REPUBLIQUE DEMOCRATIQUE DU CONGO
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET UNIVERSITAIRE
« ESU »
INSTITUT SUPERIEUR D'INFORMATIQUE ET DE GESTION
« ISIG »
BP. 841 GOMA

DEPARTEMENT D'INFORMATIQUE

PROGRAMMATION ORIENTÉE OBJET AVEC VISUAL BASIC 6.0

Par : KALUMBI ISSA ALLEN
Licencié en Sciences Informatiques
Certifié en CCNA

"La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi." A. Einstein.

ANNEE ACADEMIQUE 2010-2011

Plan du Cours

CHAP 1 - Introduction

- Rôle de VB
- Définitions - client/serveur, architecture

CHAP 2 - L'environnement Visual Basic

- Exemple d'application - "Le Football"
- Exercice de codage

CHAP 3 - Le Langage VB

- Techniques de base
- "Naming conventions"
- Les structures
- Les opérateurs
- Les fonctions: MsgBox( ), InputBox( ), IsDate( ), IsNumeric( )
- Les decisions
- Les tableaux (Arrays)
- Les boucles

CHAP 4 - Création d'un Form

- Le design
- Les controls communs
- Les propriétés des controls
- Les listes
- Les procedures
- Utilisation d'un Timer

CHAP 5 - Validation de données

- La fonction MsgBox( )
- Les "Events" en Visual Basic - Change, Validate, LostFocus
- Utilisation du "Active Control"

CHAP 6 - Menu et Debug

- Pratique - la calculatrice
- Listes de fichiers
- Création d'un menu Visual Basic

- Le Debugging

CHAP 7 - Caractères et images

- Les fonctions de manipulation de chaînes
- Les blocs de texte
- L'éditeur de texte Visual Basic
- Les graphiques
- Le multimédia

CHAP 8 - VB et bases de données

- Exemple - la base de données "BookStore"
- Le Data control de Visual Basic
- Les controls lies

- Trouver des enregistrements dans une table

- Data controls multiples

- Portabilité de l'application Visual Basic

CHAP 9 - Le Data Project

- Impression de rapports
- Le Data environment
- Le Connection object
- Le Command object
- Exemple de programmation ADO

I. Introduction à Visual Basic

Ce cours est la suite du cours d'Access que vous devez avoir vu. Vous allez maintenant continuer à appliquer les notions de développement d'applications comme dans le cours d'Access mais, vous utiliserez des outils plus puissants et vous apprendrez à penser en termes de déploiement à grande envergure des systèmes créés.
Avec *MS ACCESS*, vous avez étudié les techniques de modélisation et de création d'une base de données relationnelle en mode autonome (utilisée par une personne à la fois sur un PC). Cependant, dans la vraie vie les besoins sont beaucoup plus complexes que ça. Dans ce cours nous allons commencer à explorer (notez le mot *commencer*) les applications complexes. Nous allons étudier et appliquer les concepts tels que: **l'architecture client-serveur** et le développement **"Object-oriented".**

CONCEPTS FONDAMENTAUX

- **CLIENT/SERVEUR**

  L'architecture client/serveur:

  L'architecture client/serveur est la suite logique de la programmation modulaire. La programmation modulaire suppose qu'un gros programme est plus efficace s'il est décomposé en modules; il est plus facile à développer et à maintenir. Donc, si on décompose un logiciel en modules, on réalise qu'il n'est pas nécessaire d'exécuter tous les modules dans le même espace-mémoire. On peut créer un module ***client*** qui demande un service et un autre module ***serveur*** qui fournit le service. En plus, les modules n'ont pas à être sur la même machine ni même sur la même platforme. On peut utiliser la platforme

appropriée pour chaque tâche.

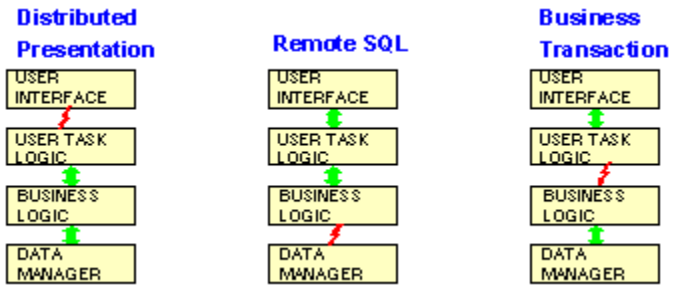


Fig. 1 DIFFÉRENTES SAVEUR DE C/S

Le client

Le client est un programme qui envoie un message à un autre programme, ***le serveur***, lui demandant d'exécuter une tâche quelconque, ***le service***. C'est le client qui gère l'interface avec l'utilisateur, valide les données, gère la communication avec le serveur et exécute certaines opérations logiques. Le client est aussi responsable de la gestion des ressources locales: moniteur, clavier et périphériques. Lorsqu'on parle du client on utilise aussi le terme ***front-end*** car c'est la partie du système qui est à l'avant, c'est à dire la plus visible à l'utilisateur. Le client fonctionne toujours en mode graphique, ***GUI***, et communique avec l'utilisateur au moyen de fenêtres.

Le serveur

Le serveur reçoit les demandes des clients, exécute les opérations d'extraction et de mise à jour de la base de données, assure l'intégrité des données et retourne les réponses aux clients. Le serveur peut aussi être appellé à exécuter des opérations logiques qui peuvent aller du simple au complexe, basées sur les règles d'affaires de l'entreprise. Le serveur

pourrait être une autre machine sur le réseau, il pourrait servir aussi de serveur de fichiers sur le réseau. Le serveur est le ***back-end*** qui gère les ressources partagées et les tâches communes à différentes applications.

- **PROGRAMMATION ORIENTÉE OBJET *(Object-oriented programming)***

  En programmation on a toujours séparé les programmes et les données. La POO regroupe les deux dans des unités réutilisables qu'on appelle ***classes***. Une classe contient des procédures (***methods***) et des données (***attributes ou properties***). On crée une ***instance*** d'une classe comme on déclarerait une variable. L'instance d'une classe est ***un objet***. Puisqu'un objet est facile à dupliquer, il est toujours réutilisable. Aussi, il est facile, à partir d'une classe, de créer des classes similaires qui vont ***hériter*** des caractéristiques de la première mais leur ajouter des fonctions ou des données (***inheritance***)...

- **GUI**

  De nos jours, toutes les applications PC sont à base de GUI (***Graphical User Interface)***). Le GUI permet à l'utilisateur de manipuler des icônes dans un environnement graphique à base de fenêtres (Windows) et offre une productivité beaucoup plus grande que ce qu'on peut obtenir dans un environnement à base de caractères, ie DOS.

- **Pilotée par les événements (Event-driven)**

  Dans l'environnement traditionnel, le programme est lancé et les instructions s'exécutent en ordre jusqu'à ce qu'il n'y ait plus de données. Mais, dans l'environnement Windows, l'utilisateur

contrôle l'exécution. Il doit faire un clic sur un bouton ou choisir une option dans un menu, etc. Le programmeur doit écrire le code en fonction de ces actions (***events***). Par exemple, "Si l'utilisateur clic sur le bouton, ouvrir formulaire X .....". C'est la responsabilité du programmeur de voir à ce les options soient activées ou désactivées au bon moment afin que les traitements s'exécutent dans l'ordre approprié. En Visual Basic, tout le codage est fait dans un environnement ***event-driven***.

II. L'environnement VB

Application : Série d'objets (fenêtres, programmes, menus, etc.) qui travaillent sur un même sujet. On appelle l'application un **Projet**. Le **Projet Scoring** servira à manipuler les données pour un match de football. On pourrait créer un **Projet Vidéo** pour gérer les opérations d'un magasin de vidéos, par exemple ...
En démarrant VB on doit choisir de travailler sur un projet existant ou d'en créer un nouveau. Il y a différentes sortes de projets mais, pour l'instant nous allons créer un ***Standard EXE***. Je vous suggère de sauvegarder le projet dès l'ouverture afin de lui donner un nom officiel (il portera l'extension .VBP et vous devriez créer un répertoire VBapps sur le C:). Notez que quand vous demanderez de sauver le Projet au début, VB vous fera d'abord sauver le Form sur lequel vous travaillez (nommez-le Scoring.FRM dans C:\VBapps).

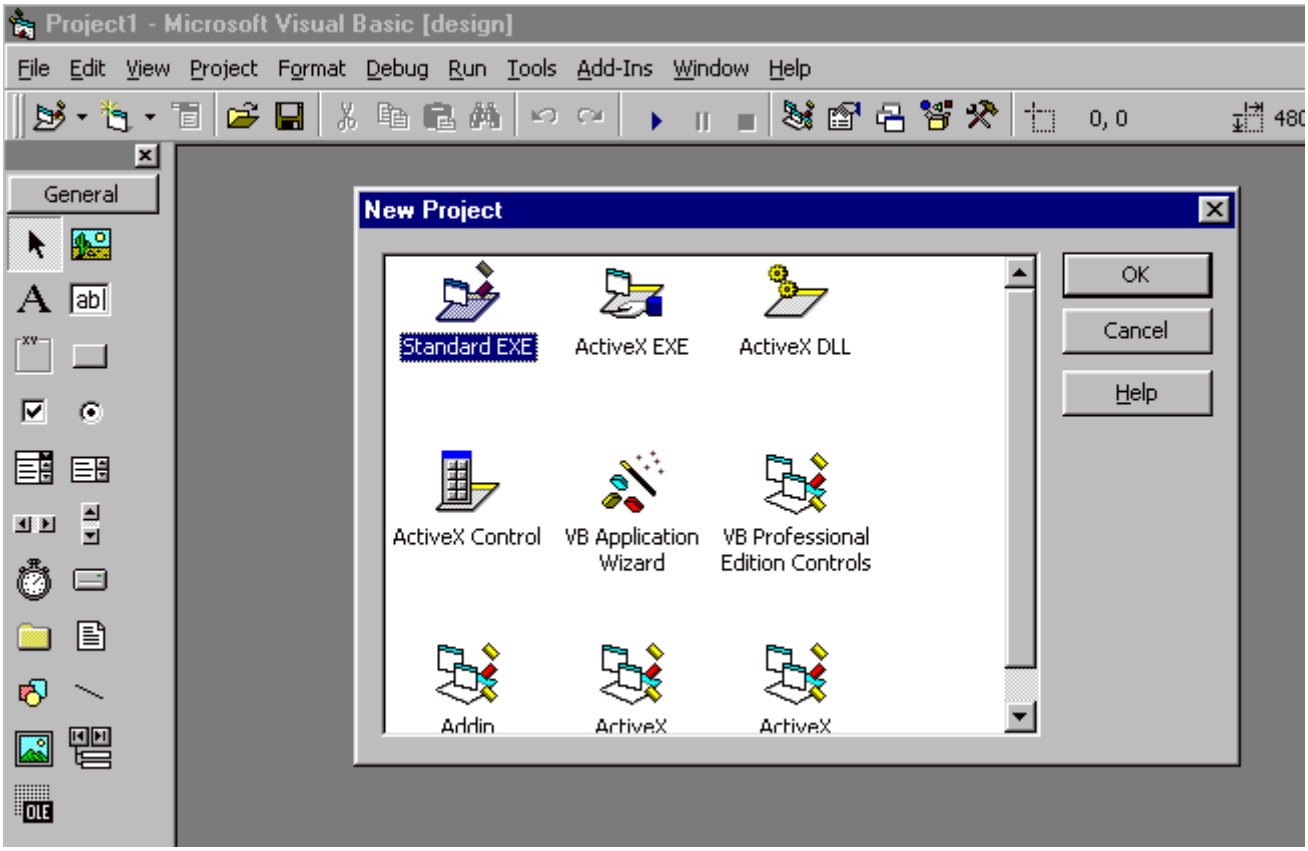


Fig. 2-0

Tout d'abord, remarquez qu'en lançant VB vous avez une première feuille, un ***Form***, qui s'ouvre pour vous. Le Form est l'objet le plus visible de VB. On utilise le Form pour créer l'interface avec l'utilisateur. Pour créer une feuille on y place des ***Controls*** tels que ceux du ***Toolbox*** à la gauche de l'écran. En vous familiarisant avec l'interface VB vous verrez aussi que vous pouvez personnaliser plusieurs des fonctions d'édition de la feuille en allant au menu ***Tools --> Options***.

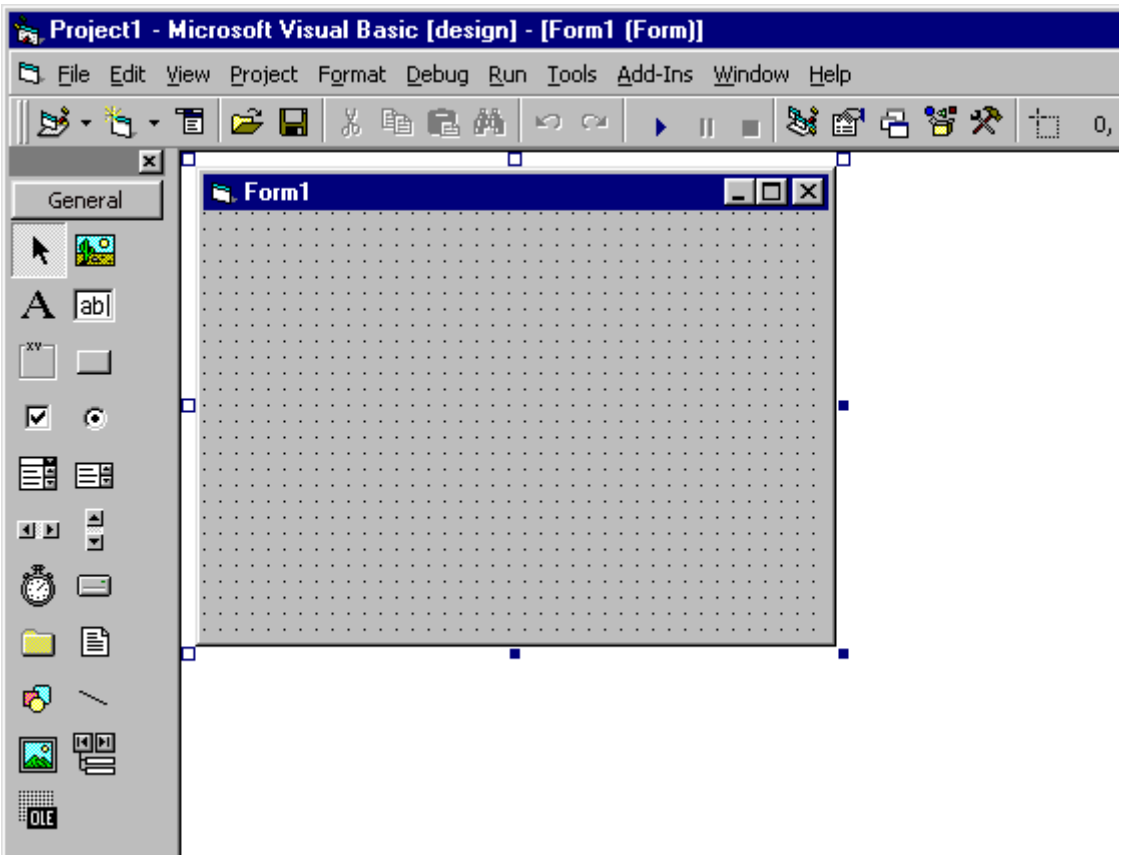
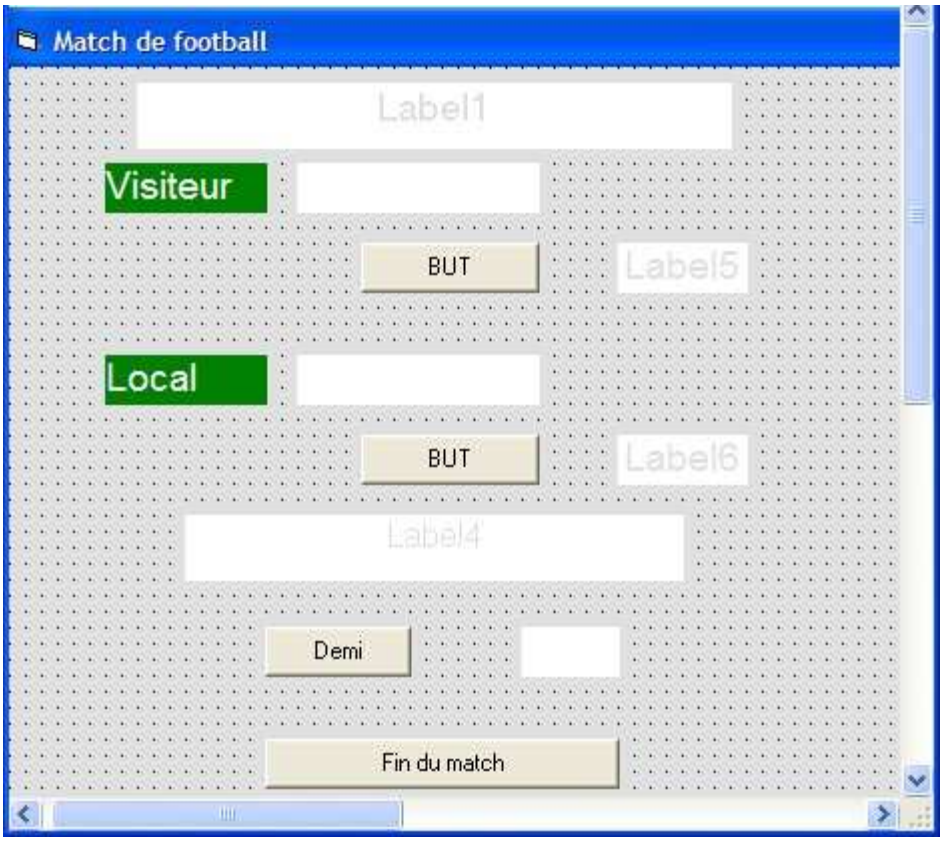


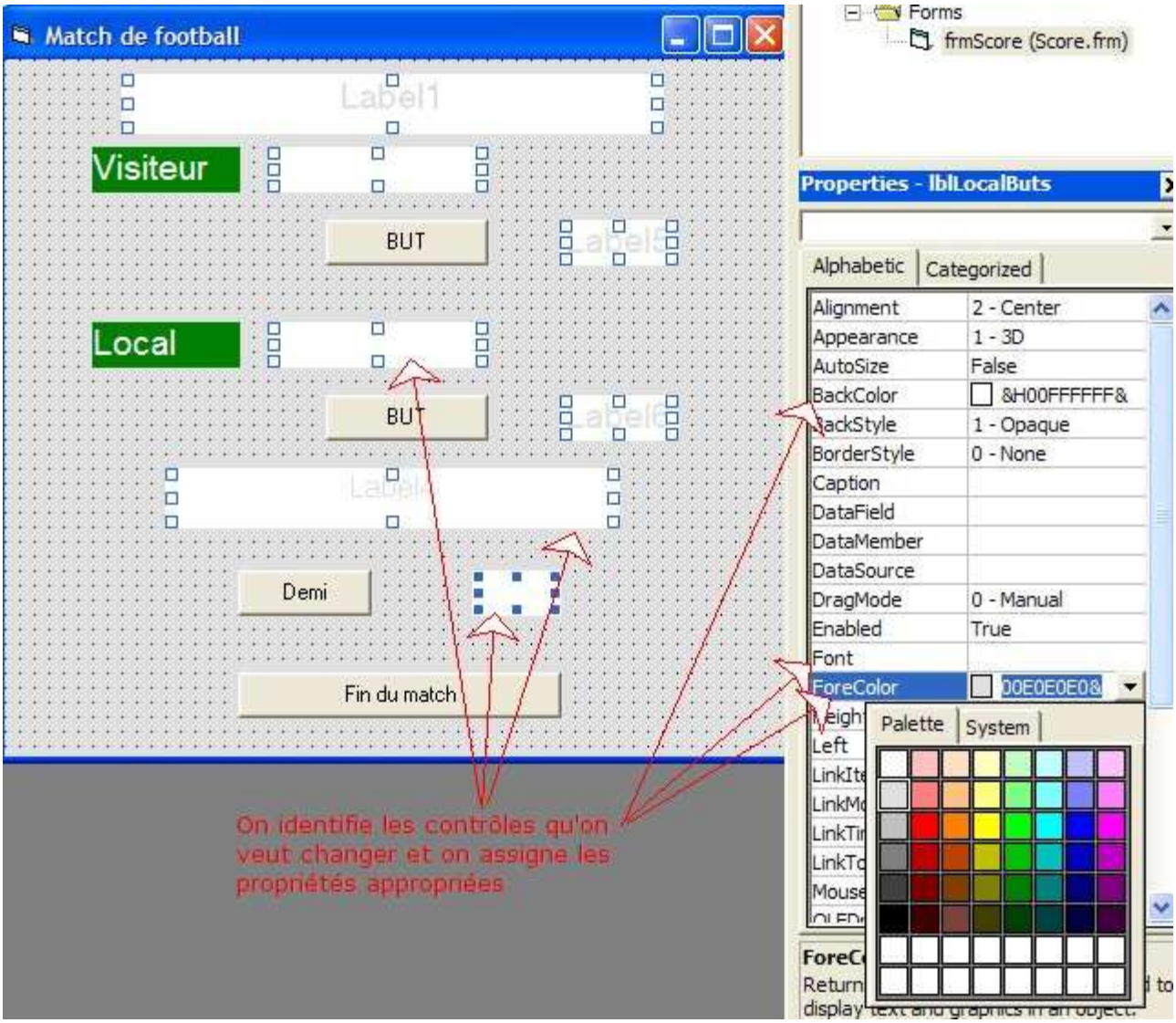
Fig. 2-1

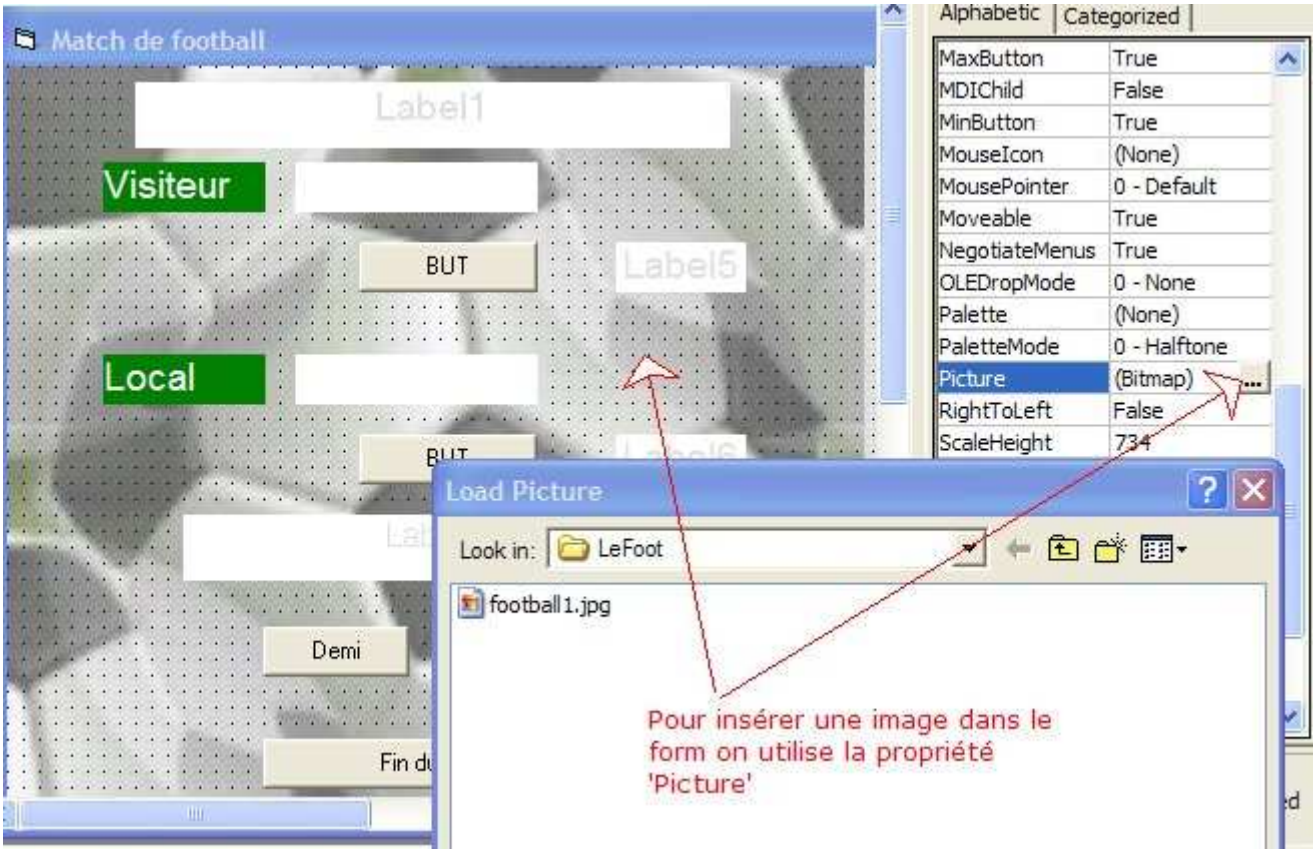
PREMIER EXERCICE

Comme premier exercice en VB vous allez créer la feuille que vous voyez. Vous devez mettre sur la feuille 4 ***CommandButton*** et 9 ***Label***. Ces contrôles devraient être alignés à peu près comme l'illustration.



Une fois les contrôles placés vous pouvez ouvrir la fenêtre ***Properties*** en cliquant avec le bouton droit de la souris. La fenêtre reste ouverte et à chaque fois que vous cliquez sur un contrôle vous voyez les propriétés de ce contrôle s'afficher. Expérimentez avec ***Caption, BackColor, ForeColor et Alignment***. Ne changez pas ***(Name)*** pour l'instant. Remarquez aussi que les propriétés ne sont pas les mêmes pour un ***Button*** et pour un ***Label*** - tous les contrôles, incluant la feuille elle-même, ont des propriétés différentes.

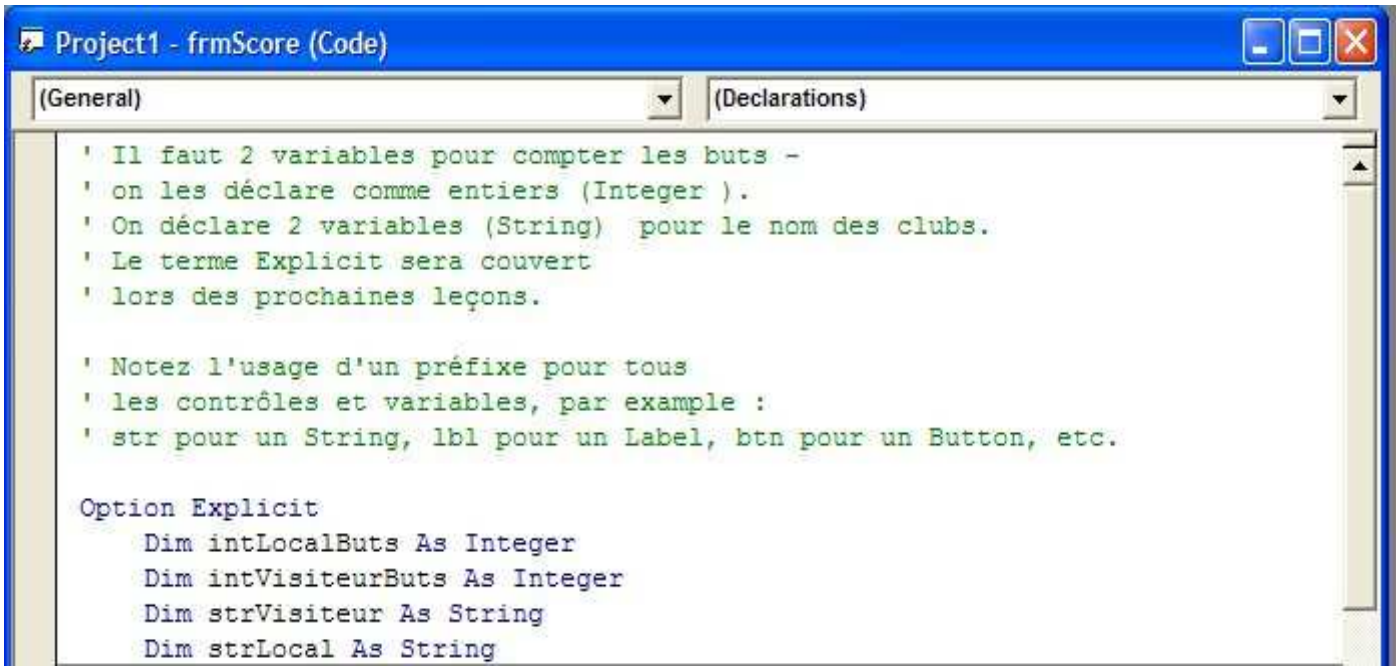

Match de football
Label1
Visiteur
BUT
Label3
Local
BUT
Label4
Label5
Demi
Fin du match
On identifie les contrôles qu'on veut changer et on assigne les propriétés appropriées
Forms
frmScore (Score.frm)
Properties - lblLocalButs
Alphabetic
Categorized
Alignment
2 - Center
Appearance
1 - 3D
AutoSize
False
BackColor
&H00FFFFFF&
BackStyle
1 - Opaque
BorderStyle
0 - None
Caption
DataField
DataMember
DataSource
DragMode
0 - Manual
Enabled
True
Font
ForeColor
00E0E0E0&
Palette
System
Left



ÉCRIRE LE CODE VB

Pour ouvrir l'éditeur de code on fait un double clique sur la feuille. La première chose à savoir est qu'on aura besoin de 2 compteurs dans le programme. Comme vous savez, un compteur est simplement une variable de type numérique. On doit donc déclarer ces variables. On le fait dans la section *General Declarations* pour que les variables soient accessibles à tous les objets de la feuille. On pourrait déclarer chaque variable à l'intérieur d'une procédure - un *Private Sub* - mais ces variables seraient alors locales et ne seraient valables que pour la procédure où elles sont nommées. Nous verrons les détails concernant les différents types de variables au prochain chapitre.

Notre application ne contient encore qu'un seul objet : le form Scoring. Quand on voudra lancer l'application, faire un *Start*, la première action qui va se passer est que la feuille Scoring va s'ouvrir. L'action d'ouvrir est un *event*, dans ce cas, le *Open event for Form Scoring*. En programmation VB on écrit toujours du code pour des *events*. Donc, si on veut exécuter certaines tâches lors de l'ouverture de la feuille, comme initialiser des variables locales ou donner des valeurs de départ aux propriétés des contrôles, on programme le *Open* de la feuille.
Dans l'éditeur de code on choisi Form dans le premier ListBox et Load dans le deuxième ListBox, ce qui génère une procédure *Private Sub Form_Load()*.
Maintenant on code les actions qu'on veut voir lorsqu'on clique sur un bouton de commande. Encore on invoque l'éditeur soit en faisant un double-clique sur le bouton lui-même ou dans l'éditeur, en choisissant le nom du bouton.
À tout moment on peut tester l'application en faisant *Start*. Si le résultat n'est pas satisfaisant, on revient au mode *Design* et on modifie l'interface.



Une fois qu'on a maitrisé la technique pour coder le premier bouton, on peut coder le deuxième facilement. Cependant, on doit faire les changements appropriés pour la situation.

```
'Ces instructions s'exécutent lorsqu'on click
'le bouton "But visiteur".

Private Sub btnVisBut_Click()
    intVisiteurButs = intVisiteurButs + 1
    lblVisButs.Caption = intVisiteurButs

    If intVisiteurButs > intLocalButs Then
        lblMessage.Caption = strVisiteur & " EN AVANCE!"
    ElseIf intLocalButs > intVisiteurButs Then
        lblMessage.Caption = strLocal & " EN AVANCE!"
    Else
        lblMessage.Caption = "ÉGALITÉ!"
    End If
End Sub
```

III. Le Langage VB

Comme vous savez de l'exercice précédent, pour ouvrir l'éditeur de code de VB vous devez soit cliquer sur l'icon **View Code** dans la fenêtre Project Explorer, cliquer sur View-->Code dans le menu ou cliquer sur View code avec le bouton droit de la souris ou encore faire un double-clique sur un objet.
Vous noterez que dans la ligne de menu les fonctions ***Delete, Cut, Copy, Paste, Find, Replace, etc.*** fonctionnent toutes comme dans un éditeur de texte ordinaire.

LE LANGAGE VB: TECHNIQUES DE BASE

Lignes de code

Le langage VB n'est pas très rigide: les espaces, indentations, etc. n'ont pas d'importance pour le compilateur. Cependant, elles en ont pour l'instructeur et vous devrez respecter les techniques de base concernant la lisibilité du code.
En général on écrit une commande par ligne; pour la lisibilité il est parfois préférable de mettre la commande sur 2 lignes en utilisant le caractère de continuation (espace underscore) _
La ligne suivante est correcte:

```
Data1.RecordSource = "SELECT * FROM Titles"
```

Mais je pourrais aussi l'écrire en 2 lignes dans le code:

```
Data1.RecordSource = _
    "SELECT * FROM Titles"
```

Dans le cas contraire on pourrait écrire 2 commandes sur une ligne en les séparant par un : mais ce n'est pas une pratique recommandée!
L'utilisation de majuscules ou minuscules n'a pas d'importance, sauf pour la lisibilité.

Commentaires

Le caractère de commentaire est l'apostrophe '

On recommande l'usage de commentaires dans les codes partout ou des explications sont requises.

Notez qu'on ne peut pas mettre un commentaire après un caractère de continuation.

Les commentaires peuvent être au début d'une ligne ou après le code comme: **' Ceci est un commentaire**

```
nom.Caption = "Michel"      ' Ce commentaire suit une commande
```

Noms de variables

- Le nom doit commencer par une lettre

- Maximum de 40 caractères

- Ne doit pas contenir d'espaces; peut contenir des signes excepté ceux qui décrivent un data type: ! Single, # double, % integer, $ string, & long, @ currency

(le trait d'union - quoique légal est fortement déconseillé car il porte à confusion avec l'opération moins; utilisez plutôt le underscore comme: nom_famille ou les majuscules comme: NomFamille)

-Ne doit pas être un mot réservé (qui fait partie du code)

Types de données

Data type	Storage size	Range
Byte	1 byte	0 to 255
Boolean	2 bytes	True or False
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values

Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.0000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any Object reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16 bytes	Any numeric value up to the range of a Double
Variant (with characters)	22 bytes + string length	Same range as for variable-length String
User-defined (using Type)	Number required by elements	The range of each element is the same as the range of its data type.

"NAMING CONVENTIONS"

Quand on crée des contrôles en VB, l'éditeur leur donne automatiquement un nom. Par exemple, le premier bouton sera Command1 et le vingtième sera Command20. Dans le code, quand on voudra référer au bouton pour exécuter les calculs, est-ce Command8 ou Command12? Et une variable qu'on a définie, est-ce une "String" ou une "Single" ou une "Double"? C'est difficile de se rappeler de tous les contrôles et les variables qu'on crée pour pouvoir s'y retrouver par

la suite si on n'utilise pas un système de référence.
Pour nommer les objets en VB il y 2 facteurs à considérer:

- Utiliser toujours un nom significatif pour chaque objet
- Utiliser le préfixe standard pour chaque objet tel que suggéré par le langage

Donc: **cmdExit** au lieu de Command8 pour le bouton "Exit",
txtHeures au lieu de Text3 pour la saisie du nombre d'heures
lblTitle au lieu de Label7 pour l'affichage des titres de Form.

Voici le tableau des préfixes suggérés par VB

Variable Data Type	Prefix	Example
Boolean	bln	blnFull
Byte	byt	bytDaysInMonth
Currency	cur	curPoundsSterling
Date (Time)	dtm	dtmStart
Double	dbl	dblAstronomicalDistances
Integer	int	intNumberOfEmployees
Long	lng	lngProfits
Single	sgl	sgl
String	str	strSurname
User-Defined Type	udt	udtStaffDetails
Variant	vnt	vntChartData

Control Type	Prefix	Example
Animated button	ani	aniEmptying
Check box	chk	chkWriteOnly
Combo box	cbo	cboLanguage
Command button	cmd	cmdCancel
Common Dialog	dlg	dlgSave
Data Control	dat	datStock
Data-bound Combo	dbcbo	dbcboActicleType

Data-bound Grid	dbgrd	dbgrdStockItems
Data-bound List box	dblst	dblstAccountCodes
Directory list box	dir	dirTarget
Drive list box	drv	drvSource
File list box	fil	filPick
Form	frm	frmMainMenu
Frame	fra	frmPrinters
Graph	gra	graSharePrices
Grid	grd	grdQuantities
Horizontal Scroll Bar	hsb	hsbHueColor
Image	img	imgBitMap
Label	lbl	lblHelpUser
List Box	lst	lstColorCodes
MCI	mci	mciSoundEffects
MDI Child Form	mdi	mdiJuly
Menu	mnu	mnuFileOpen
MS Tab	mst	mstDays
OLE	ole	oleExcel
Picture Box	pic	picMemoryLeft
ProgressBar	prg	prgConverting
Report	rpt	rptEndofYear
RichTextBox	rtf	rtfDiary
Shape	shp	shpSquare
Slider	sld	sldWindSpeed
Spin button	spn	spnTicketsRequired
StatusBar	sta	staInformation
Text Box	txt	txtInputText
Timer	tmr	tmrStartAlarmCount
Vertical Scroll Bar	vsb	vsbRatios

Déclaration de variables

Déclaration explicite

Définir une variable au début d'une procédure ou dans la section "Déclarations" en utilisant **Dim**

```
Dim UnEntier As Integer
Dim NomClient As String
Dim MontantDu As Currency
```

Normalement, les variables déclarées dans une procédure sont détruites lorsqu'on quitte la procédure; si on veut les garder on peut les déclarer avec **Static** au lieu de **Dim** comme:

```
Static TotalOtt As Integer
```

Par exemple, codez les deux procédures qui suivent et comparez-les:

```
Private Sub Command1_Click()
   Dim I As Integer
   I = I + 1
   Command1.Caption = I
End Sub
Private Sub Command2_Click()
   Static I As Integer
   I = I + 1
   Command2.Caption = I
End Sub
```

Déclaration implicite

Déclaration dynamique, faite "on the fly" dans le code: Dim Total1, Total2 As Integer

```
Total3 = Total1 + Total2
Total4% = 0
Montant5# = 12.34
```

MonNom$ = "Michel" En général, les **déclarations implicites ne sont pas recommandées** car elles rendent le code plus difficile à lire et à comprendre.

Constantes

Une constante est une valeur qui ne change pas au cours de l'exécution d'une procédure; on la déclare avec **Const** comme: Const ValeurPi = 3.1416

Portée des variables (Scope)

Une variable déclarée dans une procédure avec l'instruction Dim est **locale**; lorsqu'on sort de la procédure la variable n'existe plus.
Si on déclare la variable dans la section General/Déclarations avec Dim, la variable est **locale au module**; elle est disponible pour toutes les autres procédures de la feuille.
Si on déclare la variable dans la section General/Déclarations d'un module (et non d'une feuille), avec l'instruction **Public au lieu de Dim**, la variable est **globale** et elle est disponible à l'application toute entière.

Opérateurs

Les opérateurs arithmétiques habituels sont disponibles:

+ - * / ^

Le signe & est utilisé pour une concaténation de chaines: Dim alpha, beta, chaine As String
alpha = "Chaine divisée "
beta = "en deux parties"
chaine = alpha + beta Avec les variables de type Variant le signe + peut aussi dénoter une concaténation si les deux variables contiennent des chaines mais, on suggère de toujours utiliser le &
Les opérateurs de comparaison habituels sont utilisés:

= > < <= >= <> ainsi que **AND OR NOT IS** et **LIKE**

Quelques fonctions VB utiles

MsgBox()

Le format général est :

StrReponse = MsgBox(prompt[, buttons] [, title] [, helpfile, context])

Le seul paramètre obligatoire est *prompt*

Par exemple, pour afficher:

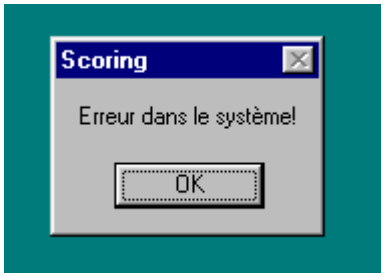


Fig. 3-3

on écrit:

DIM strReponse As String

strReponse = MsgBox("Erreur dans le système!")

InputBox()

Le format général est :

strReponse = InputBox(prompt [, title] [, default] [, xPos] [, yPos])

Le seul paramètre obligatoire est *prompt*

La valeur entrée par l'utilisateur sera stockée comme type Variant dans strReponse.

Puisque la valeur retournée est de type Variant, on pourrait aussi saisir des valeurs numériques.

Par exemple, pour saisir le nom et la note:

DIM strNom As String

DIM intNote As Integer

```
strNom = InputBox("Entrez le nom de l'étudiant")
intNote = InputBox("Entrez la note")
```

Les fonctions Is ...

Les fonctions IsNumeric, IsDate retourne True si un paramètre est de type voulu. Normalement utilisées dans une condition, comme:

```
If IsNumeric(ValeurEntree) Then ...
```

Si la valeur entrée est de type numérique, le code après le Then est exécuté.

La structure de décision

La structure de décision est comme dans tous les autres langages:

```
If (condition est vraie) Then
    (commandes)
ElseIf (autre condition est vraie)
    (commandes)
Else
    (commandes)
End If
```

La structure de cas

Aussi comme dans les autres langages:

```
Select Case Pourcent
Case Is >= 90
   Lettre = "A"
Case 60 to 89
   Lettre = "B"
Case Else
   Lettre = "F"
End Select
```

Notez qu'il y a plusieurs façons d'exprimer la condition du Case:

avec les signes < et > il faut utiliser le IS

on peut spécifier un range: 60 TO 89

on peut spécifier des valeurs: 44, 46, 55, 62

Le DO ... LOOP

```
Do While condition
    instructions
Loop
```

et aussi

```
Do Until condition
    instructions
Loop
```

Le FOR ... NEXT

```
For compteur = debut To fin
    instructions
Next
```

Les Tableaux - "Arrays"

En VB on déclare un tableau comme une variable ordinaire, avec **DIM**.

Par exemple:

```
Dim Mois(1 to 12) As String
```

Déclare un tableau Mois qui contiendra 12 valeurs avec index de 1 à 12. Mois(1) = "Janvier", etc. Le tableau:

```
Dim Département(6) As String
```

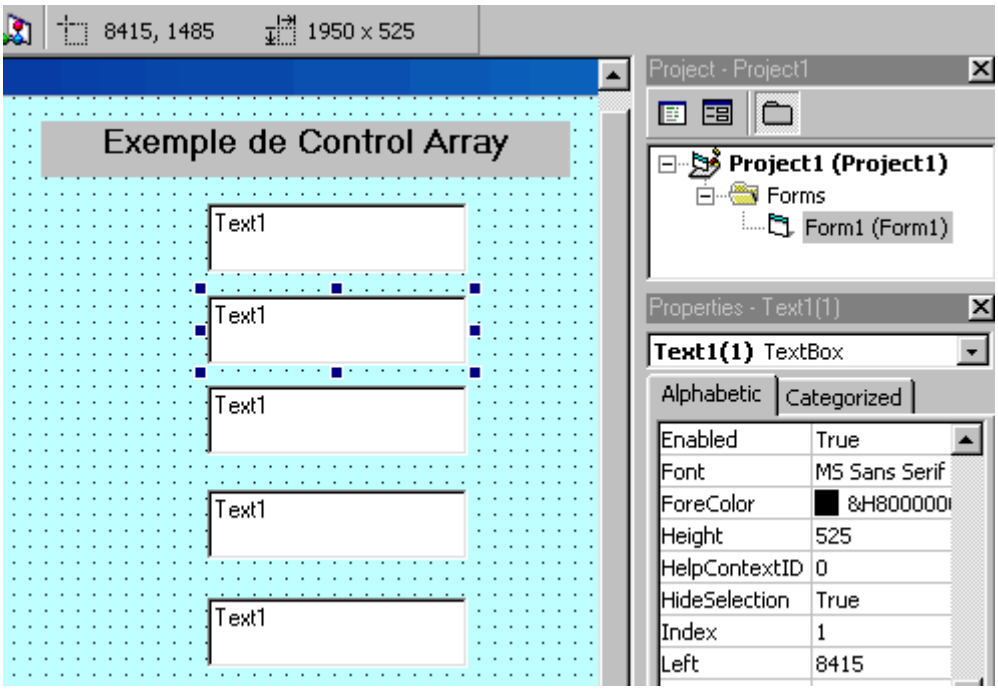
Déclare un tableau de 6 éléments avec le premier index = 0. On peut aussi déclarer un tableau de grandeur indéterminée **(unbound)** en utilisant la fonction **Array()** dans le code:

```
Dim Semaine, Jour
Semaine = Array("lundi", "mardi", "mercredi")
Jour = Semaine(1)       'retourne mardi
```

Le Control Array

Un tableau de contrôles et non un tableau qui contrôle

Si j'ai un form qui doit contenir 10 TextBox, je crée le premier et je le copie et le colle 9 fois. Je spécifie "Yes" à la question de control array. J'obtiens ceci:



Maintenant, je peux utiliser un For ... Next pour faire des opérations sur les TextBox:

Pour initialiser tous les contrôles à 0:

```
Dim i As Integer
For i = 0 to 9
    Text1(i).Text = 0
Next i
```

Utiliser plusieurs Forms dans un Projet

Un projet peut contenir plusieurs feuilles (Forms). Il s'agit d'abord de créer une nouvelle feuille en faisant Add form. Ensuite, pour l'ouvrir il y a deux façons:

1) on peut changer le **Startup object** dans les propriétés du Projet pour que notre nouvelle feuille s'ouvre en démarrant le Projet:

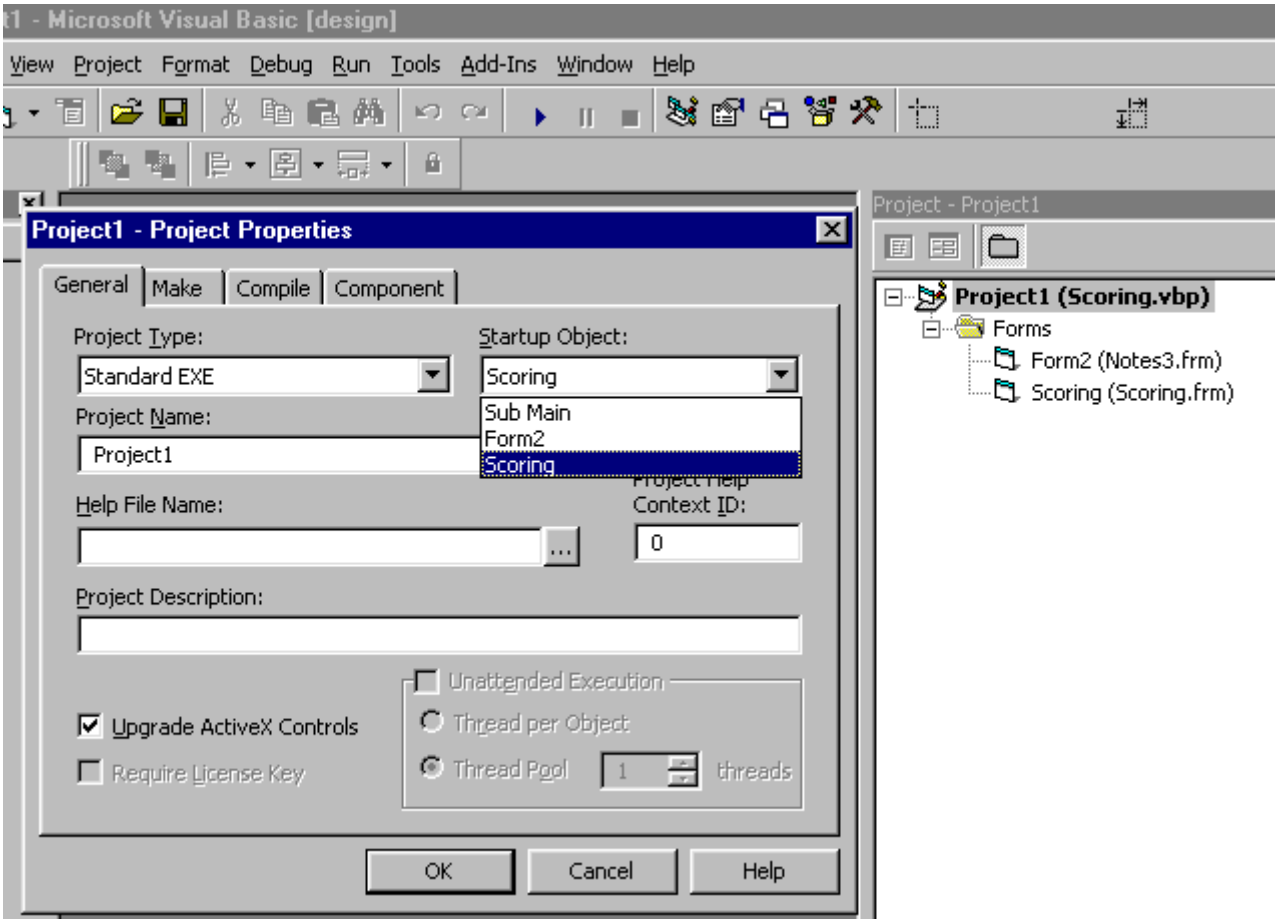


Fig. 3-1

2) on peut créer un nouveau bouton sur une feuille existante et utiliser ce bouton pour ouvrir une autre feuille:

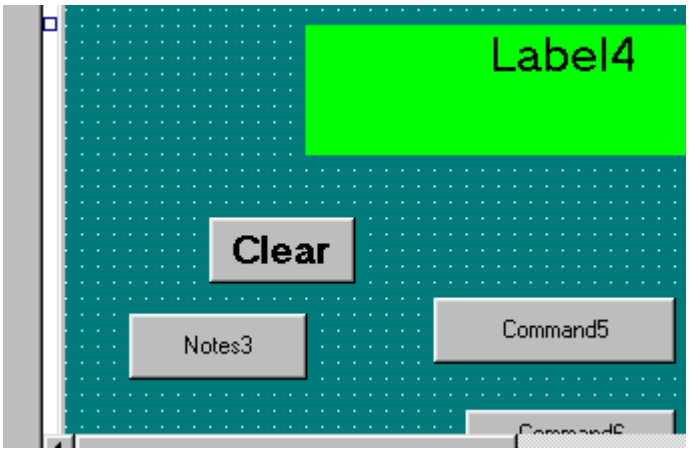


Fig. 3-2

```
Private Sub Command7_Click()
    Load Form2
    Form2.Show vbModeless
End Sub
```

Fig. 3-3

IV. Création d'un form

Design de l'application

Avant de commencer à créer les objets en VB, il faut normalement planifier le travail à faire. Cela veut dire d'utiliser un papier et un crayon et de dessiner un plan **ou** modèle **de l'application. En architecture on dessine un plan de maison. En informatique on fait** la modélisation de l'application**.**

LES NORMES DE CONCEPTION "DESIGN GUIDELINES"

Principes de design

- **L'utilisateur a le contrôle:**
  L'utilisateur doit sentir qu'il contrôle le logiciel plutôt que d'être contrôlé. L'utilisateur initie les tâches au lieu de toujours réagir aux actions du programme. L'utilisateur doit être capable de personnaliser certains aspects de l'interface - la couleur, les fonts, etc.

- **Consistence:**
  La Consistence permet à l'utilisateur de transférer des habiletés acquises à un nouvel environnement. Quand l'interface est familière et prévisible, l'utilisateur peut concentrer sur la tâche sans avoir à se soucier d'apprendre des nouvelles commandes, actions, etc.
  Consistence dans le produit: commandes agissent de la même façon partout - "Couper" veut toujours dire "Couper et mettre dans le Clipboard" - ne doit pas effacer à un endroit et couper à

un autre.
Consistence avec Windows: utiliser les actions de Windows dans la mesure du possible - elles sont connues de l'utilisateur
Consistence des symboles: utiliser les symboles appropriés- la poubelle a une signification différente de l'incinérateur - l'utilisateur reconnait qu'un document incinéré ne pourra pas être récupéré.

- **Permissivité** (en anglais on dit "**Forgiveness**"):
  Permet à l'utilisateur d'explorer, d'essayer différentes choses sans faire sauter tout le système - l'utilisateur va faire des erreurs et il faut que le logiciel pardonne ses fautes et puisse récupérer facilement - la touche "Undo" doit être disponible dans la mesure du possible.

- "**Feedback**":
  On donne toujours du feedback en réponse aux actions. Le feedback peut être visuel ou auditif. Par exemple, on change le curseur pendant qu'un document charge ou on montre une barre d'état qui affiche le progrès des opérations. L'utilisateur devient frustré au bout de quelques secondes devant un écran qui semble mort et un clavier qui ne répond pas aux commandes.

- **Esthétique**:
  Esthétique = beauté, apparence visuelle. L'environnement doit être plaisant, agréable aux yeux; l'apparence doit contribuer à la compréhension de l'information présentée.

- **Simplicité**:
  L'interface doit être simple sans être simpliste, facile à maitriser et convivial. Il doit aussi fournir toutes les fonctions dont

l'application a besoin. Il faut balancer la simplicité et la fonctionnalité de l'interface. Par exemple, en utilisant des menus contextuels (Pop-up menu), on maintient la fonctionnalité sans encombrer l'écran de trop de fonctions à la fois.

Méthodologie de design

- **L'équipe**: on requiert des habiletés en développement, design visuel, graphisme, rédaction technique, ergonomie et testing. Il est rare de trouver toutes ces habiletés chez un seul individu - il faut donc former une équipe qui regroupe des personnes ayant l'expertise dans ces différents domaines.

- **Le cycle de design**:
    - **Le design**: concevoir l'application - établir le but final, comprendre les exigences du client et comprendre les utilisateurs. Lorsqu'on sait ce qu'on doit réaliser, on dessine un premier plan sur papier et, si possible, on le valide avec l'utilisateur. On incorpore dans le dessin les suggestions du client et on raffine le plan.

    - **Le prototypage**: un prototype est un modèle physique du produit final qui n'a pas nécessairement toutes les fonctions du vrai produit. Les outils RAD (PB, VB, Delphi) sont très utiles pour cette tâche. Par exemple, on peut créer un Form (ou un Window) qui a toutes les caractéristiques du produit final mais qui n'a pas la fonctionnalité, c'est à dire qu'il n'a pas de code; mais, du point de vue visuel, on peut l'évaluer et le tester. Pour développer le produit final on modifie graduellement le

prototype de sorte à ce qu'il possède éventuellement toutes les fonctions requises.

- **Le testing**: on implique l'utilisateur dans le processus de testing - on appelle ceci "**usability testing**" car on évalue si le produit est utilisable et convivial. Il faudra aussi faire du "**quality testing**" pour trouver les bugs dans le code et s'assurer que les calculs sont exacts.

Le design visuel - création du GUI

- **Principes de base**:
  Le design visuel de l'application est beaucoup plus que de la décoration - c'est un outil de communication important. Le design visuel détermine si l'utilisateur capte le message ou bien s'il reste confus et frustré.
  En créant le design visuel on essaie d'utiliser l'écran à son maximum. On utilise la composition, les formes, les couleurs, le contraste et le focus afin de communiquer le message de la façon la plus efficace possible.

- **Composition et organisation**:
  Détermine comment les éléments sont placés à l'écran; on lit un écran comme une page - l'œil est attiré par les couleurs avant le noir et blanc, par un dessin avant un texte, etc. En plaçant les éléments il faut penser à comment on veut qu'ils soient perçus.

  - **La séquence**: placée les informations importantes en premier, placé les tâches dans l'ordre où elles sont exécutées.

- **Le focus**: attirer l'attention sur les points importants en les isolants ou avec la couleur, par exemple.
- **Relations entre les éléments**: indiquer quand différents éléments ont un rapport entre eux; par exemple, si une adresse est composée de quatre champs, mettre un cadre autours.
- **Le déroulement, "flow"**: tous les éléments doivent se suivre logiquement; ils doivent aussi avoir tous une raison d'être et contribuer à l'effet visuel total de l'interface.

- **La couleur**:
  - La couleur peut servir à augmenter l'efficacité du message en attirant l'attention, en créant des associations dans l'esprit ou en établissant un état psychologique.
  - La couleur est subjective - ce qui est plaisant pour un peut être déplaisant pour l'autre.
  - L'interprétation des couleurs peut être culturel - une couleur n'a pas toujours la même signification.
  - Un certain nombre de personne ont des difficultés avec les couleurs; environ 9% des hommes souffrent d'une forme quelconque de daltonisme.
  - La couleur devrait être secondaire dans l'interface - on le crée en monochrome d'abord et on ajoute la couleur pour faire ressortir certains points.

- Utiliser une palette restreinte - même si on a 16 million de couleurs disponibles, on n'est pas obligé de toutes les utiliser. L'utilisation de trop de couleurs rend l'interface chargé et compliqué. En général les couleurs sobres sont préférables aux couleurs voyantes. On doit aussi éviter les contrasts dans le "background" et "foreground" - du rouge sur fond vert, par exemple, est déplaisant pour l'œil.

Propriétés du Form

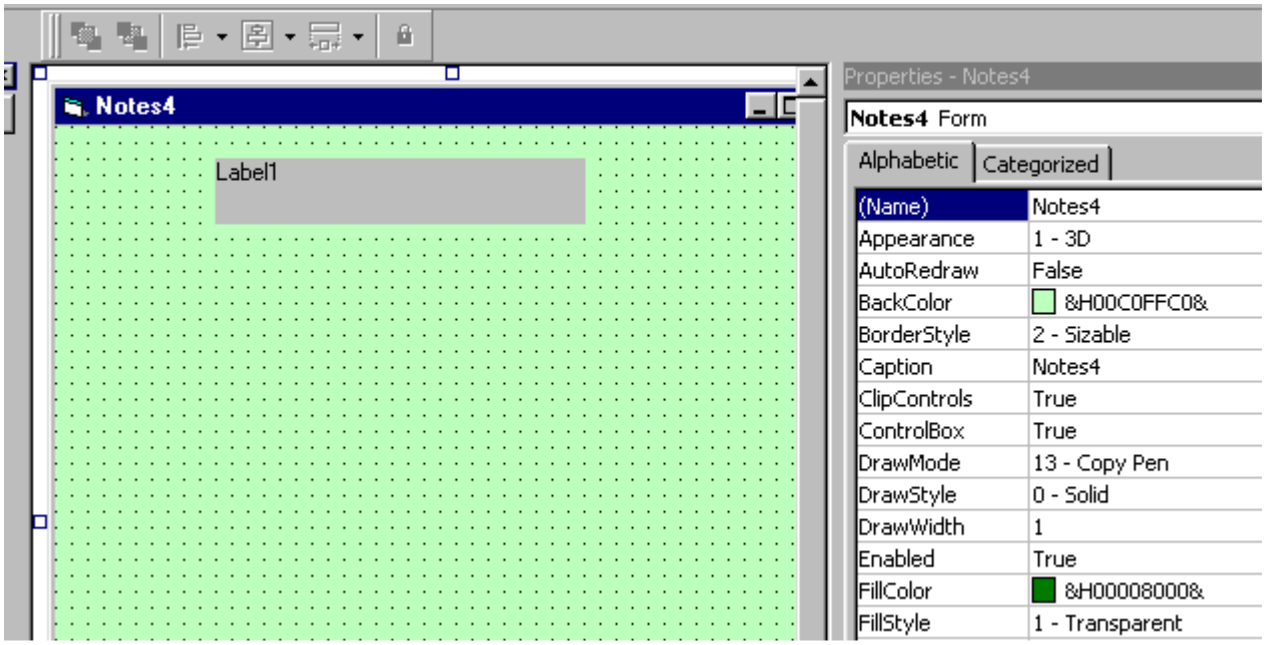


Fig. 4-1

(Name): même nom qu'on lui donne en le sauvegardant- le nom qu'on utilise lorsqu'on réfère à la feuille dans le code.
Caption: nom qui apparaît dans la ligne Titre - information seulement - pas référencé dans le code.
BorderStyle: ligne titre voulue et si la grandeur peut être changée à l'exécution - 1 et 2 sont des Forms ordinaires.
BackColor: couleur de la feuille.

StartUpPosition: où on veut que la feuille s'affiche sur l'écran à l'exécution.

Au sujet des noms (la propriété "Name"): on devrait utiliser un système de terminologie standard à travers toutes les applications VB. Ceci veut dire d'identifier tous les objets ainsi que les variables avec un préfixe qui est facile à reconnaitre partout dans le code.

Pour établir la grandeur et la position de la feuille lors de l'exécution, on le fait habituellement dans le code.

Dans le **Form_Load event** on écrit le code qui sera exécuté en lançant l'application:

```
Private Sub Form_Load()
        Me.Width = 8000
        Me.Height = 6000
        Me.Left = (Screen.Width - Me.Width) / 2
        Me.Top = (Screen.Height - Me.Height) / 2
End Sub
```

Les mesures pour l'objet **Screen** sont en **twips**:

517 twips = 1 cm

1440 twips = 1 pouce.

Ajouter des *controls* au Form

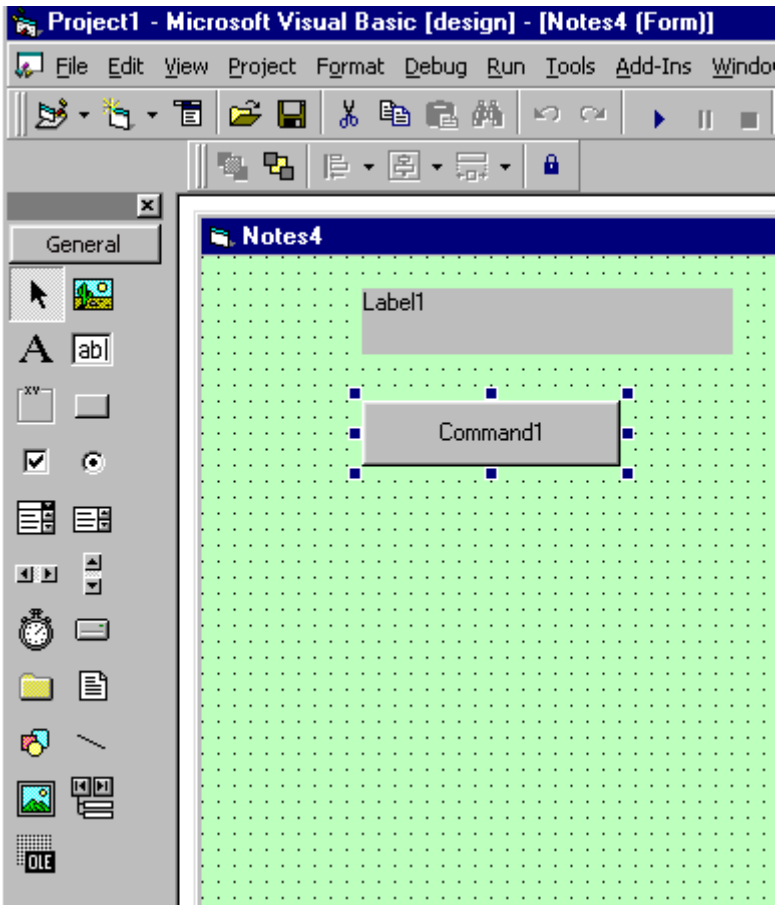


Fig. 4-2

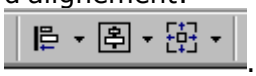
Pour mettre un control dans le form il faut d'abord sélectionner le control dans boîte d'outils qui est à la gauche de l'écran - si la boîte n'est pas là, on l'active avec le bouton Boîte d'outils.

Dans la boîte d'outils on clique sur le control voulu et puis on clique sur la feuille à l'endroit où on veut placer le control et on le traîne à la grandeur qu'on désire.

Une fois le control en place on peut changer sa grandeur, le déplacer ou l'enlever (Delete) ou faire **Undo** si on s'est trompé.

Pour faire une copie d'un control, on le sélectionne et on fait **Copy** et **Paste** - on répond "Non" à la question **Control array** parce qu'on veut des controls individuels - un nouveau nom sera assigné automatiquement au control, Label2, par exemple, mais

il faudra changer le **Caption**.
Pour sélectionner plusieurs controls on clique en haut, à gauche du premier et on traine la souris jusqu'au coin droit, en bas du rectangle de sélection; on peut aussi le faire avec **(Control)(Click)**.
Pour aligner un groupe de controls ou pour les mettre de la même grandeur, on les sélectionne tous et on utilise les boutons d'alignement:



.
Le bouton **Undo** est bien utile si on se trompe de sélection.

Propriétés communes des controls

Name: le nom interne de l'objet, tel qu'il sera utilisé dans le code.

Appearance: si le control possède ou non un aspect en relief.

BackColor, ForeColor couleur du fond et couleur du texte

Visible, Enabled valeur True/False si le control est visible et s'il est actif - par exemple, un bouton peut être désactivé (facture.enabled = false) si on ne veut pas que l'utilisateur le clique à certains moments dans le traitement.

TabIndex l'ordre d'activation des controls sur la feuille - détermine quel control aura le **Focus** quand l'utilisateur fait un Tab.

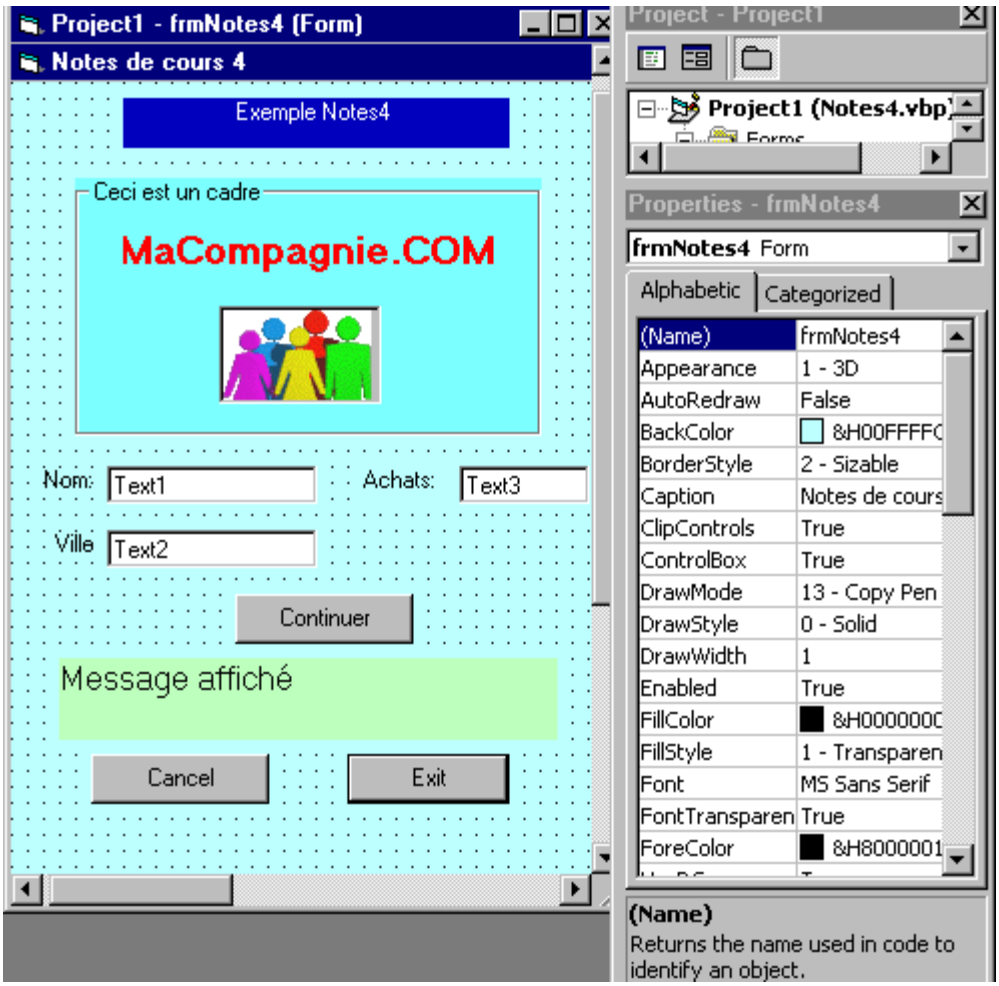
Label, Picture Box

Label: un titre, une étiquette - utilisé pour afficher une information.

PictureBox: une image - utilisé pour afficher un fichier contenant une image (.BMP, .GIF, etc) - sert pour un logo d'entreprise, par exemple

Saisie de données

TextBox: sert à saisir une donnée au clavier - la donnée saisie est gardée dans la propriété **Text** de l'objet et on peut s'en servir à partir de là:



cmdContinuer | Click

```
Option Explicit
    Dim curAchats As Currency
    Dim strNomClient As String
    Dim strVilleClient As String
Private Sub Form_Load()
    lblResultat.Caption = " "
    txtNom.Text = " "
    txtVille.Text = " "
    txtAchats.Text = " "
End Sub
Private Sub cmdContinuer_Click()
    strNomClient = txtNom.Text
    strVilleClient = txtVille.Text
    curAchats = txtAchats.Text
    lblResultat.Caption = "Client est " _
        & strNomClient _
        & " , habite à " & strVilleClient _
        & Chr(13) & "et achète pour " _
        & curAchats
End Sub
Private Sub cmdCancel_Click()
    Form_Load
End Sub
Private Sub cmdExit_Click()
    Unload Me
    End
End Sub
```

CommandButton, OptionButton et CheckBox

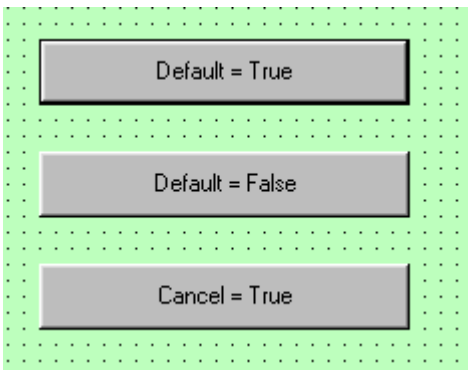


Fig. 4-5

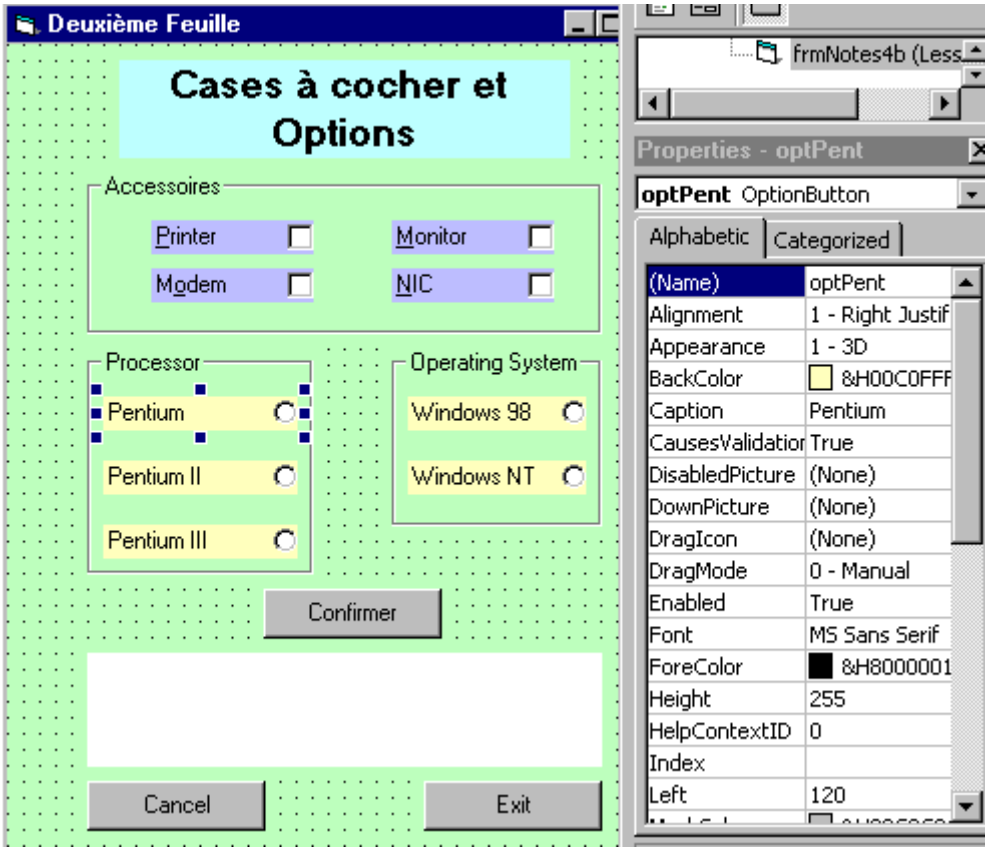
CommandButton (cmd) - un objet 3D - pas de couleur - effet de s'enfoncer quand on clique

Si **Default = True**, marge du bouton est foncée et Click sera activé par **Enter**

Si **Cancel = True** Click sera activé par **Esc**

OptionButton Par définition un bouton d'option fait partie d'un groupe; on crée d'abord un cadre (**Frame**) et on y met les boutons d'options. Le Frame détermine que les boutons font tous partie du groupe. Si j'ai besoin de 2 groupes de boutons, je dois créer 2 Frames différents.

À l'exécution on doit tester la propriété **Value** de chaque bouton - si elle retourne True, le bouton est celui qui a été sélectionné (seulement un bouton du groupe retourne True) et j'exécute l'action appropriée.



(General) (Declarations)

```
Option Explicit

Private Sub Form_Load()
    chkPrinter.Value = 0
    chkMonitor.Value = 0
    chkModem.Value = 0
    chkNic.Value = 0
    optPent.Value = False
    optPent2.Value = False
    optPent3.Value = False
    optWin98.Value = False
    optWinnt.Value = False
    lblMsg.Caption = ""
End Sub
```

cmdConfirmer Click

```
Private Sub cmdConfirmer_Click()
    Dim strPrNom As String, strOsNom As String
    Dim vntAccPr, vntAccMn, vntAccMod, vntAccNic

    'Si pas de processeur choisi, afficher erreur
    'et refaire la saisie

    If optPent.Value = False _
            And optPent2.Value = False _
            And optPent3.Value = False Then
        MsgBox ("Vous devez choisir un Processeur")
        optPent.SetFocus
    Else
        If optPent.Value = True Then
            strPrNom = "Pentium"
        ElseIf optPent2 = True Then
            strPrNom = "Pentium II"
        Else
            strPrNom = "Pentium III"
        End If
    End If

    'SetFocus est une Method qui retourne le Focus
    '(le curseur) à un objet spécifié,
    'dans ce cas-ci, un bouton d'option
    'Voir "SetFocus Method" dans Help.
```

CheckBox: on se sert de la propriété **Value** pour tester si la boîte est vide (Value = 0), cochée (Value = 1) et on peut aussi regarder pour Indéfini (Value = 2)...

Voici le code pour l'application de boutons et de cases:

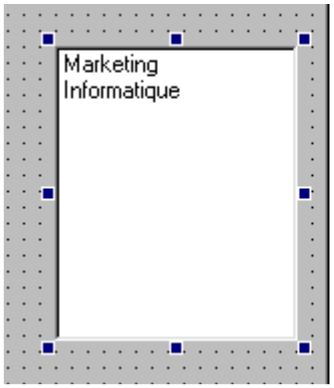
```
Option Explicit

Private Sub Form_Load()
   chkPrinter.Value = 0
   chkMonitor.Value = 0
   chkModem.Value = 0
   chkNic.Value = 0
   optPent.Value = False
   optPent2.Value = False
   optPent3.Value = False
   optWin98.Value = False
   optWinnt.Value = False
   lblmsg.Caption = ""
End Sub

Private Sub cmdConfirmer_Click()
   Dim strPrNom As String, strOsNom As String
   Dim vntAccPr, vntAccMn, vntAccMod, vntAccNic

   'Si pas de processeur choisi, afficher erreur
   'Et refaire la saisie

   If optPent.Value = False _
         And optPent2.Value = False _
         And optPent3.Value = False Then
      MsgBox ("Vous devez choisir un Processeur")
      optPent.SetFocus
   Else
      If optPent.Value = True Then
         strPrNom = "Pentium"
      ElseIf optPent2 = True Then
         strPrNom = "Pentium II"
      Else
         strPrNom = "Pentium III"
      End If
   End If

     'SetFocus est une Method qui retourne le Focus
   '(Le curseur) à un objet spécifié,
   'Dans ce cas-ci, un bouton d'option
   'Check if OS was selected - if no
   'Display error message; if yes, get its name.
   If optWin98.Value = False _
         And optWinnt.Value = False Then
```

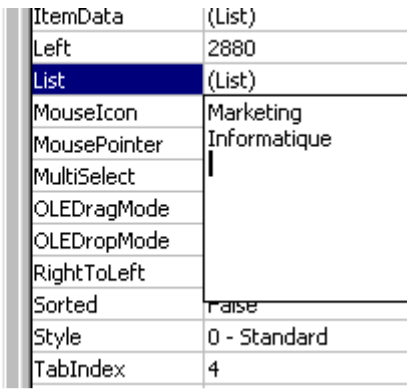
```
        MsgBox ("You must select an Operating system")
            optWin98.SetFocus
    Else
        If optWin98.Value = True Then
            strOsNom = "Windows 98"
        Else
            strOsNom = "Windows NT"
        End If
    End If

    'Verify which accessories were checked in order
    'to build output label.
    If chkPrinter.Value = 1 Then
        vntAccPr = "printer"
    End If
    If chkMonitor.Value = 1 Then
        vntAccMn = " monitor"
    End If
      If chkModem.Value = 1 Then
        vntAccMod = " modem"
    End If
    If chkNic.Value = 1 Then
        vntAccNic = " NIC"
    End If

    lblmsg.Caption = "You selected a " & PrName _
      & " with " & strOsNom & Chr(13) _
      & "and accessories: " & vntAccPr & vntAccMn _
      & vntAccMod & vntAccNic

     'If you want to force a line change in a Label,
     'insert a Chr(13) - the carriage return character-
     'In the string.
End Sub

Private Sub cmdCancel_Click()
    Form_Load
End Sub

Private Sub cmdExit_Click()
    Unload Me
    End
End Sub
```

Essayer le programme ci-dessus. Si vous avez des doutes, n'hésitez pas à contacter: *wasingface@gmail.com*

Les Listes

ListBox:
affiche une liste de choix dans une boîte - on sélectionne l'élément voulu en cliquant dessus - la liste peut être créée lors de la création du Form ou elle peut être créée dynamiquement pendant l'exécution du code (on verra la technique un peu plus loin).





Pour utiliser la liste il faut comprendre que la liste est en fait un tableau **(Array)** et que chaque item dans la liste possède un **index** qui permet de le retrouver.

Au sujet des tableaux: en VB on déclare un tableau comme une variable ordinaire, avec **DIM**. par exemple:
Dim Mois (1 to 12) As String
déclare un tableau Mois qui contiendra 12 valeurs avec index de 1 à 12. Mois(1) = "Janvier", etc. Le tableau:
Dim Département(6) As String
déclare un tableau de 6 éléments avec le premier index = 0. On peut aussi déclarer un tableau de grandeur indéterminée **(unbound)** en utilisant la fonction **Array ()** dans le code:

```
Dim Semaine, Jour
Semaine = Array ("lundi", "mardi", "mercredi")
Jour = Semaine(1)       'retourne mardi
```

Le ListBox est essentiellement un tableau "unbounded" où le premier item est ListIndex 0, etc.

Dans le **Click event du ListBox** on peut identifier l'index qui a été sélectionné: **ListIndex** ainsi que l'item lui-même avec l'expression: **objet.List(objet.ListIndex)**


```
Private Sub List1_Click()
    Dim nom As String
    Dim taux As Single

    nom = List1.List(List1.ListIndex)
    lb_NomDept.Caption = nom

    Select Case nom
        Case "Informatique"
        taux = 12.5
        Case "Ventes"
        taux = 8
        Case Else
        taux = 10
    End Select

    lb_Taux.Caption = taux
End Sub
```

Le **ComboBox** est une combinaison (de là le terme combo) d'un **TextBox** et d'un **ListBox**. Il permet à l'utilisateur de cliquer sur un item de la liste ou d'en entrer un nouveau. On utilise les 2 events **Click** et **Change** pour déterminer ce qui se passe.


```
Private Sub Combo1_GotFocus()
   Combo1.ListIndex = 1
   Combo1.BackColor = RGB(128, 0, 128)
   Combo1.ForeColor = RGB(255, 255, 255)
End Sub

Private Sub Combo1_Change()
   Lab_Saisie = Combo1.Text
End Sub

Private Sub Combo1_Click()
   Lab_Saisie = Combo1.List(Combo1.ListIndex)
End Sub

Private Sub Combo1_LostFocus()
   Combo1.ListIndex = -1
   Combo1.BackColor = RGB(255, 255, 255)
   Combo1.ForeColor = 0
End Sub
```

Si on veut charger la liste de la boîte Combo au moment de l'exécution, on écrit le code approprié dans le **Load event** de la feuille:

```
Private Sub Form_Load()
   Combo1.List(0) = "Marketing"
   Combo1.List(1) = "Publicité"
   Combo1.List(2) = "Ventes"
End Sub
```

V. Validation des données

Validation signifie s'assurer que les valeurs entrées sont bonnes. On valide afin de s'assurer qu'on passe des valeurs correctes aux calculs et pour éviter de faire "crasher" l'application.
Par exemple, si on entre une donnée qui n'est pas numérique, la procédure de calcul va s'arrêter sur une erreur de "Type mismatch" et l'utilisateur restera en panne.
Le MsgBox function

Lors de la validation vous allez probablement utiliser la fonction MsgBox( ) souvent. Le MsgBox que nous avons utilisé jusqu'à maintenant est le plus simple possible. Il y a d'autres versions du MsgBox qui vous permettent de préciser les intentions de l'utilisateur.
Par exemple:

```
DIM intMsg AS Integer
intMsg = MsgBox ("Erreur dans valeur", vbOKCancel)
If intMsg = 1 Then
    txtValeur.SetFocus
Else
    Exit Sub
End If
```

Si intMsg est 1, l'utilisateur a cliqué sur OK et on obtient une nouvelle valeur. Si intMsg est 2, l'utilisateur a fait Cancel et on veut quitter la procédure.
Les constantes qui sont utilisées:
VbOkOnly
vbOkCancel
vbAbortRetryIgnore
vbYesNoCancel
vbYesNo
vbRetryCancel
et les valeurs retournées:
1 vbOk Ok
2 vbCancel Cancel
3 vbAbort Abort
4 vbRetry Retry
5 vbIgnore Ignore
6 vbYes Yes
7 vbNo No

Il y a 4 événements qui sont généralement utilisés pour faire la validation:

Le **_Change**

Le **_KeyPress**

Le **_LostFocus**

Le **_Validate**

Le "Change event"

À noter au sujet de Change: l'événement Change est invoqué à **chaque changement** qu'on fait: entrer un caractère est un changement donc, Change s'exécute **à chaque caractère** qu'on frappe.

Voici un exemple de Change:

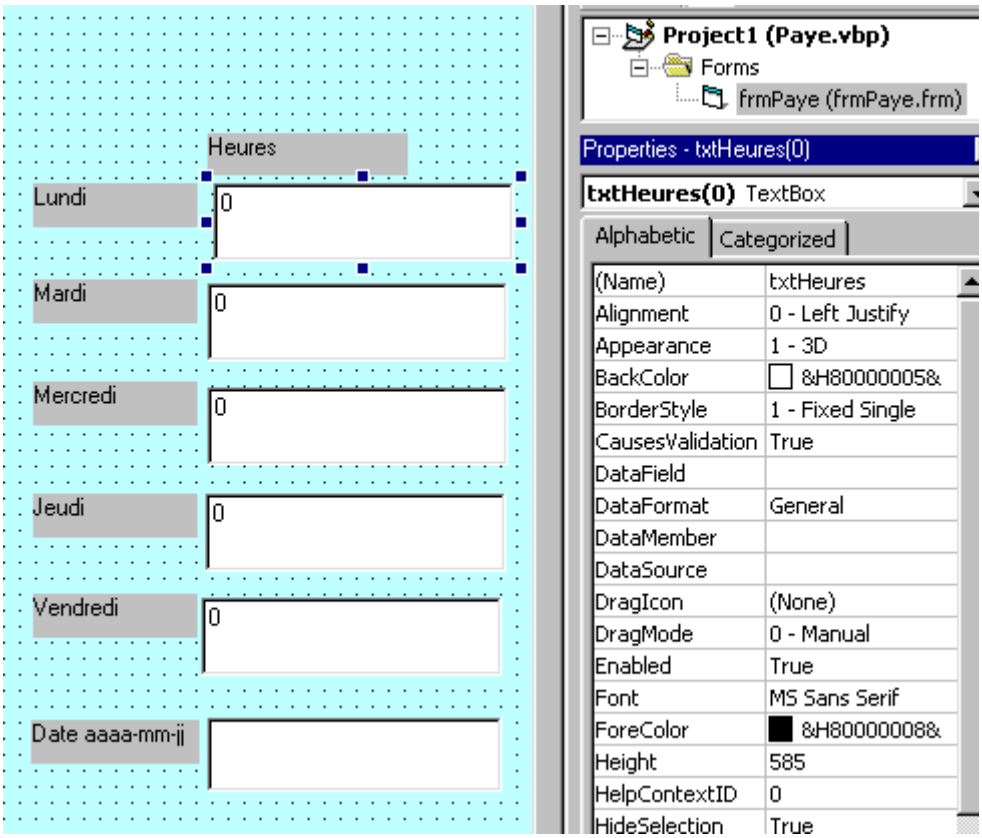
```
DIM intRep AS Integer
Private Sub txtHeures_Change ()
   If Not IsNumeric (txtHeures.Text) Then
  IntRep = MsgBox ("Doit être numérique", vbOKCancel)
   End If
   If intRep = vbCancel Then
           Exit Sub
   End If
End Sub
```

Le problème avec cette structure est que la lettre est affichée et il faut l'enlever avant de continuer.
Il y a un autre événement qu'on pourrait utiliser qui est plus facile: le KeyPress

Le "KeyPress event"

Le KeyPress saisi le code ASCII de la touche frappée, avant qu'elle soit affichée dans le TextBox. Donc, on peut la vérifier et l'ignorer si elle n'est pas bonne.

L'exemple illustre comment on valide dans un "control array".




```
Private Sub txtHeures_KeyPress(Index As Integer, KeyAscii As Integer)
    'Les chiffres 0 à 9 ont les codes ASCII 48 à 57
    '43 est le signe +, 45 est le signe - et 46 est le point.
    If (KeyAscii < 48 or KeyAscii > 57) _
        And Not KeyAscii = 43 _
        And Not KeyAscii = 45 _
        And Not KeyAscii = 46 Then
            KeyAscii = 0
    End If
End Sub
```

Voici comment on ferait la même chose avec Change

```
Private Sub txtHeures_Change(Index As Integer)
    If Not IsNumeric(txtHeures(Index)) Then
        MsgBox ("Doit être numérique")
    End If
End Sub
```

Le "Validate event"

Le TextBox a une propriété **CausesValidation** qui peut être True ou False. Si elle est True, le Validate event sera invoqué dès que j'essaie de quitter la boîte. Si je décide de ne pas valider, en réponse à une question, par exemple, je peux mettre le CausesValidation à False.

Voici un exemple de Validate avec un TextBox individuel:

```
Private Sub txtHeures_Validate(Cancel As Boolean)
  If txtHeures < 0 Or txtHeures > 100 Then
MsgBox ("Doit être un nombre entre 0 et 100")
Cancel = True
  End If
End Sub
```

Voici le Validate si le TextBox fait partie d'un control array - il faut inclure l'index:

```
Private Sub txtHeures_Validate(Index As Integer, Cancel As Boolean)
   If txtHeures(Index) < 0 Or txtHeures(Index) > 100 Then
         MsgBox ("Doit être un nombre entre 0 et 100")
    Cancel = True
  End If
End Sub
```

"Cancel = True" sert à garder le focus dans le champ actuel.

Donc, tant qu'il y a une erreur l'utilisateur ne peut pas quitter le TextBox.

On se sert aussi du Validate pour valider si la donnée entrée est une date avant d'aller plus loin:

```
Private Sub txtDate_Validate(Cancel As Boolean)
          If Not IsDate(txtDate.Text) Then
          MsgBox ("Doit être une date valide")
          Cancel = True
          End If
End Sub
```

Le "GotFocus event" et le "LostFocus event"

Voici comment on pourrait utiliser _GotFocus et _LostFocus pour changer la couleur de la boite, par exemple:

```
Private Sub txtHeures_GotFocus()
                txtHeures.BackColor = RGB(0, 0, 128)
                txtHeures.ForeColor = RGB(255, 255, 255)
        End Sub

        Private Sub txtHeures_LostFocus()
                txtHeures.BackColor = RGB(255, 255, 255)
        txtHeures.ForeColor = 0
        End Sub
```

Ou bien ceci pour sélectionner le texte déjà dans le TextBox - très utile quand on ne veut pas avoir à effacer le 0 initial avant de pouvoir entrer la valeur.

```
       Private Sub txtHeures_GotFocus()
                txtHeures.SelStart = 0
                txtHeures.SelLength = Len (txtHeures.Text)
        End Sub
```

Il est aussi possible de combiner ces fonctions dans une procédure et, en plus, utiliser le terme générique **ActiveControl** pour rendre la procédure encore plus utile.

Le terme **ActiveControl**, comme son nom l'indique, réfère au control actif en ce moment, sans avoir à le nommer.

J'appelle la procédure de sortie dans Validate plutôt que LostFocus parce qu'avec LostFocus, l'ActiveControl est déjà disparu au moment où l'événement est invoqué.

```
Private Sub txtHeures_GotFocus(Index As Integer)
   FocusIn
End Sub

Private Sub txtHeures_Validate(Index As Integer, Cancel As Boolean)
  If txtHeures < 0 Or txtHeures > 100 Then
        MsgBox ("Doit être un nombre entre 0 et 100")
     Cancel = True
  End If
  FocusOut
```

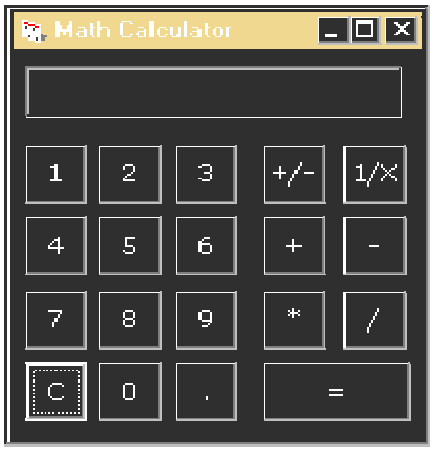
```
End Sub

Private Sub FocusIn()
   ActiveControl.BackColor = RGB(0, 128, 0)
   ActiveControl.ForeColor = RGB(255, 255, 255)
   ActiveControl.SelStart = 0
   ActiveControl.SelLength = Len(ActiveControl.Text)
End Sub

Private Sub FocusOut()
   ActiveControl.BackColor = RGB(255, 255, 255)
   ActiveControl.ForeColor = 0
End Sub
```

Finalement, notons qu'il y a certaines erreurs qui ne peuvent pas être validé par les événements des TextBox. Par exemple, si je veux m'assurer que tous les TextBox ont été remplis, je ne peux pas mettre le code dans le Validate ou le Change du TextBox car si l'utilisateur n'a pas touché ces boîtes, les événements ne seront jamais invoqués.

Il faut donc mettre ces validations dans le bouton Calcul, par exemple.

VI. Menu et Debug

Pratique avec les contrôles: la calculatrice
On vous demande de créer une application relativement simple mais, qui demande la maîtrise de plusieurs contrôles standards et l'utilisation de techniques de codage élémentaires.
Vous devez créer et faire fonctionner une calculatrice mathématique ordinaire qui aurait l'air de ceci, par exemple:



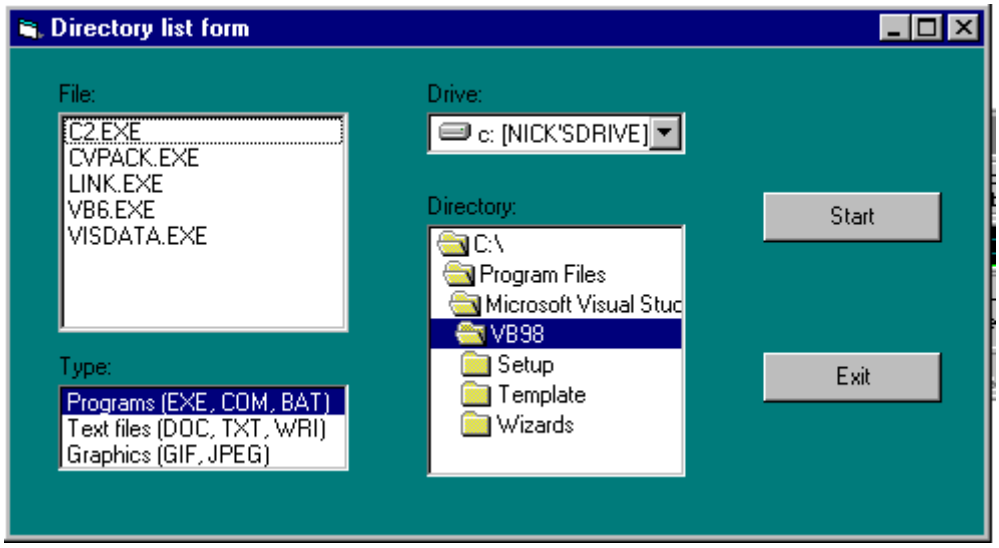

```
Option Explicit
Dim Operand1 As Double, Operand2 As Double
Dim Operator As String
Dim ClearDisplay As Boolean

Private Sub ClearBttn_Click()
   Display.Caption = ""
End Sub

Private Sub Digits_Click(Index As Integer)
   If ClearDisplay Then
      Display.Caption = ""
      ClearDisplay = False
   End If
   Display.Caption = Display.Caption + Digits(Index).Caption
End Sub

Private Sub Div_Click()
   Operand1 = Val(Display.Caption)
   Operator = "/"
   Display.Caption = ""
End Sub

Private Sub DotBttn_Click()
   If ClearDisplay Then
      Display.Caption = ""
      ClearDisplay = False
   End If
   If InStr(Display.Caption, ".") Then
      Exit Sub
   Else
      Display.Caption = Display.Caption + "."
   End If
End Sub
```

```
Private Sub Equals_Click()
Dim result As Double

On Error GoTo ErrorHandler
   Operand2 = Val(Display.Caption)
   If Operator = "+" Then result = Operand1 + Operand2
   If Operator = "-" Then result = Operand1 - Operand2
   If Operator = "*" Then result = Operand1 * Operand2
   If Operator = "/" And Operand2 <> "0" Then result = Operand1 / Operand2
   Display.Caption = result
   ClearDisplay = True
   Exit Sub
ErrorHandler:
   MsgBox "The operation resulted in the following error" & vbCrLf & Err.Description
   Display.Caption = "ERROR"
   ClearDisplay = True
End Sub

Private Sub Minus_Click()
   Operand1 = Val(Display.Caption)
   Operator = "-"
   Display.Caption = ""
End Sub

Private Sub Over_Click()
   If Val(Display.Caption) <> 0 Then Display.Caption = 1 / Val(Display.Caption)
End Sub

Private Sub Plus_Click()
   Operand1 = Val(Display.Caption)
   Operator = "+"
   Display.Caption = ""
End Sub

Private Sub PlusMinus_Click()
   Display.Caption = Val(Display.Caption)
End Sub

Private Sub Times_Click()
   Operand1 = Val(Display.Caption)
   Operator = "*"
   Display.Caption = ""
End Sub
```

Essayer le programme ci-dessus. Si vous avez des doutes, n'hésitez pas à contacter: *wasingface@gmail.com*

Listes de disques, répertoires et fichiers

Pour gérer vos disques, répertoires et fichiers, VB vous offre 3 contrôles standards: le **DriveListBox**, le **DirListBox** et le **FileListBox**. Dans une feuille normale ils apparaîtraient comme:



DriveListBox:

La propriété **.Drive** contient le nom du lecteur choisi et on peut s'en servir pour changer le lecteur comme dans: Drive1.Drive = "C"

Le fait de faire une sélection dans la liste provoque un événement **Change**.

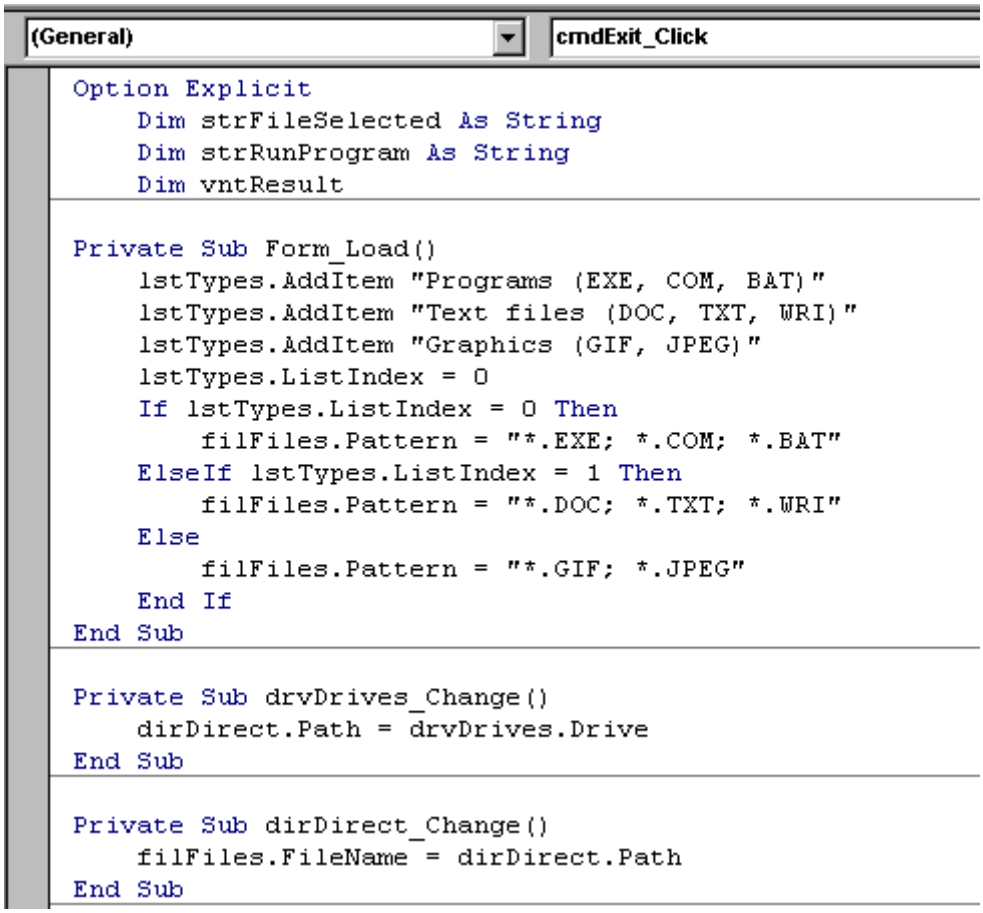
DirListBox:

La propriété **.Path** contient le nom du répertoire courant et on peut s'en servir pour changer le lecteur comme dans: Dir1.Path = "C:\Exemples\chap06.HTM"

Le fait de faire une sélection dans la liste provoque un événement **Change**.

FileListBox:

La propriété **.Path** contient le nom du répertoire courant. Le nom du fichier courant est dans **.FileName**. Cependant un test additionnel est nécessaire pour vérifier si le fichier est dans le root du disque (si oui, il se termine par \) ou bien s'il est dans un sous-répertoire.




```
Option Explicit
    Dim strFileSelected As String
    Dim strRunProgram As String
    Dim vntResult

Private Sub Form_Load()
    lstTypes.AddItem "Programs (EXE, COM, BAT)"
    lstTypes.AddItem "Text files (DOC, TXT, WRI)"
    lstTypes.AddItem "Graphics (GIF, JPEG)"
    lstTypes.ListIndex = 0
    If lstTypes.ListIndex = 0 Then
        filFiles.Pattern = "*.EXE; *.COM; *.BAT"
    ElseIf lstTypes.ListIndex = 1 Then
        filFiles.Pattern = "*.DOC; *.TXT; *.WRI"
    Else
        filFiles.Pattern = "*.GIF; *.JPEG"
    End If
End Sub

Private Sub drvDrives_Change()
    dirDirect.Path = drvDrives.Drive
End Sub

Private Sub dirDirect_Change()
    filFiles.FileName = dirDirect.Path
End Sub
```

```
Private Sub cmdStart_Click()
    If filFiles.FileName = "" Then
        MsgBox ("Select a file to run")
        Exit Sub
    End If
    strFileSelected = filFiles.Path
    If Right(strFileSelected, 1) = "\" Then
        strFileSelected = strFileSelected & filFiles.FileName
    Else
        strFileSelected = strFileSelected & "\" & filFiles.FileName
    End If

    Select Case lstTypes.ListIndex
    Case 0
        vntResult = Shell(strFileSelected, vbNormalFocus)
    Case 1
        RunProgram = "C:\Program Files\Accessories\Wordpad.exe"
        vntResult = Shell(RunProgram & " " & strFileSelected, vbNormalFocus)
    Case 2
        RunProgram = "D:\Viewer\lviewpro.exe"
        vntResult = Shell(RunProgram & " " & strFileSelected, vbNormalFocus)
    End Select
End Sub
```

```
Private Sub lstTypes_Click()
    If lstTypes.ListIndex = 0 Then
        filFiles.Pattern = "*.EXE; *.COM; *.BAT"
    ElseIf lstTypes.ListIndex = 1 Then
        filFiles.Pattern = "*.DOC; *.TXT; *.WRI"
    Else
        filFiles.Pattern = "*.GIF; *.JPEG"
    End If
End Sub

Private Sub filFiles_DblClick()
    cmdStart_Click
End Sub

Private Sub cmdExit_Click()
    Unload Me
End Sub
```

Notes sur le programme:

- La première chose à faire dans le Form_Load est de charger les items dans la liste des types de fichier - on veut seulement voir les exécutables, les fichiers de texte ou les fichiers graphiques - le .EXE est choisi par défaut (Listindex = 0)
- La propriété **Pattern** du Filelist identifie le filtre pour la sélection
- Si on entre une valeur dans le **Drive** ou le **Directory**, un événement **Change** est déclanché
- En cliquant sur Start, il faut d'abord voir si un fichier a été choisi - si non, afficher un message
- La fonction **Right()** (sera étudiée plus tard) est utilisée pour vérifier si le fichier est dans le Root
- Selon le type de fichier choisi, on exécute le **Shell function** qui appelle un fichier exécutable; vbNormalFocus signifie d'exécuter dans un window normal
- Un double-clique sur un fichier est égal à cliquer sur le bouton Start

CRÉATION D'UN MENU

Pour créer un menu dans une feuille on utilise l'éditeur de menu dans Tools --> Menu editor. L'éditeur ouvre une fenêtre dans laquelle on crée le menu qu'on veut voir affiché dans la feuille.

On utilise les flèches et le bouton Insert pour créer la cascade d'items dans le menu.

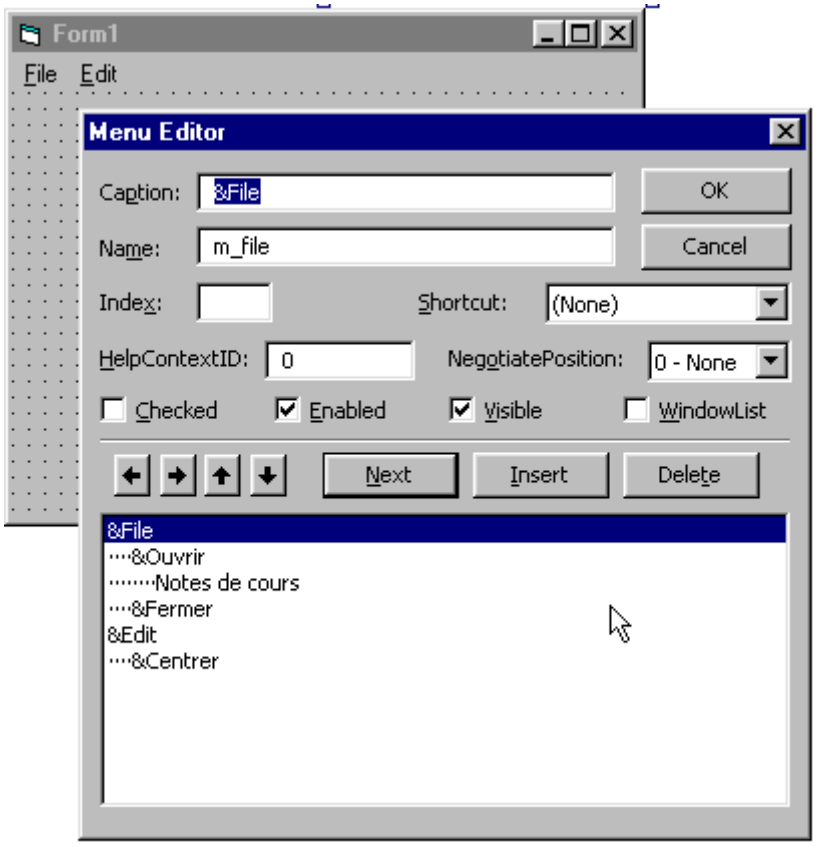


Fig. 5-5

Ensuite il ne reste qu'à écrire le code pour activer les fonctions du menu. Chaque item du menu n'a qu'un événement possible et c'est le **Click**.

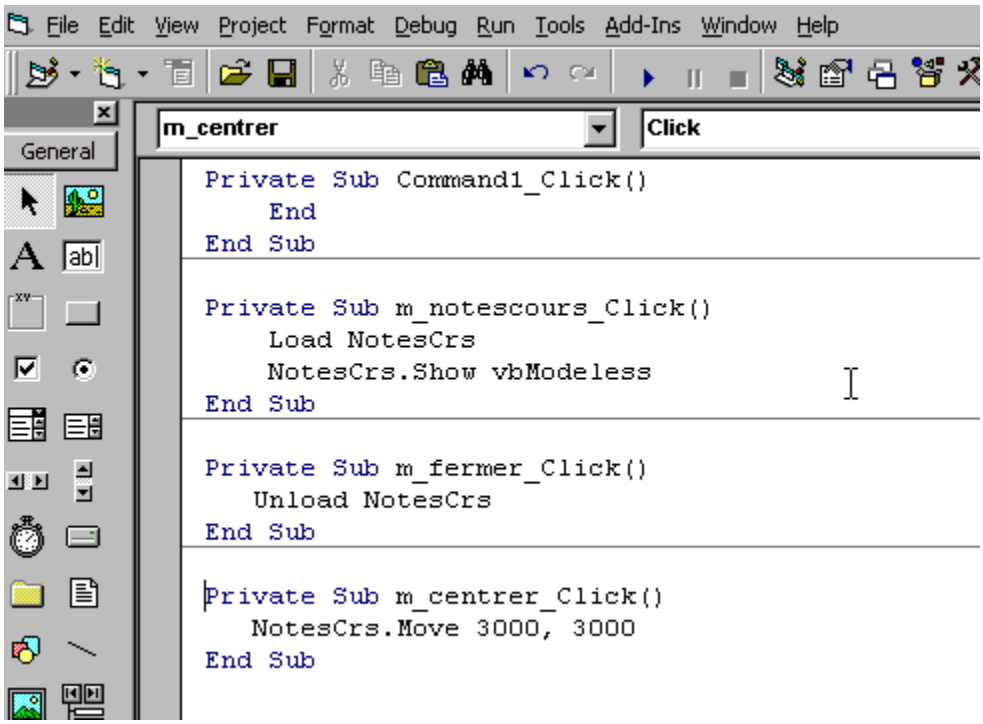


Fig. 5-6

LE "DEBUGGING"

Il arrive assez fréquemment qu'un programmeur fasse une erreur en écrivant le code. Les erreurs de syntaxe sont habituellement assez évidentes et le compilateur les identifie à l'exécution. Les erreurs de logique sont souvent plus difficiles à identifier. C'est pourquoi la plupart des logiciels ont une fonction de debugging qui aide à la correction des erreurs dans un programme.

La façon la plus simple d'utiliser debug en VB est de créer des **breakpoints** dans le code et puis de regarder l'état du programme au moment où il s'arrête. Un **breakpoint** est un point d'arrêt dans le code - l'exécution du programme arrête avant d'exécuter la ligne qui contient le breakpoint. Lorsque le code s'arrête, une fenêtre **immediate** s'ouvre au bas de l'écran. Dans cette fenêtre on peut regarder le contenu des variables, des propriétés, des objets, etc. Pour

continuer l'exécution après avoir étudier la situation on peut faire **Start** à nouveau. Cependant, il sera parfois très utile de regarder la séquence des instructions qui suivent. Pour cela on fait le **Step...** et la manière la plus facile de le faire est avec la touche **(F8)**.

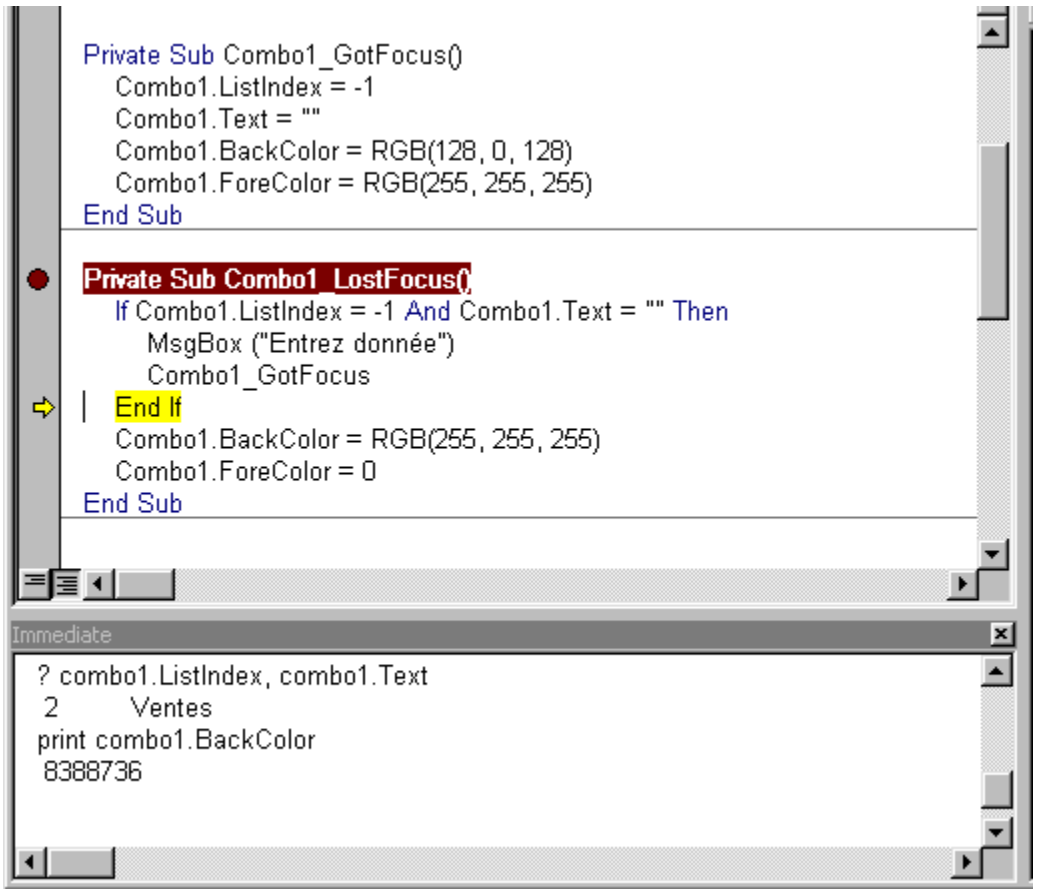


Fig. 5-7

AUTRES TECHNIQUES

Branchements:

Le GoTo: Parfois dans un module il est nécessaire de changer l'ordre d'exécution des commandes. La façon la plus simple de le faire est d'utiliser un **GoTo** comme:

```
Private Sub command1_Click()
Debut:
    (code)
    If erreur = 1 Then
       GoTo Debut
    End If
End Sub
```

Cependant, veuillez notez que l'usage du GoTo n'est **jamais recommandé** car c'est une vieille technique qui risque de créer des gros problèmes de lisibilité. Il y a d'autres structures plus efficaces pour faire la même chose.

Le Do ... Loop: crée une boucle de traitement entre le **Do** et le **Loop** - on peut s'en servir pour traiter une condition d'erreur comme:

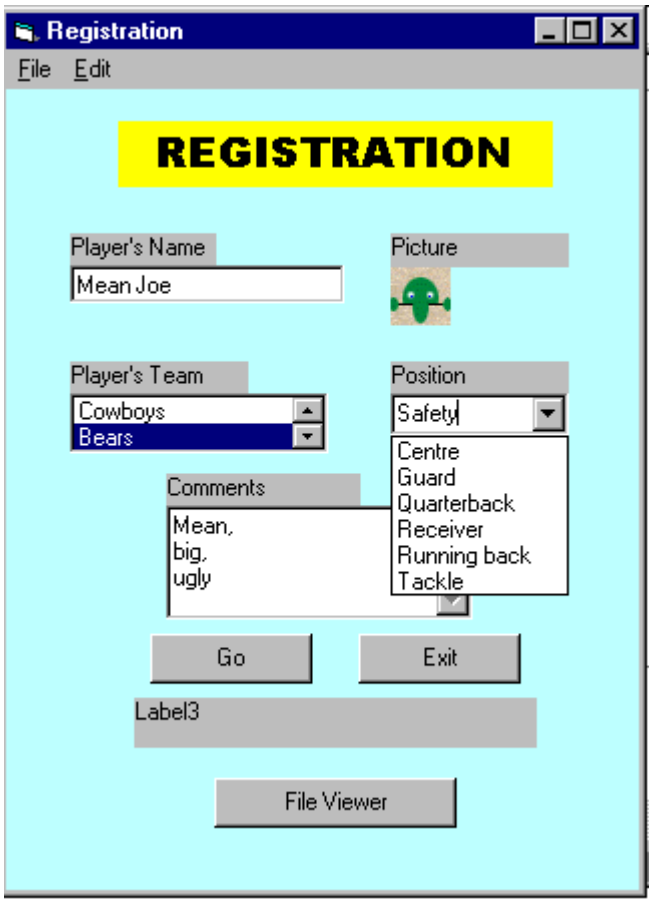
```
Private Sub command1_Click()
    erreur = 0
    Do While erreur = 0
        erreur = 1
        If valeur > x Then
            erreur = 0
        End If
    Loop
    .....
```

La structure Do While ... Loop peut être remplacée par la structure équivalente:

```
While condition
    instructions
end
```

AUTRE EXEMPLE

Voici un exemple que nous utiliserons à plusieurs reprises dans les prochains cours. Il regroupe la plupart des techniques que nous avons étudiées et certaines qui s'en viennent. Je vous suggère de recréer l'application pour vous-même afin de tester les commandes.



```
Option Explicit
    Dim strTeam As String
    Dim strPosition As String

Private Sub Form_Load()
    'When the form loads, the first thing
    'we do is to assign the names to the
    'Listbox of teams
    lstTeam.AddItem "Giants"
    lstTeam.AddItem "Redskins"
    lstTeam.AddItem "Cowboys"
    lstTeam.AddItem "Bears"
    lstTeam.AddItem "Jets"
    'Next, load the Combo for Position.
    'To have them appear in order,
    'use the "Sorted" property of the combo
    cboPosition.AddItem "Guard"
    cboPosition.AddItem "Tackle"
    cboPosition.AddItem "Quarterback"
    cboPosition.AddItem "Receiver"
    cboPosition.AddItem "Centre"
    cboPosition.AddItem "Running back"
End Sub
```

```
Private Sub cmdGo_Click()
    strTeam = lstTeam.Text
    strPosition = cboPosition.Text
    Label3.Caption = strPosition _
            & ", " & strTeam
    If txtName.Text = "" Then
        mnuViewer.Enabled = False
        cmdViewer.Enabled = False
        txtName.SetFocus
    Else
        mnuViewer.Enabled = True
        cmdViewer.Enabled = True
    End If
End Sub
Private Sub cmdViewer_Click()
    Load frmDirList
    frmDirList.Show vbModeless
End Sub
Private Sub cmdExit_Click()
    Unload Me
    End
End Sub
```

```
'Click on menu Exit same as click on button Exit
Private Sub mnuExit_Click()
    cmdExit_Click
End Sub
'Invoke Move method for this form (Me)
'Look at Form object --> Methods in Help
Private Sub mnuMove_Click()
    Me.Move 0, 0
End Sub

'Invoke PrintForm method for this form (Me)
'Sends image of form to printer - useful for hardcopy
Private Sub mnuPrint_Click()
    Me.PrintForm
End Sub
'Parameters of Move are: left edge, top edge, width, height
'Measurements in twips (see Lesson 7)
Private Sub mnuResize_Click()
    Me.Move 3000, 3000, 6000, 5000
End Sub
'Load and Show form DirList
Private Sub mnuViewer_Click()
    Load frmDirList
    frmDirList.Show vbModeless
End Sub
```

VII. Caractères et images

Gérer le texte

Il est souvent nécessaire quand on saisie des données ou qu'on valide des informations, de pouvoir manipuler les chaînes de caractères de différentes façons. Voyons quelques fonctions utiles pour exécuter ces manipulations:

Len(chaine): retourne la longueur - nombre de caractères dans la chaine.

Left(chaine, nombre): retourne le nombre de caractères spécifié à partir de la gauche de chaine.

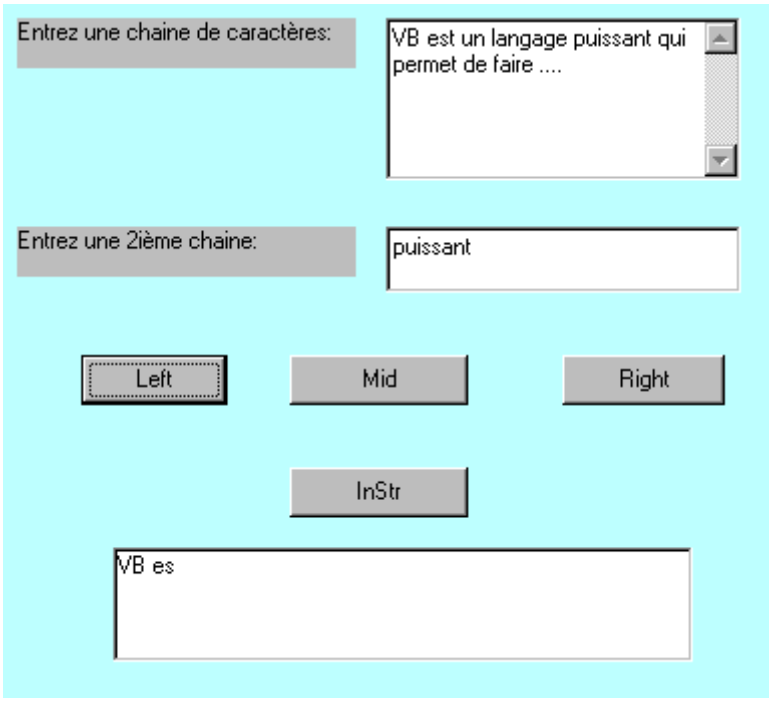
Right(chaine, nombre): retourne le nombre de caractères à partir de la droite de la chaine.

Mid(chaine, position, nombre): retourne le nombre de caractères spécifié à partir de position dans la chaine.

InStr(chaine1, chaine2): retourne la position dans la chaine1 où chaine2 commence - retourne 0 si chaine2 pas trouvée

LTrim(chaine), RTrim(chaine) et Trim(chaine): retourne la chaine avec les espaces non-significatifs enlevés, à gauche, à droite ou les deux.

LCase(chaine), UCase(chaine): LCase retourne la chaine toute en minuscules (lower case) et UCase retourne toutes des majuscules (upper case).




```
Option Explicit
    Dim strCh1 As String, strCh2 As String

Private Sub cmdLeft_Click()
    strCh1 = txtChaine1.Text
    'extraire les 5 caractères de gauche
    lblSortie.Caption = Left(strCh1, 5)
End Sub

Private Sub cmdMid_Click()
    strCh1 = txtChaine1.Text
    'extraire 7 caractères à partir
    'du 5ième
    lblSortie.Caption = Mid(strCh1, 5, 7)
End Sub

Private Sub cmdRight_Click()
    strCh1 = txtChaine1.Text
    'extraire les 8 caractères de droite
    lblSortie.Caption = Right(strCh1, 8)
End Sub

Private Sub cmdIn_Click()
    strCh1 = txtChaine1.Text
    strCh2 = txtChaine2.Text
    'la position de Chaine2 dans Chaine1
    lblSortie.Caption = InStr(strCh1, strCh2)
End Sub
```

Format(chaine, format): retourne la chaine formatée selon le format spécifié; les caractères de formatage utilisés pour les données numériques sont:

0 représente un chiffre, avec les zéros non-significatifs
représente un chiffre, sans les zéros non-significatifs
. pour la position du décimal
, pour les groupes de milliers
+ - ( ) espace affichés littéralement

Par exemple:

Format(3456.7, "00000.00")	--> 03456.70
Format(3456.7, "#####.##")	--> 3456.7
Format(003456.75899 begin_of_the_skype_highlighting 003456.75899 end_of_the_skype_highlighting, "##,##0.00")	--> 3,456.76
Format(456.7, "##,##0.00$")	--> 456.70$

Pour les données de type date ou heure on utilise les caractères suivants:
yy --> l'année sans le centenaire - eg: 98
yyyy --> l'année avec centenaire - eg: 1998

m --> le mois numérique - eg: 10
mmm --> le mois abrévié - eg: oct
mmmm --> le nom du mois au long - eg: octobre
d --> le jour du mois, sans zéro - eg: 8
dd --> le jour du mois, avec zéro - eg: 08
dddd --> le nom du jour de la semaine - eg: lundi
h --> l'heure, sans zéro - eg: 7
hh --> l'heure avec zéro - eg: 07
mm --> minutes - eg: 45
ss --> secondes - eg: 50
Il existe aussi plusieurs formats prédéfinis qu'on peut utiliser. Par exemple:
"general date", "short date", "long date" pour les dates
"general number", "currency", "standard" pour les nombres.
On les utilise comme:

```
lblDateEmb.caption = Format(dtmEmbauche, "short date")
lblSalaire.caption = Format(sglSalaire, "currency")
```

Blocs de texte

Il est souvent utile dans une application de pouvoir travailler avec des blocs de texte. Les contrôles **TextBox et ComboBox** possèdent des propriétés qui permettent de manipuler les blocs de texte. Il s'agit de:
SelStart: entier long qui identifie le début du bloc - 0 signifie le début du texte et un nombre égal à la longueur signifie tout le texte
SelLength: un entier long qui identifie le nombre de caractères à sélectionner.
SelText: un string qui contient les caractères sélectionnés.

Par exemple, le code suivant sélectionne tout le texte dans un TextBox:

```
Text1.SetFocus
Text1.SelStart = 0
Text1.SelLength = Len(Text1.Text)
'
' Si je veux remplacer le texte choisi
' j'assigne une nouvelle valeur à SelText
'
Text1.SelText = "Nouvelle valeur"
'
' Notez que ce genre de manipulation est habituellement lancé
' par les événements MouseDown, MouseUp ou
' MouseMove associés au contrôle.
```

Voici un exemple de bloc, tel que décrit dans l'Aide:

```
Private Sub Form_Load ()
    Text1.Text = "Two of the peak human experiences"
    Text1.Text = Text1.Text & " are good food and classical music."
End Sub

Private Sub Form_Click ()
    Dim Search, Where ' Declare variables.
    ' Get search string from user.
    Search = InputBox("Enter text to be found:")
    Where = InStr(Text1.Text, Search) ' Find string in text.
    If Where Then ' If found,
        Text1.SelStart = Where - 1 ' set selection start and
        Text1.SelLength = Len(Search) ' set selection length.
```

```
    Else
        MsgBox "String not found." ' Notify user.
    End If
End Sub
```

Objets spéciaux
Screen: représente l'environnement Windows au complet - permet l'accès aux feuilles et aux contrôles. Ses **propriétés** importantes: **ActiveControl** retourne le nom du contrôle qui a le focus et **ActiveForm** retourne le nom de la feuille courante **Clipboard**: le presse-papier du système - permet de manipuler (couper, copier, coller) du texte et des graphiques de l'application. Ses **méthodes** importantes: **Clear** vide le clipboard, **SetText** met une chaine de texte dans le clipboard, **GetText** retourne une chaine de texte du clipboard.

Voici un exemple qui montre comment utiliser **l'objet Clipboard**, le presse-papier, pour manipuler des blocs de texte. Remarquez qu'on fait la manipulation au moyen d'un menu puisque c'est possible de maintenir le contrôle choisi en focus pendant qu'on fait la sélection du texte.

Notez aussi que le fait d'utiliser des références à **l'objet Screen** plutôt qu'à des contrôles spécifiques nous permet d'incorporer le code dans n'importe quelle application, peu importe les noms qu'on a donné aux différents contrôles.

This example shows how the Clipboard object is used in cut, copy, paste, and deletes operations. To try this example, create a form with

a TextBox control and use the Menu Editor to create an Edit menu (for each of the commands, set the Caption property = Cut, Copy, Paste, and Delete, respectively; set the Name property = EditCut, EditCopy, EditPaste, and EditDelete, respectively).

```
Private Sub EditCut_Click ()
    ' Clear the contents of the Clipboard.
    Clipboard.Clear
    ' Copy selected text to Clipboard.
    ClipBoard.SetText Screen.ActiveControl.SelText
    ' Delete selected text.
    Screen.ActiveControl.SelText = " "
End Sub
Private Sub EditCopy_Click ()
    ' Clear the contents of the Clipboard.
    Clipboard.Clear
    ' Copy selected text to Clipboard.
    ClipBoard.SetText Screen.ActiveControl.SelText
End Sub
Private Sub EditPaste_Click ()
    ' Place text from Clipboard into active control.
    Screen.ActiveControl.SelText = ClipBoard.GetText ()
End Sub
Private Sub EditDelete_Click ()
    ' Delete selected text.
    Screen.ActiveControl.SelText = ""
End Sub
```

GÉRER LES GRAPHIQUES

Le système de coordonnées

Le système de coordonnées de la feuille est une grille à 2-dimensions qui définit la position sous forme de (x,y). la valeur de x représente la distance à partir du bord gauche et y représente la distance à partir du haut. La position (0,0) est donc le coin supérieur gauche de la feuille. En VB l'unité de mesure pour les coordonnées est le **twip**.

1 twip = 1/20 d'un point d'impression

1440 twips = 1 pouce

567 twips = 1 centimètre

Cependant, il est possible de changer l'unité de mesure en changeant la propriété **ScaleMode**.

ScaleMode = 1 --> twips

ScaleMode = 2 --> points

ScaleMode = 3 --> pixels

ScaleMode = 4 --> caractères

ScaleMode = 5 --> pouces

ScaleMode = 6 --> millimètres

ScaleMode = 7 --> centimètres

On peut changer ScaleMode dans **FormLoad**, ce qui appliquera l'échelle à la feuille elle-même ou bien, on peut changer ScaleMode pour un objet spécifique comme:

Image1.ScaleMode = 5 'en pouces

Image1.Move 2, 2, 2, 2

Déplacement et grandeur

Par exemple, pour déplacer ou modifier la grandeur d'une feuille on utilise la méthode **Move**, comme dans:

```
Form3.Move 3000, 3000  'à environ 5 cm de la gauche et 5 cm du haut
Form3.Move(3000, 3000, 5670, 5670)   'haut. = 10 cm, larg. = 10 cm
Form3.ScaleMode = 7
Form3.Move 5, 5, 10, 10
```

Les images

Une image peut être affichée de 3 façons:

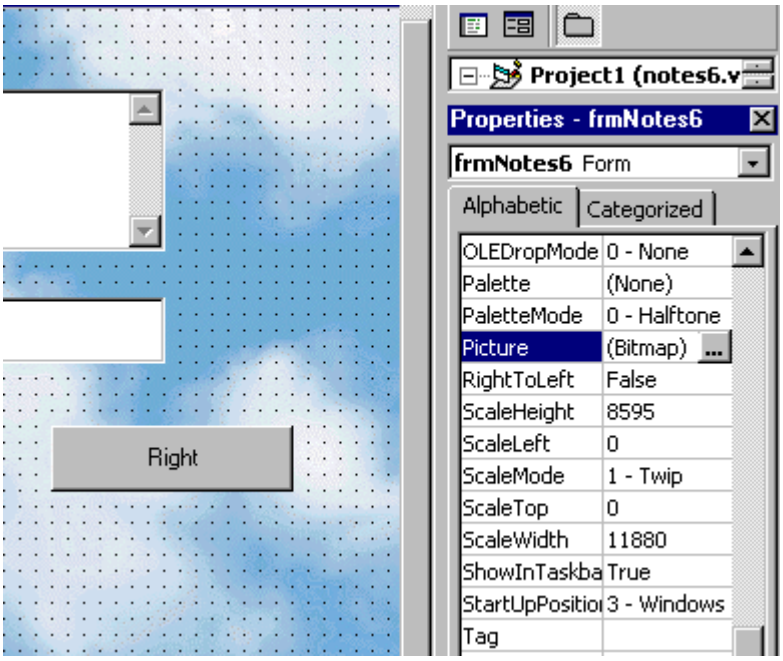
1) sur une feuille directement

2) dans un contrôle **PictureBox**

3) dans un contrôle **Image**

Pour afficher l'image dans un des contrôles on doit spécifier le nom du fichier qui contient l'image dans la propriété **Picture** du contrôle.

Dans le cas d'un Form, l'image devient le "background". Il faut que l'image soit de la bonne grandeur avant de l'afficher.



Pour charger une image lors de l'exécution on utilise la fonction **LoadPicture** comme:

picLogo.Picture = LoadPicture("C:\images\auto.bmp")

Pour enlever une image lors de l'exécution on utilise aussi LoadPicture, avec le paramètre nul comme:
picLogo.Picture = LoadPicture("")

Dans un PictureBox, si l'image est trop grande pour le contrôle dans lequel on l'affiche, elle est coupée à droite et en bas.
Si on veut que le contrôle s'étende pour recevoir l'image, on met la propriété **AutoSize** à **True**. Le contrôle **Image** n'a pas de propriété AutoSize mais, il s'agrandit automatiquement à la grandeur de l'image. Le **Form** n'a pas d'AutoSize et ne s'ajuste pas à la grandeur de l'image.
Dans un **Image**, on peut mettre la propriété **Stretch** à **True** si on veut que l'image ajuste sa grandeur à la grandeur du contrôle.

Multimédia

Multimédia réfère à des unités autres que l'écran ou l'imprimante pour produire des sons, regarder des vidéos ou écouter de la musique. Pour ce faire, on utilise un nouveau contrôle: le **Multimédia control**. Mais ne le chercher pas dans la boite à outils; il faut l'ajouter d'ailleurs.
Pour la pratique, créer une nouvelle feuille: frmMultimed.frm dans un projet existant. Dans le menu **Project --> Components**, trouvez "**Microsoft Multimédia Control 6.0**" et cochez la case puis faites OK. Maintenant, le Multimédia control fait partie de votre boite à outils (mais seulement pour ce projet).

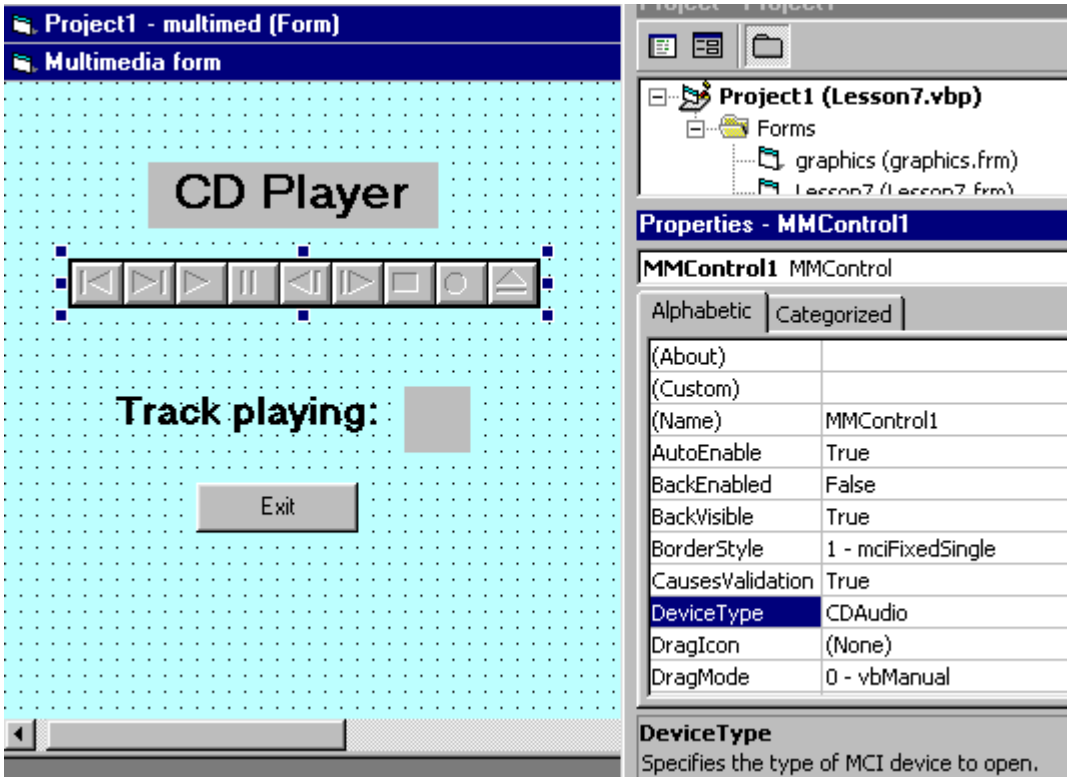
Si vous mettez un Multimédia control sur la feuille, vous obtenez une barre de bouton comme vous voyez pour tous les équipements de ce genre. Dans la **DeviceType property** vous spécifiez quelle sorte de lecteur ce control opère:

Device Type	**Device**
CDAudio	CD Audio player
DAT	Digital audio tape player
Overlay	Overlay
Scanner	Scanner
Vcr	Videotape player and recorder
Videodisc	Videodisc player
Other	Other devices not specified

Exemple: un lecteur CD

Dans la nouvelle feuille: frmMultimed, on ajoute un Multimédia control. Puisque c'est le seul control qu'on utilise, on laisse son nom à MMControl1.
Puis on met: **CDAudio dans DeviceType**. Le device CDAudio sert à lire des CD dans le lecteur CD. Si on veut entendre le son qui est enregistré dans un fichier **.WAV** on utilise le DeviceType **WaveAudio** et on doit lui fournir un **Filename** qui contient le son.

On ajoute quelque labels pour complèter la feuille et on obtient:



Maintenant il faut écrire le code pour faire fonctionner le lecteur. D'abord, voici ce qu'il faut savoir au sujet du MM Control: Il y a un **Track property** qui contient le numéro de la piste courante. Mais la propriété la plus importante est **Command property** qui peut accepter plusieurs valeurs et qui, en fait, opère le lecteur.

Command value	**Meaning**
Open	Opens the device
Close	Closes the device
Eject	Ejects the CD
Play	Plays the device
Pause	Pauses the device
Next	Goes to next track
Prev	Goes to beginning of current track. If used within 3 seconds of most recent Prev, goes to

	beginning of previous track
Record	Initializes recording
Save	Saves the open device file
Seek	Step backward or forward a track
Stop	Stops the device
Step	Step forward through tracks

Par exemple, pour ouvrir le lecteur, on fait:

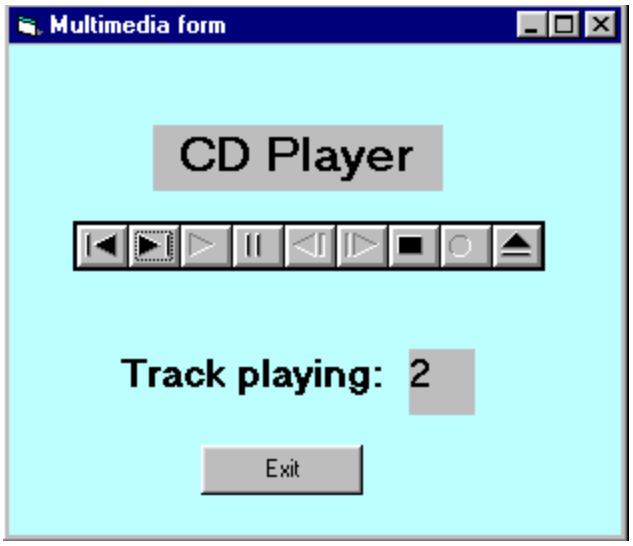
MMControl1.Command = "Open" 'assigne la valeur "Open" à la propriété Command

Pour faire une pause:

MMControl1.Command = "Pause" 'assigne la valeur "Pause" à la propriété Command

Comme vous savez, le truc de la programmation est de savoir dans quel événement on doit écrire le code. On sait que le **Form_Load event** est activé quand la feuille s'ouvre. On peut y mettre le démarrage du lecteur avec Open. Une fois le lecteur en marche, il ne s'arrêtera pas même si on ferme la feuille, à moins qu'on lui dise de s'arrêter. On va donc mettre la commande Stop dans le **Form_Unload event**. Et pour voir ce qui se passe, on va utiliser le **StatusUpdate event** pour afficher le compteur de piste; "change track, pause, play" sont toutes des actions qui lancent un StatusUpdate.

Vous remarquerez que vous pouvez utiliser les boutons du MM Control pour contrôler le lecteur.



MMControl1 | StatusUpdate

```
Private Sub Form_Load()
    MMControl1.Command = "Open"
    MMControl1.Command = "Play"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub

Private Sub MMControl1_StatusUpdate()
    lbl_track.Caption = MMControl1.Track
End Sub

Private Sub cb_exit_Click()
    Unload Me
    End
End Sub
```

VIII. VB et Bases de données

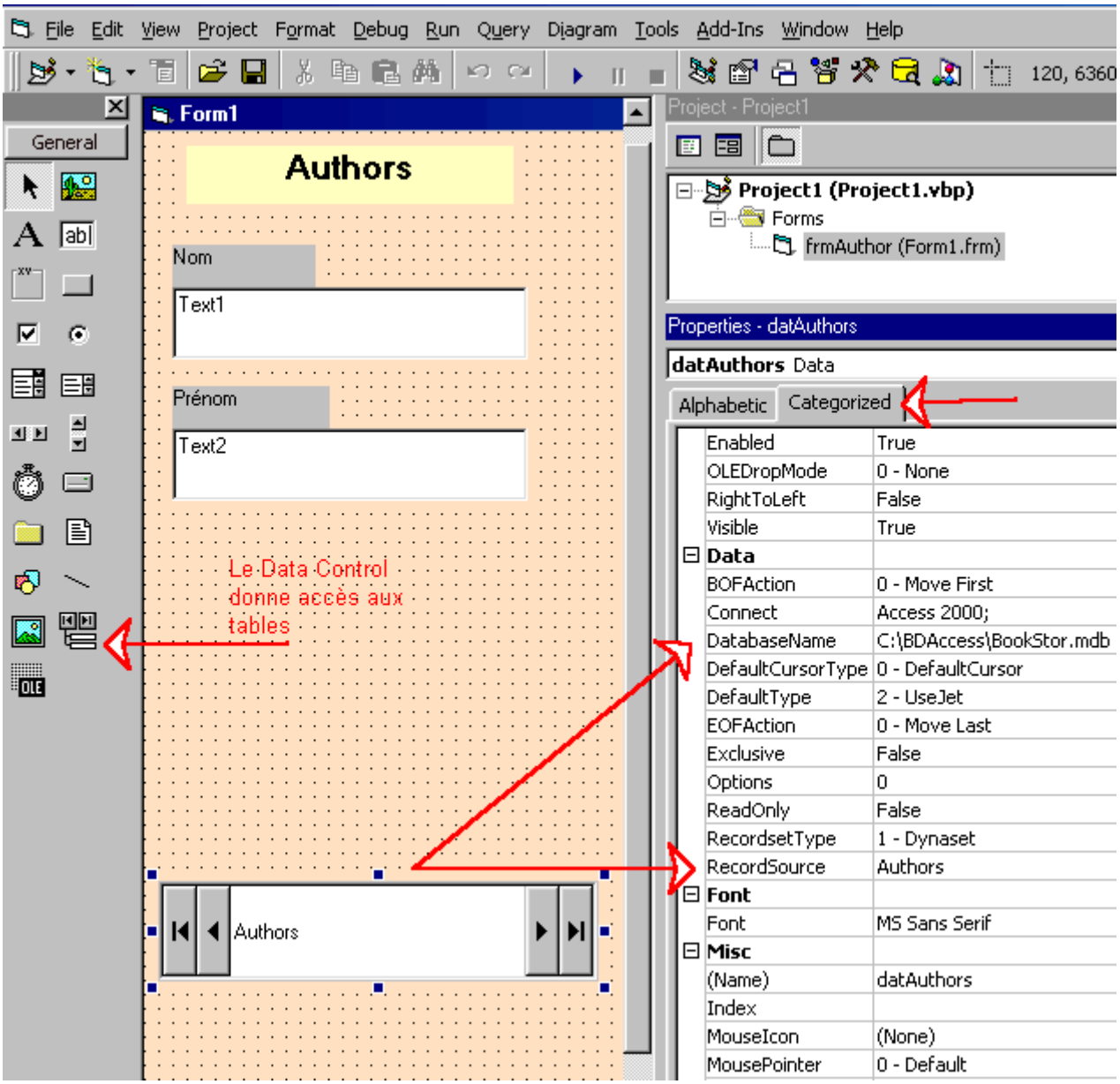
L'application "BookStore"

Dans ce premier exercice nous allons nous rattacher à une base de données en Access. Puisque VB et Access sont de proches parents, cette connection est la plus simple possible.
Plus tard, nous utiliserons d'autres techniques pour communiquer avec une bd en MySQL.
Pour ces exemples nous allons utilisons la base de données "**Book Store**". La base de données provient de sources américaines et contient plusieurs tables dont certaines ne seront pas utiles pour le moment. Vous pouvez vous servir de la base pour tous vos tests. Utilisez le format Access 97 ou Access 2000, selon votre logiciel.
Notre application devra pouvoir faire toutes les tâches de maintenance et de transactions requises. On crée un Form pour chaque table fondamentale dans l'application. Et puisqu'on se retrouvera avec plusieurs Forms il faudra naviguer d'une façon organisée à travers ces Forms. La meilleure technique pour accomplir cela est d'utiliser un **Menu d'application** pour activer les différents Forms au besoin.

Le Data Control

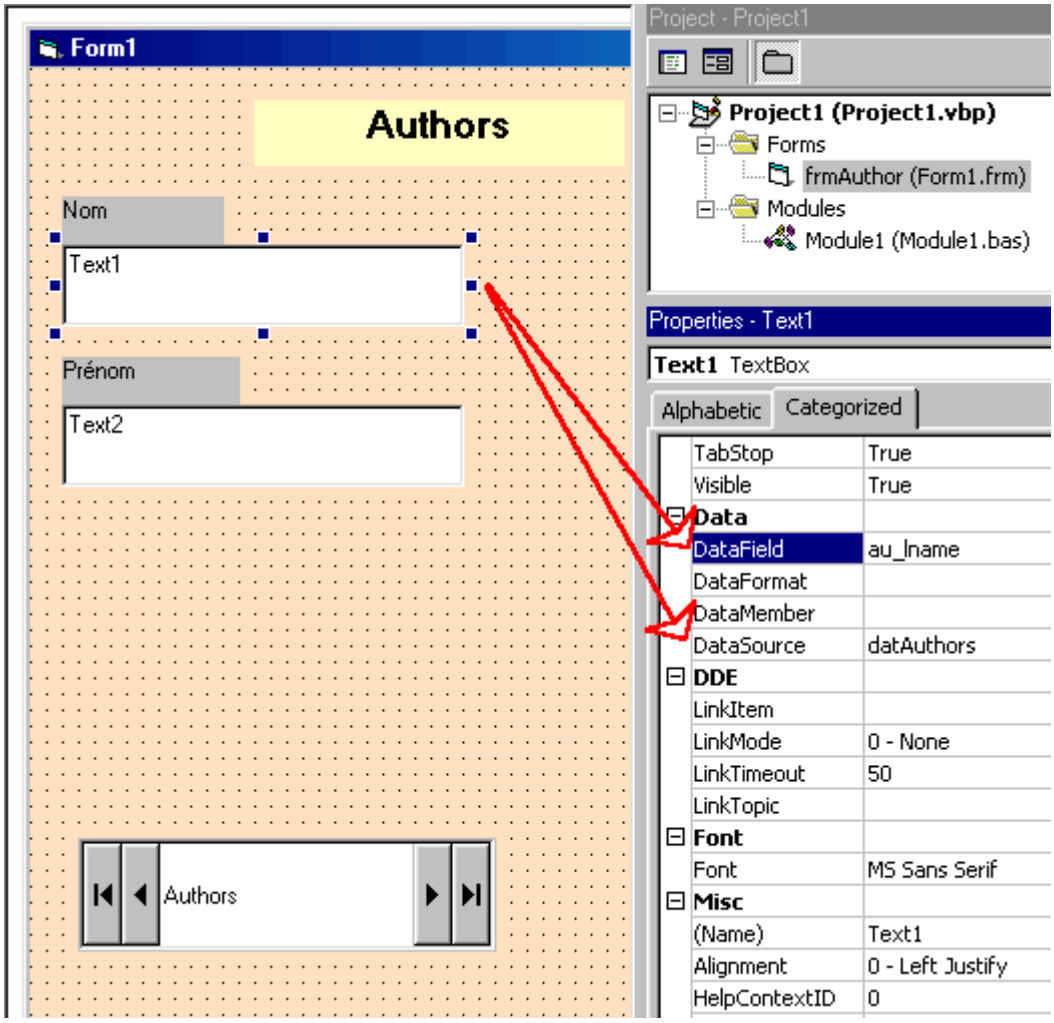
Le **data control** est l'objet qui relie un Form et une base de données. Pour avoir accès à "BookStor.mdb" pour faire la maintenance sur les tables on crée un nouveau Form qu'on appelle "Authors" et le premier control qu'on y place est le "Data control". Puis on spécifie les propriétés "Data" du control:

Connect = sorte de base de données - normallement Access 2000
DatabaseName = le nom de la bd qu'on attache
RecordSource = le nom de la table qu'on utilise



Les controls liés

Après le Data control on place des controls pour afficher les champs de la table. Pour chacun de ces controls on doit décrire les propriétés: **DataSource** et **DataField** qui spécifient de passer par le Data control pour accéder au champ qu'on veut manipuler.



Tout changement que l'on fait sur un control lié est automatiquement éffectué dans la table dès que l'on quitte l'enregistrement. Dans certains cas cependant on peut vouloir écrire du code pour traiter les changements.

Pour **Ajouter** un enregistrement à la table, il y a deux méthodes:

- dans le DataControl datAuthors, mettre la propriété **EOFAction = 2**

  Ceci fait en sorte que quand on fait "Suivant" après "Dernier", un nouvel enregistrement est généré.

- créer un **bouton "Ajouter"** qui exécute la commande:

  datAuthors.Recordset.AddNew

  Ceci a l'avantage que l'enregistrement est inséré dans le recordset à l'endroit où on fait le Addnew.

  La commande **Update** réécrit l'enregistrement sans le quitter.

  Update doit être précédé d'un **AddNew** ou d'un **Edit**.

  On peut coder le bouton **Update** comme:

  datAuthors.Recordset.Edit

  datAuthors.Recordset.Update

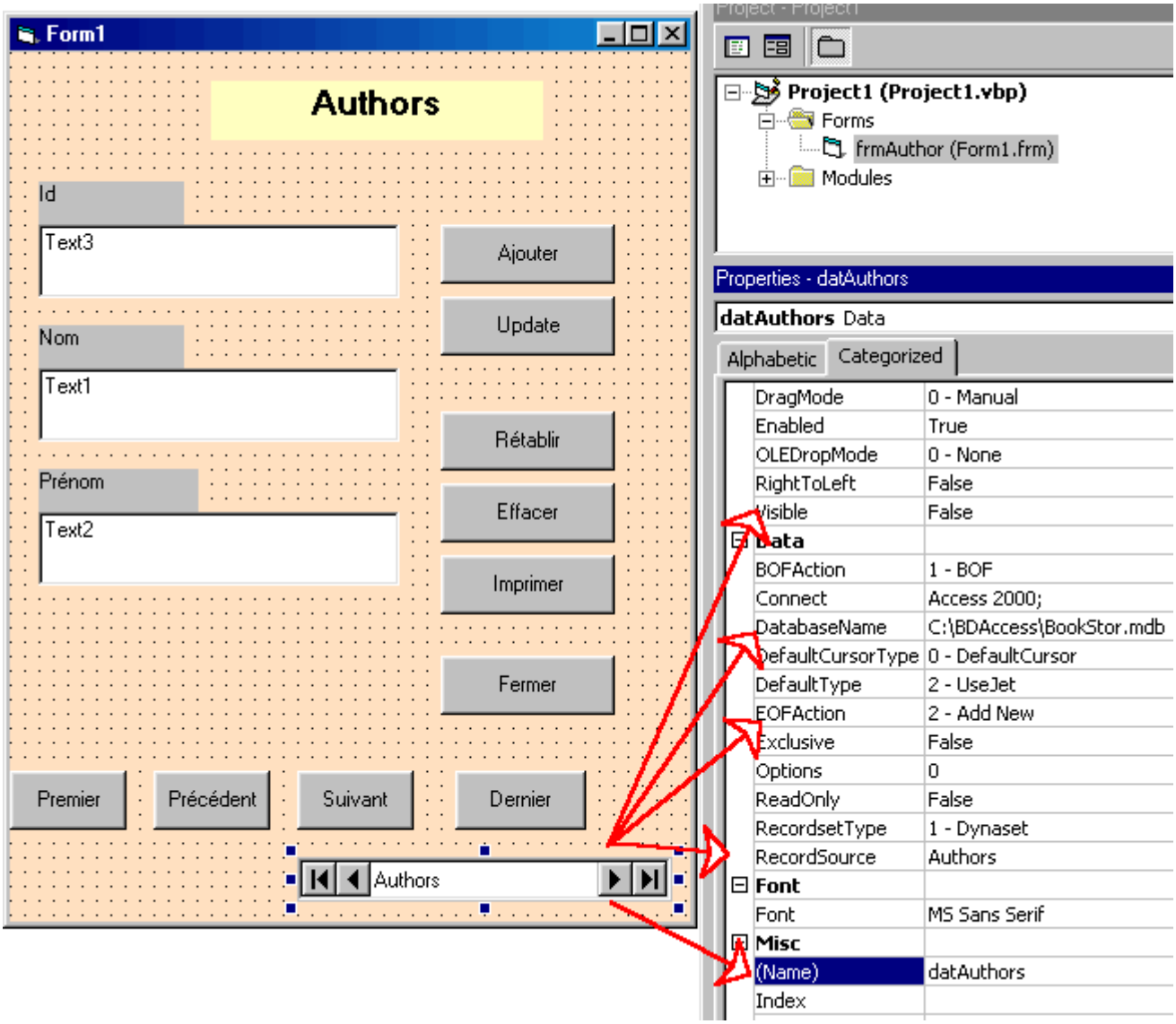
  Pour **Effacer** il est préférable d'utiliser un bouton codé comme:

  datAuthors.Recordset.Delete

  datAuthors.Recordset.MoveNext

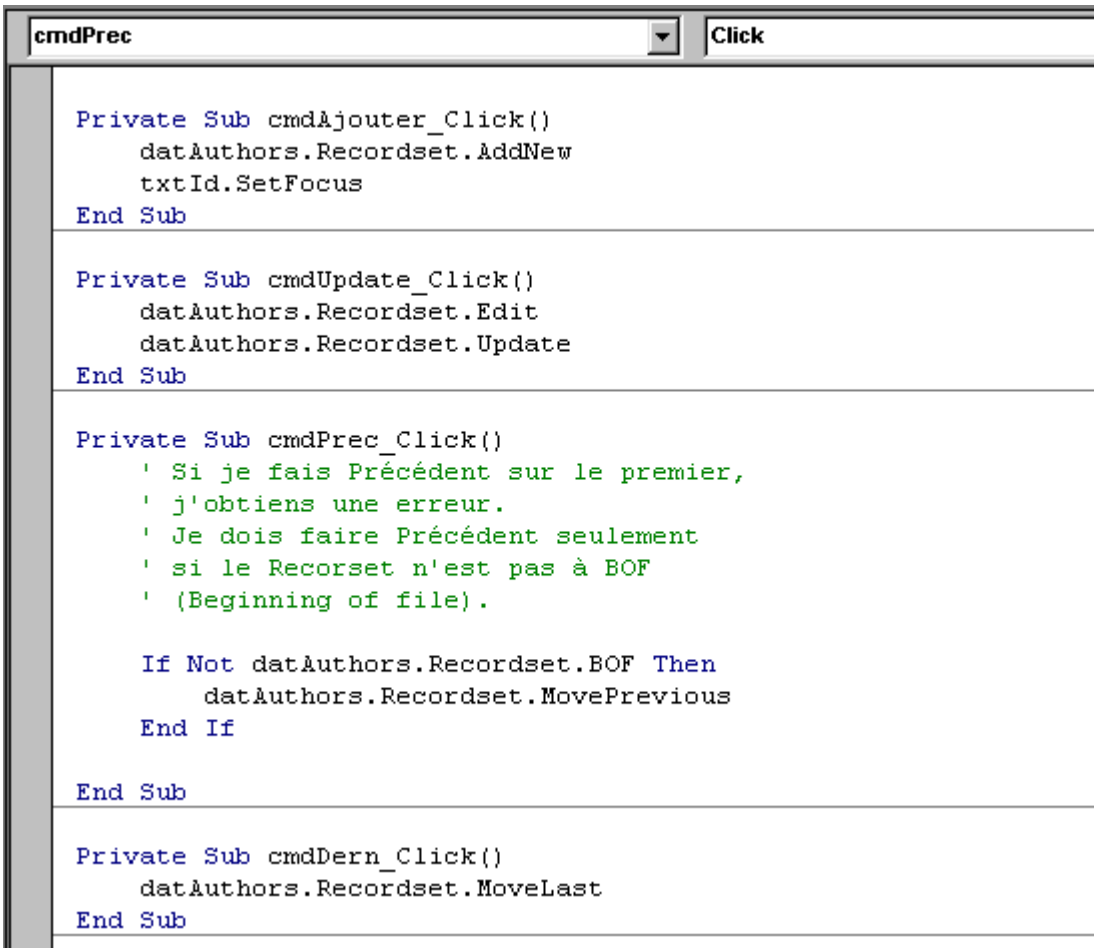
  Il se peut dans certains cas qu'on doit annuler les changements faits sur les controls - on veut donc **Rétablir** les controls à leur valeur initiale. On crée un bouton codé comme:

  datAuthors.UpdateControls



Dans le Form illustré, on a aussi ajouté un bouton pour **Fermer**. Il devra exécuter un Unload afin de fermer le Form courant et retourner au Menu.

On peut aussi ajouter les boutons de navigation. Le code pour naviguer utilise le **MoveFirst, MovePrevious, MoveNext et MoveLast**. Avec les boutons codés, on peut cacher le Data Control.

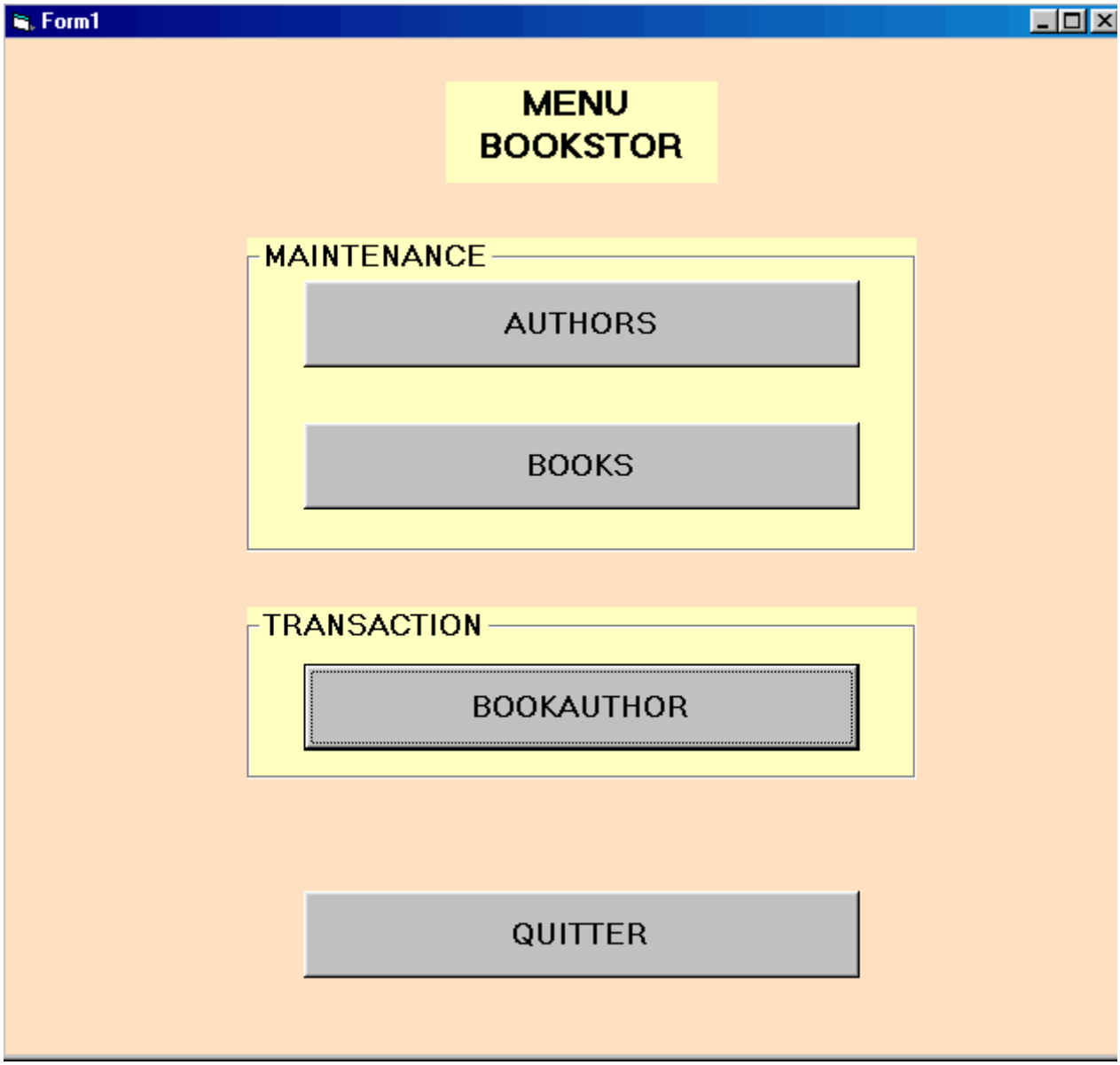


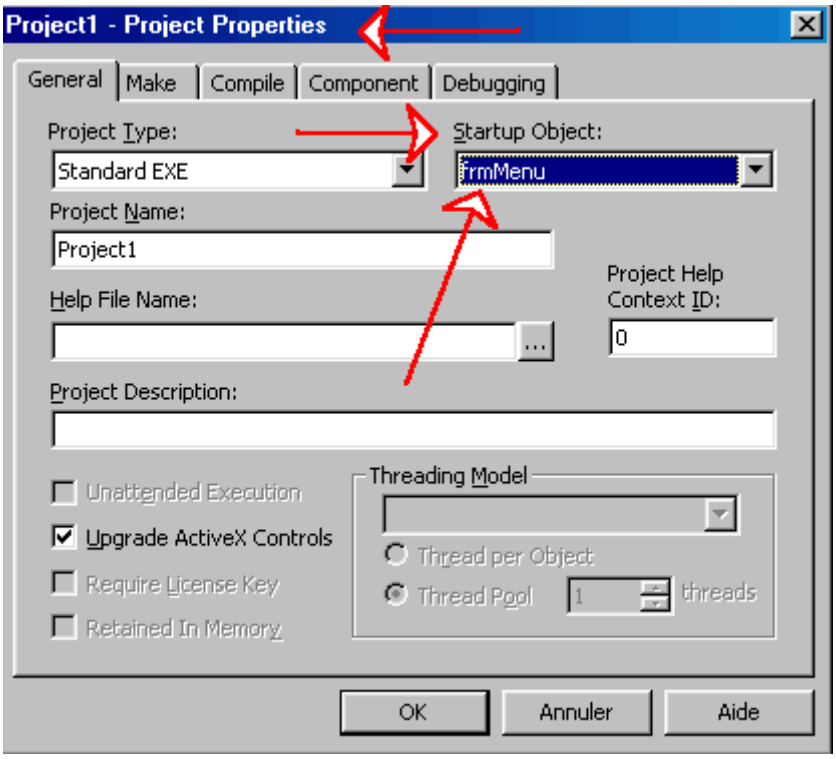
Le bouton **Imprimer** nous permet d'envoyer à l'imprimante une copie du Form. C'est très élémentaire comme rapport mais, ça nous permet de produire un "hard copy" facilement. Le code pour imprimer consiste d'exécuter la méthode **PrintForm**. La syntaxe est:

form.PrintForm

Si "form" n'est pas spécifié, elle imprime le form actuel.

Pour exécuter l'application, il faudra créer un **Menu d'application**:


Form1
MENU
BOOKSTOR
MAINTENANCE
AUTHORS
BOOKS
TRANSACTION
BOOKAUTHOR
QUITTER

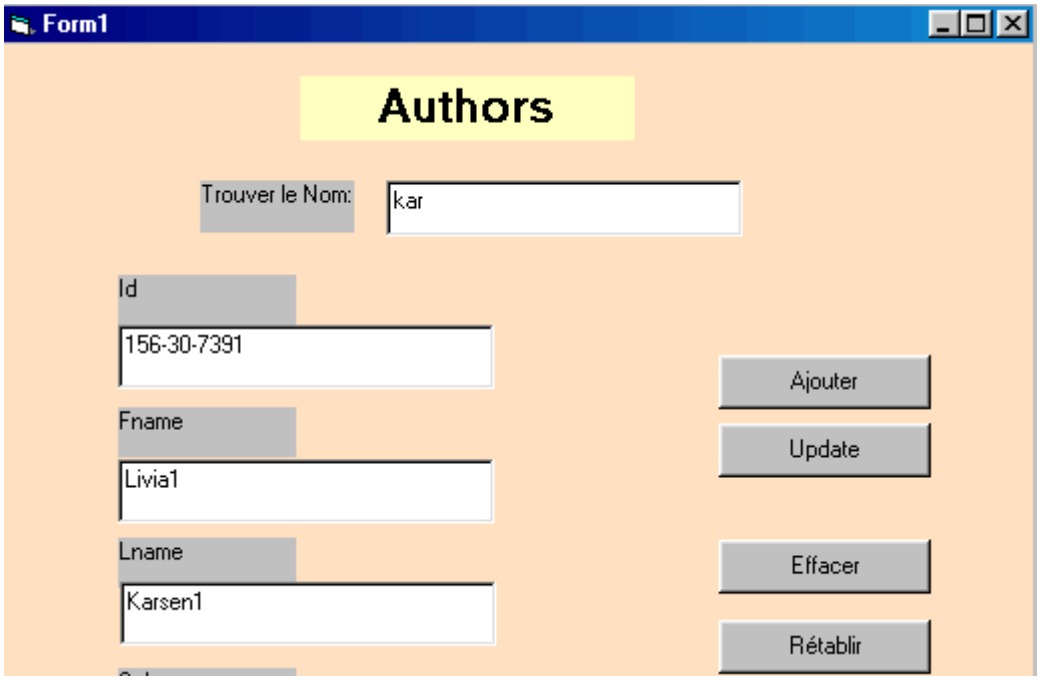


Trouver un enregistrement spécifique.

On peut chercher seulement sur un champ à la fois.

Il faudra créer un nouveau TextBox pour saisir le critère de recherche.

On met le TextBox de critère comme TabIndex = 0. La recherche sera activée par le LostFocus du Textbox. Si on ne fait pas de recherche (le critère est vide) la fonction ne fait rien.



La syntaxe du Find est un peu difficile - il faut surveiller la ponctuation:

DataControl.Recordset.FindFirst "fieldname" = 'searchstring'

C'est pourquoi c'est préférable de construire le 'searchstring', comme dans le code suivant:

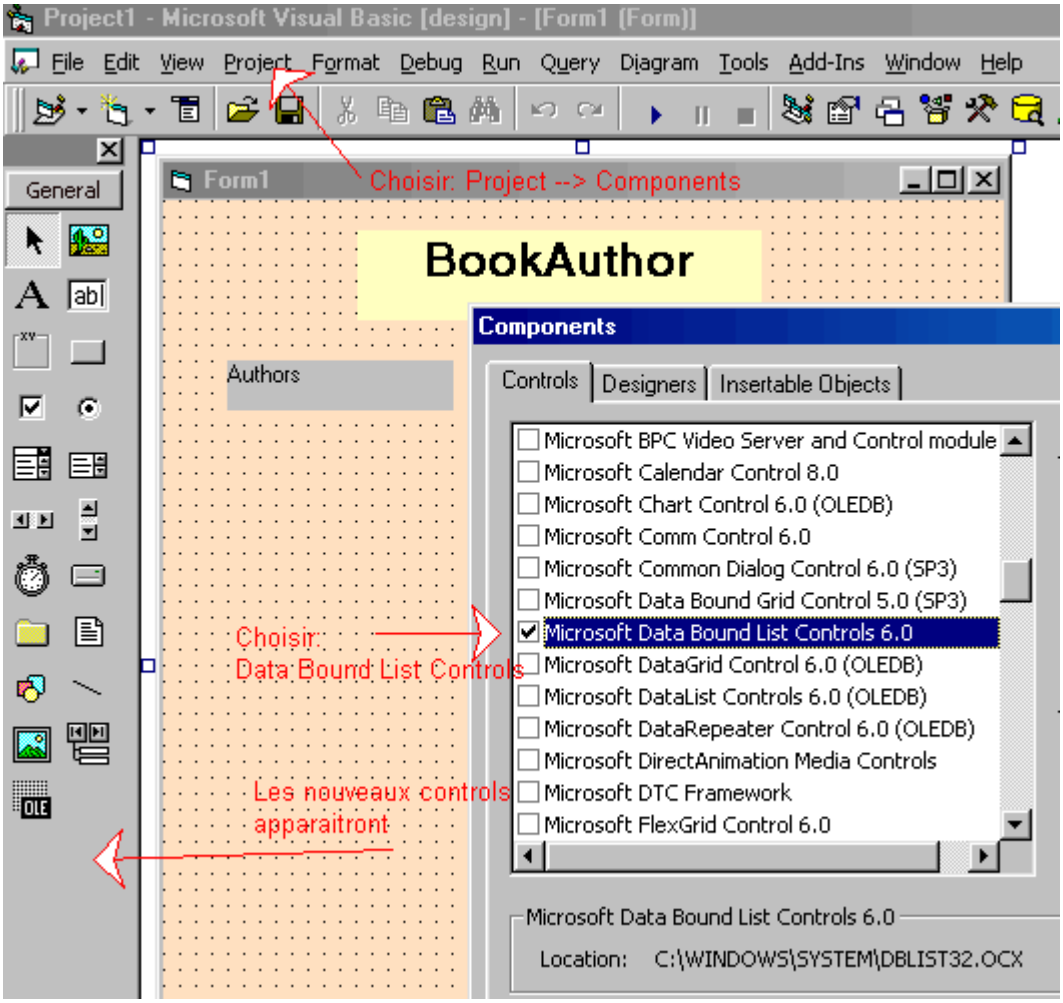
```
Private Sub txtTrouver_LostFocus()
    Dim strNom As String
    strNom = Trim(txtTrouver.Text) & "*"
    strNom = "au_Lname like '" & strNom & "'"
    If txtTrouver.Text <> "" Then
        datAuthors.Recordset.FindFirst strNom
    End If
End Sub
```

Utiliser plusieurs data controls

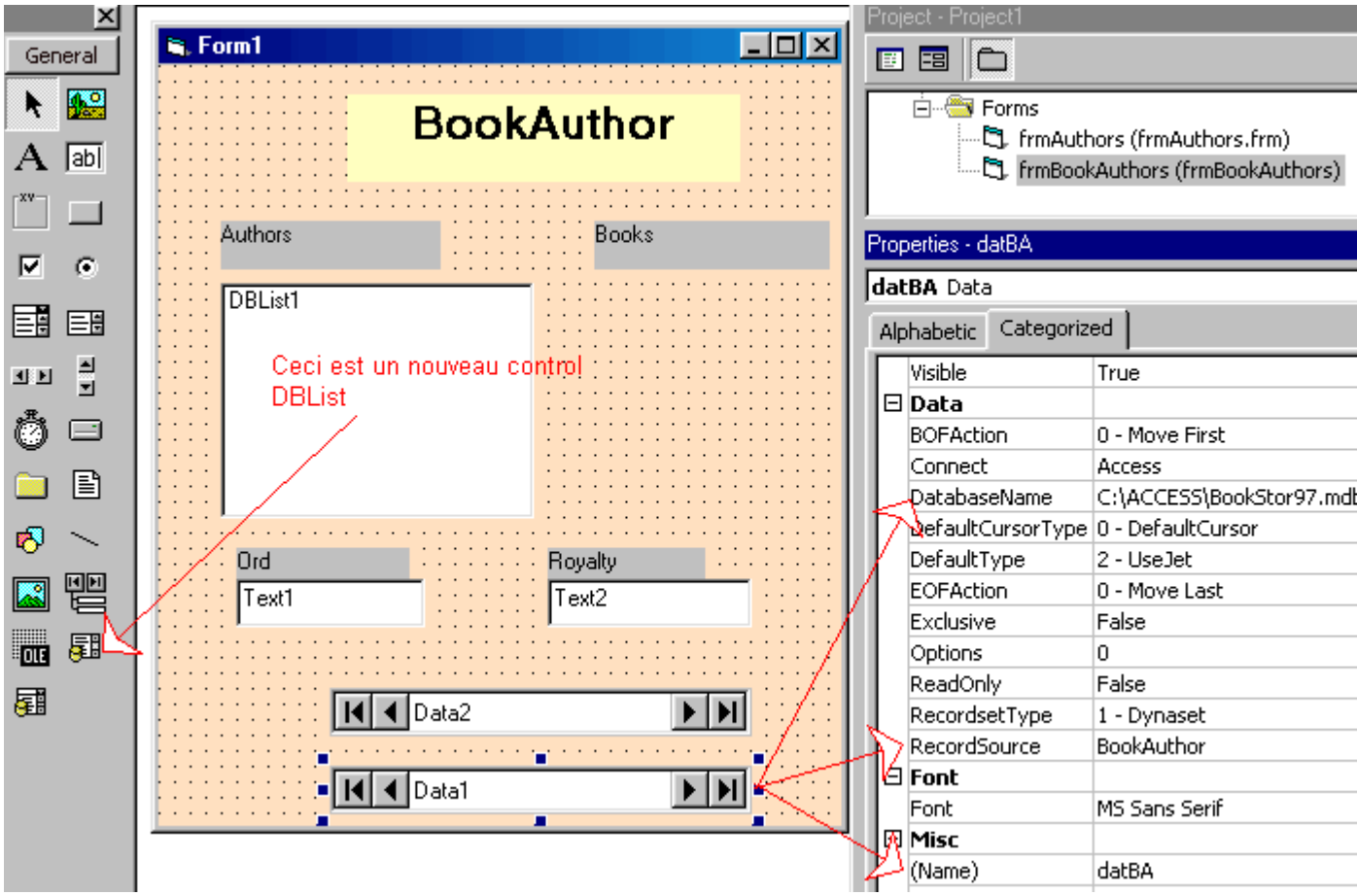
Pour utiliser plusieurs tables dans un même Form il faut créer plusieurs DataControls.

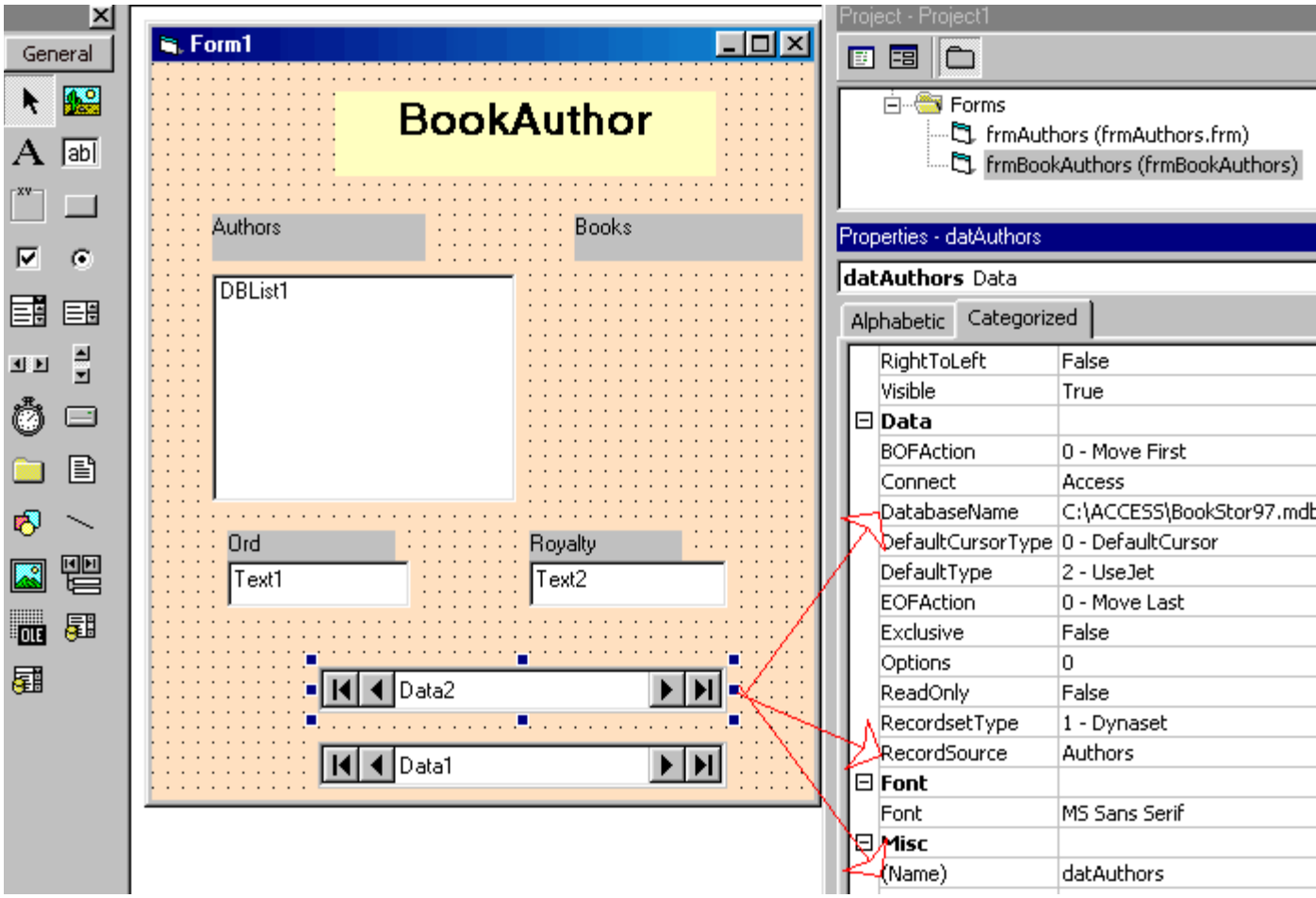
Chaque Data control est rattaché à une seule table.

Prenons le form "BookAuthor", par exemple:



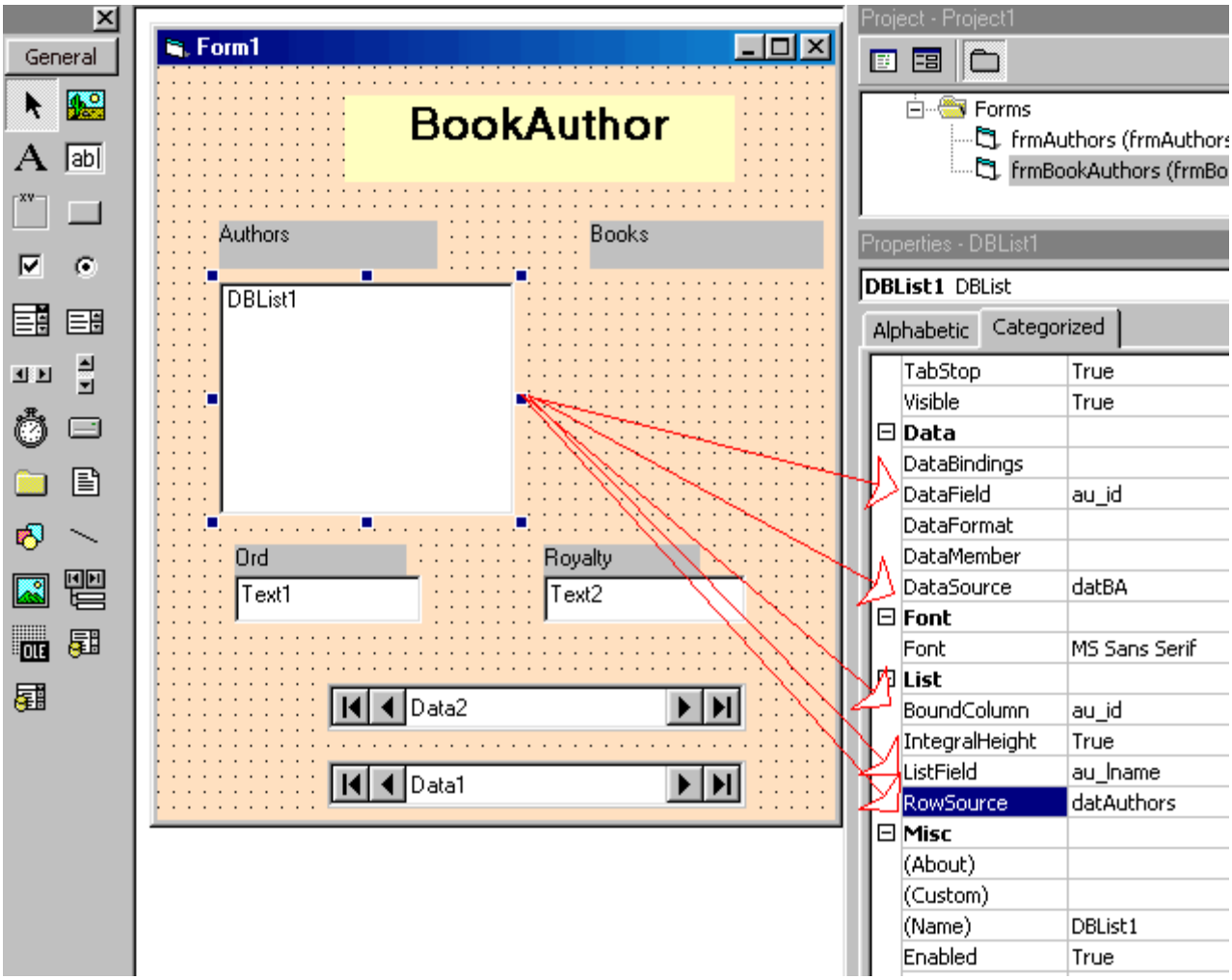
Les deux nouveaux controls sont: **DBList** et **DBCombo**. Ils fonctionnent comme le List et le Combo normal mais, ils peuvent être rattachés à une table, ce que les autres ne peuvent pas faire.


General
Form1
BookAuthor
Authors
Books
DBList1
Ceci est un nouveau control
DBList
Ord
Text1
Royalty
Text2
Data2
Data1
Project - Project1
Forms
frmAuthors (frmAuthors.frm)
frmBookAuthors (frmBookAuthors)
Properties - datBA
datBA Data
Alphabetic
Categorized
Visible
True
Data
BOFAction
0 - Move First
Connect
Access
DatabaseName
C:\ACCESS\BookStor97.mdb
DefaultCursorType
0 - DefaultCursor
DefaultType
2 - UseJet
EOFAction
0 - Move Last
Exclusive
False
Options
0
ReadOnly
False
RecordsetType
1 - Dynaset
RecordSource
BookAuthor
Font
Font
MS Sans Serif
Misc
(Name)
datBA



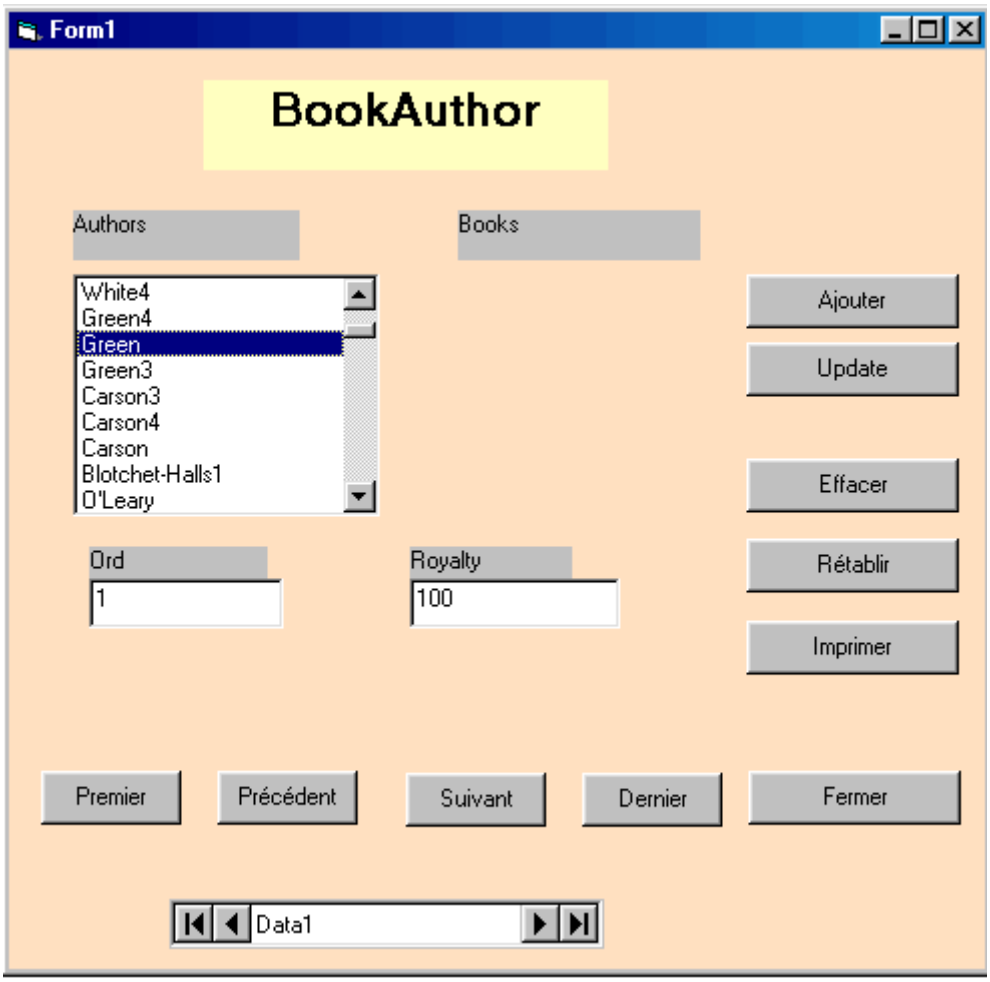
Il faut rattacher chacun des data controls à la table appropriée.

Notez que dans une base Access, une requête est traitée exactement comme une table. Un data control peut être rattaché à une Requête.

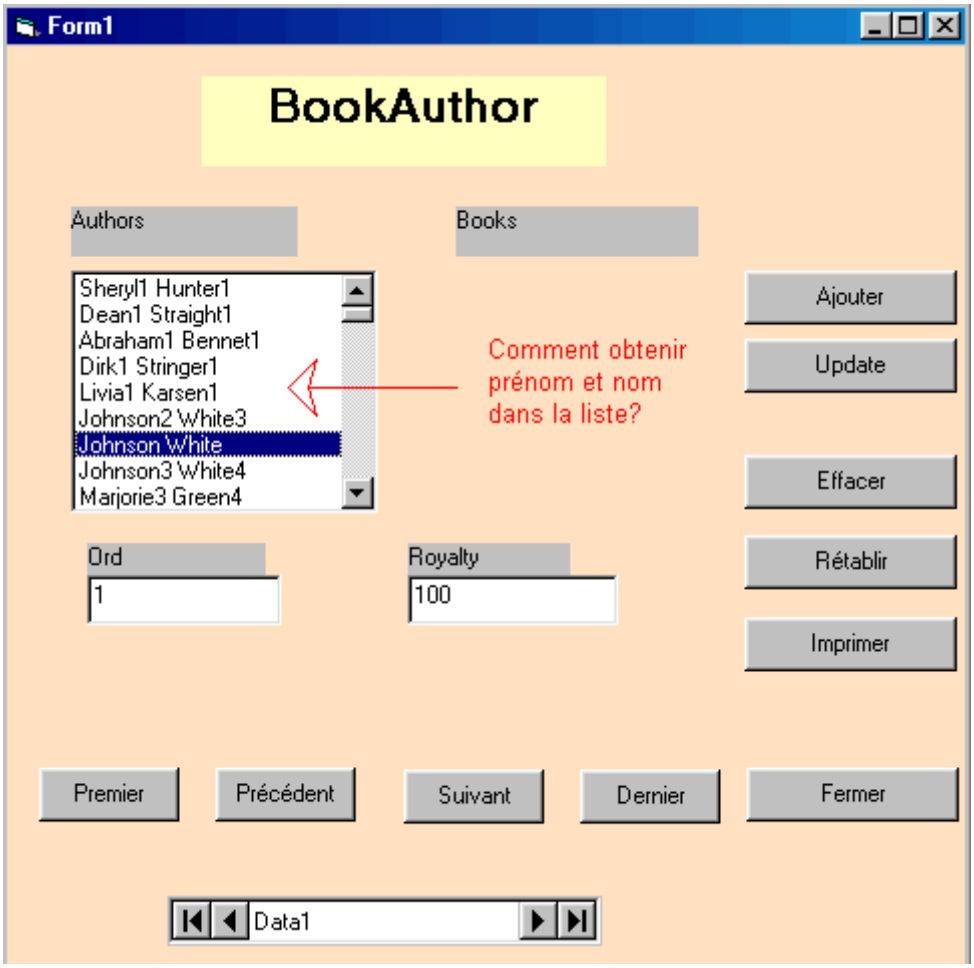


Pour afficher la liste des auteurs dans le DBList, il faut spécifier les propriétés: **List**

Les propriétés **Data** servent à spécifier où l'information saisie sera stockée.



Quiz: Comment pourriez-vous obtenir le résultat suivant dans le DBList pour Authors:



La portabilité de l'application

Pour pouvoir exécuter l'application, il faut avoir accès à la base de données.

Si le path de la b.d. est spécifié dans les propriétés on peut seulement l'appeler du même répertoire que celui où elle a été créée.

En mettant le path dans le code l'application est portable.

```
Private Sub Form_Load()
    datBA.DatabaseName = App.Path & "\bookstor97b.mdb"
    datBA.RecordSource = "bookauthor"
    datAuthors.DatabaseName = App.Path & "\bookstor97b.mdb"
    datAuthors.RecordSource = "authors"
End Sub
```

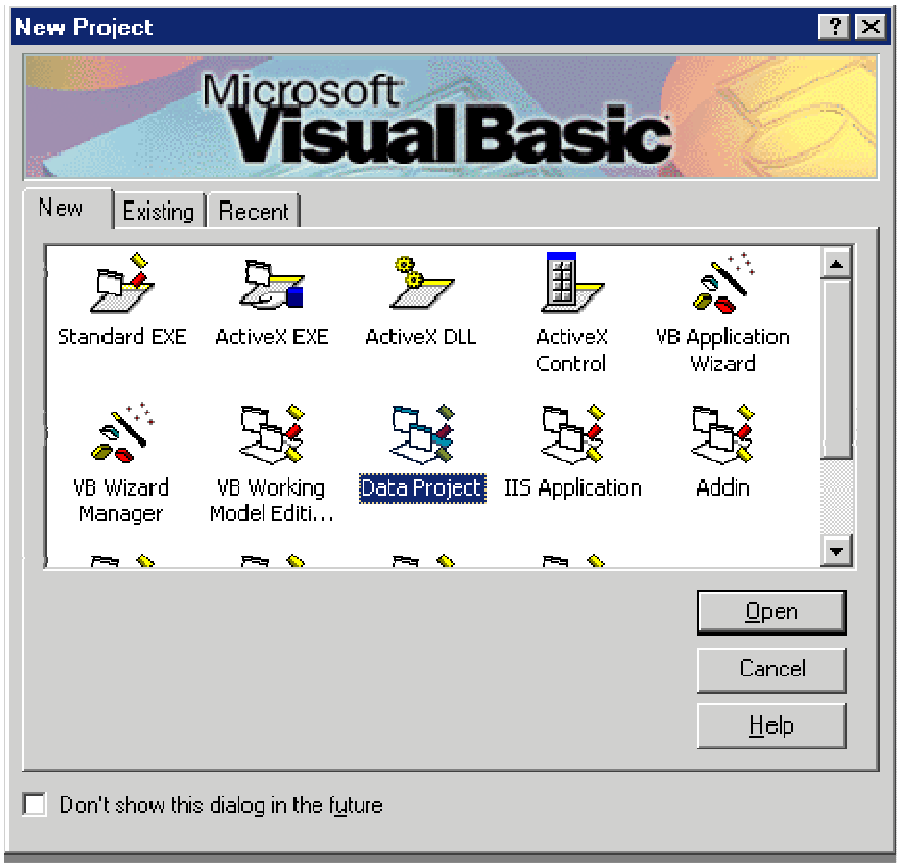
IX. Le Data Project

Impression de rapports

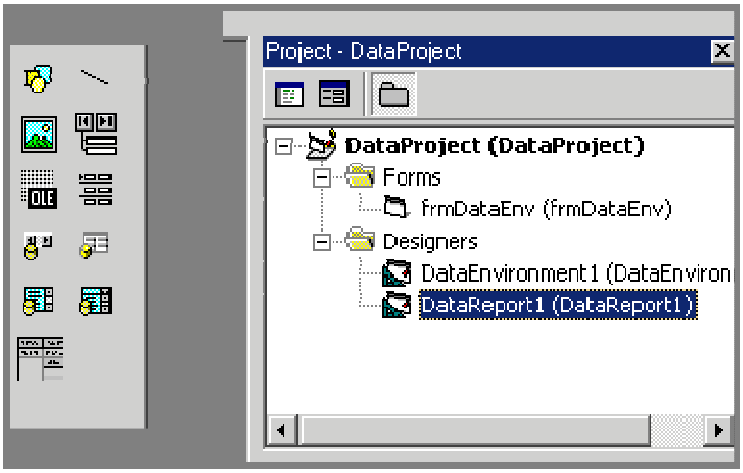
Dans la plupart des applications traitant de bases de données l'utilisateur voudra produire des imprimés des informations contenues dans la base.
Comme vous avez constaté en SQL, le langage SQL ne se prête pas vraiment à la production de rapports. VB normal n'est pas beaucoup mieux.
Par le passé, si on voulait produire des rapports en VB on utilisait habituellement un add-in. Le plus connu des add-ins pour produire les rapports est **Crystal Reports de Seagate**. Jusqu'à la version 5 de VB, Crystal Reports était inclu dans l'installation standard. En version 6 il faut l'installé séparément. Crystal Reports est un logiciel à part entière qu'il faut apprendre à manipuler. Quoi qu'il fonctionne avec une application VB et les bases de données Access, ce n'est pas comme écrire du code VB.
La façon que nous allons étudier dans ce cours est de passer par **ADO** de VB pour créer un **Data Project** et les objets qui produisent les rapports.

Étapes du Projet:

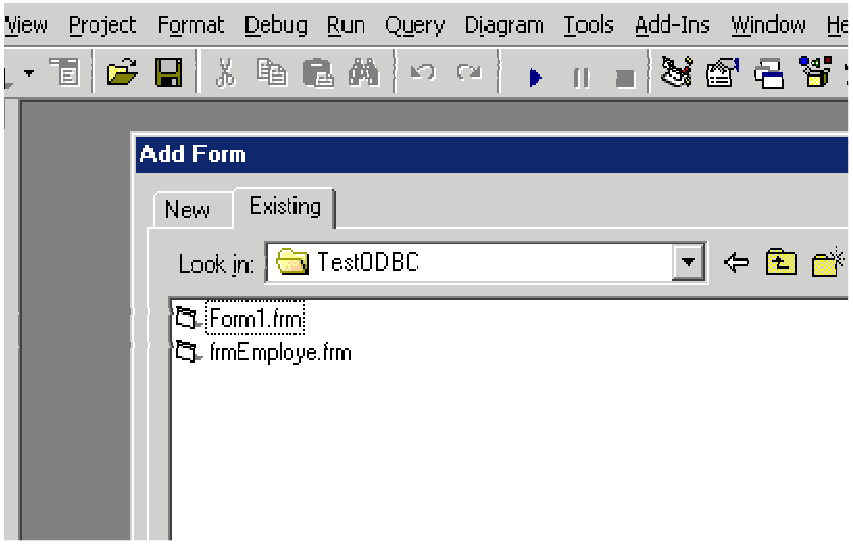
1. On crée un nouveau Projet en utilisant **Data Project**



Comme vous pouvez voir, le projet contient des contrôles additionnels dans le Toolbox et des objets dans le Project Explorer.

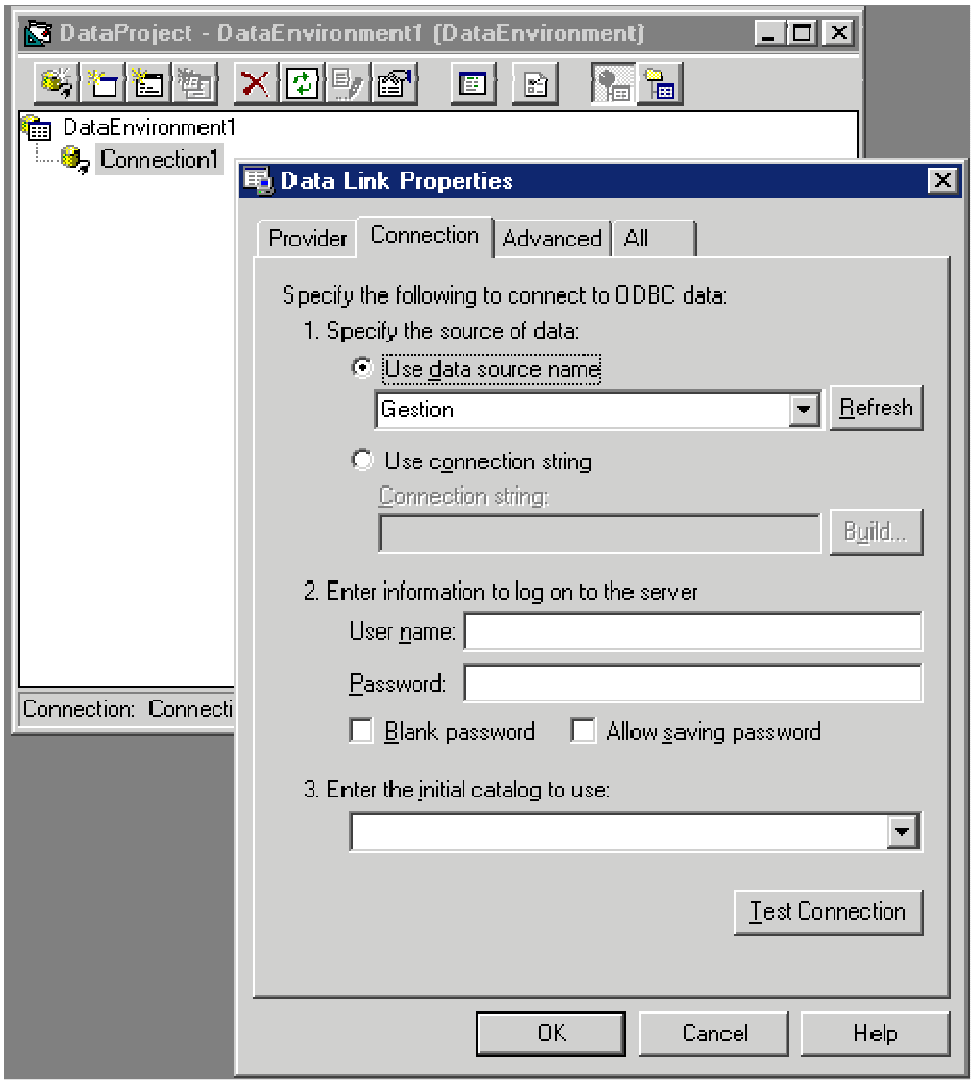


2. Si on veut transférer les forms de notre projet Gestion à celui-ci, on passe par **Menu-->Project-->Add form** et on choisi les forms qu'on veut.



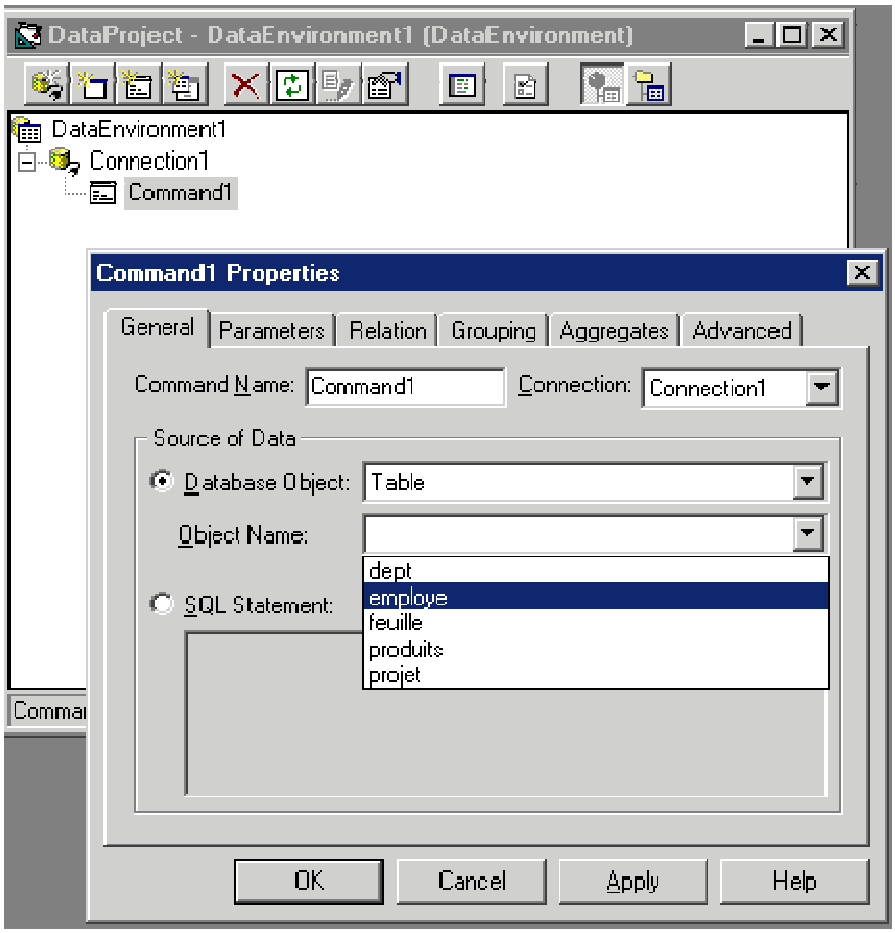
3. Il faut maintenant ouvrir la connection à la bd

- Ouvrir **DataEnvironment1** avec un double-click
- Faire un **right-click** sur **Connection1** et choisir **Properties**
- Dans la fenêtre, choisir le **DSN** pour la base de données qu'on veut ouvrir
  et faire **Test Connection** pour s'assurer que la connection est bonne

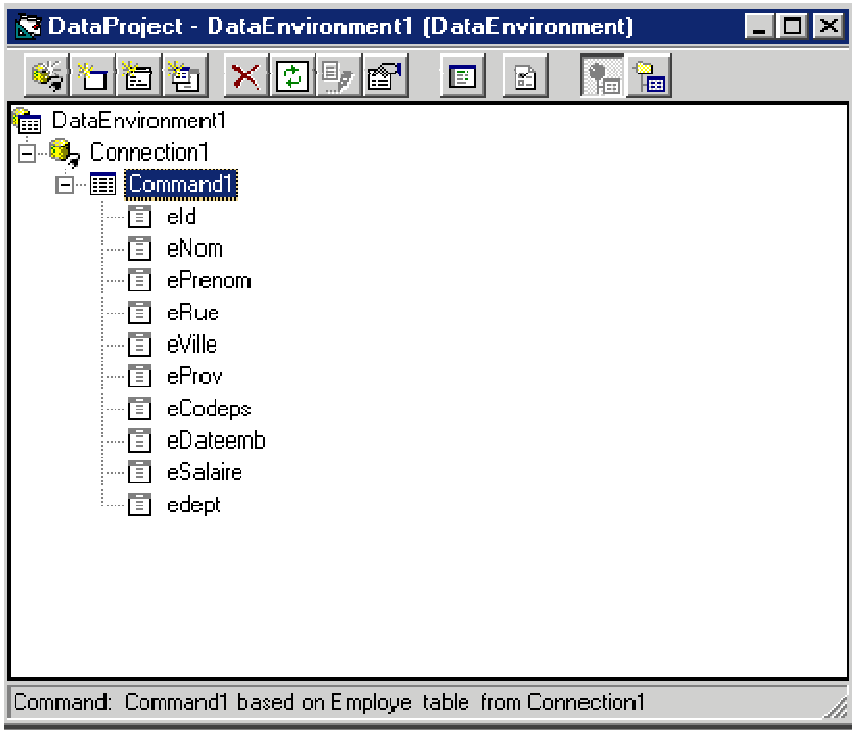


4. Ajouter un **Command object**. Le command object décrit les informations qu'on veut obtenir de la bd. En somme, le command object contient une commande **SELECT ...** qui sortira les colonnes de la table qu'on veut.

- **Right-click** sur Connection1 et choisir **Add command**
- **Right-click** sur Command1 et choisir **Properties**
- Dans fenêtre Properties, choisir: **Database Object --> Table** et **Object name --> employé** et **OK**



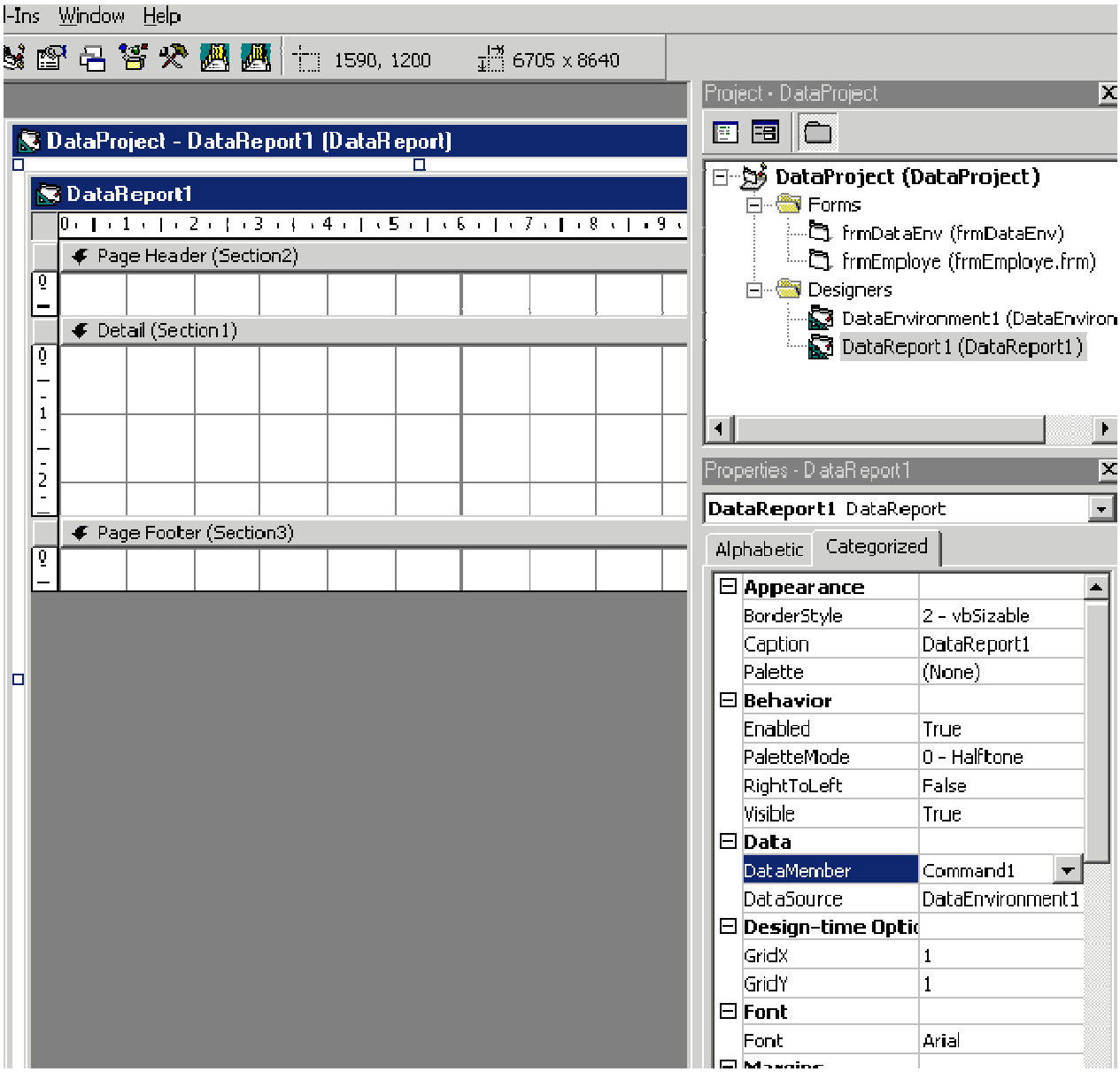
Maintenant on peut ouvrir Command1 et voir que tous les champs sont inclus:



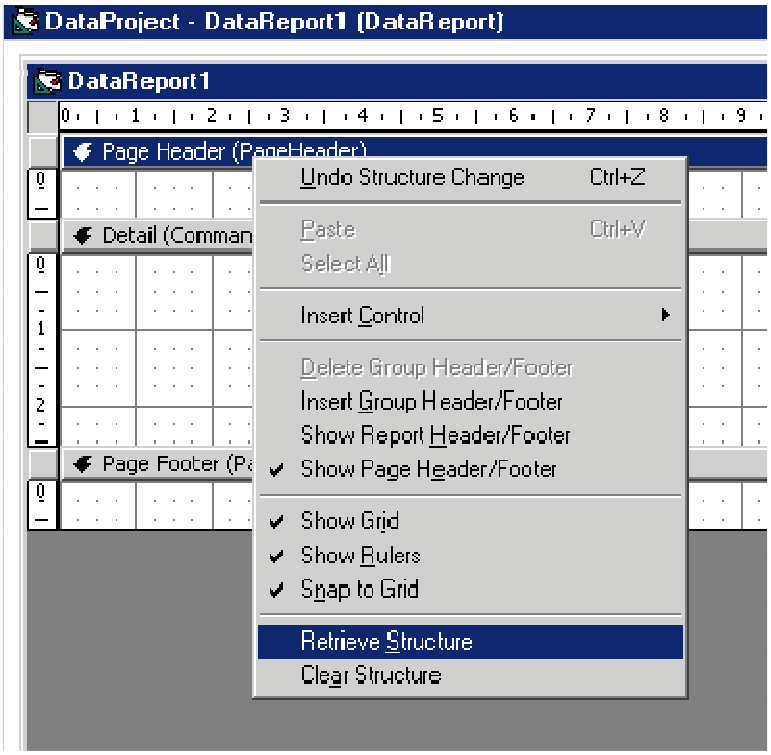
5. On ouvre **DataReport1** et on spécifie ses propriétés:

- le **DataSource** est DataEnvironment1
  et le **DataMember** est Command1

  - on change **GridX et GridY** à 4 - ceci produit des plus petits carrés de grid et c'est plus facile de positionner les objets.

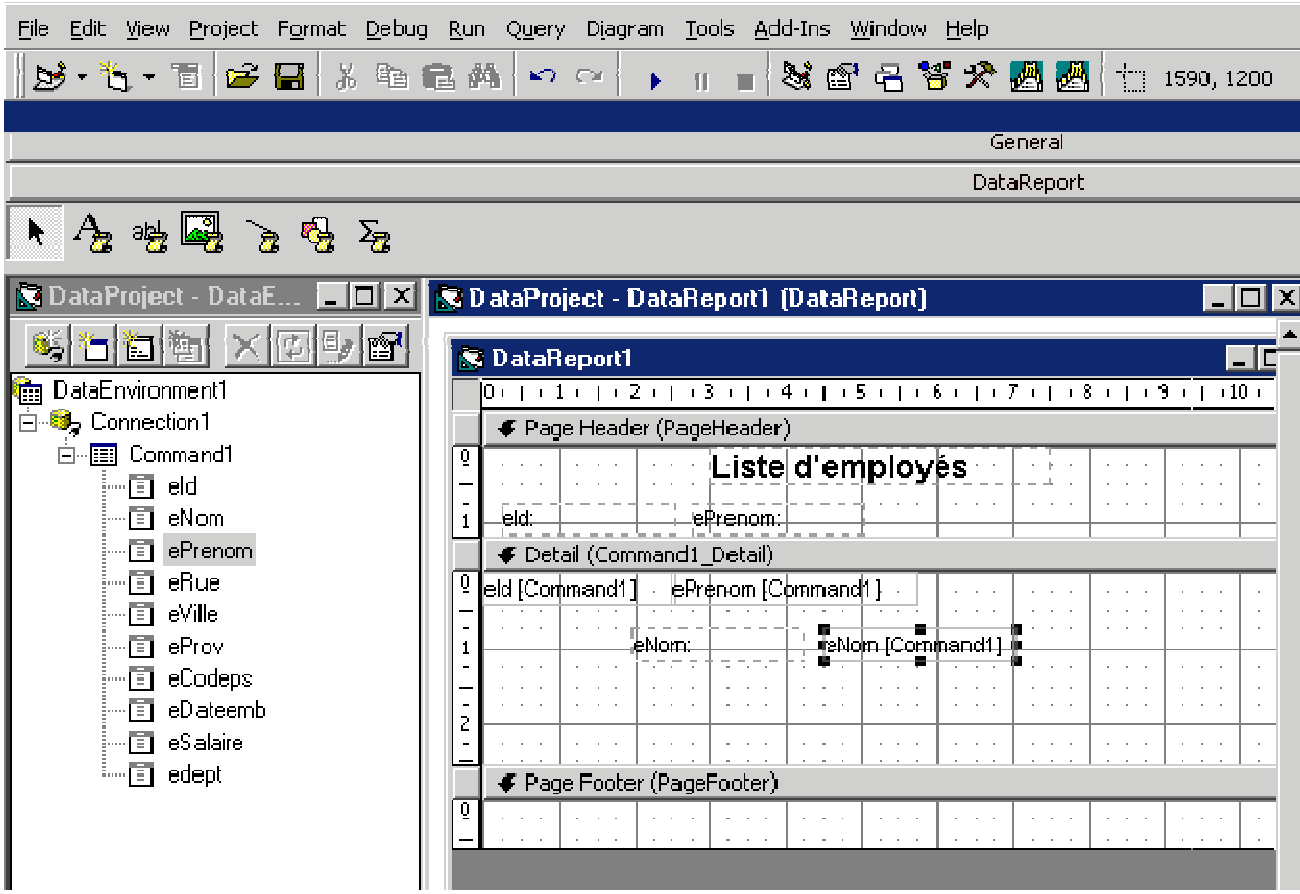


6. Avec right-click sur le report, faire: **Retrieve structure**. L'apparence ne changera pas beaucoup mais, le rapport est maintenant conforme à la structure qu'on a décrite dans le Command1.



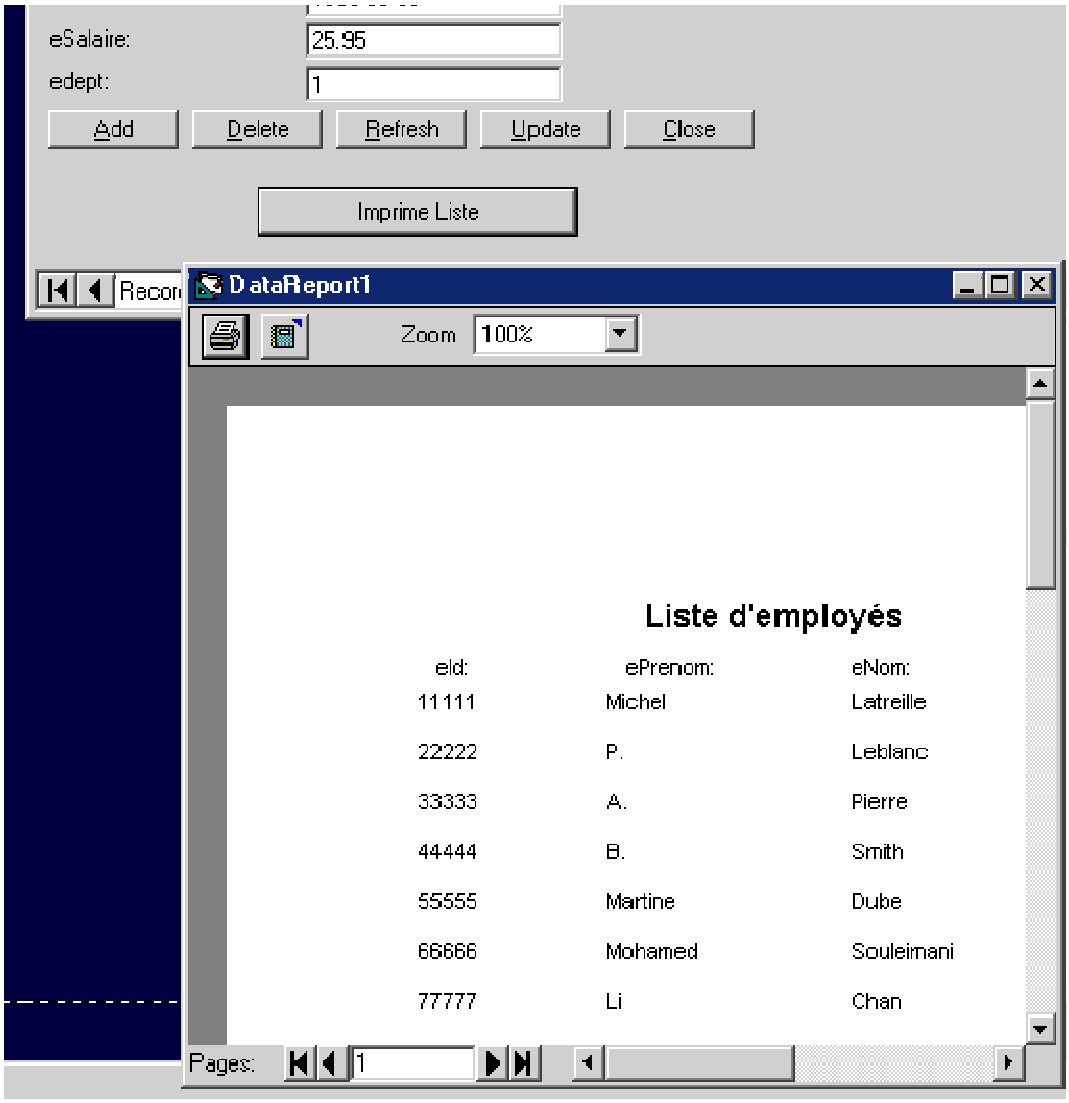
7. Pour créer le rapport, on ouvre Command1 et Data Report côte à côte et on "**drag and drop**" les champs qu'on veut imprimer de Command1 à Data Report.

On ajoute les titres appropriés, on fait la mise en page sur le rapport, on place les colonnes, etc.



8. Pour ouvrir le DataReport, il faut mettre un bouton sur une feuille, soit le menu ou la form Employé, par exemple.

Le code du bouton est: **DataReport1.Show**



FIN

NB : Pour être très à jour avec la programmation orientée objet, rendez vous l'année prochaine avec VB.net. Pour toutes autres informations relatives au domaine n'hésitez pas à contacter : wasingface@gmail.com