateur de projet

Dans la fenêtre, cliquez avec le bouton de droite sur le nom de votre projet et vous verrez ces fenêtres apparaître :



Cliquez sur Ajouter puis Module

Quand votre module est ajouté, vous y placez le démarrage de votre projet c'est-à-dire la connexion à votre première feuille.

Pour cela, dans le *Sub Main* du module, tapez :

```
Sub main()
        Set fMainForm = New FrmPrincipale
        fMainForm.Show
```

```
End Sub
```

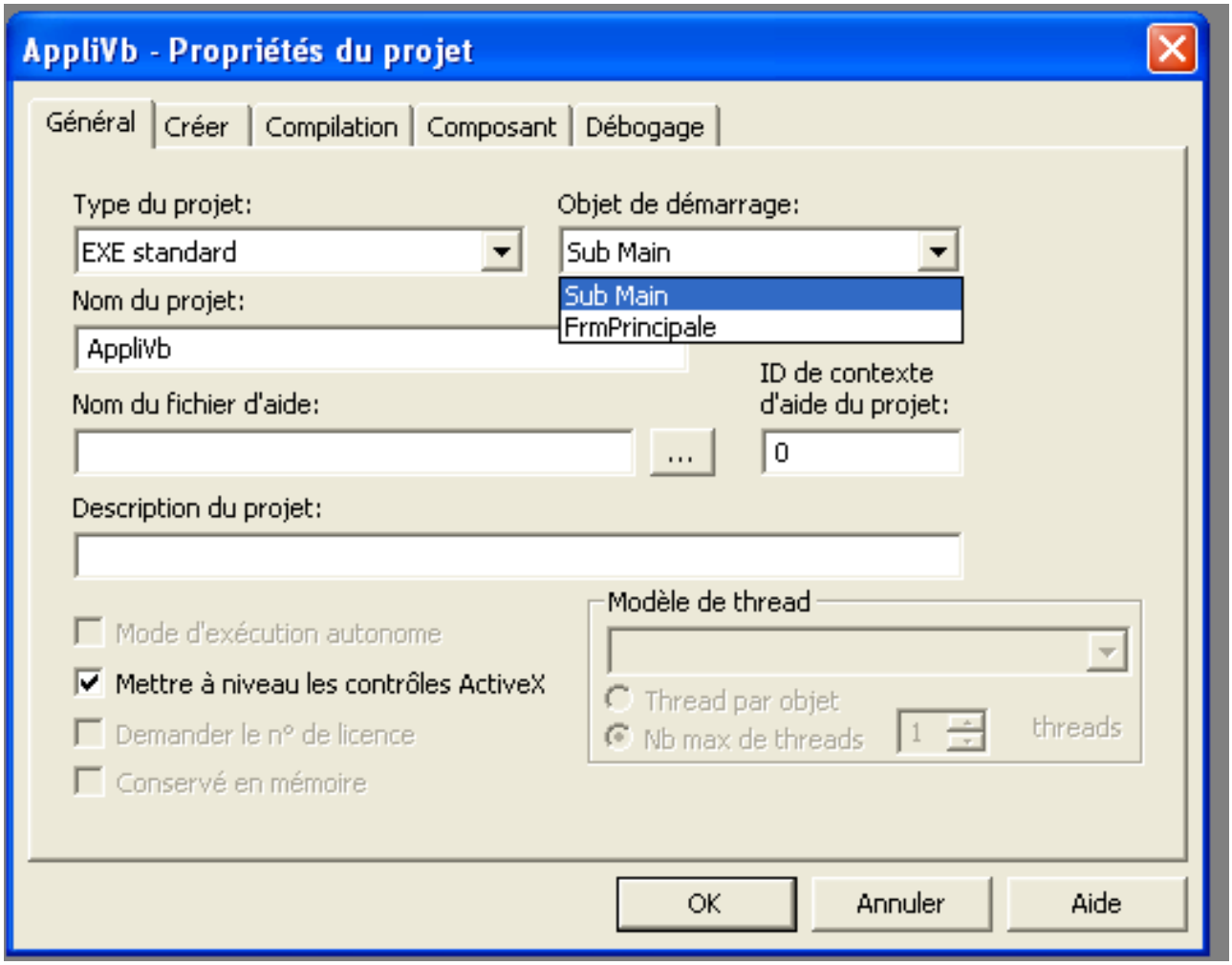
et dans la partie *déclaration générale* du module :

```
Option Explicit
Public fMainForm As FrmPrincipale
```

Ensuite, dites au projet de démarrer sur le module afin de prendre en compte les paramètres que vous y indiquerez :

Dans la barre de menu, *Projet* → *Propriétés de "nom de projet"*

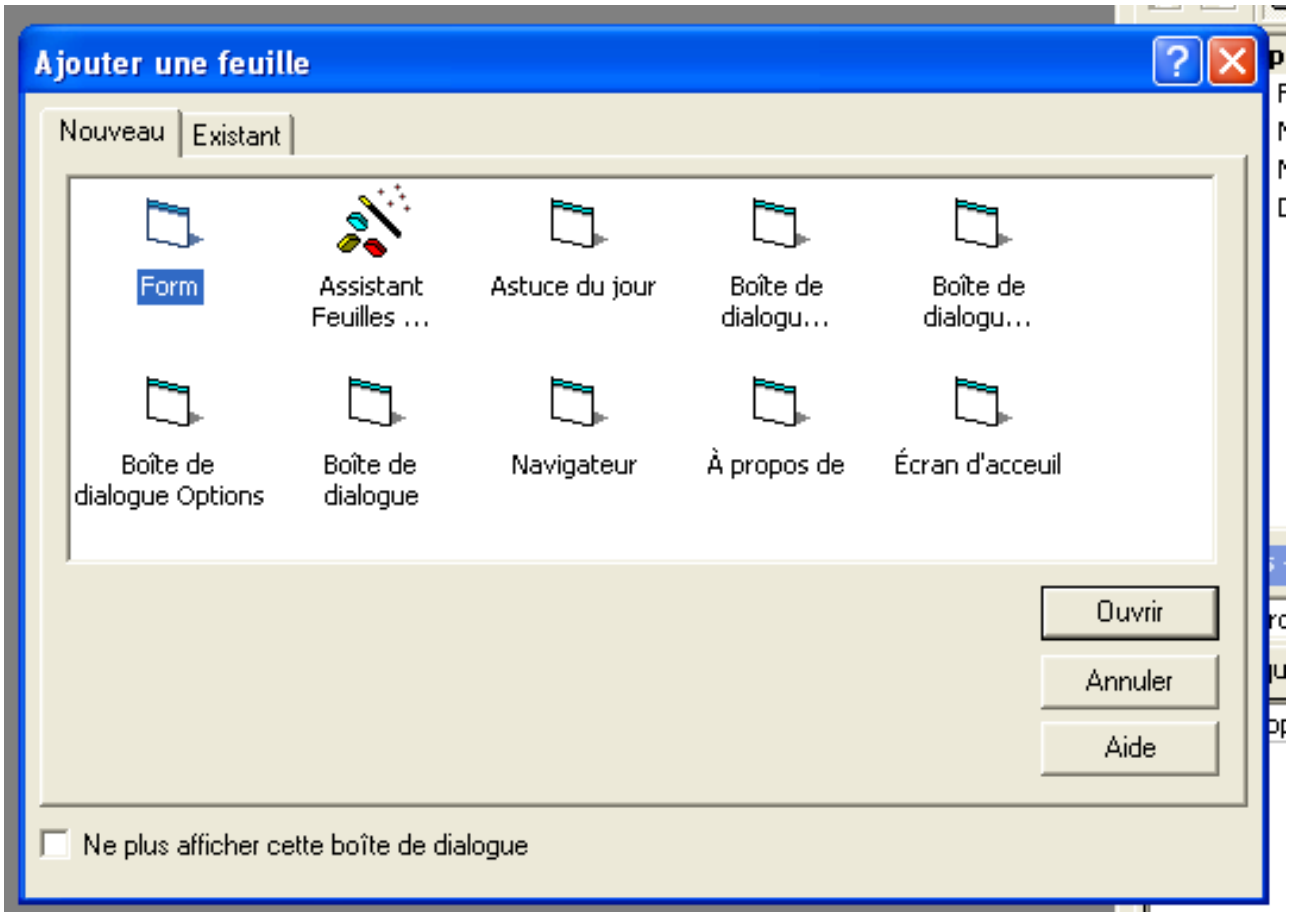
Cette fenêtre s'ouvre :



Il suffit de sélectionner *Sub Main* pour que le programme démarre automatiquement sur le module BAS, initialise les variables ou paramètres qui y sont placés et affiche la fenêtre renseignée sous *fMainForm*.

2 - Ajouter une feuille (form)

Pour ajouter une feuille dans notre projet, le principe est entièrement le même. Cliquez sur le bouton droit de la souris sur le nom de votre projet, choisir Ajouter (Add) puis Feuille (Form). Cette petite fenêtre s'ouvre et vous permet de choisir le style de feuille que vous désirez insérer.



Les plus intéressantes

Form :feuille normale standard (pas MDI)

Astuce du jour :les petites astuces affichées par certain logiciel (Tips of the day)

A propos de :informations sur la version, les auteurs, ... (About)

Ecran d'accueil :le Splash Screen bien connu

IMPORTANT

Veillez à bien *Enregistrer sous* les ajouts que vous avez fait à votre projet. J'insiste absolument sur le fait de sauvegarder régulièrement via cette méthode votre application même si vous trouvez fastidieux de toujours confirmer les noms des objets que vous enregistrez. Je pense que cela vous évitera bien des désagréables surprises lors de votre développement.

Vous avez probablement remarqué que je me suis simplement attachée à l'onglet *Nouveau* et qu'il existe un onglet *Existant*. Je me permets de vous renvoyer au chapitre VII pour aborder ses fonctionnalités.

d - L'utilité du module BAS

Le module BAS est l'endroit où l'on peut regrouper des informations d'initialisation, de répétition, # d'une application. Il permet de sortir les objets des limites de la feuille à laquelle ils auraient appartenu. Il faut cependant faire très attention à la portée des fonctions qui y prennent place car elles seront accessibles de toutes les feuilles de votre projet.

Il va vous servir de référence pour le passage de paramètres en direct, pour les routines répétitives et bien d'autres choses encore que vous allez imaginer vous-même dès que le principe du paramétrage sera expliqué.

Par exemple :

J'ai un fond d'écran uni spécifique à chaque feuille de mon application. Dans le *Form_Load* de chacune de mes feuilles, je charge un contrôle image (propriété *Stretch* = *True*) avec le fond désiré :

```
ImgFond.Picture = LoadPicture(App.Path & "\graphiques\fondfeuille1.jpg")
```

Le graphique de fond n'a évidemment pas la taille de ma feuille car il ne la connaît pas. Dans le module, j'ai donc créé une routine qui dimensionne mon fond à la totalité de ma feuille :

```
Public Sub ImageFond(Feuille As Form)
    Feuille.ImgFond.Move 0, 0, Feuille.ScaleWidth, Feuille.ScaleHeight
End Sub
```

Je l'appelle simplement dans chacune de mes feuilles :

```
Private Sub Form_Resize()
   ImageFond Me
End Sub
```

Voici donc un exemple parmi bien d'autres de l'utilité du module BAS.

e - Le paramétrage

Le paramétrage est important pour la portabilité du projet aussi bien que pour votre propre utilisation. Imaginez que vous changez le nom d'un répertoire pour une quelconque raison. Dès lors, vous devez passer tout votre programme en revue afin d'effectuer les changements d'adressage du logiciel, des fichiers, des bases de données, des graphiques,etc

Non !!! En préparant à l'avance les styles d'adressages que vous allez utiliser, il vous suffira de changer un seul fichier texte et votre application fonctionnera toujours correctement. Ce fichier est en général appelé un fichier INI.

Une instruction va vous servir tout au long de l'application pour référencer l'adressage désiré : **App.Path** Cette nomenclature donne, à tout moment, l'adresse d'installation de votre application. A vous de la compléter avec les

répertoires que vous avez créés :

```
MonChemin = App.Path & ¨\MonRepertoire\MonFichier.ext¨
```

Dans le chapitre suivant, nous allons voir en détail comment créer, lire et écrire dans un tel fichier.

III - Le fichier INI

a - Création

C'est un simple fichier texte qui peut être créé avec le Bloc Notes. Il suffit de lui attribuer une extension INI au lieu de TXT. Ce qui le différencie d'un fichier texte, c'est sa structure. Il est composé de sections et de sous-sections qui permettent, de part le nom qu'on leur attribue, de savoir quels types de données vont y être sauvegardées. Soyez le plus explicite possible dans le choix de vos noms, vous vous faciliterez le travail lors de la programmation.

Le plus simple est de prendre un exemple :

Le nom de la section se place entre crochets. Ici, je choisis de créer la section qui a trait à la rubrique " A propos " de l'application :

Je crée les sous-sections qui correspondent aux données dont j'ai besoin pour afficher ma feuille " A propos " et qui sont sujet à d'éventuelle modification. La plus connue étant sûrement le numéro de la version.

Attribuez maintenant les valeurs à vos sous-sections :

Puisque ces données peuvent être lues à tout moment par l'utilisateur de votre projet lorsqu'il clique sur le menu " A propos ", la routine de lecture de ce fichier DOIT se trouver dans le module BAS. Vous voyez une autre utilisation courante du module BAS.

Ouvrez la fenêtre de code de votre module et tapez :

Vous avez maintenant toutes les références pour utiliser votre fichier.

b - Lecture

Dans votre feuille "A propos", il vous suffit de lire les données du fichier les unes après les autres et de les afficher dans les zones prévues à cet effet. Pas besoin de " fouiller " dans le code pour effectuer les changements ultérieurs, il faudra juste modifier la(les) valeur(s) dans le fichier INI pour qu'elle(s) soi(en)t changée(s) lors de l'exécution du programme.

Pour lire vos données :

```
Private Sub Form_Load()

        Dim CheminFichierIni As String

        CheminFichierIni = App.Path &  "\MonFichier.ini"

        TxtDateCreation.Text = LitDansFichierIni("APROPOS", "DateCreation", CheminFichierIni)
        LblAuteur1.Caption = LitDansFichierIni("APROPOS", "Auteur1", CheminFichierIni)

End Sub
```

Vous constatez qu'il faut passer le nom de la section, le nom de la sous-section et l'adressage complet du fichier INI pour obtenir la valeur inscrite en correspondance. C'est pourquoi j'insiste pour que vous utilisiez des noms

concrets de section et sous-section.

c - Ecriture

Le principe est le même que la lecture mais en utilisant la routine d'écriture. Tapez donc dans votre module le code suivant :

```
Public Function EcritDansFichierIni(Section As String, Cle As String, _
                                    Valeur As String, Fichier As String) As Long

        EcritDansFichierIni = WritePrivateProfileString(Section, Cle, Valeur, Fichier)

End Function
```

Et à n'importe quel endroit de votre application :

```
      EcritDansFichierIni "MaSection", "MaSousSection", MaValeur, CheminFichierIni
```

Vous avez deviné que vous pouvez utiliser ce système pour sauvegarder des valeurs, d'une feuille à une autre, d'un projet à un autre, etc.

NB : il n'est pas nécessaire que le fichier existe pour écrire une section/sous-section. Le fichier sera automatiquement créé lors du premier appel à la routine d'écriture.

Vous pouvez donc imaginer tout ce que vous voulez en respectant la structure Section / Sous-section.

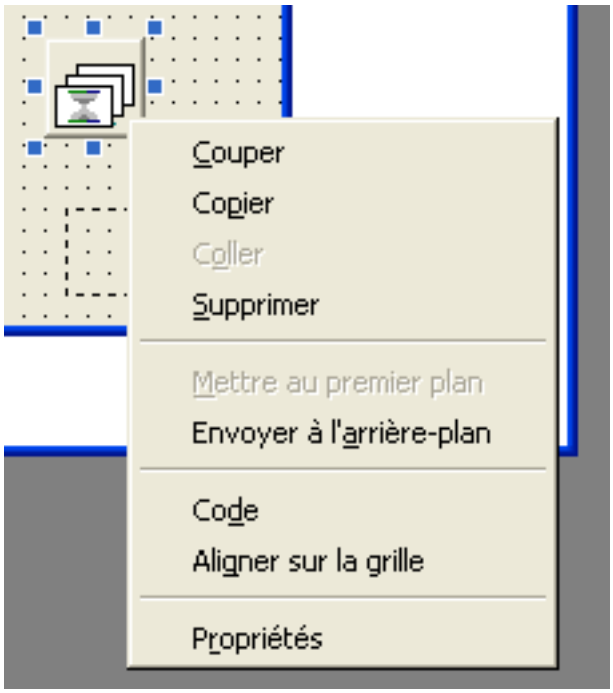
IV - Le contrôle ImageList

a - Utilité

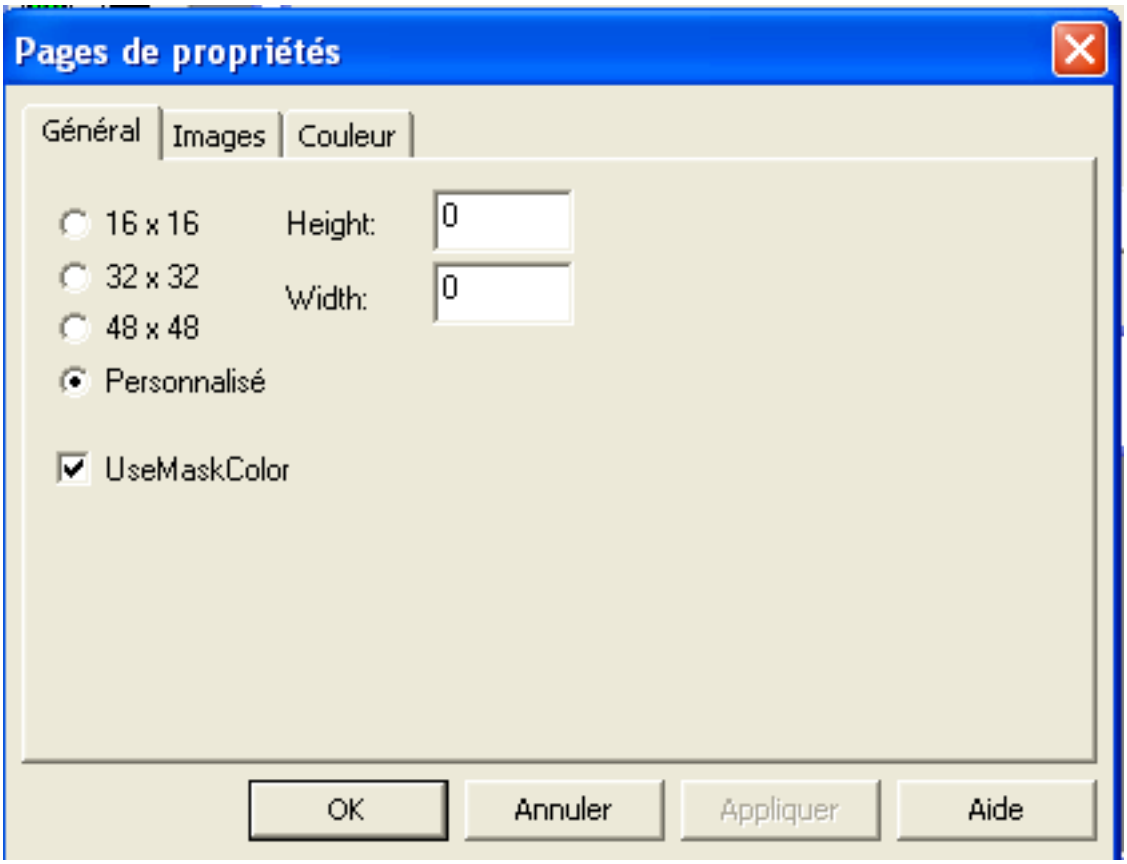
Un contrôle tout simple que vous pouvez initialiser dans la fenêtre de démarrage de votre application et qui permet de lister et référencer les icônes de votre projet. Du moins, c'est la fonction que personnellement je lui attribue dans mon développement.

b - Initialisation

Il se pose sur votre feuille initiale comme n'importe quel contrôle VB et est invisible à l'exécution. Il se nomme par défaut *ImageList1* et correspond à cette icône  . Cliquez droit dessus et vous obtenez cette fenêtre :

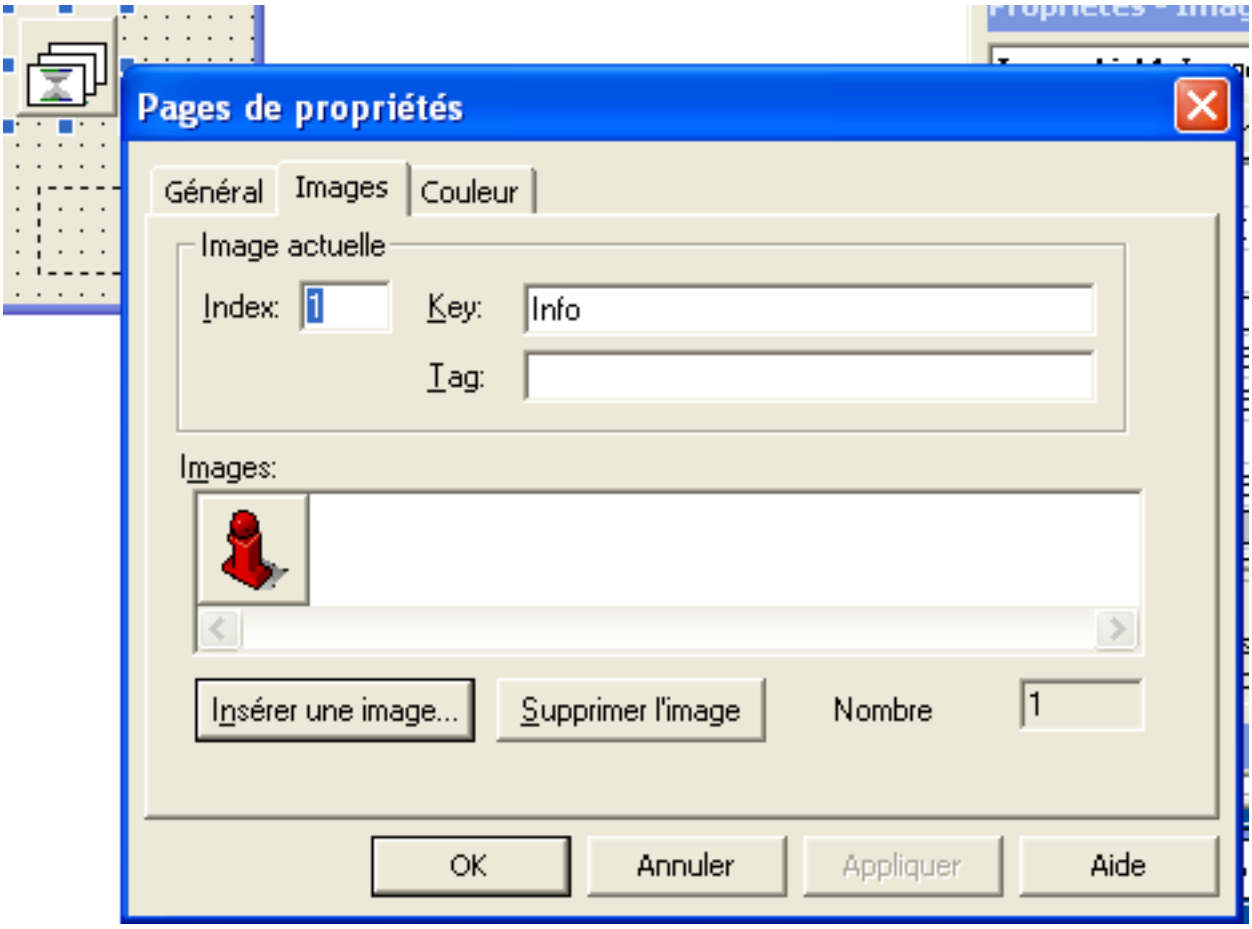


L'action qui nous intéresse le plus est *Propriétés*. Cliquez dessus et vous aller vous retrouver devant une petite fenêtre comme celle-ci :



Dans l'onglet *Général*, vous allez pouvoir choisir la taille de vos icônes. Faites des essais afin de déterminer ce qui vous intéresse le mieux.

Choisissez l'onglet *Images* et cliquez sur *Insérer une image*. Dès lors, vous allez pouvoir parcourir votre disque dur à la recherche de l'icône à insérer. Dès que vous avez choisi l'icône, vous pouvez constatez qu'elle porte le n° d'index 1, ne l'oubliez pas. Nommez-la en lui attribuant une *Key* (clé) comme dans l'exemple ci-dessous.



Placez une référence dans le *Tag* si vous connaissez cette utilisation, sinon, ce n'est pas obligatoire. Choisissez *Appliquer* pour compléter l'insertion de cette icône. Recommencez ce processus autant de fois que vous désirez insérer une icône.

c - Utilisation

Lorsque vous avez composé votre liste d'icônes, vous pouvez vous en servir dans toutes les feuilles de votre projet en les appelant soit par leur *index*, soit par leur clé (*Key*).

Imaginons un *contrôle Image* nommé *ImgIcone* avec une propriété *Stretch* = *True*. Dans la feuille où se trouve le *contrôle ImageList1*, il suffit de taper :

```
ImgIcone.Picture = Me.ImageList1.ListImages(1).Picture
```

Et dans toutes les autres feuilles, sachant que la feuille principale s'appelle *FrmPrincipale* :

```
ImgIcone.Picture = FrmPrincipale.ImageList1.ListImages(1).Picture
```

Cas intéressant

Si vous désirez que pour certains contrôles, le curseur afficher une icône différente lorsque vous pointez dessus, il

y a une solution très simple :

Par exemple, je désire que l'icône *main* (clé) que j'aurais inclu dans mon *ImageList1* apparaisse lors du passage de la souris sur tous les contrôles boutons (*CommandButton*) de ma feuille :

```
Dim Ctrl As Control

For Each Ctrl In Me.Controls
        If TypeOf Ctrl Is CommandButton Then
            Ctrl.MousePointer = vbCustom
            Set Ctrl.MouseIcon = FrmPrincipale.ImageList1.ListImages("main").Picture
        End If
Next
```

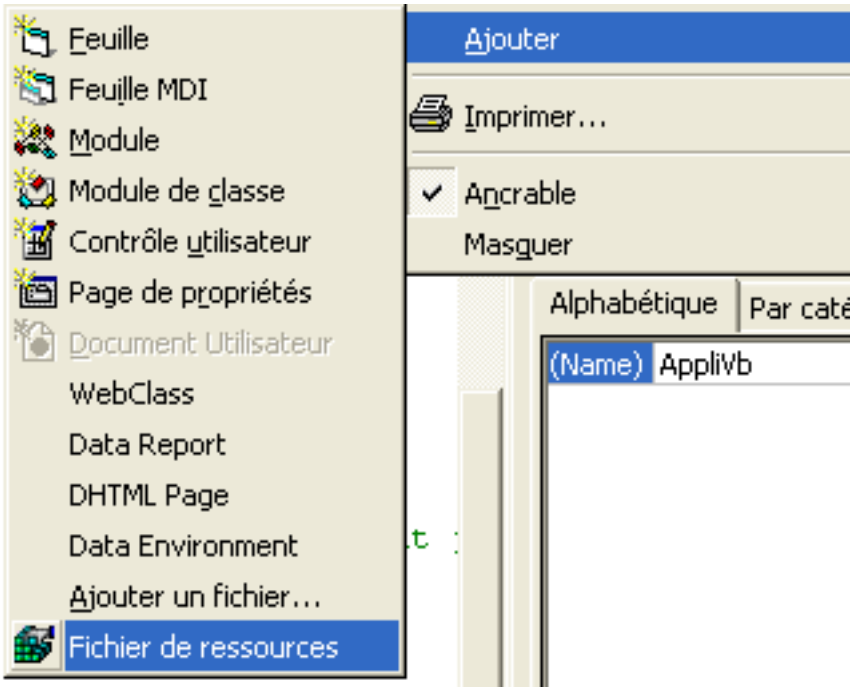
V - Le fichier RES

a - Création

Il s'agit d'un *Compléments* et correspond à l'icône  . Si vous ne voyez pas cette icône, cliquez :

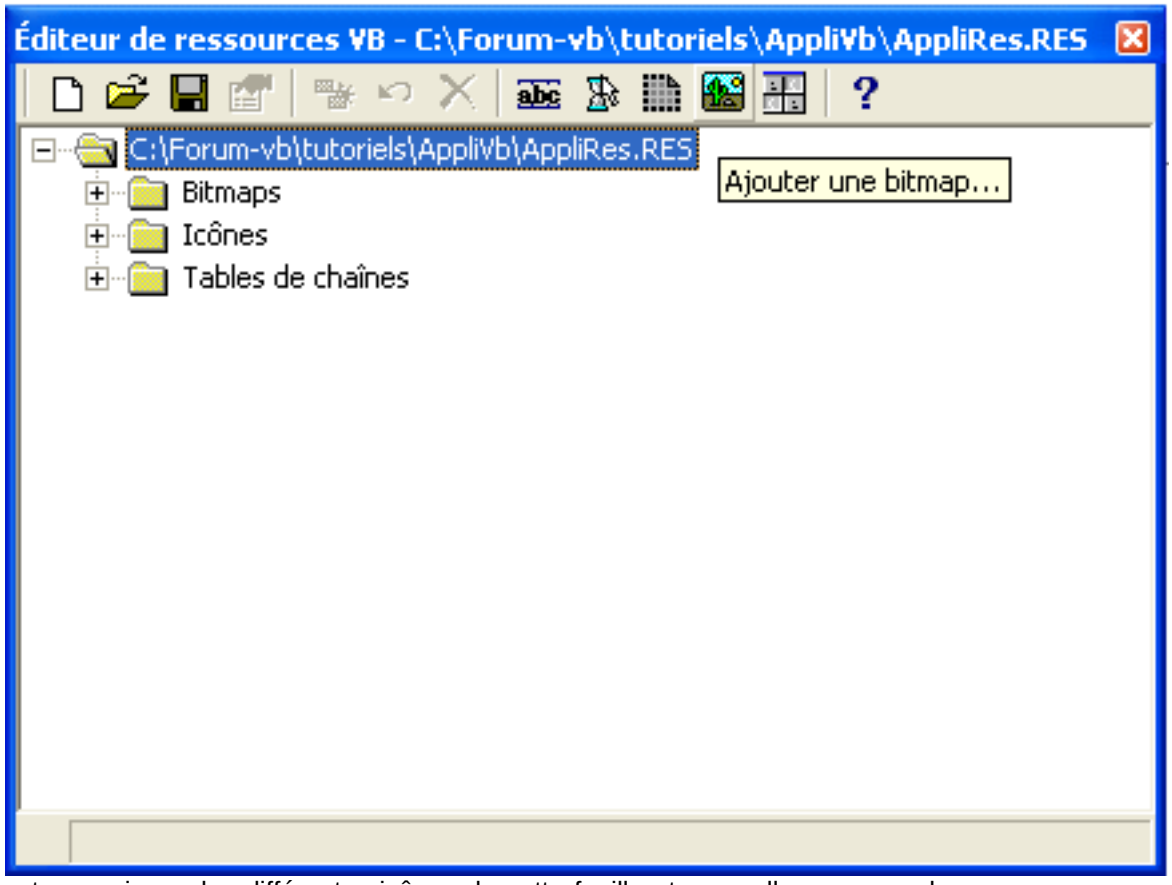
Compléments → *Gestionnaire de compléments* → *Editeur de ressources VB6*

Pour l'ajouter à votre projet, procédez comme pour une feuille : *Projet - Ajouter - Fichier de ressources* ou par le clic droit sur le nom de projet dans l'explorateur de projets :



Nommez-le et sauvegardez-le dans le répertoire de votre application.

Cliquez sur l'icône  et vous allez avoir une fenêtre qui s'ouvre :



Passez votre souris sur les différentes icônes de cette feuille et vous allez comprendre que vous pouvez ajouter beaucoup de choses à ce ficher de ressources. Il faut savoir que toutes les images, toutes les icônes, # que vous insérez dans ce fichier seront compilées et compressées dans ce fichier et, donc, incluses dans votre application lors de son déploiement sur un autre poste. A condition, bien entendu, d'être passé par un empaquetage VB.

b - Lecture

Pour utiliser les données encodées dans votre fichier de ressources, rien de plus simple :

```
Me.Caption = LoadResString(100)  'pour le nom de la feuille en cours

ImgFond.Picture = LoadResPicture("FondImage", 0)  'pour l'image de fond d'une feuille
```

Remarquez l'utilisation de *LoadResString* ou *LoadResPicture* suivant le style de données auxquelles on accède. Le plus intéressant, à mon avis, est la *Table de chaînes* car, si vous y réfléchissez bien, elle peut vous permettre de déployer votre application en plusieurs langues sans aucun effort particulier, si ce n'est la création de *Tables de chaînes* différentes.

VI - Le module de classe

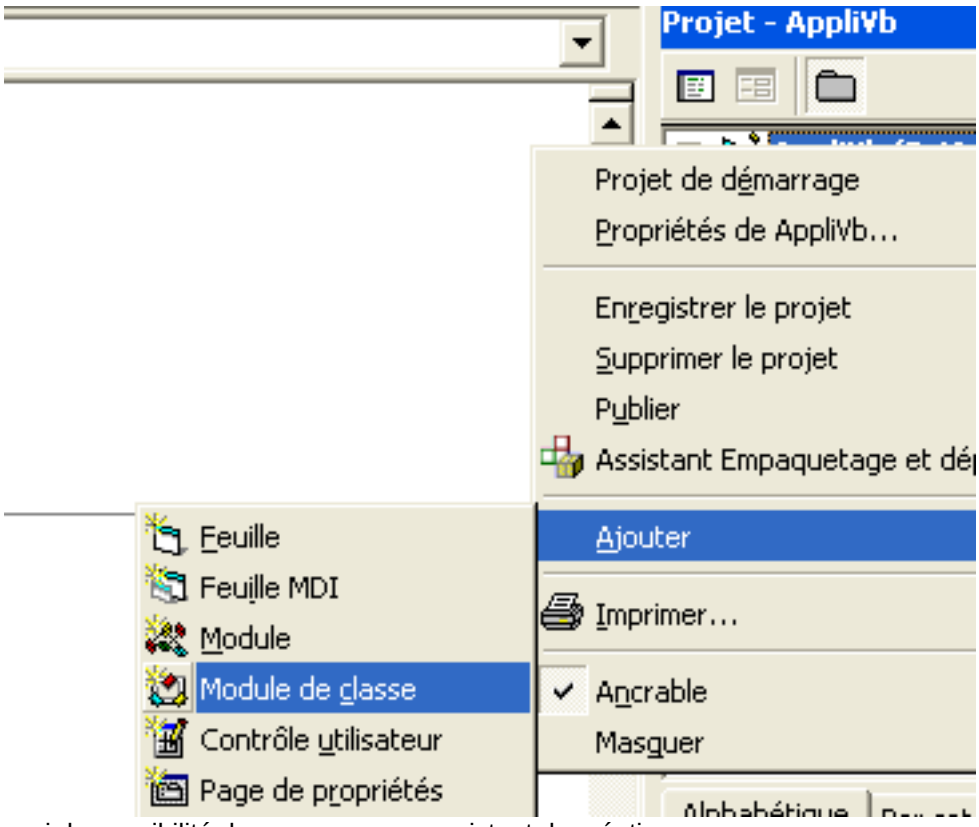
a - Création

Le module de classe est utilisé lors de traitements complexes et/ou répétitifs. Avant de voir comment créer un module de classe, il faut préciser certains points :

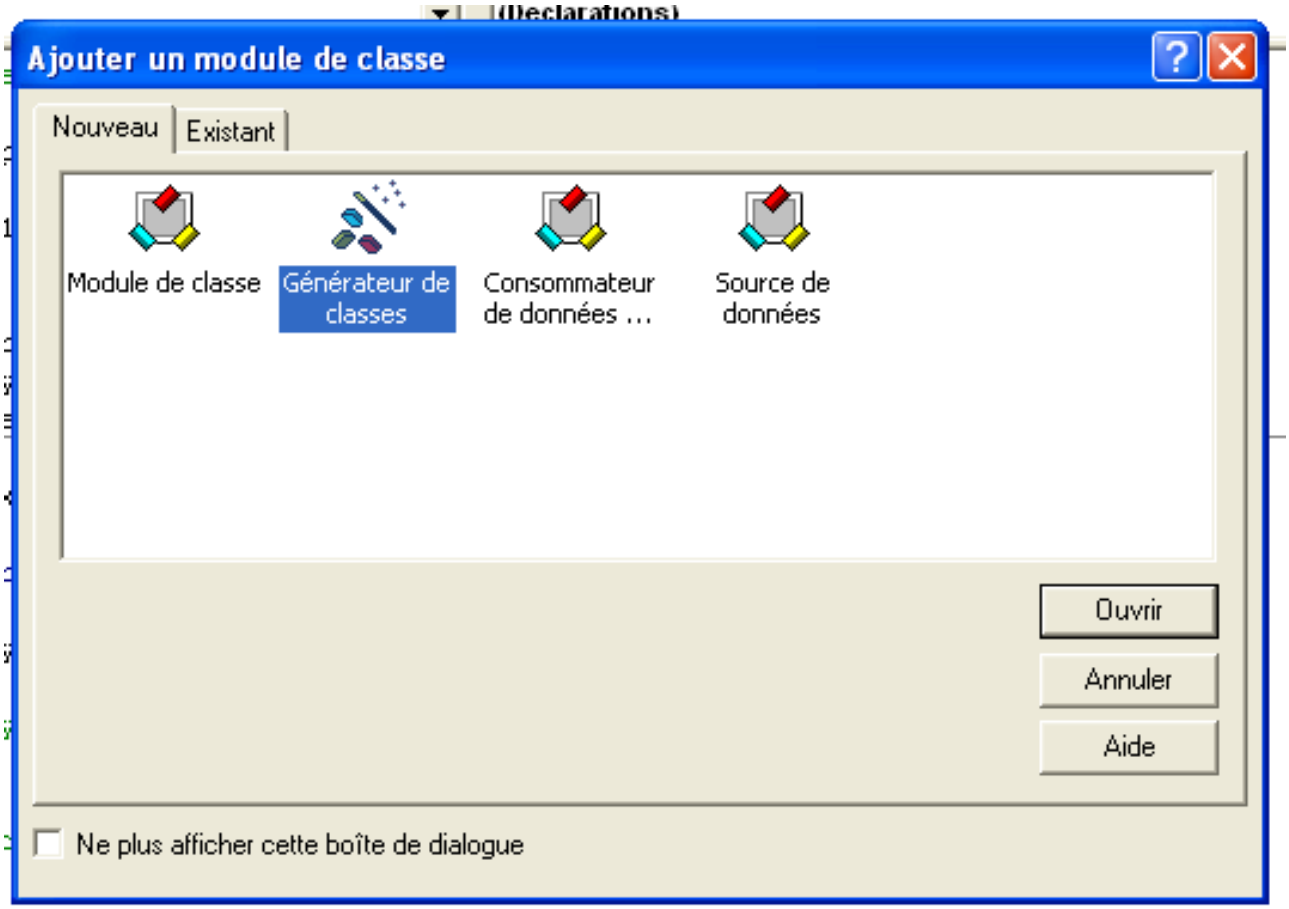
Il possède des propriétés, des méthodes et des événements :

- Les propriétés sont du style Lecture - Ecriture correspondant au vocable LET - GET.
- Les méthodes renvoient des valeurs de retour (sous-routine ou fonction)
- Les événements ne seront pas abordés dans ce cours

Pour ajouter un module de classe, vous pouvez procéder comme pour une feuille ou un module BAS.



Puis, vous allez avoir la possibilité de passer par un assistant de création :



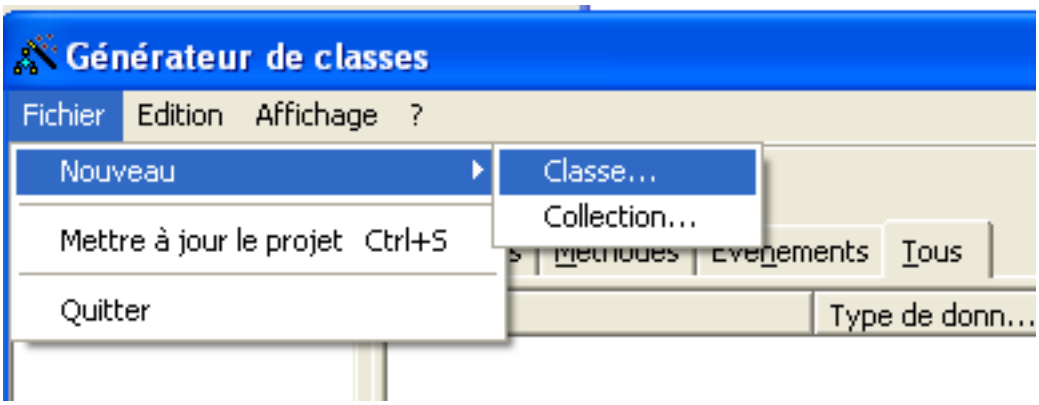
Avant de commencer à créer votre classe, prenez le temps de recenser quelles sont vos entrées - sorties.

- Que dois-je fournir à la classe ?
- Que doit-elle me retourner ?
- Quel type de données sont nécessaires ?

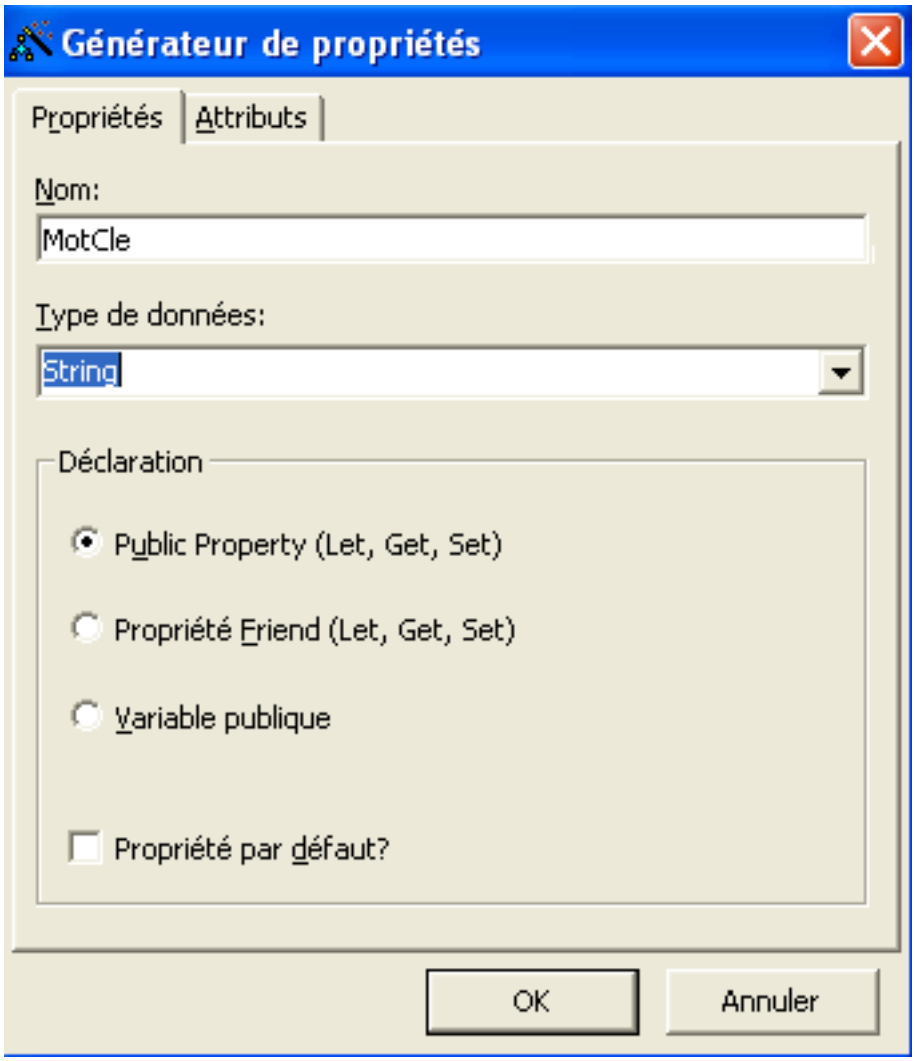
Choisissez le nom de vos variables avec soin afin de bien cerner ce qui est en entrée et ce qui vous est retourné dans l'application appelante.

Passons maintenant à la création réelle de notre classe.

Dès que vous ouvrez le *Générateur de classes*, choisissez *Ajouter une nouvelle classe* et nommez-la de préférence en commençant par *Cls* afin de suivre les conventions VB.

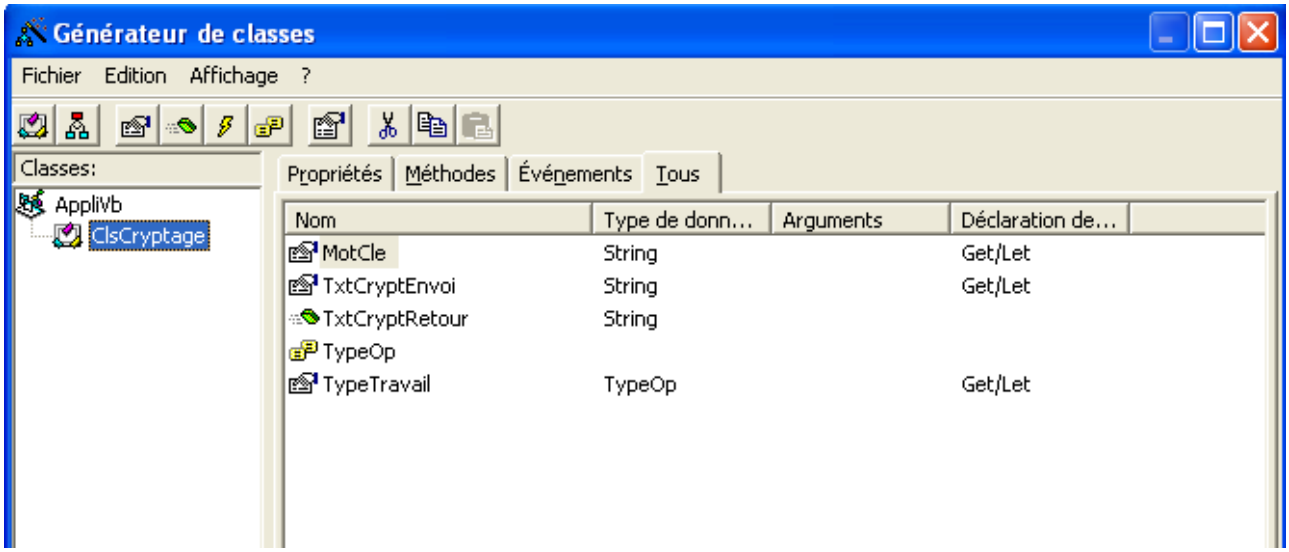


Ensuite, commencez la création des propriétés. *Fichier - Nouveau - Propriété*. Vous devrez la nommer et lui attribuer son rôle dans l'écran suivant.



Dans notre cas, il s'agit d'une variable de type *String* qui répond aux méthodes *Let* et *Get*. (*Set* ne sera pas abordé dans le cadre de ce cours).

Vous créez de la même manière toutes les propriétés de votre classe et vous arrivez finalement à une fenêtre du style :

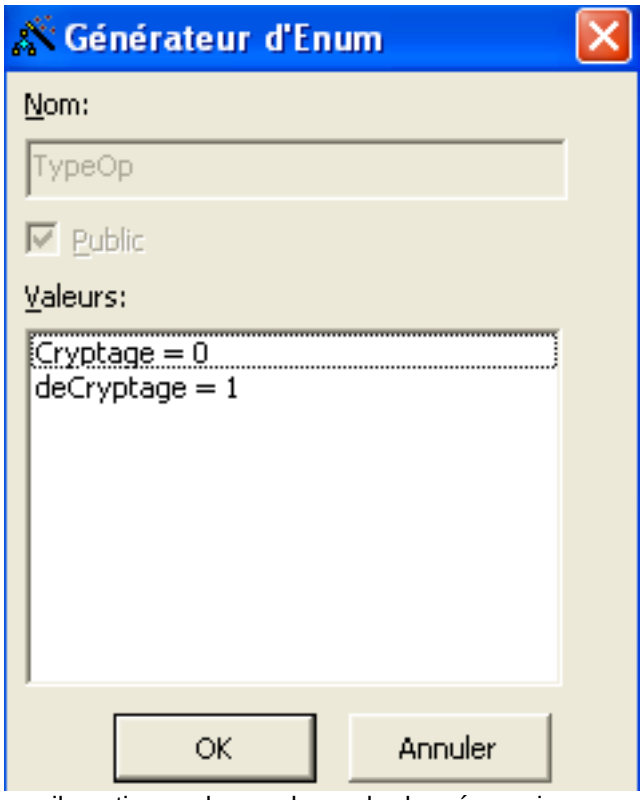


Vous remarquerez que :

MotCle et *TxtCryptEnvoi* sont de type *String* et de déclaration *Get* et *Let* (Ecriture/Lecture). Ils serviront respectivement à passer le mot clé de cryptage à la classe ainsi que le texte à crypter.

TxtCryptRetour correspond au retour d'une fonction de type *String*. Il s'agira de récupérer le texte crypté.

TypeOP est une énumération définie comme suit (lors de la création de vos propriétés) :



Il est normal que, dans notre cas, il contienne deux valeurs de données puisque nous allons aussi bien appeler la classe pour crypter que pour décrypter. Il faudra donc, à l'appel de la classe, préciser le type de travail à effectuer par le vocable *Cryptage* ou *deCryptage*. Vous remarquez que j'ai parlé de *type de travail* et, comme par hasard, la dernière propriété de la classe s'appelle *TypeTravail* qui est de type *TypOp* (type d'opération), la boucle est bouclée.

Laissons l'assistant générer le code en cliquant sur :

Fichier - Mettre à jour le projet

Nous trouvons dans ce code :

- La déclaration des variables *MotCle*, *TypeTravail*, *TxtCryptEnvoi* et l'énumération *TypOp*. En faisant précéder le nom des variables définies lors de la création de la classe par le préfixe *mvar* cela indique que ces variables sont construites pour un usage local donc, dans la classe seulement.

- Les fonctions *Property Get* et *Property Let* respectivement pour chaque variable **sortante** et **entrante** dans la classe. Le type est aussi respecté : *TypeTravail* est de type *TypOp*, *MotCle* est de type **String** et **TxtCryptEnvoi** également. Dans ces fonctions, vous voyez l'utilité des variables locales créées dans la déclaration.

- Une fonction publique *Public Function* pour la seule variable **sortante** de la classe, *TxtCryptRetour* qui est bien de type *String*. En toute rigueur, on pourrait employer *Property Get* pour cette fonction.

Il nous reste à créer :

- La fonction publique de retour *Public Function TxtCryptRetour() As String* du texte crypté ou décrypté suivant le choix dans *TypeTravail*.

- Les fonctions internes à la classe de cryptage et de décryptage appelées dans l'exemple :

FctCryptage : Private Function FctCryptage() As String

FctDecryptage : Private Function FctDecryptage() As String

La plus grande partie du travail est terminée. Vous allez voir qu'envoyer et recevoir les informations est relativement simple.

b - Lecture

Dans le programme appelant, si vous tapez le nom de la classe suivi d'un point, vous verrez qu'elle se comporte comme un contrôle VB et liste ses propriétés et méthodes. Cela sert aussi de contrôle afin de s'assurer que la classe est bien créée. En effet, si les propriétés et méthodes ne sont pas listées après le point, vous aurez un problème à l'exécution.

Il faut donc lui fournir un *MotCle* et un *TxtCryptEnvoi*, un texte à crypter (ou décrypter) et le travail à effectuer *TypeTravail*.

```
ClassCryptage.MotCle = "Developpez"
ClassCryptage.TxtCryptEnvoi = LblTexte.Caption
ClassCryptage.TypeTravail = Cryptage
```

c - Ecriture

Cela équivaut à un simple appel de fonction.

```
   LblCrypter.Caption = ClassCryptage.TxtCryptRetour
```

Voilà, vous avez créé une classe que vous pouvez utiliser à tout moment dans votre projet. Combinez le cryptage à l'écriture dans un fichier INI et vous obtenez des informations illisibles sans le mot clé de décryptage.

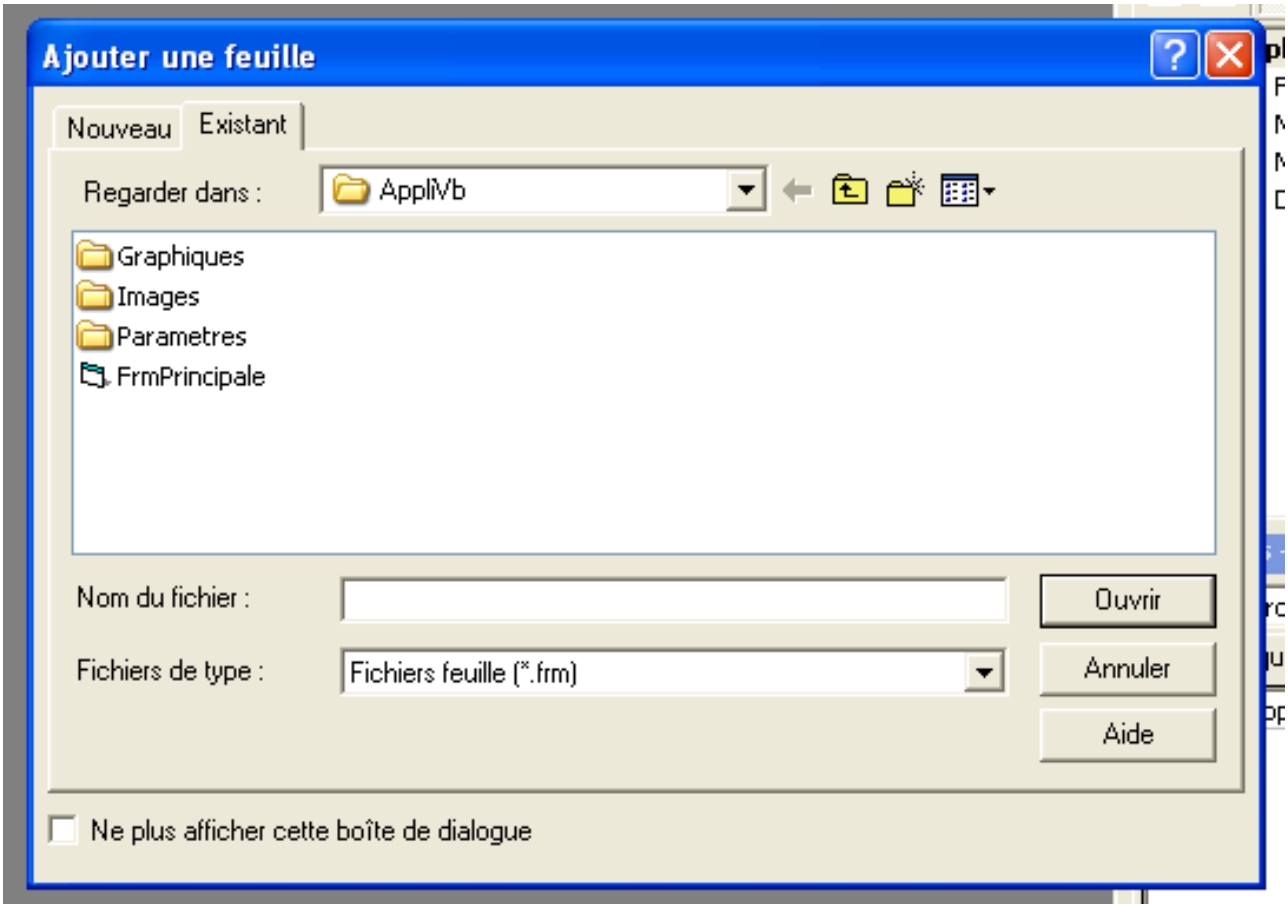
L'exemple est simpliste dans ce cours mais, vous avez le loisir de compliquer l'utilisation des classes en y ajoutant des événements propres.

VII - Copie d'objets existants

a - Récupération de feuilles/modules créés dans d'autres projets

Nous avons vu au chapitre II-c-2 comment ajouter des feuilles/modules à un projet. A ce moment, l'utilisation de l'onglet *Existant* n'a pas été expliquée. Nous allons y remédier dans ce chapitre.

En cliquant sur l'onglet *Existant*, nous voyons qu'il y a moyen de récupérer une (ou des) feuille(s) créée(s) dans un autre projet. Cela évite de refaire tout l'interface graphique lorsque une nouvelle feuille ressemble d'assez près à une déjà créée auparavant dans le **même projet** ou dans un **autre projet**.



Dans le cas de la récupération d'une feuille d'un **autre projet** et **pour autant que** la feuille à récupérer **ne porte pas le même nom** qu'une feuille déjà existante dans votre projet, il suffit via cet onglet, de parcourir vos répertoires à la recherche de la feuille ciblée, de la sélectionner et de l'ouvrir. Elle sera automatiquement ajoutée à votre projet actuel.

IMPORTANT

Vous devez **impérativement** sauver cette feuille dans le répertoire de votre projet en cours après lui avoir donné le nom souhaité pour ce projet sinon, TOUTE MODIFICATION sur cette feuille sera enregistrée dans LE PROJET D'ORIGINE de cette feuille. Conséquences : le projet actuel sera erroné lors du transport car ne pourra pas charger une feuille se trouvant à un emplacement peut-être inexistant sur un autre poste mais, **plus grave**, vous allez altérer la feuille du projet dans lequel vous avez été la récupérer.

En suivant cette méthode correctement et en renommant puis sauvegardant les feuilles au fur et à mesure de leur intégration, vous pouvez même copier une feuille dans le même projet. Changez la propriété *Name* dès que insérée.

Dans le cas de copie d'une **même** feuille dans **un même projet**, vous devez :

- Renommez la feuille concernée et la sauvegarder sous son nouveau nom.

FrmPrincipale → *FrmAccueil*

Enregistrer FrmAccueil *sous* → FrmAccueil (dans le dossier du projet)

- Ouvrir la feuille FrmPrincipale via l'onglet existant. Elle sera intégrée dans votre projet.

Vous aurez donc **2 feuilles identiques** (FrmPrincipale et FrmAccueil) dans votre projet. Faites bien attention de modifier la bonne feuille, normalement *FrmAccueil* puisque *FrmPrincipale* est votre feuille originelle.

Le principe pour un module BAS ou un module de classe CLS est le même que pour une feuille. Par contre, pour les concepteurs, il faut procéder autrement.

b - Récupération d'un DataReport/concepteur créé dans d'autres projets

Si vous essayez de procéder de la même manière pour un concepteur, vous allez vous apercevoir que l'onglet *Existant* n'est pas accessible dans ce cas précis.

Pour récupérer un DataReport existant, vous devez :

- Ajouter un *DataReport* à votre projet
- Vous verrez apparaître *DataReport1* dans votre explorateur de projet
- Nommer votre DR (DataReport) comme celui déjà existant ailleurs
- *Enregistrer sous* votre DR
- Aller dans le dossier où ce concepteur existant se trouve
- Faire un *Copier/Coller* dans le répertoire de votre nouveau projet
- Il vous sera demander si vous voulez le remplacer
- Dites *OUI/YES* et vous avez alors accès au DR créé précédemment
- Vous **devez** le réenregistrer

Vous pouvez dès lors le modifier pour qu'il soit propre à votre nouvelle application

VIII - Synthèse

Un petit tableau de synthèse avant de nous quitter.

Nom	Type	Icône	Description	Chapitre	Accès
Module BAS	Elément de VB		Initialisation et démarrage	II d	Projet - Ajouter un module
Fichier INI	fichier externe		Sauvegarde de paramètres, données, ...	III	
ImageList	Contrôle VB		Liste d'images, icônes, ...	IV	Projet - Composants - Micorsoft Common Controls 6.0 (SP6)
Fichier RES	Compléments		Fichier de ressources Interne à VB	V	Projet - Ajouter un fichier de ressources
Module de classe	Eléments de VB		Classe objet	VI	Projet - Ajouter un module de classe

IX - Conclusion

Nous voici arrivés à la fin de ce petit tour d'horizon non exhaustif d'un exemple d'implémentation d'une application Visual Basic 6.0. Tout ce qui est décrit ici l'est à titre indicatif au niveau utilisation. Il n'y a aucune obligation de procéder de cette manière si vous utilisez déjà un système performant pour vos applications.

Au début de mon étude de VB6, j'ai du développer quelques applications assez conséquentes et gourmandes en ressources. De là sont nées les solutions expliquées dans ce cours afin d'éviter les messages du type *Mémoire insuffisante* dus entre autres au nombre trop important d'images/ icônes figées dans les propriétés des contrôles.

J'espère avoir pu aider quelques uns d'entre vous à créer des applications génériques ou, du moins, vous aider à mieux cerner les possibilités de ce genre de programmation.

Bon travail !

X - Téléchargements