# VISUAL BASIC 6

DESS IAIE, ISTIA, Université d'Angers
2003/2004, 30h
Jean-Louis Boimond

---

**Annexes :**



**Bibliographies :**

1) *MSDN Library Visual Studio 6.0*

2) *Visual Basic 6: How to Program*, H.M. Deitel, P.J. Deitel, T.R. Nieto, Prentice Hall, Upper Saddle River, New Jersey 07458, 1015 pages, www.prenhall.com/deitel
3) *Visual Basic 6*, Peter Wright, Eyrolles, 767 pages, www.eyrolles.com
4) *Visual Basic 6*, M. Franke, Micro Application, Paris, 493 pages, www.microapp.com
5) *Visual Basic 6 : Le guide du programmeur*, G. Frantz, Edition OEM, 1272 pages, *www.oemweb.com*
6) *Programmer avec Visual Basic*, Alain Godon, cours $M^2$AI - ISTIA, 2001/2002, 28 pages
7) *Visual Basic version 5*, Roland Guihur, oct. 1999, 47 pages
8) *T.P. de Visual Basic*, Serge Tahé, Innovation - ISTIA

---

# 1 INTRODUCTION

Visual Basic s'est développé à partir du langage BASIC (Beginner's All-purpose Symbolic Instruction Code, Larousse : " Langage de programmation conçu pour l'utilisation interactive de terminaux ou de micro-ordinateurs ", 1960). Le but du langage BASIC était d'aider les gens à apprendre la programmation. Son utilisation très répandue conduisit à de nombreuses améliorations. Avec le développement (1980-1990) de l'interface utilisateur graphique (*Graphical User Interface - GUI*) de Windows Microsoft, BASIC a naturellement évolué pour donner lieu à Visual Basic (1991). Depuis, plusieurs versions ont été proposées, Visual Basic 6 est apparue en 1998.

Visual Basic est un langage de programmation existant actuellement en trois versions (*Learning, Professional, Entreprise*). Les programmes (aussi appelées *applications*) sont créés dans un environnement de développement intégré (*Integrated Development Environment - IDE*), ceci dans le but de créer, exécuter et déboguer les programmes d'une manière efficace. Ce langage est réputé pour permettre un développement rapide d'applications. Outre une interface utilisateur graphique, il dispose de caractéristiques telles que la manipulation d'événements, un accès à Win32 API, la programmation orientée objet, la gestion d'erreurs, la programmation structurée. C'est un langage interprété, notons que les éditions *Professional* et *Entreprise* permettent une compilation en *code natif* (*code machine*) (voir annexe A pour plus de détails).

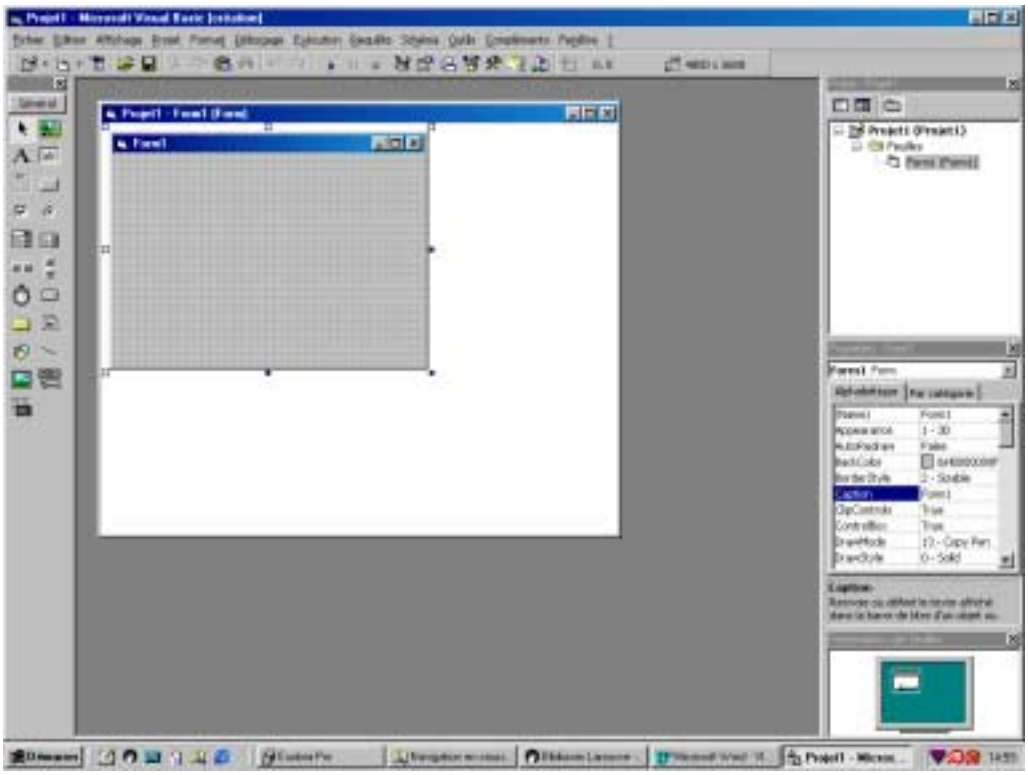
# 2 ENVIRONNEMENT DE DÉVELOPPEMENT INTÉGRÉ (EDI)

L'environnement de développement intégré de Visual Basic permet de créer, exécuter et déboguer des programmes Windows dans une seule application (à savoir Visual Basic).

## 2.1 Présentation de l'Environnement de Développement Intégré (EDI)

Au démarrage de Visual Basic, la boîte de dialogue suivante, intitulée *Nouveau projet* (cf. barre de titre), s'affiche. Elle permet de choisir le type de projet que l'on souhaite créer.


Double-cliquer sur l'option *EXE Standard* (surlignée par défaut) de l'onglet *Nouveau* afin de créer un projet (ensemble de fichiers permettant de créer une application Windows). Certaines options de cette boîte de dialogue sont brièvement décrites en annexe B. L'interface de l'EDI de Visual Basic s'affiche (voir la recopie d'écran dans la figure suivante).

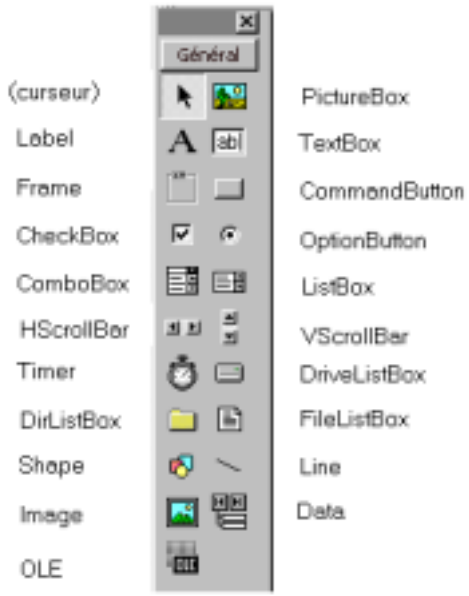


Cet environnement, situé dans une fenêtre intitulée *Projet1 - Microsoft Visual Basic [création]* (cf. barre de titre), est formé d'une *barre de menu*, d'une *barre de contrôles* et de plusieurs fenêtres.

- Le tableau suivant récapitule les menus accessibles à partir de la *barre de menu*, située en haut de l'écran. En dessous sont situés, dans une barre d'outils, les raccourcis graphiques (icônes) représentant les commandes les plus courantes.

| Menu | Description |
|---|---|
| Fichier | Contient des options pour ouvrir, fermer, écrire, … des projets. |
| Edition | Contient des options telles que couper, copier, coller, etc. |
| Affichage | Contient des options concernant l'EDI. |
| Projet | Contient des options pour ajouter au projet des particularités telles que des feuilles. |
| Format | Contient des options pour aligner et verrouiller les contrôles d'une feuille. |
| Débogage | Contient des options pour déboguer le programme. |
| Exécution | Contient des options pour exécuter, stopper, … un programme. |
| Requête | Contient des options pour manipuler des données récupérées d'une base de données. |
| Schéma | Contient des options pour éditer des bases de données. |
| Outils | Contient des options pour particulariser l'EDI. |
| Compléments | Contient des options pour utiliser, installer et désinstaller des compléments (programmes visant à augmenter la performance de Visual Basic). |
| Fenêtres | Contient des options pour disposer les fenêtres à l'écran. |
| Aide | Contient des options pour obtenir de l'aide. |

- La *barre de contrôles*, située dans la partie gauche de l'écran, contient des contrôles permettant de réaliser l'Interface Utilisateur Graphique (IUG) (i.e., la partie de la feuille visible par l'utilisateur). Ces derniers permettent une programmation accélérée. La figure et le tableau suivants résument les contrôles contenus dans cette barre.




| Contrôle | Description |
|---|---|
| Curseur | Permet d'interagir avec les contrôles de la feuille (en fait, ce n'est pas un contrôle). |
| PictureBox (zone d'image) | Permet l'affichage d'un fichier image (bmp, ico, wmf, …). |
| Label (étiquette) | Permet l'affichage d'un texte non modifiable directement par l'utilisateur. |
| TextBox (zone de saisie) | Permet à l'utilisateur d'entrer du texte. |
| Frame (cadre) | Permet de regrouper d'autres contrôles tels que des cases à option. |
| CommandButton (bouton) | Ce contrôle est représenté par un bouton que l'utilisateur peut presser, ou cliquer, pour exécuter une action sous-jacente. |
| CheckBox (case à cocher) | Permet de fournir un bouton de choix (*checked* ou *unchecked*). |
| OptionButton (case à option) | Les cases à option sont utilisées en groupe (dans le même cadre), sachant que seulement une case peut être sélectionnée (*True*) à la fois. |
| ListBox (liste) | Permet l'affichage de différents items (éléments). |
| ComboBox (liste combinée) | Permet de combiner les fonctionnalités des zones de saisie et des listes. |
| HScrollBar | Une barre de défilement horizontale. |
| VScrollBar | Une barre de défilement verticale. |
| Timer (horloge) | Permet la répétitivité de tâches (ce contrôle est non visible pour l'utilisateur). |
| DriveListBox | Permet un accès aux différents disques du système. |
| DirListBox | Permet un accès à des répertoires. |
| FileListBox | Permet d'accéder aux fichiers d'un répertoire. |
| Shape | Permet de dessiner des cercles, des rectangles, des carrés ou des ellipses. |
| Line | Permet de dessiner des lignes. |
| Image (dessin) | Similaire au contrôle PictureBox avec des capacités moindres. |
| Data | Permet la connexion à une base de données. |
| OLE | Permet d'interagir avec d'autres applications Windows. |

- L'EDI d'un projet *EXE Standard* contient les fenêtre suivantes :
  - *Projet1 – Form1 (Form)*
  - *Présentation des feuilles*

- *Propriétés – Form1*
- *Projet – Projet1*

- La fenêtre *Projet1 – Form1 (Form)* contient une feuille (en anglais *form*) vierge nommée, par défaut, *Form1*, dans laquelle le programme de l'IUG sera conçu. L'IUG est la partie visible du programme (boutons, cases à cocher, cases à option, zones de saisie, etc.), elle permet :
  - à un utilisateur de fournir des données (appelées *entrées*) au programme,
  - au programme de restituer des résultats (appelés *sorties*) à l'utilisateur.
- La fenêtre *Présentation des feuilles* (en anglais *Form Layout*) permet de spécifier - à l'aide de la souris - la position souhaitée de la feuille (dans le cas présent *Form1*) sur l'écran lors de l'exécution du programme.
- La fenêtre *Propriétés – Form1* décrit les propriétés (taille, position, couleur, etc.) d'une feuille ou d'un contrôle. Dans le cas présent, sont listées les propriétés liées à la feuille *Form1*. Chaque type de contrôle a son propre ensemble de propriétés. Certaines propriétés, telles que *Height* et *Width* sont communes aux feuilles et aux contrôles, alors que d'autres propriétés sont uniques à une feuille, ou un contrôle.
- La fenêtre *Projet – Projet1* est l'explorateur de projets, elle regroupe par type les différents fichiers composant le projet en cours. La barre de menu de cette fenêtre comporte 3 boutons : *Code* afin de visualiser la fenêtre contenant le code du programme, *Afficher l'objet* pour faire apparaître la feuille, *Basculer les dossiers* cache, ou montre, le dossier *Feuilles*. Par défaut, le projet a pour nom *Projet1* et est constitué d'une seule feuille, nommée *Form1*.

## 2.2 Un premier exemple : Affichage d'une ligne de texte

Réalisons un programme qui écrit le texte " Premier exemple " à l'écran. Ce programme, de part sa simplicité, se fait sans écrire une seule ligne de code. En fait, nous allons utiliser les techniques de la programmation visuelle dans laquelle, à travers différentes manipulations (tel qu'un clic de souris), vont être fournies les informations suffisantes à Visual Basic pour qu'il puisse automatiquement générer le code de notre programme.

Voici la démarche. Nous allons ajouter un contrôle *Label* sur l'IUG d'une feuille : Le fait de double-cliquer sur ce contrôle fait apparaître un contrôle *Label*, nommé de manière standard *Label1*, au centre de l'IUG de la feuille. L'utilisation de la souris permet de le positionner et de le dimensionner à volonté (la grille, visible en mode *création*, permet un alignement des contrôles, cf. onglet *Général* du menu *Outils* | *Options*).
La propriété *Caption* du contrôle *Label1*, visible dans la fenêtre des propriétés (fenêtre *Propriétés – Label1*), détermine le texte affiché par ce contrôle. Il suffit donc de remplacer le texte " Label1 ", mis par défaut, par celui souhaité (à savoir " Premier exemple ").

**Remarques :**
- La propriété *Caption* ne doit pas être confondue avec la propriété *Name* du contrôle, bien que leurs contenus soient par défaut les mêmes !
- Le fait de préfixer le *Name* de chaque *Label* avec les lettres (minuscules) *lbl* permet une meilleure identification des contrôles *Label*. Il en sera de même pour les autres contrôles. Cette convention, largement adoptée, permet une meilleure " lisibilité " du programme.

**Attention :** Avant de tenter une exécution, il est prudent de sauvegarder le projet (cf. menu *Fichier* | *Enregistrer le projet sous …*). En ce qui concerne notre exemple, la sauvegarde du projet, nommé *Projet1* par défaut, donne lieu à la création de deux fichiers (texte) :

- l'un, avec l'extension *.frm*, pour mémoriser les propriétés de la feuille, nommée *Form1* par défaut. En fait, à chaque élément du projet (feuille ou module) va correspondre un fichier (*.frm* ou *.bas*).
- l'autre, avec l'extension *.vbp*, pour mémoriser les paramètres des variables d'environnement et des noms de fichiers des éléments constituant le projet.

Jusqu'à présent, nous avons travaillé dans le mode *création* de l'EDI (i.e., le programme n'est pas exécuté). En plaçant l'EDI en mode *exécution* (obtenu en cliquant le bouton *Exécuter*, ou en activant le menu *Exécution | Exécuter*), le programme est exécuté (avec une possible interaction *via* le programme de l'IUG). Il s'ensuit alors que :

- la fenêtre de l'EDI est intitulée *Projet1 - Microsoft Visual Basic [exécution]* (cf. barre de titre),
- la plupart des fenêtres utilisables dans le mode *création* sont indisponibles (c'est le cas, par exemple, de la fenêtre *Propriétés*). Notons l'apparition d'une fenêtre nommée *Exécution*, habituellement utilisée pour déboguer le programme.

# 3 INTRODUCTION A LA PROGRAMMATION DE VISUAL BASIC

La réalisation d'une application s'appuie essentiellement sur l'association de deux éléments : Les objets et les événements.

## 3.1 La programmation orientée objet

Visual Basic permet le développement orienté objet. Au lieu de résoudre un problème en le décomposant en problèmes plus petits, il s'agit de le scinder sous forme d'objets qui existent chacun indépendamment les uns des autres. Chaque objet possède certaines caractéristiques (appelées *propriétés*) et fonctions qu'il serait en mesure d'effectuer (appelées *méthodes*).
A titre d'illustration, la feuille " *Form1* " est un objet ayant, entre autres, une propriété appelée " *Caption* " qui permet de spécifier le texte affiché dans la barre de titre de la feuille, et deux méthodes appelées " *Show* " et " *Hide* " qui permettent respectivement d'afficher et de cacher la feuille.

Très brièvement, on peut dire que les objets " encapsulent " les données (les attributs) et les méthodes (les comportements), sachant que les données et les méthodes sont intimement liées. Les objets ont la propriété de " cacher l'information ", au sens où la communication d'un objet avec un autre ne nécessite pas de connaître comment ces objets sont " implémentés " (à titre d'illustration, il est possible de conduire une voiture sans connaître en détail son fonctionnement mécanique).

Alors que dans un langage procédural, la fonction joue le rôle de l'unité de programmation (les fonctions en langage objet sont appelées méthodes), l'unité de programmation en langage objet est la *classe* à partir laquelle les objets sont " instanciés " (créés) (tout objet appartient à une classe). En fait, la communication avec un objet se fait notamment à travers ses propriétés et ses méthodes. Une propriété a une valeur que le programmeur peut consulter, ou modifier ; une méthode est une procédure permettant d'agir sur les objets d'une classe. Le programmeur peut faire référence à la propriété *prop*, respectivement à la méthode *meth*, de l'objet *obj* en écrivant **obj.prop**, respectivement **obj.meth**.

De nombreuses classes sont prédéfinies dans Visual Basic. Par exemple, chaque icône de contrôle, situé dans la barre de contrôles de l'EDI, représente en fait une classe. Lorsqu'un contrôle est déplacé vers une feuille, Visual Basic crée automatiquement un objet de cette classe, vers lequel votre programme pourra envoyer des messages ; ces objets pouvant aussi générer des événements. Les versions récentes de Visual Basic permettent le développement de ses propres classes (et contrôles).

## 3.2 Programmation événementielle

Visual Basic permet la création d'IUG, sous une forme standard dans l'environnement Windows, par simple pointage et cliquage de la souris, ce qui a pour intérêt d'éliminer l'écriture du code permettant de générer l'IUG d'une feuille, de fixer ses propriétés, de créer les contrôles contenus dans l'IUG.
Il s'agit pour le programmeur de créer l'IUG d'une feuille et d'écrire le code décrivant ce qui se produit lorsque l'utilisateur interagit avec l'IUG (clic, double-clic, appuie sur une touche, etc.). Ces actions, appelées événements (en anglais *events*), sont reliées au programme *via* le système d'exploitation de Microsoft Windows (voir l'annexe C pour plus de précisions). En fait, il est possible pour chacun des contrôles contenus dans l'IUG de répondre, de manière appropriée, aux différents événements auxquels sont sensibles les contrôles.
Pour cela, il va correspondre à chacun de ces événements une procédure de réponse (en anglais *event procedure*), dans laquelle le programmeur insérera des lignes de code approprié. Une telle programmation de code en réponse à ces événements est dite programmation événementielle.
Ainsi, ce n'est plus le programmeur mais l'utilisateur qui maîtrise l'ordre d'exécution du programme (le programme ne dicte plus la conduite de l'utilisateur).
Dans un programme Visual Basic, tous les contrôles (zones de texte, boutons de commande, etc.) possèdent un ensemble d'événements prédéfinis auxquels on peut lier du code (pour ne pas réagir à un événement prédéfini, il suffit de ne pas écrire de code correspondant). A titre d'illustration, l'événement appelé " *Load* " de la feuille " *Form1* " est déclenché à chaque premier chargement de la feuille (ici " *Form1* "), juste avant son affichage. Il sert habituellement à fixer les valeurs des propriétés de la feuille, ou à exécuter du code lors de son apparition. Voir l'annexe D pour plus de précisions.

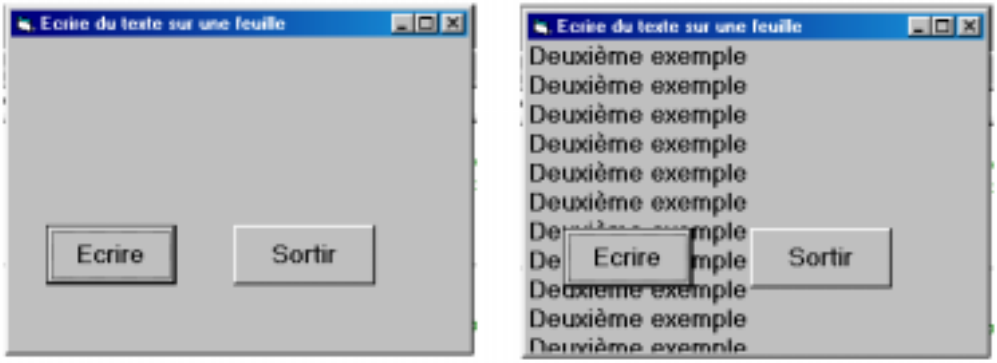
En bref, un programme (une application) Visual Basic est créé à partir d'un projet constitué, notamment, de feuilles, lesquelles comprennent :
- une IUG composée de contrôles,
- les procédures de réponses aux événements associées à ces contrôles.

Notons qu'une feuille peut aussi contenir des procédures et des déclarations locales à la feuille, sans lien direct avec un contrôle contenu dans l'IUG. Nous verrons par la suite, qu'en plus des feuilles, un projet peut être constitué de modules.

## 3.3 Deux exemples

### 3.3.1 Écriture d'une ligne de texte sur une feuille

L'IUG consiste en deux boutons : **Ecrire** et **Sortir**. Il s'agit d'écrire la ligne " Deuxième exemple " autant de fois que l'utilisateur clique sur le bouton **Ecrire**. Le fait de cliquer sur le bouton **Sortir** permet de terminer l'exécution du programme. Les recopies d'écran qui suivent montrent, à gauche, la feuille avant un clic sur le bouton **Ecrire**, à droite, la feuille après plusieurs clics sur ce bouton.



Le tableau qui suit liste les objets et certaines de leurs propriétés (seules les propriétés ayant été modifiées sont listées).

| Objet | Propriété | Valeur de la propriété | Description |
|---|---|---|---|
| Form | Name | frmExemple2 | Identification de la feuille. |
| | Caption | Ecrire du texte sur une feuille | Spécification du texte affiché dans la barre de titre de la feuille. |
| | Font | MS Sans Serif Gras 12 pt | Spécification de la police de caractères de la feuille. |
| CommandButton **Ecrire** | Name | cmdDisplay | Identification du bouton **Ecrire**. |
| | Caption | Ecrire | Texte qui apparaît sur le bouton. |
| | Font | MS Sans Serif Gras 12 pt | Spécification de la police de caractères du texte Caption. |
| | TabIndex | 0 | Numéro d'ordre de Tab. |
| CommandButton **Sortir** | Name | cmdExit | Identification du bouton **Sortir**. |
| | Caption | Sortir | Texte qui apparaît sur le bouton. |
| | Font | MS Sans Serif Gras 12 pt | Spécification de la police de caractères du texte Caption. |
| | TabIndex | 1 | Numéro d'ordre de Tab. |

La propriété *TabIndex* permet de contrôler l'ordre dans lequel les contrôles, constituant l'IUG, reçoivent le *focus* (i.e., deviennent actifs). Le contrôle, ayant une valeur de *TabIndex* égale à 0, a le *focus* au départ.

Le code programme qui suit permet :
- d'afficher dans la feuille le texte " Deuxième exemple " avec la police de caractères souhaitée lors d'un clic sur la touche **Ecrire** (cf. la procédure de réponse à l'événement, nommée *cmdDisplay_Click*).
- de terminer le programme lors d'un clic sur la touche **Sortir** (cf. la procédure de réponse à l'événement, nommée *cmdExit_Click*).

---

```
Private Sub cmdDisplay_Click()
    ' A chaque fois que le bouton "Ecrire" est cliqué,
    ' le message "Deuxième exemple" est affiché sur la feuille
    Print "Deuxième exemple"
End Sub

Private Sub cmdExit_Click()
    End         ' Termine l'exécution du programme
End Sub
```

---

*Private Sub* et *End Sub* marquent respectivement le début et la fin d'une procédure. Le code que le programmeur veut exécuter lors d'un clic sur la touche **Ecrire**, est placé entre le début et la fin de la procédure, i.e., *Private Sub cmdDisplay_Click()* et *End Sub*.

```
    ' A chaque fois que le bouton "Ecrire" est cliqué,
    ' le message "Deuxième exemple" est affiché sur la feuille
```

sont des lignes de commentaires.

La ligne

**Print "Deuxième exemple"**

affiche le texte " Deuxième exemple " sur la feuille (en cours) en utilisant la méthode *Print*. En toute rigueur, il faudrait faire précéder la méthode *Print* du nom de la feuille, soit la ligne

**frmExemple2.Print "Deuxième exemple"** (nom de l'objet.nom de la propriété)

Notons que cette méthode n'est pas la plus appropriée lorsque la feuille contient des contrôles, il serait préférable d'écrire le texte dans un contrôle de type *TextBox* (détaillé au § 8.2).

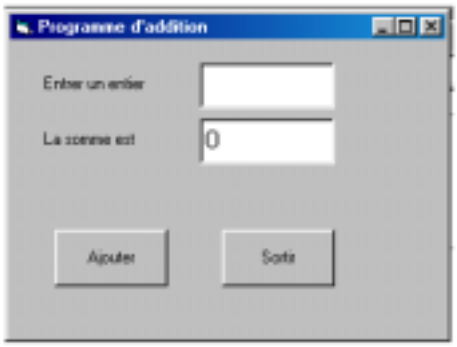
La ligne

**End**

située dans la procédure *cmdExit*, termine l'exécution du programme (i.e., place l'EDI en mode création).

### 3.3.2 Addition d'entiers

A partir d'entiers introduits par l'utilisateur, il s'agit d'afficher leur somme successive. L'IUG consiste en deux étiquettes (*Label*), deux zones de saisie (*TextBox*) et deux boutons (*CommandButton*), cf. la recopie d'écran dans la figure suivante.




Le tableau qui suit liste les objets et certaines de leurs propriétés (seules les propriétés ayant été modifiées sont listées).

| Objet | Propriété | Valeur de la propriété | Description |
|---|---|---|---|
| Form | Name | frmExemple3 | Identification de la feuille. |
| | Caption | Programme d'addition | Spécification du texte affiché dans la barre de titre de la feuille. |
| CommandButton **Ajouter** | Name | cmdAjouter | Identification du bouton **Ajouter**. |
| | Caption | Ajouter | Texte qui apparaît sur le bouton. |
| CommandButton **Sortir** | Name | cmdSortir | Identification du bouton **Sortir**. |

| | | | |
|---|---|---|---|
| | Caption | Sortir | Texte qui apparaît sur le bouton. |
| Label | Name | lbl1 | Identification du label. |
| | Caption | Entrer un entier | Texte qui apparaît dans le label. |
| Label | Name | lbl2 | Identification du label. |
| | Caption | La somme est | Texte qui apparaît dans le label. |
| TextBox | Name | txtEntree | Identification du TextBox. |
| | Font | MS Sans Serif Gras 12 pt | Spécification de la police de caractères. |
| | MaxLength | 5 | Limite le nombre maximum de caractères (la valeur 0, mise par défaut, indique qu'il n'y a pas de limite). |
| | TabIndex | 0 | Numéro d'ordre de Tab. |
| | Text | (vide) | Texte affiché. |
| TextBox | Name | txtSomme | Identification du TextBox. |
| | Font | MS Sans Serif Gras 12 pt | Spécification de la police de caractères. |
| | Text | 0 | Texte affiché. |
| | Enabled | False | Autorisation / non autorisation. |

Le contrôle *TextBox* permet à l'utilisateur, *via* la propriété *Text*, d'entrer du texte (cf. *TextBox txtEntree*), il permet aussi d'afficher du texte (cf. *TextBox txtSomme*).
La propriété *Enabled* du contrôle *TextBox txtSomme* positionnée à *False* fait que ce contrôle ne réagit à aucun événement, il s'ensuit que le texte représentant la somme est de couleur grise.

Le code du programme est le suivant :

---

```
Dim sum As Integer

Private Sub cmdAjouter_Click()
    sum = sum + txtEntree.Text
    txtEntree.Text = ""
    txtSomme.Text = sum
End Sub

Private Sub cmdSortir_Click()
    End
End Sub
```

---

La ligne

```
Dim sum As Integer
```

déclare une variable, nommée *sum*, de type entier (2 octets, entre –32768 et 32767), initialisée par défaut à 0.

**Remarque :** La déclaration des variables dans Visual Basic est facultative, une variable non déclarée sera par défaut de type *Variant* (cf. § 4.4). Toutefois, il est conseillé, pour des raisons d'aide à la mise au point de programmes, de rendre obligatoire la déclaration des variables. Pour cela, il faut la présence dans la partie " déclaration générale " de la ligne suivante :

**Option Explicit**

Cette ligne peut être soit tapée directement dans la partie " déclaration générale ", soit introduite automatiquement par Visual Basic en ouvrant, avant toute programmation, le menu *Outils* | *Options*, puis en cliquant l'onglet nommé *Editeur* et en cochant la case *Déclaration des variables obligatoire* (case non cochée par défaut).

La ligne

**sum = sum + txtEntree.Text**

ajoute le texte de *txtEntree* à la variable *sum*, puis place le résultat dans la variable *sum*. Avant l'opération d'addition (+), la chaîne de caractères située dans la propriété *Text* est convertie implicitement en une valeur entière.
La ligne

**txtSomme.Text = sum**

met le contenu de la variable *sum* dans *txtSomme.Text*, Visual Basic convertit implicitement la valeur entière contenue dans *sum* en une chaîne de caractères.

**Remarque :** De manière plus rigoureuse que les conversions implicites (qui peuvent donner lieu à des méprises), il existe des fonctions Visual Basic permettant de convertir une chaîne de caractères en une valeur entière, et réciproquement (cf. fonctions *Val* et *Str$*).

## 3.4 Règles de priorité des opérateurs arithmétiques

Les opérations arithmétiques sont effectuées selon un ordre fixé par les règles de précédence d'opérations suivantes :

| Opération | Ordre d'évaluation |
|---|---|
| () parenthèses | Evalué en 1er. S'il y a plusieurs paires de parenthèses " de même niveau ", elles sont évaluées de gauche à droite. |
| ^ exponentielle | Evalué en 2e. S'il y en a plusieurs, elles sont évaluées de gauche à droite. |
| - négation | Evalué en 3e. S'il y en a plusieurs, elles sont évaluées de gauche à droite. |
| * ou / multiplication et division | Evalué en 4e. S'il y en a plusieurs, elles sont évaluées de gauche à droite. |
| \ division entière | Evalué en 5e. S'il y en a plusieurs, elles sont évaluées de gauche à droite. |
| Mod modulo<br>(17 Mod 3 = **2** car 17 = 3*5 + **2**) | Evalué en 6e. S'il y en a plusieurs, elles sont évaluées de gauche à droite. |
| + ou - addition et soustraction | Evalué en dernier. S'il y en a plusieurs, elles |

| | |
|---|---|
| | sont évaluées de gauche à droite. |

### 3.5 Opérateurs de comparaison

| Dans l'algèbre standard | Equivalent dans Visual Basic |
|---|---|
| a = b | a = b |
| c ≠ d | c < > d |
| e > f | e > f |
| g < h | g < h |
| i ≥ j | i >= j |
| k ≤ l | k <= l |

Ces opérateurs permettent au programme de prendre une décision basée sur la vérification, ou non, (*True* ou *False*) d'une condition.

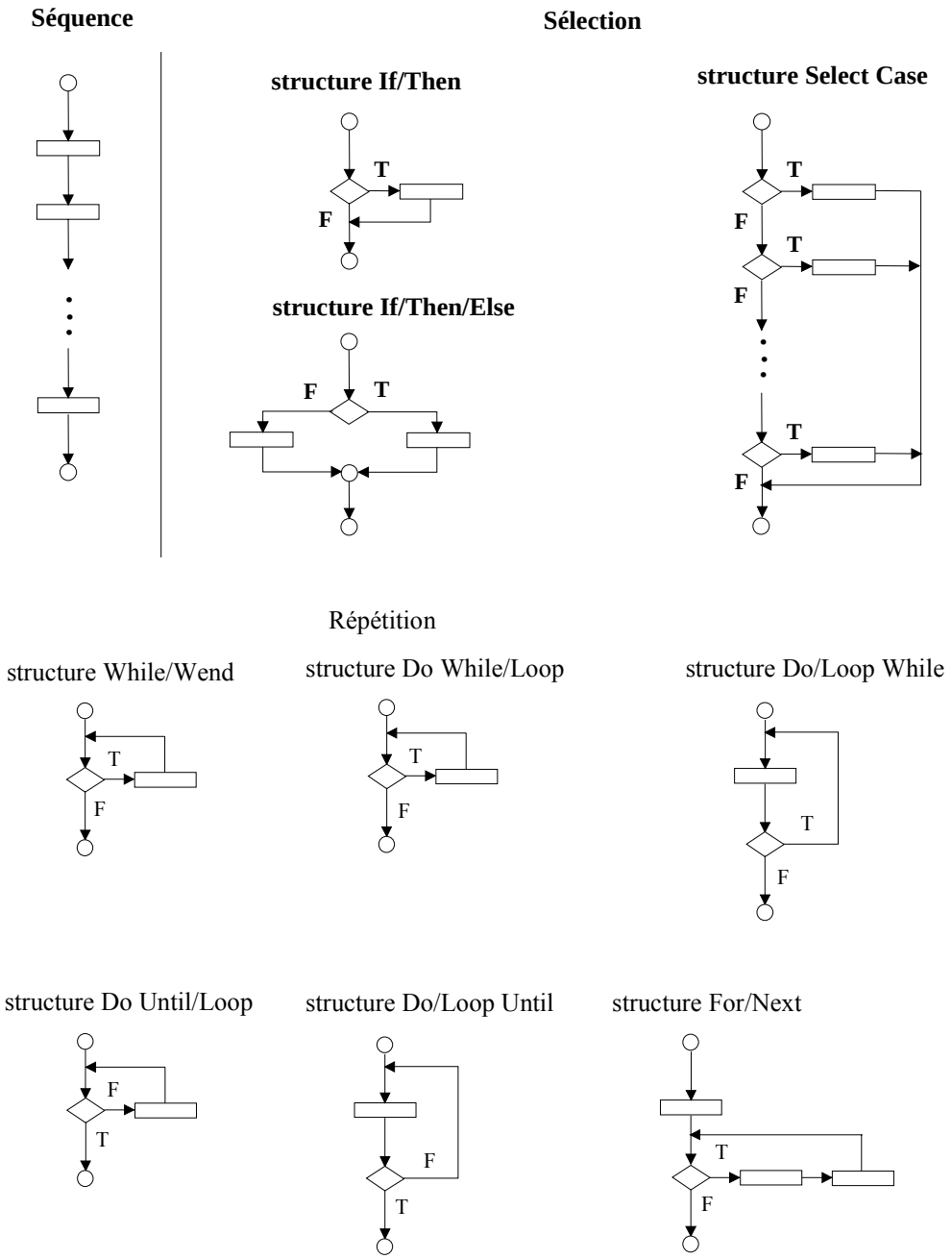
## 4 STRUCTURES DE CONTRÔLE

Afin de concevoir un programme, nous allons décrire les différents types de blocs de construction possibles et rappeler les principes de la programmation structurée. N'importe quel problème informatique peut-être résolu en exécutant une série d'actions dans un ordre spécifique, une telle procédure est appelée un *algorithme*.

Lorsque les instructions d'un programme sont exécutées l'une à la suite de l'autre, dans l'ordre où elles ont été écrites, l'exécution est dite *séquentielle*. Certaines instructions permettent au programmeur de spécifier que la prochaine instruction à exécuter peut être différente de celle qui est située dans la séquence, ces instructions effectuent un *transfert de contrôle*.
Durant les années 1960, l'utilisation de l'instruction de transfert de contrôle *goto* fut bannie comme étant la source de nombreux problèmes de programmation (difficultés vis-à-vis de la compréhension, du test, de la modification d'un programme). La programmation dite *structurée* (" *sans goto* ") fut présentée comme une alternative. En fait tous les programmes peuvent être construits par combinaison de seulement trois structures de contrôle : La structure *en séquence*, la structure *de sélection* et la structure *de répétition*.

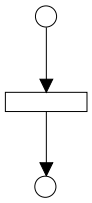
Visual Basic propose :
- trois types de structures de sélection : *If/Then, If/Then/Else, Select Case*,
- six types de structures de répétition : *While/Wend, Do While/Loop, Do/Loop While, Do Until/Loop, Do/Loop Until, For/Next.*

La figure suivante récapitule les différentes structures de contrôle de Visual Basic.




La connexion arbitraire de telles structures peut induire des programmes non structurés. Les règles suivantes permettent la construction de programmes structurés (le symbole rectangle peut être utilisé pour indiquer une action, y compris l'entrée/sortie) :

1. Commencer avec le diagramme (*flowchart*) suivant :



2. Tout rectangle (action) peut être remplacé par deux rectangles en séquence.

3. Tout rectangle (action) peut être remplacé par une structure de contrôle (séquence, *If/Then, If/Then/Else, Select Case, While/Wend, Do While/Loop, Do/Loop While, Do Until/Loop, Do/Loop Until, For/Next*).
4. Les règles 2 et 3 peuvent être appliquées autant de fois que nécessaire et dans n'importe quel ordre.

### 4.1 Structures de sélection

- Structure de type *If/Then*

```
If note >= 10 Then
        lblNote.Caption = "Accepté"
End If
```

- Structure de type *If/Then/Else*

```
If note >= 10 Then
        lblNote.Caption = "Accepté"
Else
        lbl.Caption = "Refusé"
End If
```

- Structure de type *Select Case*

Cette structure permet de ne pas utiliser des *If* imbriqués. Les conditions sont examinées dans l'ordre d'écriture, dès qu'une condition est vérifiée, les suivantes ne sont pas examinées.

```
Select Case code_d_acces
        Case Is < 1000
                Message = "Accès refusé"
        Case 1542, 1645 To 1689
                Message = "Personnel technique"
        Case Else
                Message = "Accès refusé"
End Select
```

### 4.2 Structures de répétition

- Structure de type *While/Wend*

```
Private Sub cmdBouton_Click()
    Dim produit As Integer
    produit = 2
    While produit <= 1000
            Print produit
            produit = produit * 2
    Wend
 End Sub
```

- Structure de type *Do While/Loop*

Cette structure se comporte comme la précédente.

```
Private Sub cmdBouton_Click()
    Dim produit As Integer
    produit = 2
    Do While produit <= 1000
            Print produit
            produit = produit * 2
    Loop
 End Sub
```

- Structure de type *Do/Loop While*

Cette structure est similaire à la précédente si ce n'est que la condition de test de la boucle est faite après que le corps de la boucle ait été exécuté.

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    compteur = 1
    Do
            Print compteur & Space$(2) ;        ' permet d'écrire 1  2  3  4  …  10
            compteur = compteur + 1
    Loop While compteur <= 10
End Sub
```

- Structure de type *Do Until/Loop*

```
Private Sub cmdBouton_Click()
    Dim produit As Integer
    produit = 2
    Do Until produit > 1000
            Print produit
            produit = produit * 2
    Loop
End Sub
```

Au contraire des trois précédentes structures, les instructions, situées dans le corps de la boucle, sont exécutées autant de fois que la condition de test de la boucle est fausse.

- Structure de type *Do/Loop Until*

Cette structure est similaire à la précédente si ce n'est que la condition de test de la boucle est faite après que le corps de la boucle ait été exécuté.

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    compteur = 1
    Do
            Print compteur & Space$(2) ;
            compteur = compteur + 1
    Loop Until compteur = 10
```

```
End Sub
```

- Structure de type *For/Next*

A titre d'exemple, réécrivons la portion de programme suivante qui utilise la structure *Do While/Loop* :

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    compteur = 3
    Do While compteur <= 20
            Print compteur
            compteur = compteur + 2
    Loop
End Sub
```

En fait l'initialisation, la condition de répétition et l'incrémentation sont incluses dans la structure d'entête *For* :

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    For compteur = 3 To 20 Step 2
            Print compteur
    Next compteur        ' la présence du nom de la variable compteur est facultative
End Sub
```

**Remarque :** L'incrément, ici égal à 2, peut-être négatif. Par défaut, sa valeur est unitaire.

**Remarque :** Les instructions *Exit Do* et *Exit For* permettent la sortie immédiate de ces structures (*Exit For* est réservée à la structure *For/Next*).

### 4.3 Opérateurs logiques

Liste des opérateurs permettant d'effectuer des opérations logiques à partir d'expressions à valeurs booléennes *True* (vraie) ou *False* (fausse) :

| | |
|---|---|
| **Not** *Expression* | Etablit la négation logique de *Expression* |
| *E1* **And** *E2* | *True* si les conditions *E1* et *E2* sont *True* |
| *E1* **Or** *E2* | *True* si l'une au moins des conditions *E1, E2* est *True* |
| *E1* **Xor** *E2* | *True* si seulement une seule des deux conditions *E1, E2* est *True* |

### 4.4 Types de données

- Les types de données décrivent l'information qu'une variable stocke. Les types prédéfinis dans Visual Basic sont listés dans le tableau ci-dessous.

| Type | Occupation mémoire | Portée des valeurs |
|---|---|---|
| Boolean | 2 octets | *True* ou *False* |
| Byte | 1 octet | 0 à 255 ($2^8$) |
| Currency | 8 octets | -922 337 203 685 477.580 8 à 922 337 203 685 477.580 7 |
| Date | 8 octets | 1er janvier 100 au 31 décembre 9999<br>0:00:00 à 23:59:59 |

| | | |
|---|---|---|
| Double | 8 octets | -1.797 693 134 862 32 E308 à – 4.940 656 458 412 47 E-324<br>4.940 656 458 412 47 E-324 à 1.797 693 134 862 32 E308 |
| Integer | 2 octets | -32 768 à 32 767 ($2^{16}$, signé) |
| Long | 4 octets | -2 147 483 648 à 2 147 483 647 ($2^{32}$, signé) |
| Object | 4 octets | Pointeur sur un objet (voir chp. 12, annexe F) |
| Single | 4 octets | -3.402 823 E38 à -1.401 298 E-45<br>1.401 298 E-45 à 3.402 823 E38 |
| String | 10 octets + taille de la chaîne | La longueur (variable) de la chaîne est composée d'au plus 2 147 483 648 caractères |
| String*_n_ | Taille de la chaîne | La taille de la chaîne, fixée par *n*, est comprise entre 1 et 65536 |
| Variant | 16 octets | Une valeur parmi celles listées au-dessus, mise à part *String*n* |

*Currency* est un type de données, stocké sur 8 octets, qui permet, notamment, des calculs monétaires précis dans le cas de nombres de moins de 4 décimales.
*Variant* est le type de données par défaut pour les variable dont le type n'est pas explicitement déclaré et stocke n'importe quel type de données (excepté *String*n*), y compris les types définis par le programmeur (voir ci-dessous). Ce type de données peut-être intéressant :
- dans le cadre de la conception d'une interface utilisateur, au sens où il permet de prendre en compte des données dont on ne connaît pas le type. Par exemple, à la question " entrer votre âge ", un utilisateur peut introduire un nombre, mais aussi du texte (" vingt-deux " au lieu de 22) ; le fait de stocker cette information dans une variable *Variant* permet, *via* les fonctions *VarType()* ou *TypeName()*, de connaître son type.
- pour construire des structures de données (listes chaînées, arbres), sachant que le code manipulant de telles structures peut être écrit indépendamment du type de données des éléments stockés.

- Visual Basic permet la création de ses propres types de données. Il s'agit de regrouper sous un seul type des collections de variables non nécessairement de même type (au contraire des tableaux).

Il suffit de répertorier, entre les instructions *Type* et *End Type*, l'ensemble des variables pour définir un nouveau type, voir l'exemple ci-dessous.

```
Type Client
        Nom As String*15
        Prénom As String*15
        Age As Integer
End Type
```

Le type *Client* étant défini, il est possible de déclarer une nouvelle variable basée sur ce type, par exemple :

```
Dim Untel As Client
Untel.Nom = "Dupond"
Untel.Prénom = "Léon"
Untel.Age = 40
```

Usuellement, ces types de données sont utilisés dans le cas de manipulation de fichiers à accès direct (fichiers constitués d'enregistrements de longueur fixe).

**Remarque :** Les définitions *Type* doivent être privées (*Private*) quand elles sont déclarées dans des modules feuilles, elles peuvent être publiques (*Public*) ou privées quand elles sont déclarées dans des modules code.

# 5 PROCÉDURES ET FONCTIONS

## 5.1 Introduction

L'expérience montre que le fait de fractionner un programme (important) en plusieurs parties facilite bien souvent son développement et sa maintenance. Cette approche, applicable, notamment, dans le cadre de Visual Basic, est appelée " *diviser et conquérir* ".

## 5.2 Les modules

Un projet est constitué de modules, tels que des modules *feuilles*, ou des modules *standards* (encore appelés modules *codes*). A la différence d'un module *feuille*, un module *standard* ne comporte pas d'IUG, il est uniquement composé de codes de programme.
En dehors de déclarations, les modules sont constitués de procédures, quatre types de procédures existent :

- les procédures *événementielles*, en anglais *event procedures*, en réponse à des événements (appui sur une touche du clavier, clic de souris, etc.) ;
- les procédures *Visual Basic*, en anglais *Visual Basic procedures*, (*Print, VarType*, etc.), fournies par Microsoft pour effectuer des tâches courantes (le but cherché étant de réduire le temps de développement) ;
- les procédures conçues par le programmeur (car non fournies par Visual Basic). Elles sont de deux types : Les procédures *Sub* et les procédures *Function*.

Un programmeur peut écrire des procédures pour exécuter des tâches spécifiques pouvant être utilisées à différents endroits dans un programme. A des fins de réutilisation éventuelle (en anglais, on parle de *software reusability*), chaque procédure devra être limitée à la résolution d'une tâche bien définie, il est de coutume que la taille d'une procédure ne dépasse pas une demie page.

L'appel à une procédure se fait en spécifiant le nom de la procédure et en fournissant les informations (on parle de paramètres d'entrée) nécessaires à la procédure " appelée " pour réaliser le travail demandé (restitué au programme " appelant " *via* des paramètres de sortie).

## 5.3 Les procédures *Sub*

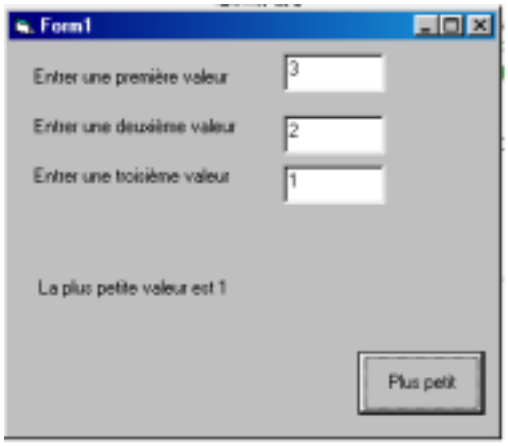
Voici un exemple de procédure *Sub* qui détermine la plus petite valeur parmi trois nombres entiers et l'affiche dans un *Label* (voir la recopie d'écran dans la figure qui suit) :

---

```
' Ce programme permet de trouver le minimum parmi trois nombres entiers
Option Explicit  ' Force les variables à être explicitement déclarées

Private Sub cmdPlusPetit_Click()
        Dim valeur1 As Long, valeur2 As Long, valeur3 As Long
        valeur1 = txtUn.Text
        valeur2 = txtDeux.Text
        valeur3 = txtTrois.Text
        Call Minimum(valeur1, valeur2, valeur3)   ' ou Minimum valeur1, valeur2, valeur3
End Sub

Private Sub Minimum (min As Long, y As Long, z As Long)
```

```
        If y < min Then
                min = y
        End If
        If z < min Then
                min = z
        End If
        lblPlusPetit.Caption = "La plus petite valeur est" & min
End Sub
```

---




Le corps de la procédure *Minimum* est placé entre l'entête et *End Sub*. A chaque appel de cette procédure (cf. instruction *Call Minimum(valeur1, valeur2, valeur3),* ou de manière équivalente, *Minimum valeur1, valeur2, valeur3*), le corps de la procédure est immédiatement exécuté, puis l'exécution du programme se poursuit par l'instruction située immédiatement après l'appel de la procédure. Toutes les entêtes des procédures contiennent des parenthèses, éventuellement vides ou contenant une liste de déclarations de variables, appelée liste des paramètres qui comprend des paramètres d'entrée et de sortie. En fait, une procédure peut modifier le contenu des variables passées en paramètre (c'est le cas du paramètre *min* de la procédure *Minimum*).

**Remarques :**
- Le fait que les procédures opèrent sur des données, explique que leurs noms correspondent bien souvent à des verbes. Il serait d'ailleurs préférable de renommer la procédure *Minimum* par *RechercherLeMinimum*.
- Par convention, une procédure débute par une lettre majuscule (par exemple : *Envoyer*).

### 5.4 Les procédures *Function*

Les procédures *Function* partagent les mêmes caractéristiques que les procédures *Sub* si ce n'est que les procédures *Function* retournent une valeur, appelée *valeur de retour*. La plupart des procédures fournies par Visual Basic sont de ce type, par exemple, la fonction *VarType (Name)* (retourne un entier indiquant le type de la variable *Name* de type *Variant*).

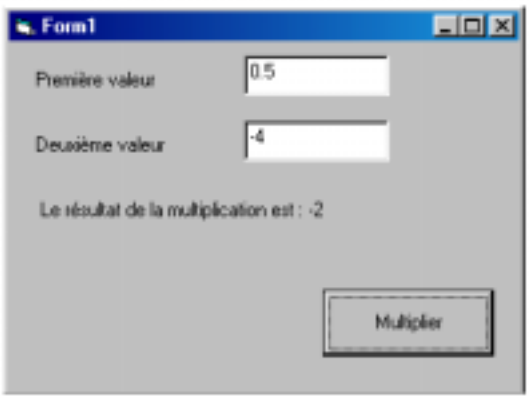
Voici un exemple de procédure *Function* qui multiplie deux valeurs réelles et affiche le résultat dans un *Label* (voir la recopie d'écran dans la figure qui suit) :

---

```
' Ce programme permet de multiplier deux nombres
```

```
Option Explicit  ‘ Force les variables à être explicitement déclarées

Private Sub cmdMultiplier_Click()
        Dim valeur1 As Single, valeur2 As Single
        valeur1 = Val(txt1.Text)     ‘ La fonction Val convertit une chaîne en un nombre
        valeur2 = Val(txt2.Text)
        lblMultiplication.Caption = "Le résultat de la multiplication est : " & _
        Multiplication(valeur1, valeur2)
End Sub

Private Function Multiplication (x As Single, y As Single) As Single
        Multiplication = x*y
End Function
```

---




La valeur, retournée par la fonction *Multiplication*, est placée dans le nom de la fonction (on parle de *pseudo-variable*).

## 5.5 Appel par valeur, appel par référence

Le passage des paramètres d’une procédure peut se faire *par valeur* (en anglais *by value*) ou *par référence* (en anglais *by reference*). Dans les deux exemples précédents, chacun des arguments a été passé *par référence*, ce qui est le mode de passage par défaut. Avec un tel mode de passage, le programme " appelant " donne à la procédure " appelée " la possibilité d’*accéder directement* aux données de " l’appelant " et donc de modifier ces données. Quand un argument est passé *par valeur*, une copie de la valeur de l’argument est faite et passée à la procédure " appelée ". Cette dernière peut manipuler cette copie, mais ne peut pas manipuler la donnée d’origine de " l’appelant ". Notons que ce mode de passage amoindrit les performances vis-à-vis du temps d’exécution et de l’espace mémoire, dans le cas d’un grand nombre de paramètres à passer.

L’entête de la fonction suivante déclare deux variables :

```
Function Calcul (ByVal x As Long, y As Boolean) As Double
```

La fonction *Calcul* reçoit *x par valeur* et *y par référence*, cette entête peut aussi s’écrire :

**Function Calcul (ByVal x As Long, ByRef y As Boolean) As Double**

## 5.6 Exit Sub et Exit Function

Le fait d'exécuter l'instruction *Exit Sub* (respectivement *Exit Function*) dans une procédure *Sub* (respectivement *Function*) provoque une sortie immédiate de la procédure. Le contrôle est retourné à " l'appelant " et l'instruction située immédiatement en séquence après l'appel est exécutée.

## 5.7 Durée de vie d'une variable

Une variable possède différents attributs : Un nom, un type, une taille et une valeur. A cela s'ajoutent d'autres attributs : Une durée de vie (en anglais *storage class*), une portée (en anglais *scope*).
L'attribut *durée de vie* d'une variable détermine la période durant laquelle elle existe en mémoire, certaines existent brièvement, d'autres sont régulièrement créées et détruites, d'autres existent tout au long de la durée d'exécution du programme.
Les variables locales (i.e., déclarées dans une procédure ou une fonction) *non statiques* (aussi appelées *variables automatiques*) sont définies par défaut et sont créées quand la procédure devient active. Ces variables existent (seulement) jusqu'à la fin d'exécution de la procédure.
Le mot clé *Static* est utilisé pour déclarer des variables locales *statiques*. De telles variables sont toujours connues seulement dans la procédure dans laquelle elles ont été déclarées (on dit qu'elles ont la même portées), mais à la différence des *variables automatiques*, les variables *statiques* conservent leurs valeurs (i.e., celles acquises lors de la dernière sortie de la procédure). Ces variables sont implicitement initialisées à zéro lors du premier appel de la procédure. Ce type de variables permet de ne pas déclarer des variables globales lorsqu'une seule procédure en a besoin.

Exemple :

```
Private Sub Exemple_Click ()
        Static N As Integer   ' N est une variable entière statique
        Dim M As Integer      ' M est une variable entière non statique

        N = N + 1
        M = M + 1
End Sub
```

Lors du premier appel de cette procédure, les variables *N* et *M* sont égales à 0, aussi lors de la première sortie de cette procédure, les variables *N* et *M* sont égales à 1.
Lors du second appel, la variable *N* est égale à 1 (valeur acquise lors de la précédente sortie) alors que la variable *M* est toujours égale à 0. Aussi lors de la seconde sortie de cette procédure, la variable *N* est égale à 2 alors que la variable *M* est toujours égale à 1.

## 5.8 Portée d'une variable, d'une procédure, d'une fonction

L'étendue d'un *identificateur* (nom d'une variable, ou nom d'une procédure définie par le programmeur) est la région dans laquelle l'*identificateur* peut-être référencé. Par exemple, une variable déclarée en local dans une procédure peut seulement être référencée dans cette procédure. Les trois portées d'un *identificateur* sont la portée *locale* (en anglais *local scope*), la portée au niveau *module* (en anglais *module scope*) et la portée au niveau *public* (en anglais *public scope*).
La portée *locale* s'applique à des variables déclarées dans le corps d'une procédure. Les variables locales peuvent être référencées à partir de l'endroit où elles ont été déclarées jusqu'à la sortie de la procédure (ou de la fonction), i.e., *End Sub* (ou *End Function*).

La portée au niveau *module*, aussi appelée portée au niveau *privé* (en anglais *Private scope*), s'applique à des variables déclarées dans la partie " déclaration générale du module " *via* le mot clé *Dim*. Ces variables peuvent seulement être référencées dans le module dans lequel elles ont été déclarées.
La portée au niveau *public* fait référence à des variables déclarées en *public* dans un module. De telles variables sont accessibles pour tous les modules.

**Remarque :** Quand une variable est déclarée au niveau *module* et a le même nom qu'une variable déclarée en local, celle qui est déclarée au niveau *module* est " cachée " tant que la variable déclarée en local est active. Une alternative à cette source d'erreur consiste à ne pas utiliser des noms de variables dupliquées.

## 5.9 Les constantes

Visual Basic permet la création de variables dont la valeur ne peut pas changer durant l'exécution d'un programme. Ces variables, particulières, sont appelées des *variables constantes*[1] et sont souvent utilisées pour améliorer la " lisibilité " d'un programme. La déclaration d'une variable constante se fait à l'aide du mot clé *Const* :

**Syntaxe**

[**Public** | **Private**] **Const** *constname* [**As** *type*] **=** *expression*

*Private* est utilisé au niveau *module* pour déclarer les constantes uniquement disponibles dans le module dans lequel la déclaration est effectuée. Ce mot clé n'est pas autorisé dans les procédures.
*Public* est utilisé au niveau *module* pour déclarer des constantes disponibles pour toutes les procédures de l'ensemble des modules. Ce mot clé n'est pas autorisé dans les procédures.

**Remarque :** Utiliser le mot clé *Dim* avec *Const* dans une déclaration est une erreur de syntaxe.

A titre d'exemple, on peut donner les lignes suivantes :

```
Const pi As Double = 3.14159        ' il est obligatoire d'assigner une valeur
Const Deux_pi As Double = pi*2
Public Const RG = "Rouge"
```

En fait, de nombreuses constantes sont reconnues dans Visual Basic, elles sont généralement préfixées par les lettres *vb*. Par exemple, dans le cadre d'une boîte de dialogue, la constante *vbOK* (renvoyée par la fonction *MsgBox*) permet de détecter un appui (un clic) sur le bouton *OK*.

## 5.10 Paramètres optionnels

Il est possible de créer des procédures qui acceptent un ou plusieurs *paramètres optionnels*. Ces paramètres sont spécifiés dans l'entête de la procédure *via* le mot clé *Optional*. Considérons, par exemple, la procédure suivante :

```
Private Sub Ajout (x As Integer, Optional y As Integer = 1)
    x = x + y
End Sub
```

L'entête de cette procédure indique que le second argument peut ne pas être demandé lors d'un appel de la procédure *Ajout*, auquel cas il sera égal à 1. Considérons les appels suivants :

[1] oxymoron : Figure de style qui réunit deux mots en apparence condratictoires (silence éloquent).

**Call Ajout**
**Call Ajout (a)**
**Call Ajout (a, b)**

Le premier appel génère une erreur de syntaxe car un argument au minimum est réclamé. Le second appel est valide, l'argument optionnel n'est pas fourni. Au retour de la procédure, la variable entière *a* est incrémentée par défaut de 1. Le troisième appel est aussi valide, la variable entière *b* est fournie en tant qu'argument optionnel. Au retour de la procédure, la variable entière *a* est incrémentée par défaut de *b*.

## 5.11 Fonctions mathématiques de Visual Basic

Certaines fonctions de calcul mathématique sont disponibles :

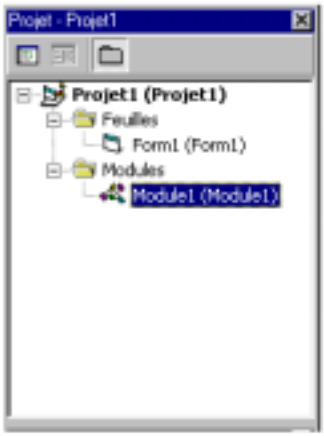
| Fonction | Description | Exemple |
|---|---|---|
| *Abs (x)* | Valeur absolue de *x* | Abs (-2) est égal à 2, Abs (3) est égal à 3 |
| *Atn (x)* | Arc tangente (en radians) de *x* | Atn (1) est égal à pi/4 |
| *Cos (x)* | Cosinus (en radians) de *x* | Cos (0) est égal à 1 |
| *Exp (x)* | Fonction exponentielle $e^x$ | Exp (1.0) est égal à 2.71828 |
| *Int (x)* | Retourne le plus grand entier inférieur ou égal à *x* | Int (-5.3) est égal à – 6<br>Int (0.893) est égal à 0<br>Int (76.45) est égal à 76 |
| *Fix (x)* | Partie entière de *x* | Fix (-5.3) est égal à –5<br>Fix (0.893) est égal à 0<br>Fix (76.45) est égal à 76 |
| *Log (x)* | Logarithme népérien de *x* | Log (2.718282) est égal à 1.0 |
| *Round (x, y)* | Retourne une valeur approchée de *x* avec *y* décimales (si *y* est omis, *x* est retourné comme étant un entier) | Round (4.84) est égal à 4<br>Round (5.73839, 3) est égal à 5.738 |
| *Sgn (x)* | Signe de *x* | Sgn (-19) est égal à –1<br>Sgn (0) est égal à 0<br>Sgn (3.4) est égal à 1 |
| *Sin (x)* | Sinus (en radians) de *x* | Sin (0) est égal à 0 |
| *Sqr (x)* | Racine carré de *x* | Sqr (9.0) est égal à 3 |
| *Tan (x)* | Tangente (en radians) de *x* | Tan (0) est égal à 0 |

## 5.12 Module standard

Un module standard (ou module code) n'a pas d'IUG (au contraire des modules feuilles, en anglais *form modules*), il contient uniquement du code. Les procédures que le programmeur souhaite utiliser dans de nombreux projets (on parle de *briques logicielles*) sont souvent placées dans des modules standards. Comme pour un module feuille, un module code admet une partie " déclaration générale ".
Par défaut, les variables d'un module et les procédures événementielles sont privées (*Private*) au module dans lesquelles elles ont été définies : Une variable, ou une procédure, privée peut seulement être utilisée dans le module où elle a été déclarée. Si le programmeur souhaite permettre à un autre module l'utilisation d'une variable, ou procédure, le mot clé *Public* doit être utilisé dans la déclaration : Les variables, ou procédures, publiques sont accessibles dans chacun des modules du projet.

**Remarques :**

- Les modules standards sont tous placés dans un classeur nommé *Modules*, alors que les feuilles sont placées dans un classeur nommé *Feuilles* (cf. figure suivante).




- Les noms des fichiers de modules standards se terminent par **.bas**.
- Visual Basic ne propose pas de module standard par défaut dans un projet, il faut l'ajouter manuellement (voir le menu *Projet | Ajouter un module*).
- Un projet peut avoir plusieurs modules codes (aussi bien que plusieurs modules feuilles).

A titre d'exemple, le programme suivant contient un module feuille et un module code. Le fait de cliquer sur le bouton *Ecrire* de la feuille (voir la saisie d'écran dans la figure qui suit) appelle la procédure *ModuleEcrire*, déclarée en *public* et située dans un module standard.

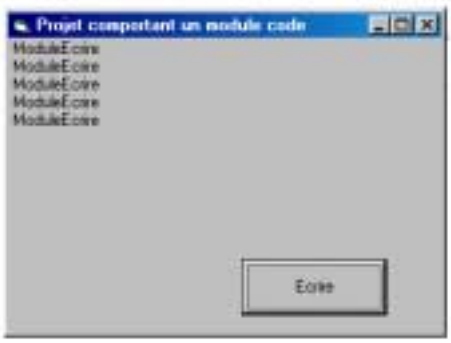
_____ *.frm* _____

```
' Utilisation d'un module code
Option explicit

Private Sub cmdEcrire_Click ()
        ' La procédure ModuleEcrire est définie dans le module code (modModule.bas)
        Call ModuleEcrire
End Sub
```

_____ *.bas* _____

```
' modModule.bas
Option Explicit

Public Sub ModuleEcrire ()
        frmFeuille.Print "ModuleEcrire"
End Sub
```

_____




## 6 LES TABLEAUX

Un tableau peut se représenter comme étant un groupe consécutif (une série) d'emplacements mémoires de même nom et de même type (voir § 4.4). La référence à un emplacement particulier, ou élément dans le tableau, se fait à travers le nom du tableau et sa position (son index) dans le tableau. Le traitement des variables contenues dans un tableau peut alors se faire à l'aide de boucle.

Par exemple, soit un tableau, mono dimensionnel, nommé *Nombre* contenant 6 éléments de type entiers :

| | |
|---|---|
| Nombre (0) | 55 |
| Nombre (1) | 22 |
| Nombre (2) | 15 |
| Nombre (3) | 1 |
| Nombre (4) | 89 |
| Nombre (5) | 56 |

Notons qu'il existe deux catégories de tableaux : Les tableaux *statiques* (en anglais, *static arrays*) dont le nombre maximum d'éléments est fixé, et les tableaux *dynamiques* (en anglais, *dynamic arrays*, aussi appelé *redimmable arrays*) dont le nombre d'éléments peut varier durant le déroulement du programme.

### 6.1 Leurs déclarations

Les tableaux peuvent être déclarés comme *Public* uniquement dans un module code. Les tableaux actifs au niveau d'un module sont déclarés dans la partie " déclarations générales " du module *via* le mot clé *Dim* ou *Private*. Les tableaux actifs au niveau *local* sont déclarés dans une procédure *via* le mot clé *Dim* ou *Static*.

La déclaration

**Dim Nombre (5) As Integer**

déclare un tableau *Nombre* constitué de six entiers. Les fonctions *LBound (Nombre)* et *UBound (Nombre)* retournent respectivement l'indice le plus petit et l'indice le plus grand du tableau, l'indice étant de type *Long*. Par défaut, le plus petit indice d'un tableau est égal à 0. Le fait de spécifier une valeur pour le plus grand indice d'un tableau, dans notre exemple 5, indique que le tableau est statique. Par défaut, les six entiers du tableau sont égaux à 0, la ligne

**Nombre (0) = 55**

permet de fixer la valeur 55 à l'élément du tableau *Nombre* d'indice 0.

La déclaration

**Dim Tab (-5 To 4) As String**

déclare un tableau *Tab* constitué de 10 chaînes de caractères, *LBound (Tab)* et *UBound (Tab)* sont respectivement égaux à –5 et 4.

Les tableaux peuvent être multi-dimensionnels, la déclaration

**Dim Tab3 (50 To 100, 8, 7 To 15)**

déclare un tableau *Tab* de dimension 3, *LBound (Tab, 3)* est égal à 7, *UBound (Tab, 1)* est égal à 100.

## 6.2 Les tableaux dynamiques

Les tableaux dont la taille peut augmenter, ou diminuer, en cours d'exécution sont appelés tableaux *dynamiques*. Aucune taille n'est donnée lors de la déclaration d'un tel tableau. Par exemple, la ligne

**Dim TabDynamique () As Double**

déclare le tableau *TabDynamique* comme étant un tableau dynamique constitué de réels doubles. La taille d'un tableau dynamique est spécifiée en cours d'exécution à l'aide du mot clé *ReDim* (notons que la dimension du tableau ne peut pas être modifiée). Par exemple, la ligne

**ReDim TabDynamique (10)**

alloue 11 éléments à *TabDynamique*. Le fait d'exécuter *ReDim* provoque la perte des valeurs contenues dans le tableau. Toutefois, les valeurs déjà dans le tableau peuvent être conservées en plaçant le mot clé *Preserve* après *ReDim*.

Le mot clé *Erase Tab* permet, en cours d'exécution, de supprimer de la mémoire le tableau dynamique *Tab*.

## 6.3 Passage de tableaux dans les procédures

Pour passer tous les éléments d'un tableau en paramètre à une procédure, il suffit de spécifier le nom du tableau suivi d'une paire de parenthèses vide. Par exemple, si le tableau *TemperatureMensuelle* est déclaré comme suit :

**Dim TemperatureMensuelle (12) As Integer**

l'appel

**Call ModifierTableau (TemperatureMensuelle ( ))**

passe tous les éléments du tableau *TemperatureMensuelle* à la procédure *ModifierTableau*. Les tableaux sont automatiquement passés par référence – l'appelé peut donc modifier la valeur des éléments du tableau fourni par l'appelant.
Pour passer un élément d'un tableau à une procédure, utiliser l'élément du tableau comme paramètre, par exemple :

**Call PasserUnElement (TemperatureMensuelle (4))**

Pour une procédure recevant un tableau, la liste des paramètres de la procédure doit spécifier le passage d'un tableau. Par exemple, l'entête de la procédure *ModifierTableau* devra s'écrire comme suit :

**Private Sub ModifierTableau (a( ) As Integer)**

ainsi, *ModifierTableau* s'attend à recevoir un tableau d'entier dans le paramètre *a* (la taille du tableau n'est pas spécifiée entre les parenthèses).

# 7 LES CHAÎNES

Chaque caractère est représenté en interne par un nombre entier, compris entre 0 et 255 (par exemple, 65 représente la lettre *A*). Cet ensemble de valeurs entières est appelé l'ensemble des caractères *American National Standards Institute (ANSI)*. Les 128 premiers caractères *ANSI* (de 0 à 127) correspondent aux valeurs de l'*American Standard Code for Information Interchange (ASCII)*. Les valeurs *ANSI* de 128 à 255 représentent un ensemble de caractères spéciaux, d'accents, de fractions, etc. Une chaîne est une série de caractères formant une entité. Une chaîne dans Visual Basic a un type de donnée *String*. Deux types de chaînes sont possibles : Les chaînes de longueur variable (en anglais, *variable-length strings*), les chaînes de longueur fixe (en anglais, *fixed-length strings*).
Par défaut, les variables *String* sont de longueur variable, exemple :

```
Dim ch As String
ch = "02 41 – abc"
```

la variable *ch* est déclarée de type *String*, elle contient la chaîne "02 41 – abc".
La chaîne (*NomVariable*) de longueur fixe (*TailleChaîne*) est déclarée comme suit :

**Dim** *NomVariable* **As String *** *TailleChaîne*

La fonction *Len* (pour *length*) est utilisée pour déterminer la longueur d'une chaîne, par exemple, *Len* ("*02 41 – abc*") est égal à 11.

## 7.1 Concaténation avec & (esperluette) et +

```
ch1 = "Pro"
ch2 = "gramme"
ch3 = ch1 & ch2
ch4 = ch1 + ch2
```

*ch3* et *ch4* contiennent la chaîne "*Programme*". Les opérateurs & et + sont équivalents dans le cas où les opérandes sont des chaînes. L'utilisation de l'opération + entre des opérandes de type différent, par exemple, la ligne

**ch = "bonjour" + 22**

provoque une erreur car Visual Basic tente de convertir la chaîne "*bonjour*" en un nombre, de manière à l'additionner (+) à 22. Pour cette raison, il est préférable d'utiliser l'opération & pour concaténer des chaînes.

## 7.2 Comparaison de chaînes

Il est fréquent de devoir comparer deux chaînes. Pour cela, on dispose de la fonction *StrComp* et des opérateurs <, <=, >, >=, =, <>. La comparaison entre deux chaînes est basée sur les différentes valeurs *ANSI* codant les chaînes.
La fonction *StrComp (ch1, ch2)* retourne 0 lorsque les chaînes *ch1* et *ch2* sont égales, -1 si la chaîne *ch1* est inférieure à *ch2*, 1 si la chaîne *ch1* est supérieure à *ch2*. A titre d'exemple, *StrComp ("A", "B")* est égal à –1 car le code *ANSI* de la lettre *A* (= 65) est inférieur à celui de la lettre *B* (= 66). Une option permet d'être *case-sensitive* ou *case-insensitive*.

- L'opérateur *Like* fournit un autre moyen de comparer deux chaînes.

## 7.3 Manipulation de caractères dans une chaîne

Visual Basic fournit différents moyens permettant la manipulation de caractères dans une chaîne.

La fonction *Mid$ (ch, pos, nb)* retourne une sous-chaîne de la chaîne *ch* constituée de *nb* caractères à partir du caractère indiqué par la position *pos* (la position du premier caractère est égale à 1).
Par exemple :

```
sous_ch = Mid$ ("programme", 2, 3)      ' sous_ch = "rog"
```

Cette fonction permet aussi de remplacer une portion de chaîne par une autre. Par exemple, les lignes

```
ch = "Bonjour monsieur"
Mid$ (ch, 9, 1) = "M"
```

change le contenu de *ch* en "*Bonjour Monsieur*".

## 7.4 *Left$, Right$, InStr, InStrRev, Split, Join*

La fonction *Left$ (ch, nb)* retourne une chaîne composée de *nb* caractères situés dans la partie gauche de la chaîne *ch*, la fonction *Right$ (ch, nb)* fait de même pour la partie droite de la chaîne. A titre d'exemple, considérons les lignes suivantes

```
ch = "Bonjour madame"
ch1 = Left$ (ch, 7)          ' ch1 = "Bonjour"
ch2 = Right$ (ch, 6)         ' ch2 = "madame"
```

La fonction *InStr (pos, source, rech)* retourne, si possible, un nombre entier représentant la position dans la chaîne de base *source* de la chaîne cherchée *rech*, cette recherche débutant à partir de la position *pos*. Si la chaîne *rech* n'est pas détectée, la fonction *InStr* renvoie une valeur nulle. A titre d'exemple, considérons les lignes suivantes

```
ch = "Bonjour madame"
i = InStr (1, ch, "on")          ' i = 2
j = InStr (2, ch, "on")          ' j = 2
k = InStr (3, ch, "on")          ' k = 0
```

La fonction *InStrRev* a un mode de fonctionnement similaire à celui de *InStr*, si ce n'est que la recherche débute à partir de la fin de la chaîne (i.e., de droite à gauche dans la chaîne de base).

La fonction *Split (expression, délimiteur)* retourne dans un tableau (en fait un vecteur) des sous-chaînes de caractères. L'extraction des sous-chaînes de la chaîne *expression* se fait au vue du *délimiteur,* par exemple :

```
Dim tableau() As String
tableau = Split ("ab" "c" "d1f", " ")
```

retourne les sous-chaînes "*ab*" dans *tableau(0),* "*c*" dans *tableau(1)* et "*d1f*" dans *tableau(2).*

La fonction *Join (tableau source, délimiteur)* est la fonction duale de *Split.* Elle retourne une chaîne joignant les sous-chaînes contenues dans le tableau (en fait un vecteur) *tableau source,* les sous-chaînes étant séparées par un *délimiteur*. Par exemple,

```
Dim ch As String
ch = Join (tableau, ".")
```

retourne la chaîne "*ab.c.d1f*" dans la variable *ch*.

### 7.5 *LTrim$, RTrim$* et *Trim$*

Les fonctions *LTrim$, RTrim$* et *Trim$* (en anglais, le verbe *trim* signifie *couper, tailler*) ôtent les (éventuels) espaces situés respectivement à gauche, à droite, à la fois à gauche et à droite, d'une chaîne. A titre d'exemple, considérons les lignes suivantes

```
ch = "    Bonjour     "
ch1 = LTrim$ (ch)          ' ch1 = "Bonjour     "
ch2 = RTrim$ (ch)          ' ch2 = "    Bonjour"
ch3 = Trim$ (ch)           ' ch3 = "Bonjour"
```

### 7.6 *String$* et *Space$*

La fonction *String$ (nb, ascii)* retourne une chaîne de *nb* caractères dont le code *ASCII* est spécifié par *ascii*. A titre d'exemple, considérons les lignes suivantes

```
ch1 = String$ (4, "A")     ' ch1 = "AAAA"
ch1 = String$ (4, 66)      ' ch1 = "BBBB"
```

La fonction *Space$(nb)* retourne une chaîne composée de *nb* caractères d'espace.

### 7.7 Autres fonctions de traitement de chaînes

La fonction *Replace (source, ch_à_remplacer, ch_de_remplacement)* retourne une chaîne dans laquelle la sous-chaîne *ch_à_remplacer* a été remplacée plusieurs fois par la sous-chaîne *ch_de_remplacement*.

La fonction *StrReverse (ch)* retourne une chaîne contenant des caractères dont l'ordre a été inversé par rapport à la chaîne *ch*.

Les fonctions *UCase$ (ch)* et *LCase$ (ch)* convertissent tous les caractères de la chaîne *ch* respectivement en majuscules (en anglais, *upper-case letters*) et minuscules (en anglais, *lower-case letters*). Les caractères qui ne sont pas des lettres demeurent inchangés.

### 7.8 Fonctions de conversion

- ***Asc* et *Chr$***

La fonction *Asc (ch)* retourne un nombre entier correspondant au code *ASCII* du premier caractère de la chaîne *ch*. Réciproquement, la fonction *Chr$ (ascii)* retourne le caractère correspondant au code *ASCII* donné (nombre entier *ascii*).

- ***IsNumeric, Val, Str$, Hex$* et *Oct$***

La fonction *IsNumeric (ch)* retourne *True* si la chaîne *ch* peut représenter un nombre numérique.

La fonction *Val (ch)* convertit la chaîne *ch* en une valeur numérique (la lecture de la chaîne s'arrête au premier caractère ne faisant apparemment pas partie d'un nombre). A titre d'exemple, considérons les lignes suivantes

```
Dim ch1 As String, ch2 As String
Dim x As Integer, y As Double, z As Double
ch1 = "  2.35 a"
x = Val (ch1)          ' x = 2
y = Val (ch1)          ' y = 2.35
ch2 = "a12"
z = Val (ch2)          ' z = 0
```

La fonction *Str$ (valeur)* convertit une valeur numérique en chaîne.

Il est possible de convertir des valeurs numériques en chaîne sous une forme hexadécimale (base 16), ou octale (base 8), *via* les fonctions *Hex$* et *Oct$* respectivement.

Visual Basic fournit plusieurs autres fonctions permettant de convertir une chaîne en un autre type de donnée.

# 8 INTERFACE UTILISATEUR GRAPHIQUE : LES BASES

Les Interfaces Utilisateurs Graphiques (IUG) sont construites à partir de *contrôles*. Un *contrôle* est un objet (un objet s'utilise à travers ses propriétés, ses méthodes et ses événement associés) avec lequel un utilisateur interagit *via* la souris, ou le clavier. Les contrôles Visual Basic sont de deux types :

- les contrôles *intrinsèques*, aussi appelés contrôles *standards*, disponibles par défaut dans la barre de contrôles de l'EDI. Une description rapide de ces contrôles est donnée au § 2.1.
- les contrôles *ActiveX*. Ces contrôles sont mis à disposition dans la barre de contrôles, en ouvrant le menu *Projet* | *Composants*, puis en cliquant l'onglet nommé *Contrôles* et en cochant la case correspondant au contrôle *ActiveX* à insérer. A titre d'exemple, des contrôles *ActiveX* sont disponibles dans le cadre des bases de données, des réseaux, sachant que certains des contrôles sont spécifiques aux différentes versions (*Learning, Professional, Enterprise*) de Visual Basic.

Une description complète des contrôles disponibles dans Visual Basic est accessible dans l'aide en ligne (taper sur la touche F1). Décrivons ici quelques-uns des contrôles les plus utilisés, à travers une description de leurs propriétés, méthodes et événements associés.

## 8.1 Le contrôle *Label*

Les contrôles *label* (en français, étiquette) servent à afficher du texte à l'écran, en mode création ou exécution. Ce texte est non modifiable directement par l'utilisateur.

**Ses propriétés**
En dehors de la propriété *Name* (par défaut, *Label1, Label2, …*) qui permet de différencier les objets entre eux, la principale propriété du contrôle *Label* est *Caption* (notons que le texte par défaut correspond à celui mis par défaut dans la propriété *Name*) qui permet l'affichage (si nécessaire) d'un texte.
Comme pour un module feuille, les propriétés *BackColor* et *ForeColor* déterminent respectivement la couleur du fond et celle du texte affiché dans le contrôle *Label*.
La propriété *Font* permet de spécifier la police utilisée pour afficher le texte (nom, style (standard, italique, …), taille, …).
La propriété *Alignment* permet au choix de cadrer le texte affiché à droite, à gauche ou au centre.
La propriété *Enabled* permet d'activer, ou non, le contrôle lors de l'exécution de l'application.
La propriété *Visible* permet, par exemple en réponse à un événement, de cacher (*False*), ou non (*True*), le contrôle.

**Ses méthodes**
Nous retiendrons seulement la méthode *Move* qui permet le déplacement du *Label*.

**Ses événements associés**
Parmi les événements liés à la souris, on peut retenir les événements : *MouseDown, MouseUp, MouseMove, Click, DblClick*.

**Remarques :**
- Les événements *MouseDown* et *MouseUp* se produisent, dans l'ordre, lorsque l'utilisateur enfonce (*MouseDown*), ou relâche (*MouseUp*), un bouton de la souris.
- L'événement *MouseMove*, généré continuellement lorsque le pointeur de la souris se déplace sur des objets. La génération de cet événement s'intercale entre *MouseDown* et *MouseUp*.
- L'événement *Click* se produit lorsque l'utilisateur clique un bouton de la souris puis le relâche sur un objet. Lorsque l'on effectue un clic, la séquence d'événements générés est, dans l'ordre, *MouseDown, MouseUp, Click*.
- L'événement *DblClick* se produit lorsque l'utilisateur appuie sur, et relâche, un bouton de la souris deux fois de suite sur un objet. Lorsque l'on effectue un double clic, la séquence d'événements générés est, dans l'ordre, *MouseDown, MouseUp, Click, DblClick, MouseUp*.

## 8.2 Le contrôle *TextBox*

Le contrôle *TextBox* (en français, *zone de saisie*) permet, *via* une zone à l'écran, à un utilisateur l'introduction, ou l'affichage, d'informations. Le tableau ci-dessous regroupe les principales propriétés, méthodes et événement associés à ces contrôles.

| Propriété, méthode, événement | Description |
|---|---|
| | |
| **Propriété** | |
| *Enabled* | Spécifie si l'utilisateur peut, ou non, agir sur le contrôle. |
| *MaxLength* | Spécifie le nombre maximal de caractères pouvant être entrés (la valeur 0, mise par défaut, indique qu'il n'y a pas de limite). |
| *MultiLine* | Permet quand elle est *True* d'entrer du texte sur plusieurs lignes. |
| *PasswordChar* | Spécifie le caractère affiché à la place des caractères entrés. |
| *Text* | Spécifie le texte entré. |
| | |
| **Méthode** | |
| *SetFocus* | Place le *focus* sur le contrôle. |
| | |

| **Evénement** | |
|---|---|
| *Change* | Evénement appelé à chaque fois que le contenu de la zone est modifié (par l'utilisateur ou par le programme). |
| *Click, DblClick, MouseDown, MouseMove, MouseUp* | Evénements liés à la souris. |
| *GotFocus, LostFocus* | Evénements générés lors de la réception, resp. de la perte, du *focus*. |
| *KeyDown, KeyPress, KeyUp* | Evénements liés au clavier (générés dans l'ordre : Touche pressée, appuyée, relachée). |

### 8.3 Le contrôle *CommandButton*

Les contrôles *CommandButton* sont représentés par des *boutons*, appelés aussi " boutons-poussoirs ", qu'un utilisateur peut cliquer pour exécuter une action.
La propriété *Caption* permet d'écrire un texte sur le bouton. La propriété *Enabled* indique si le bouton est actif (*True*), ou non (*False*). Suite à un clic sur un bouton (actif), l'événement *Click* est appelé.

### 8.4 Les contrôles *ListBox, ComboBox*

Les listes permettent de visualiser une liste de différents items (éléments). Pour réaliser des listes, Visual Basic propose deux types de contrôles : *ListBox* (liste) ou *ComboBox* (liste combinée, ce contrôle permet de combiner une liste avec une zone de saisie).

- **Le contrôle *ListBox***

Quand une liste (*ListBox*) contient plus d'items qu'elle ne peut en afficher, une barre de défilement (en anglais, *scrollbar*) verticale apparaît automatiquement.

| **Propriété, méthode, événement** | **Description** |
|---|---|
| | |
| **Propriété** | |
| *Columns* | (entier) Spécifie si le contrôle a une barre de défilement horizontal, si oui, spécifie le nombre de colonnes. Une valeur égale à 0 indique qu'il n'y a pas de défilement horizontal, une valeur supérieure à 0 spécifie le nombre de colonnes dans lesquelles les items sont listés horizontalement. |
| *Enabled* | (booléen) Spécifie si l'utilisateur peut, ou non, agir sur le contrôle. |
| *List* | (tableau de *String*) Contient les éléments de la liste, le plus petit indice du tableau est égal à 0, le plus grand indice est égal à *ListCount* - 1. |
| *ListCount* | (entier) Contient le nombre d'éléments de la liste. |
| *ListIndex* | (entier de -1 à *ListCount* - 1) (pour les listes à sélection simple) Contient l'indice de l'élément actuellement sélectionné. La valeur - 1 signifie qu'aucun élément n'est sélectionné. |
| *MultiSelect* | (entier de 0 à 2) Spécifie si l'utilisateur peut sélectionner plus d'un item à la fois. Une valeur égale à :<br>- 0 autorise la sélection d'un seul élément à la fois (sélection simple),<br>- 1 autorise plusieurs choix, chaque clic sélectionne/désélectionne un item (sélection multiple),<br>- 2 autorise un choix étendu d'items – possibilité d'utiliser les touches *shift* et *contrôle* (sélection multiple étendu). |
| *Selected* | (tableau de booléens) (pour les listes à sélection multiple) L'élément d'indice *i* est actuellement sélectionné si *Selected* (*i*) est *True*. |

| | |
|---|---|
| *SelCount* | (entier) (pour les listes à sélection multiple) Indique le nombre d'éléments sélectionnés. |
| *Sorted* | (booléen) Permet d'avoir une liste dont les éléments sont triés par ordre alphabétique. |
| *Text* | (*String*) Spécifie l'élément sélectionné, correspond à *List* (*ListIndex*). |
| | |
| **Méthode** | |
| *AddItem* | *AddItem item* [, *index*] permet d'ajouter (en fin de liste) l'élément *item* dans la liste. Le paramètre optionnel *index* permet d'insérer l'élément *item* dans la liste à l'indice *index*. |
| *Clear* | Enlève tous les éléments de la liste. |
| *RemoveItem* | *RemoveItem index* permet d'enlever l'élément d'indice *index* de la liste. |
| | |
| **Evénement** | |
| *Click* | Evénement activé à chaque fois qu'un élément de la liste est sélectionné. |

- **Le contrôle *ComboBox***

La liste combinée permet l'affichage d'une liste (à sélection simple) et d'une zone de saisie. Elle permet à l'utilisateur de sélectionner un élément dans une liste, ou d'entrer une donnée dans une zone de saisie.

Il existe trois types de listes combinées, définis par la valeur (entière) de la propriété *Style* :

- 0 (*VbComboDropDown*) (Valeur par défaut.) *Liste déroulante modifiable*. Comprend une zone de saisie avec, à côté, une flèche déroulante (*drop down*) permettant de faire apparaître une liste, dite déroulante. L'utilisateur peut sélectionner une option dans la liste, ou taper ce qui convient dans la zone de texte.
- 1 (*VbComboSimple*) *Liste modifiable simple*. Comprend une zone de texte et une liste non déroulante (toujours visible). L'utilisateur peut sélectionner une option de la liste, ou taper ce qui convient dans la zone de texte. La taille d'une liste modifiable simple inclut les parties texte et liste. Par défaut, une liste modifiable simple est dimensionnée de sorte que la liste ne s'affiche pas. Augmenter la valeur de la propriété *Height* pour afficher une plus grande partie de la liste.
- 2 (*VbComboDrop-DownList*) *Liste déroulante* (sans saisie possible). Ce type de liste permet seulement à l'utilisateur de sélectionner une option dans la liste déroulante (semblable à 0 mais la saisie est interdite).

On retrouve des propriétés et méthodes similaires à celles des contrôles *TextBox*.

### 8.5 Les contrôles *Frame, CheckBox, OptionButton*

Les contrôles *Frame* (en français, cadre) permettent de regrouper (souvent de manière fonctionnelle) plusieurs contrôles dans un cadre afin d'en faire un *groupe* identifiable. Un tel contrôle est souvent couplé aux cases à option. La propriété *Caption* permet de donner un intitulé au cadre, les propriétés *Enabled* et *Visible* permettent de rendre le cadre respectivement inactif (de même pour les contrôles qu'il contient), et non visible - caché - (de même pour les contrôles qu'il contient).

Les contrôles *CheckBox* (en français, case à cocher) peuvent être sélectionnés, ou non. Ils sont habituellement utilisés pour exprimer des attributs optionnels, par exemple l'attribut *Marié* d'une personne. La propriété *Value* permet de savoir si une case est désélectionnée - non cochée - (0, *vbUnChecked*), ou sélectionnée (1, *vbChecked*), ou indisponible - la case est alors ombrée - (2, *vbGrayed*) (cette valeur ne peut être fixée que par programmation).
L'événement *Click* est appelé lorsqu'une case à cocher est sélectionnée, ou désélectionnée.

Les contrôles *OptionButton* (en français, case à option) fonctionnent sensiblement selon le même principe, mais par *groupe*. Dans un même *groupe*, seule une case à option peut être sélectionnée à la fois, i.e., les cases à option d'un même groupe sont exclusives les unes par rapport aux autres. Les cases à option forment un groupe lorsqu'elles sont dans le même *cadre*.
La propriété *Value* permet de savoir si une case à option est sélectionnée - pointée - (*True*), ou désélectionnée (*False*).
L'événement *Click* est appelé lorsqu'une case à option est sélectionnée, ou désélectionnée.

## 8.6 Les menus

Visual Basic fournit un moyen simple de créer des menus *via* l'*éditeur de menus* (*Menu Editor*), voir le menu *Outils | Créateur de menus*. L'*éditeur de menus* est, en fait, une manière d'affecter les propriétés d'un menu. Une fois un menu créé, ses propriétés et ses événements associés sont visibles dans les fenêtres *Propriétés* et *Code*.

La boîte de dialogue de l'*éditeur de menus* contient les zones de saisies *Caption* et *Name*, pour indiquer respectivement le nom du menu visible par l'utilisateur (par exemple, pour entrer le menu *Fichier*, entrer *&Fichier*), et le nom de la variable utilisée par le programmeur (par exemple, *mnuFichier*). Il est bien sûr possible de créer :

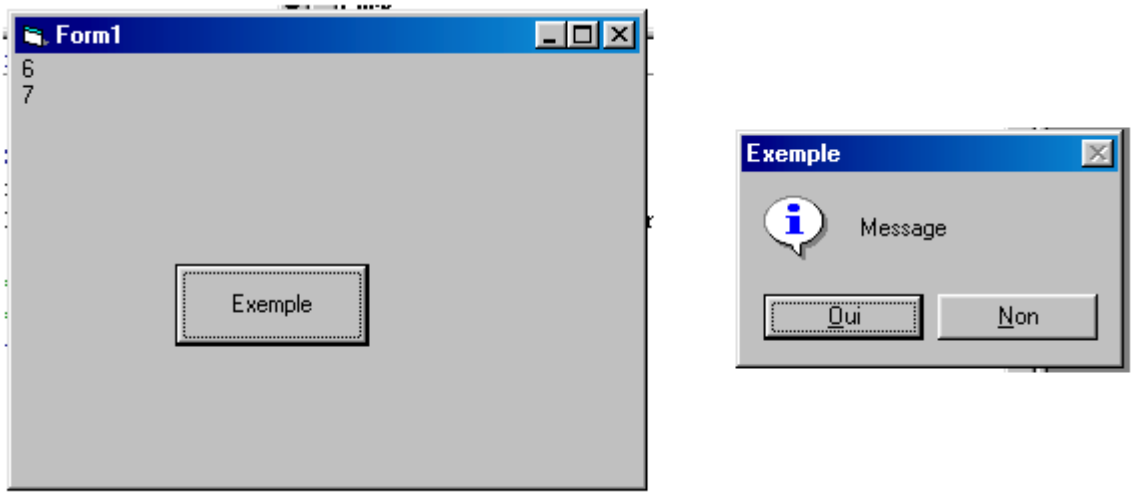
- des menus déroulants (listes déroulantes d'options qui n'apparaissent qu'à la suite d'un clic sur un titre de menu) ;
- des menus imbriqués (cinq niveaux de retrait au maximum).

## 8.7 La fonction *MsgBox*

La fonction *MsgBox* (boîte de message) permet d'afficher, de manière standard sous Windows, une boîte de dialogue fournissant un message à destination de l'utilisateur à propos de l'état d'exécution du programme. La boîte de message *MsgBox* peut, selon sa configuration, afficher un message (un texte), un icône de même que des boutons (ces derniers permettant de fournir des informations au programme).

A titre d'exemple, le programme suivant permet - par un appui sur un bouton de commande - de faire apparaître une boîte de message (voir la figure qui suit). Selon la réponse (appui sur bouton *Yes* ou *No*) de la boîte de message, le contenu de la variable en retour de la fonction *MsgBox* est affiché dans la fenêtre.

```
Option Explicit

Private Sub CmdExemple_Click()
   Dim r As Integer
   r = MsgBox("Message", vbYesNo + vbInformation + vbApplicationModal, _
              "Exemple")
   ' vbYesNo permet l'affichage des boutons Yes et No
   ' vbInformation permet l'affichage de l'icône information
   ' vbApplicationModal indique que la boîte de message est modale (i.e., l'utilisateur
   ' ne peut pas interagir sur une fenêtre tant que la boîte de message n'est pas fermée)
   ' r = 6 (vbYes) si le bouton Yes a été pressé
   ' r = 7 (vbNo) si le bouton No a été pressé
   Print r
End Sub
```




## 8.8 Le contrôle *Timer*

Le contrôle *Timer* (minuterie) pilote le système d'horloge, il permet le déclenchement de certaines actions à intervalles réguliers. La propriété *Enabled* permet de lancer, ou d'arrêter, l'horloge, tandis que la propriété *Interval* permet de définir le nombre de millisecondes (compris entre 0 et 64 767 (~ 64,8 *sec*)) entre les événements de la minuterie.
Le seul événement disponible est *Timer*, il se produit lorsque l'intervalle de temps prédéfini est écoulé. Cet événement est donc périodique, il suffit de lui associer une procédure pour que celle-ci soit déclenchée périodiquement. Notons que ce contrôle n'est pas visible en mode exécution.

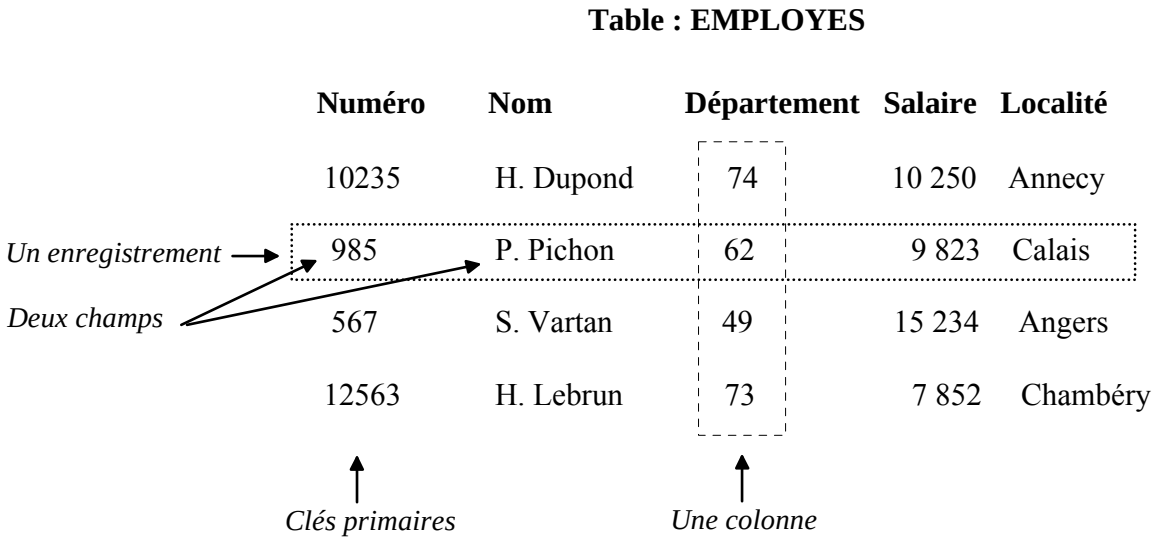
# 9 BASE DE DONNÉES : ACCÈS

L'utilisation de fichiers de *type séquentiel* est appropriée dans le cas d'applications exploitant la plupart des informations contenues dans le fichier. L'emploi des fichiers *à accès direct* est plus adapté à des applications n'utilisant qu'une faible partie des données du fichier, et où il est important de pouvoir localiser rapidement une donnée. Visual Basic fournit de nombreuses manipulations sur de tels fichiers, qui, faute de temps, ne sont pas abordées dans ce cours.
Le principal inconvénient lié à l'utilisation des fichiers (*séquentiels* ou *à accès direct*) vient de ce qu'ils permettent simplement un accès aux données, sans possibilité de recherche plus efficace. Au contraire, les systèmes de bases de données[2] vont organiser les données de manière à faciliter la demande de requêtes appropriées à la recherche des données souhaitées.

Dans ce chapitre, nous abordons la gestion des bases de données *relationnelles*. L'accès aux informations dans de telles bases, largement utilisées, peut se faire dans l'environnement de Visual Basic *via* l'utilisation de requêtes[3] *SQL* (*Structured Query Language*, ce langage permet d'effectuer des requêtes, i.e., de rechercher des informations satisfaisant un critère donné).

[2] Une *base de données* est une collection de données. Un *système de base de données* regroupe les données, le matériel sur lequel les données résident, le logiciel (appelé *moteur de base de données*) qui contrôle le stockage - l'extraction - de données, et les utilisateurs.
[3] *Requête* se traduit par *query* en anglais.

Les bases de données *relationnelles* sont des représentations logiques de données qui permettent d'établir des relations entre les données, ceci sans se soucier de l'implantation physique des structures de données. Une base de données relationnelle est composée de *tables* (voir la figure suivante à titre d'exemple).




Les lignes d'une table sont appelées des *enregistrements* (en anglais, *records*), ou *lignes* (en anglais, *rows*). Dans l'exemple, la table *EMPLOYES* est constituée de 4 *enregistrement*s. Chaque *colonne* d'une table représente un *champ* différent. Dans l'exemple, la valeur du champ *Numéro* de chacun des enregistrements est appelée la *clé primaire*, au sens où elle permet de référencer les données dans la table. Dans Visual Basic, une table est manipulée comme un objet *Recordset*.

## 9.1 Introduction de l'ADO Data Control 6.0 et du DataGrid Control 6.0

Une fois un contrôle *ADO Data Control* mis en place (connecté à une table d'une base de données), il est possible de lier à ce contrôle plusieurs contrôles, dits *dépendants*, permettant l'affichage et la manipulation de données, le tableau suivant liste quelques-uns de ces contrôles.

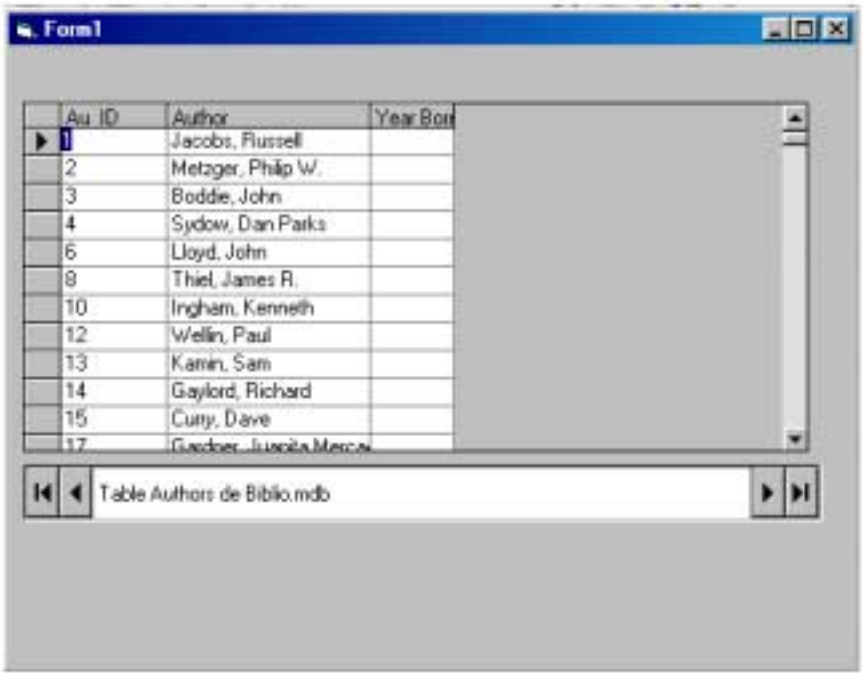
| Contrôle | Description |
|---|---|
| *CheckBox* | Peut être rattaché à un champ booléen d'une table. |
| *ComboBox* | Peut être lié à une colonne d'une table. |
| *DataCombo* | Similaire à *ComboBox*, il permet de passer un champ donné à un autre contrôle de données. |
| *DataGrid* | Permet l'affichage et la manipulation d'une table, ou d'un résultat d'une requête. |
| *DataList* | Similaire au *ListBox*, il permet de passer un certain champ à un autre contrôle de données. |
| *Hierarchical FlexGrid* | Ce contrôle est similaire au *DataGrid,* mais l'utilisateur peut seulement visualiser les données à travers un *FlexGrid*. |
| *Image* | Affiche des images sous différents formats. |
| *ImageCombo* | Similaire à *ComboBox*, il peut afficher des images dans une liste. |
| *Label* | Peut être utilisé pour afficher la valeur d'un champ d'une table. |
| *Listbox* | Peut être lié à une colonne d'une table. |
| *MaskedEdit* | Similaire à *TextBox*, il permet de contrôler le format de données dans un champ. |
| *Microsoft Chart* | Permet la représentation d'un tableau de données sous forme d'un graphe. |
| *MonthView* | Affiche des dates dans un format approprié. |
| *PictureBox* | Une version plus élaborée du contrôle *Image*. |

| | |
|---|---|
| *RichTextBox* | Permet d'afficher, ou d'entrer, un texte formaté. |
| *TextBox* | Permet d'afficher, ou d'entrer, un texte. |

Ces contrôles *dépendants* sont liés au contrôle *ADO Data Control via* les propriétés *DataSource, DataField* et éventuellement *DataFormat*, afin de spécifier le nom du contrôle Data, un champ de la base de données, et éventuellement le format des données.

Le contrôle *ADO Data Control* est un moyen de contrôler l'accès à une base de données, le contrôle *DataGrid* affiche sous la forme d'une table, et permet la manipulation par un utilisateur, des données demandées à partir d'une base de données.

Prenons comme exemple la feuille présentée par la figure suivante. Sa construction se fait en utilisant les contrôles *ADO Data Control* et *DataGrid*, elle ne nécessite aucune programmation pour établir la connexion avec, par exemple, la table *Authors* de la base de données *Biblio.mdb* (accessible dans le répertoire *\Microsoft Visual Studio\VB98*). Notons qu'il existe trois autres tables : *Publishers, Title Author* et *Titles* (utiliser le logiciel *Access* pour visualiser *Biblio.mdb*).




Le contrôle *ADO Data Control* permet de connecter l'application à une base de données, et rend possible la visualisation et la manipulation des données par différents contrôles. Le contrôle *DataGrid* permet un accès aisé à un enregistrement.

Il est nécessaire d'ajouter ces contrôles dans la *barre de contrôles* (ces contrôles ne sont pas *intrinsèques*). Pour cela, sélectionner *Composants…* à partir du menu *Projet* afin d'afficher la boîte de dialogue *Composants*. Après avoir glissé ces deux contrôles (à savoir *Microsoft ADO Data Control 6.0 (OLEDB)* et *Microsoft Data Grid Control 6.0 (OLEDB)*, sur une feuille, il est possible de visualiser leurs propriétés. Celles qui sont relatives au contrôle *ADO Data Control* sont, notamment, accessibles *via* un clic à droite sur le contrôle, suivi d'une sélection de *Propriétés du contrôle ADODC*, ce qui permet l'affichage d'une boîte de dialogue, intitulée *Pages de propriétés*.

Les étapes suivantes permettent de créer une connexion à la base de données *Biblio.mdb* (cette procédure se ramène à créer une *source de données OLE*[4] *DB*) :

1. Dans le cadre *Source de la connexion*, sélectionner l'option *Utiliser une chaîne de connexion* et cliquer sur le bouton *Créer* pour afficher la boîte de dialogue *Data Link Properties* (une chaîne de connexion comprend, notamment, le *fournisseur de la base de données*, la localisation et le nom de la base de données). *Biblio.mdb* est une base de données de Microsoft Access.
2. Dans l'onglet *Provider*, sélectionner *Microsoft Jet 3.51 OLE DB Provider*.
3. Dans l'onglet *Connection*, on introduit, dans le champ *Select or enter a database name*, le répertoire et le nom de la base de données.
4. Cliquer sur le bouton *Test Connection* afin de déterminer si la connexion est établie.
5. Dans l'onglet *RecordSource* de la boîte de dialogue *Pages de propriétés*, sélectionner, dans le champ *Type de commande*, *2 – adCmdTable* pour spécifier qu'une table dans la base de données va être la source des données. Sélectionner *Authors*, dans le champ *Nom de procédure stockée ou de table*, pour spécifier que les données seront retrouvées à partir de la table *Authors* dans la base de données.

Une fois la *chaîne de connexion* spécifiée, il reste, avant de pouvoir exécuter l'application, à fixer les valeurs des propriétés comme indiquées dans le tableau suivant :

| Contrôle | Propriété | Valeur |
|---|---|---|
| *Adodc1* | *Caption* | *Table Authors de Biblio.mdb*. |
| *DataGrid1* | *DataSource* | Sélectionner *Adodc1*. |
| | *AllowUpdate* | Sélectionner *False* pour empêcher une modification par l'utilisateur. |

A titre d'essai, on pourra mettre :

- une valeur *True* à la propriété *AllowUpDate* de *DataGrid1* pour permettre une modification de la base de données (au préalable, penser à effectuer une copie du fichier *Biblio.mdb* modifiable à loisir, ce qui sous-entend de changer le contenu du champ *Select or enter a database name*),
- un nom de table différent dans la propriété *RecordSource* pour permettre l'affichage de cette table.

## 9.2 Survol du langage SQL

A titre d'exemple, nous allons considérer la base de données *Biblio.mdb*. Elle regroupe quatre tables, intitulées *Authors, Publishers, Title Author* et *Titles*.

La table *Authors* regroupe trois colonnes contenant, pour chaque auteur, dans le champ :

- *Au_ID* : Son numéro (unique) d'identification *ID*, ce numéro est la clé primaire pour cette table,
- *Author* : Son nom,
- *Year Born* : Son année de naissance.

La table *Publishers* regroupe dix colonnes contenant, pour chaque éditeur, dans le champ :

- *PubID* : Son numéro (unique) d'identification *ID*, ce numéro est la clé primaire pour cette table,
- *Name* : Son nom,
- *Company Name* : Son nom étendu,
- *Address* : Le nom de sa rue,

[4] *Object Linking and Embedding* (*OLE*) est une technologie - sur laquelle s'appuient les technologies *ActiveX* - qui permet d'associer des applications entre elles. Par exemple, il est possible d'insérer une feuille *Excel* dans un document *Word*, ceci sans avoir explicitement ouvert le logiciel *Excel*. *OLE* permet notamment à une application de contrôler d'autres applications (cette fonction s'appelle *Automation*).

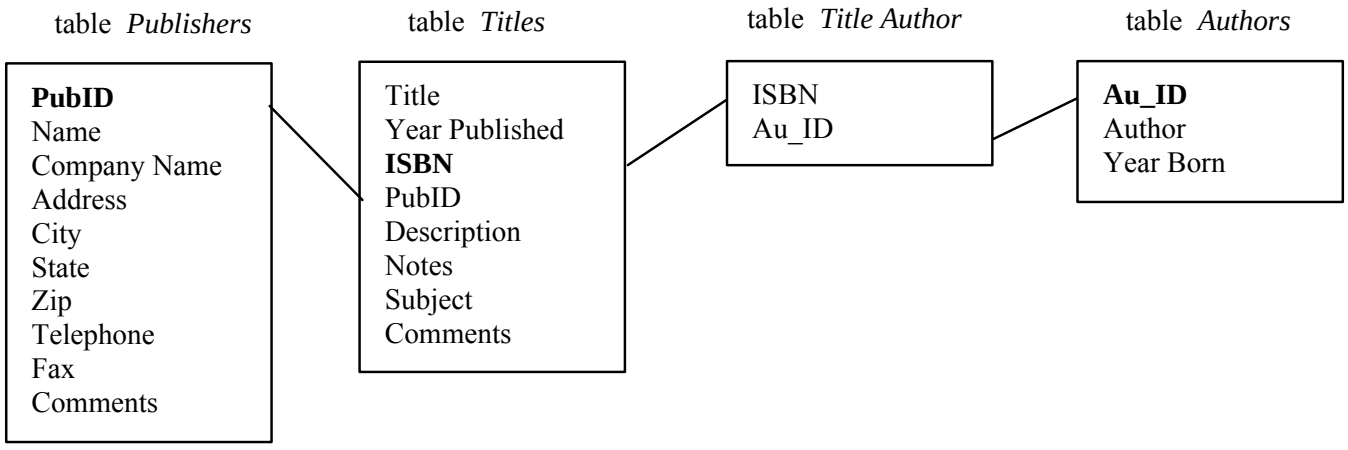
- *City* : Sa ville,
- *State* : Son état,
- *Zip* : Son code postal,
- *Telephone* : Son numéro de téléphone,
- *Fax* : Son numéro de fax,
- *Comments* : D'éventuels commentaires.

La table *Title Author* regroupe deux colonnes contenant dans le champ :
- *ISBN* : Le numéro ISBN d'un ouvrage,
- *Au_ID* : Le numéro *ID* de l'auteur, ou d'un auteur, de cet ouvrage.

Cette table permet de lier les ouvrages à leurs auteurs (notons qu'un ouvrage peut être écrit par plusieurs auteurs, et qu'un auteur peut écrire plusieurs ouvrages).

La table *Titles* regroupe huit colonnes contenant, pour chaque livre, dans le champ :
- *Title* : Son titre,
- *Year Published* : Son année de parution,
- *ISBN* : Son numéro ISBN, ce numéro est la clé primaire pour cette table,
- *PubID* : Son numéro d'identification *ID*, ce numéro doit correspondre avec un numéro ID dans la table *Publishers*,
- *Description* : Son prix,
- *Notes* : D'éventuelles notes,
- *Subject* : Des mots-clés sur le contenu du livre,
- *Comments* : Une description du livre.

La figure suivante montre les relations entre ces tables. Le champ indiqué en **gras** dans chaque table est la clé primaire de la table.




Le langage *SQL* permet la définition et l'extraction de données à partir de bases de données relationnelles. Dans le cadre de ce cours, nous ferons référence au langage *SQL* qu'aux travers des instructions listées dans le tableau suivant :

| Mot-clé SQL | Description |
|---|---|
| *SELECT* | Sélectionne (retrouve) des champs à partir d'une, ou plusieurs tables. |
| *FROM* | Pour indiquer les tables contenant les champs (associé à *SELECT*). |
| *WHERE* | Critère de sélection pour déterminer les enregistrements à retrouver. |
| *ORDER BY* | Critère pour mettre en ordre les enregistrements. |

| | |
|---|---|
| *INNER JOIN … ON* | Permet de lier (fusionner) des tables entre elles. |

- **Instructions SELECT, FROM**

Le format de base d'une requête *SELECT* est :

**SELECT * FROM *TableName***

L'astérisque (*) indique que tous les champs de la table *TableName* sont sélectionnés. Par exemple, la sélection de tous les champs de la table *Authors* utilise la requête :

**SELECT * FROM Authors**

Pour sélectionner seulement les colonnes *Au_ID* et *Author*, on utilise la requête :

**SELECT Au_ID, Author FROM Authors**

- **Instruction WHERE**

Le langage *SQL* permet la sélection d'enregistrements satisfaisant un critère dit de *sélection* en utilisant le l'instruction optionnelle *WHERE*. Le format de base d'une requête *SELECT* avec un critère de sélection est :

**SELECT * FROM *TableName WHERE* critère**

Par exemple, la sélection des champs de la table *Authors* pour lesquels l'année de naissance (*Year Born*) de l'auteur est supérieure, ou égale, à 1950, se fait à travers la requête :

**SELECT * FROM Authors WHERE [Year Born] >= 1950**

La requête :

**SELECT * FROM Authors WHERE Author Like 'a_r%'**

permet de localiser les champs de la table *Authors* pour lesquels le nom de l'auteur (*Author*) commence avec la lettre *a*, laquelle est suivie d'un caractère quelconque (spécifié par _), lequel est suivi de la lettre *r*, laquelle est suivie d'un nombre quelconque de caractères (spécifié par %).

- **Instruction ORDER BY**

Le résultat d'une requête peut être visualisé selon l'ordre alphabétique, ou dans le sens inverse, en utilisant l'instruction optionnelle *ORDER BY*.

Par exemple, pour obtenir l'affichage des champs de la table *Authors* dans l'ordre alphabétique des noms d'auteurs (*Author*), on utilise la requête :

**SELECT * FROM Authors ORDER BY Author ASC**

*ASC* (pour ASC*ending*) spécifie l'ordre alphabétique (croissant), *DESC* (pour DESC*ending*) spécifie l'ordre alphabétique inverse.

Notons que les instructions précédentes peuvent se combiner pour ne former qu'une requête. Par exemple, pour localiser les enregistrements de tous les auteurs dont le nom commence par la lettre *a*, et ceci par ordre alphabétique, on utilise la requête :

```
SELECT *
FROM Authors
WHERE Author Like 'a%'
ORDER BY Author ASC
```

Une requête dans Visual Basic consiste en une (longue) chaîne.

- **Instruction INNER JOIN … ON**

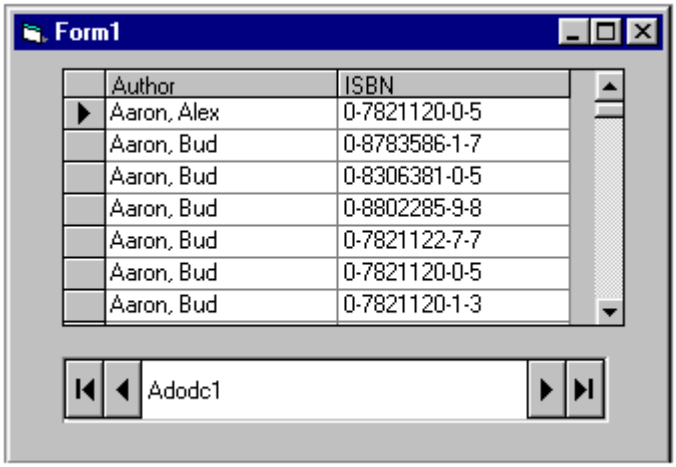
Il est souvent nécessaire de fusionner des données issues de plusieurs tables, ceci est possible en utilisant le mot-clé *INNER JOIN* dans l'instruction *FROM* d'une requête *SELECT*.
*INNER JOIN* réunit les enregistrements appartenant à deux, ou plusieurs, tables en testant les valeurs assorties (en accord) dans un champ qui est commun aux tables considérées.

Le format de base est :

**SELECT * FROM *Table1* INNER JOIN *Table2* ON *Table1.field* = *Table2.field***

La partie " **ON *Table1.field* = *Table2.field*** " spécifie les champs considérés dans les tables qui devront être comparés afin de déterminer les enregistrements retenus. Par exemple, pour fusionner le champ *Author* de la table *Authors* avec le champ *ISBN* de la table *Title Author*, dans un ordre alphabétique par *Author*, de manière à disposer du, ou des, numéro(s) ISBN du, ou des, livre(s) écrit(s) par chacun des auteurs (cf. figure suivante), on utilise la requête :

```
SELECT Author, ISBN
FROM Authors INNER JOIN [Title Author]
ON Authors.Au_ID = [Title Author].Au_ID
ORDER BY Author ASC
```

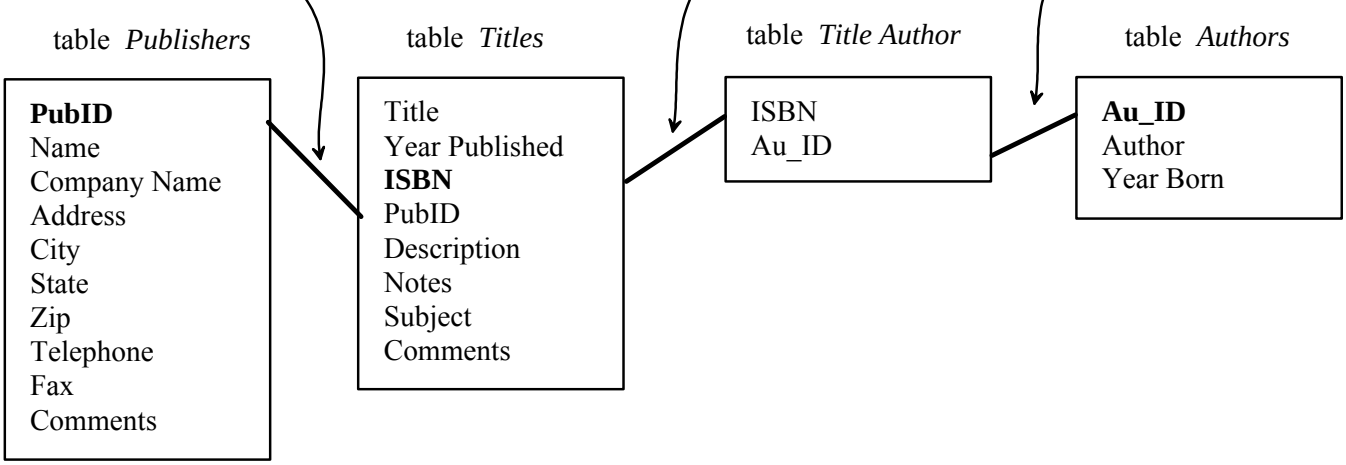



On peut se représenter la fusion effectuée dans l'exemple précédent, ainsi :

1. On commence par le premier enregistrement de la table *Authors*.
2. On cherche le champ *Au_ID*.
3. On va dans la table *Title Author* pour trouver le, ou les, enregistrement(s) dont le champ *Au_ID* correspond à celui trouvé dans l'étape 2.
4. On considère l'enregistrement suivant dans la table *Authors*, puis on repasse à l'étape 2.

**Remarque :** La base de données *Biblio.mdb* contient une requête prédéfinie, intitulée *All Titles*, qui permet, pour chaque ouvrage, l'affichage de son titre, son numéro ISBN, l'auteur (nom, prénom), son année de parution, et le nom de l'éditeur, *via* la requête (visible sous Access) suivante :

```
SELECT Titles.Title, Titles.ISBN, Authors.Author,
        Titles.[Year Published], Publishers.[Company Name]
FROM
Publishers INNER JOIN
        (Authors INNER JOIN
                ([Title Author] INNER JOIN Titles
                        ON [Title Author].ISBN = Titles.ISBN)
        ON [Title Author].Au_ID = Authors.Au_ID)
ON Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```



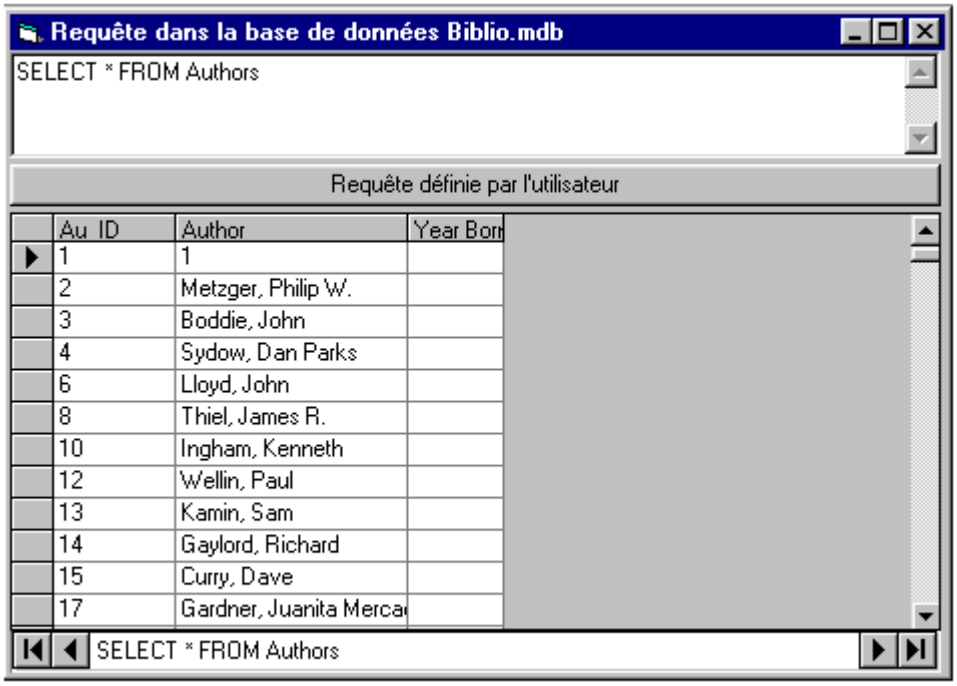

Notons que la requête produit, s'il y a lieu, un enregistrement pour chacun des auteurs d'un même ouvrage.

## 9.3 Description de l'ADO Data Control 6.0 et du DataGrid Control 6.0

Le tableau suivant décrit quelques propriétés, méthodes et événements associés au contrôle *ADO Data Control 6.0*.

| Propriété, méthode, événement | Description |
|---|---|
| | |
| **Propriété** | |
| *ConnectionString* | Spécifie une chaîne contenant des paramètres permettant de se connecter à la base de données. |
| *UserName* | Spécifie le nom de l'utilisateur lors d'une connexion à la base de données. |
| *Password* | Spécifie le mot de passe lors d'une connexion à la base de données. |
| *RecordSource* | Spécifie une chaîne contenant une requête, ou un nom de table (utilisé pour extraire des données). |
| *CommandType* | Spécifie le type de commande utilisée dans le *RecordSource* (*adCmdTable* pour une table, *adCmdText* pour une requête). |
| *LockType* | Spécifie la façon de bloquer les enregistrements pour empêcher des utilisateurs de modifier des données en même temps. |
| *Mode* | Spécifie les droits de connexion (lecture seule, écriture seule, lire/écrire, etc.). |
| *MaxRecords* | Spécifie le nombre maximum d'enregistrements à retourner dans un objet *Recordset*. |
| *ConnectionTimeout* | Spécifie le temps d'attente d'une connexion avant de générer une erreur. |
| | |
| **Méthode** | |
| *Refresh* | Rafraîchit les données dans un objet *Recordset* à la suite d'un changement d'une propriété dans le contrôle *ADO Data Control*. |
| | |
| **Evénement** | |
| *EndOfRecordset* | Déclenché lors d'une tentative d'accès après le dernier enregistrement du *Recordset*. |
| *WillChangeField* | Appelé avant qu'une opération change la valeur d'un, ou plusieurs, champ(s). |
| *FieldChangeComplete* | Appelé après le changement de la valeur d'un, ou plusieurs, champ(s). |
| *WillChangeRecord* | Appelé avant qu'une opération change la valeur d'un, ou plusieurs, enregistrement(s). |
| *RecordChangeComplete* | Appelé après le changement de la valeur d'un, ou plusieurs, enregistrement(s). |
| *WillChangeRecordset* | Appelé avant qu'une opération change l'objet *Recordset*. |
| *RecordsetChangeComplete* | Appelé après un changement de l'objet *Recordset*. |
| *WillMove* | Appelé avant une opération changeant la position courante dans le *Recordset*. |
| *MoveComplete* | Appelé après un changement de la position courante dans le *Recordset*. |

Une particularité fondamentale du contrôle *ADO Data Control* est de pouvoir exécuter des requêtes SQL en cours d'exécution *via* la propriété *RecordSource*, ce qui permet un changement dynamique des données manipulées par ce contrôle. A titre d'exemple, on va créer un programme permettant l'exécution de requêtes *SQL*, introduites *via* un contrôle *TextBox* par l'utilisateur, et visualisées à l'aide d'un contrôle *DataGrid* (cf. la saisie d'écran dans la figure suivante).




Ce programme est listé ci-dessous.

---

```
' Demande de requêtes sur la base de données Biblio.mdb
' à travers le contrôle ADO Data Control

Option Explicit

Private Sub Form_Load()
    ' affichage du texte de la requête mise par défaut
    txtRequête.Text = Adodc1.RecordSource
    Adodc1.Caption = Adodc1.RecordSource
End Sub

Private Sub cmdRequête_Click()
    On Error Resume Next  ' l'exécution reprend à l'instruction qui suit immédiatement la dernière instruction
                          ' ayant appelée la procédure contenant la routine de gestion d'erreur
    cmdRequête.Enabled = False          ' désactivation (temporaire) du bouton
    Adodc1.RecordSource = txtRequête.Text
    Call Adodc1.Refresh       ' soumission de la requête à la base de données, i.e., rafraîchissement du Recordset;
                              ' à l'achèvement de la requête, les résultats sont automatiquement affichés dans le
                              ' contrôle DataGrid
    Adodc1.Caption = Adodc1.RecordSource
End Sub

Private Sub Adodc1_MoveComplete( _          ' signification des paramètres en annexe E
    ByVal adReason As ADODB.EventReasonEnum, _
    ByVal pError As ADODB.Error, _
```

```
    adStatus As ADODB.EventStatusEnum, _
    ByVal pRecordset As ADODB.Recordset)
    cmdRequête.Enabled = True          ' réactivation du bouton pour permettre une nouvelle requête
End Sub
```

---

Les propriétés, fixées lors de la phase de création de l'application, sont données dans le tableau qui suit.

| Contrôle | Propriété | Valeur |
|---|---|---|
| *Adodc1* | *ConnectionString* | Spécifie la chaîne de connexion. |
| | *RecordSource* | Cliquer le bouton **…** de cette propriété. Sélectionner *1 - adCmdText* dans le champ *CommandType*, et introduire "SELECT * FROM Authors" dans le champ *CommandText (SQL)*. |
| *DataGrid1* | *DataSource* | Adodc1 |
| *cmdRequête* | *Caption* | Requête définie par l'utilisateur. |
| *txtRequête* | *Multiline* | True |
| | *ScrollBars* | 2 - Vertical |

Relativement au contrôle *DataGrid* :

1. La propriété *AllowAddNew* mise à *True* (à *False* par défaut) permet d'utiliser la dernière ligne du contrôle *DataGrid* pour entrer un nouvel enregistrement.
2. Si la propriété *AllowDelete* est à *True* (mise à *False* par défaut), l'utilisateur peut détruire un enregistrement en cliquant le rectangle gris situé à gauche de la ligne pour sélectionner la ligne dans la table, et en appuyant sur la touche *Suppr* du clavier.
3. La propriété *AllowUpdate* mise à *True* (à *False* par défaut) permet à l'utilisateur de modifier les données.

## 9.4 L'objet Recordset

*Recordset* est en fait un objet sur lequel s'appuie un contrôle Data pour manipuler les enregistrements d'une base de données.

- L'objet *Fields* permet de manipuler des colonnes de données dans un *Recordset*. Par exemple,

```
[Data control].Recordset.Fields ("Nom_d_un_champ")
```

retourne un objet *Fields* représentant la valeur dans l'enregistrement courant de la colonne *Nom_d_un_champ*.
Il existe d'autres propriétés de l'objet *Fields*, notamment, *Attributes, Size, Type* et *Value*.

La propriété *Bookmark* d'un *Recordset* peut être utilisée pour pointer vers l'enregistrement courant. Pour cela, il suffit de déclarer une variable de type *Variant* et de lui assigner la valeur de *Bookmark*. Par exemple, en supposant que *Localisation_bookmark* est de type *Variant*, l'instruction

```
Localisation_bookmark = [Data control].Recordset.Bookmark
```

mémorise l'emplacement de l'enregistrement courant.

Les propriétés *BOF* et *EOF* permettent de savoir si on se trouve au début, respectivement à la fin, du jeu d'enregistrements, i.e., avant le premier enregistrement, respectivement après le dernier enregistrement.

- Un objet *Recordset* possède cinq méthodes *Move* qui permettent de se déplacer au sein d'un jeu d'enregistrements, à savoir :

| Méthode | Description |
|---|---|
| *MoveFirst* | Permet de se positionner sur le premier enregistrement. |
| *MoveLast* | Permet de se positionner sur le dernier enregistrement. |
| *MoveNext* | Permet de se positionner sur l'enregistrement suivant. |
| *MovePrevious* | Permet de se positionner sur l'enregistrement précédent. |
| *Move n* | Permet de se déplacer de *n* enregistrements (vers l'avant ou l'arrière, selon le signe de *n*). |

Les quatre premières méthodes se rapportent aux quatre boutons dessinés sur le contrôle Data (voir la figure qui suit).


**Illustration :** A titre d'exercice, réaliser un programme dont le comportement correspond à *Ex_Recordset.exe*.

## 10 MULTIPLE DOCUMENT INTERFACE

Dans les chapitres précédents, ont été développées des applications dites *Single Document Interface* (*SDI*). Quand une application *SDI* telle que *Notepad* s'exécute, seul un document à la fois est affiché (*Notepad* est un éditeur de texte très simple permettant l'édition d'un seul document (en fait un fichier *text*) à la fois).
Au contraire, une application telle que *Word* permet l'édition de plusieurs documents en même temps, chacun de ces documents ayant sa propre fenêtre. *Word* est un exemple d'application dite *Multiple Document Interface* (*MDI*), i.e., une application dans laquelle plusieurs documents peuvent être ouverts en même temps. Notons qu'un document n'est pas nécessairement un fichier texte (*text*), cela peut-être un fichier image (*bitmap*), un fichier compressé, etc.
Chaque fenêtre contenue dans une application *MDI* est appelée fenêtre *fille* (en anglais : *child*), alors que la fenêtre de l'application - qui est unique - est appelée fenêtre *mère* (on parle aussi de fenêtre *parente*, ou *conteneur*). Une fenêtre *fille* se comporte comme n'importe quelle fenêtre, excepté qu'elle ne peut que rester à l'intérieur de la fenêtre *mère* (ce qui augmente la cohérence de l'application). Notons qu'une fenêtre *mère* peut contenir des fenêtres *filles* ayant chacune une fonctionnalité propre, par exemple, une fenêtre *fille* peut permettre l'édition d'une image, alors qu'une autre va permettre d'éditer un texte.
Une fenêtre *MDI fille* est représentée dans Visual Basic avec une feuille *MDI* (en anglais : *MDI form*). A partir d'un projet de type *EXE Standard*, Visual Basic crée automatiquement une feuille *Form1*, l'ajout au projet d'une feuille *MDI mère*, nommée par défaut *MDIForm1*, se fait en sélection *Projet* | *Ajouter une feuille MDI* (notons qu'un projet ne peut contenir qu'une seule feuille *MDI mère*).
La création d'une application *MDI* sous-entend l'utilisation de plusieurs feuilles (au minimum une feuille *mère* et une feuille *fille*). En fait, une feuille *fille* est une feuille standard (telle que *Form1*) pour laquelle la propriété *MDIChild* est mise - en mode création - à *True* (l'ajout de feuille supplémentaire se fait en sélectionnant *Projet* | *Ajouter une feuille*).

Au contraire des feuilles standards, une feuille *MDI mère* peut seulement contenir des contrôles :

- disposant d'une propriété *Align*, c'est le cas des contrôles *PictureBox* (notons que rien n'empêche à ces derniers de contenir des contrôles n'ayant pas la propriété *Align*, par exemple, des *CheckBox*, des *ComboBox*). La propriété *Align* décrit où un contrôle est localisé (au sommet, à gauche, etc.) ;
- ou, non visibles en mode exécution, par exemple un contrôle *Timer* (contrôle *minuterie*).

Comme pour les feuilles standards, les feuilles *mères*, ou *filles*, peuvent contenir des menus (la création d'un menu est facilitée par l'emploi de la boîte de dialogue *Créateur de menus*, accessible à partir du menu *Outils*. Le menu de la feuille *mère* est affiché seulement si la fenêtre *fille* active (i.e., ayant le *focus*) ne comporte pas de menu. Si la fenêtre *fille* active a un menu, il apparaîtra dans la fenêtre *mère*, au détriment du menu de la fenêtre *mère*.
Notons que le fait d'agrandir au maximum (clic sur le bouton *Agrandir*) une fenêtre *fille* fait que son *Caption* est automatiquement combiné avec le *Caption* de la fenêtre *mère*.

Lorsqu'un programme contient plusieurs feuilles, on utilise la boîte de dialogue *Projets* | *Propriétés du projet* pour spécifier un *objet de démarrage*. Cet objet est souvent la feuille *MDI mère* dans le cas d'une application *MDI*. Ainsi en mode exécution, la feuille *mère* est automatiquement chargée en premier et affichée, cette feuille pouvant ensuite gérer les feuilles *filles*.
Pour certaines applications, il est parfois difficile d'identifier la feuille active au départ. Aussi, Visual Basic fournit une procédure, intitulée *Main* (procédure placée dans un module standard (i.e., dans un fichier *.bas*) dont l'entête est : *Private Sub Main* ()), qui, une fois positionnée *via* la boîte de dialogue *Projets* | *Propriétés du projet* pourra être l'*objet de démarrage*.

Il est souvent utile de permettre la création dynamique de feuilles. En utilisant des variables objets, il est possible de créer plusieurs copies, ou instances, d'une même feuille, chaque nouvelle instance possédant les mêmes propriétés, méthodes et événements associés. Par exemple, la procédure événementielle suivante permet l'affichage d'une feuille qui est une copie de la feuille *Form1*.

```
Private Sub mnuFNouveau_Click ()
        Dim NouvelleFeuille As Form              ' voir annexe F
        Set NouvelleFeuille = New Form1
        NouvelleFeuille.Show
End Sub
```

Notons que la variable objet *NouvelleFeuille*, de type *Form*, est locale à la procédure *mnuFNouveau*. Mais, même si cette variable n'existe plus en dehors de la procédure, la feuille créée continue d'exister. Par contre, elle ne peut plus être référencée par le nom *NouvelleFeuille*. Une solution consiste à utiliser le mot clé *Me*, ce mot clé permet de référencer une feuille, pour autant qu'elle soit active (ayant le *focus*). Il est aussi possible d'accéder à une feuille active en utilisant la propriété *ActiveForm*. Notons que si un objet *MDIForm* est actif, cette propriété spécifie la feuille *MDI fille* active.

## 11 RÉSEAUX, INTERNET ET *WEB*

Les applications réseaux, décrites ici, utilisent une communication basée sur un modèle d'interaction *client-serveur*. L'application qui établit le contact (de manière active) s'appelle le *client*, alors que celle qui attend (passivement) un contact s'appelle le *serveur*.
Dans le cas où la connexion, réclamée par le *client*, est acceptée (par le *serveur*), alors le *client* et le *serveur* communiquent *via* des *sockets*[5]. Ainsi, la gestion de réseaux par une application est semblable à celle de transferts de données dans un fichier – un programme peut lire à partir d'un *socket*, ou écrire sur un *socket*, aussi simplement que s'il s'agissait de lire, ou d'écrire, dans un fichier.

[5] Un *socket* est défini par un couple adresse IP – numéro de port.

Deux formes de communication sont supportées par les protocoles de transport : Avec ou sans connexion. Dans le premier cas, deux applications doivent établir une connexion avant de pouvoir s'en servir pour transmettre des données. A titre d'exemple, *TCP* (*Transmission Control Protocol*) fournit aux applications une interface orientée connexion : Une application doit d'abord demander à *TCP* d'ouvrir une connexion avec une autre application, ensuite les deux applications peuvent s'échanger des données. A la fin de la communication, la connexion doit être close. Cette communication est similaire à celle d'une conversation téléphonique.
L'autre choix consiste en une interface sans connexion qui permet à une application d'envoyer à tout moment un message à n'importe quelle destination. Dans chacun de ses messages, l'application émettrice doit en spécifier la destination. A titre d'exemple, *UDP* (*User Datagram Protocol*) fournit un transport sans connexion. Cette communication est similaire à des envois de lettres par la poste.
Le protocole *UDP* ne garantit pas une bonne réception des paquets : Des paquets peuvent se perdre, se dupliquer, ou arriver hors délai, ce qui nécessite l'utilisation de programmes supplémentaires pour traiter ces problèmes.

Visual Basic fournit plusieurs contrôles permettant le développement d'applications orientées réseaux. Le tableau qui suit décrit les trois principaux contrôles, détaillés par la suite.

| Contrôle | Description |
|---|---|
| *WebBrowser* | Permet aux applications de disposer d'un navigateur *Web*[6], de visualiser des documents et de transférer des fichiers. Ce contrôle permet des développements semblables à *Internet Explorer*. |
| *Internet Transfert* | Permet aux applications d'utiliser les protocoles *HTTP* (*HyperText Transfer Protocol*) ou *FTP* (*File Transfer Protocol*). |
| *Winsock* | Permet la programmation d'applications *client-serveur* à travers *UDP* ou *TCP*. |

## 11.1 Le contrôle *WebBrowser*

Internet utilise plusieurs protocoles. Le protocole *HTTP*, omniprésent à travers le *Web*, utilise des *URLs* (*Universal Resource Locators*[7]) comme références externes pour localiser des ressources sur Internet. Visual Basic permet la manipulation d'*URL* à travers le contrôle *WebBrowser*. Ce contrôle, non *intrinsèque*, est accessible à travers le composant *Microsoft Internet Controls*.

A titre d'exemple, on propose le programme suivant qui correspond à une version rudimentaire d'un navigateur *Web*. En fait, il s'agit d'un simple interpréteur *HTML*[8] pour l'affichage des documents.

```
' Utilisation du contrôle WebBrowser pour faire un navigateur Web (élémentaire)
Option Explicit

Private Sub Form_Load()
  ' Affiche la page par défaut (home page) lors de l'enregistrement de la feuille
  Call WebBrowser1.GoHome
```

[6] Un navigateur est un programme interactif qui permet de visualiser des informations issues du *Web*.
[7] Les *URLs* permettent aux navigateurs d'extraire le protocole utilisé pour accéder à l'élément, le nom de l'ordinateur qui héberge l'élément, et le nom de ce dernier. Par exemple, le fait qu'une *URL* commence par *http://* indique au navigateur un accès à l'élément à l'aide du protocole *HTTP*.
[8] La plupart des documents Web sont écrits en langage *HTML* (*HyperText Markup Language*), en plus du texte, un document contient des balises qui en définissent la structure et le formatage.

```
End Sub

Private Sub cmdGo_Click()
  ' Aller à l'URL spécifiée dans txtURL.Text (si txtURL.Text est non vide !!)
  On Error Resume Next
  If txtURL.Text <> "" Then
    Call WebBrowser1.Navigate(txtURL.Text)
  End If
End Sub

Private Sub cmdBack_Click()
  ' Aller à la page précédente contenue dans la liste Historique
  On Error Resume Next
  Call WebBrowser1.GoBack
End Sub

Private Sub cmdForward_Click()
  ' Aller à la page suivante contenue dans la liste Historique
  On Error Resume Next
  Call WebBrowser1.GoForward
End Sub

Private Sub cmdHome_Click()
  ' Aller à la page par défaut (home page)
  On Error Resume Next
  Call WebBrowser1.GoHome
End Sub

Private Sub Form_Resize()
  ' Dimensionnements de txtURL et WebBrowser1
  On Error Resume Next
  WebBrowser1.Width = ScaleWidth - WebBrowser1.Left
  WebBrowser1.Height = ScaleHeight - WebBrowser1.Top
  txtURL.Width = ScaleWidth - txtURL.Left
End Sub

Private Sub WebBrowser1_DocumentComplete( _
    ByVal pDisp As Object, URL As Variant)
  ' Affichage de l'URL dans txtURL.Text une fois le transfert de la page terminée
  txtURL.Text = URL
End Sub
```

---

La figure qui suit est une recopie d'écran du navigateur *Web*.


Les valeurs des propriétés de la page et des contrôles constituant l'application sont indiquées dans le tableau suivant :

| Contrôle | Propriété | Valeur |
|---|---|---|
| *frmWebBrowser* | *Caption* | Exemple de navigateur *Web* |
| | *WindowState* | 2 – Maximized (permet un agrandissement maximal) |
| *lblURL* | *Caption* | URL : |
| *txtURL* | *Text* | (vide) |
| *cmdGo* | *Caption* | Aller à l'URL |
| *cmdBack* | *Caption* | Précédente |
| *cmdForward* | *Caption* | Suivante |
| *cmdHome* | *Caption* | Démarrage |
| *WebBrowser1* | | Valeurs par défaut |

**Remarques :**

- La méthode *GoHome* de *WebBrowser1*, appelée dans la procédure *Form_Load*, permet le chargement de la page par défaut (cette page correspond à celle par défaut dans *Internet Explorer*, spécifiée dans *Internet Explorer via* le cadre *Page de démarrage* de la boîte de dialogue *Outils | Options Internet*…).
- L'affichage de l'*URL*, effectué dans la procédure événementielle *WebBrowser1_DocumentComplete* une fois la page affichée, peut être différent de l'*URL* introduite par l'utilisateur (quelquefois un serveur *Web* redirige les *URLs* vers d'autres emplacements).
- Liés au contrôle *WebBrowser*, il existe d'autres propriétés, méthodes et événements, tels que :
  pour les propriétés : *Busy, LocationName, LocationURL, Offline,*
  pour les méthodes : *GoSearch, Refresh, Stop,*
  pour les événements : *DownloadBegin, DownloadComplete, ProgessChange*.

### 11.2 Le contrôle Internet Transfer

Le contrôle *Internet Transfer* opère à un niveau plus bas que celui du contrôle *WebBrowser*, il supporte deux des protocoles les plus connus : *HTTP* (*HyperText Transfer Protocol*) et *FTP* (*File Transfer Protocol*).

A la différence du contrôle *WebBrowser* qui permet l'affichage d'une page *Web*, le contrôle *Internet Transfer* fournit uniquement le document *HTML* (accessible par son *URL*), ceci pour, par exemple, *parser* (en français, effectuer une analyse grammaticale) le texte en vue d'une détection spécifique.
Le protocole *FTP* permet le transfert de fichier entre ordinateurs *via* Internet. Le contrôle *Internet Transfer* permet un transfert synchrone (le programme attend la fin du transfert) ou asynchrone (l'exécution du programme se poursuit et un événement se produit pour indiquer au programme la fin du transfert).

Ce contrôle, non *intrinsèque*, est accessible à travers le composant *Microsoft Internet Transfer Control 6.0*.

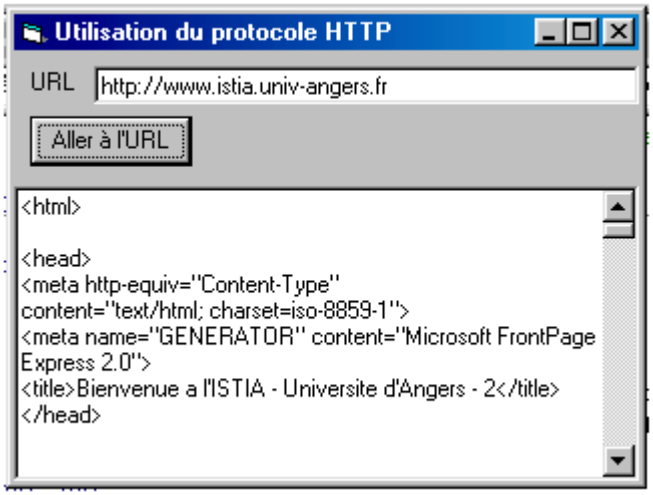
- **Réalisation d'un client *HTTP* à travers la méthode *OpenURL***

Dans le programme suivant, l'utilisateur introduit l'adresse *URL* d'un serveur *HTTP*. Le fait de cliquer sur le bouton *Aller à l'URL* provoque l'affichage du document, au format *HTML*, spécifié par l'*URL* (cf. la saisie d'écran dans la figure suivante).

---

```
' Utilisation du contrôle Internet Transfer pour afficher le contenu d'un fichier
' à travers une connexion HTTP
Option Explicit

Private Sub cmdGo_Click()
  ' Aller à l'URL spécifiée dans txtURL.Text (si txtURL.Text est non vide !!)
  On Error Resume Next
  If txtURL.Text <> "" Then
    txtOutput.Text = "Recherche du fichier ..."
    txtOutput.Text = Inet1.OpenURL(txtURL.Text)
  End If
End Sub

Private Sub Form_Resize()
  ' Dimensionnements de txtURL et txtOutput
  On Error Resume Next
  txtOutput.Width = ScaleWidth - txtOutput.Left
  txtOutput.Height = ScaleHeight - txtOutput.Top
  txtURL.Width = ScaleWidth - txtURL.Left
End Sub
```

---




Les propriétés des contrôles sont décrites dans le tableau qui suit.

| Contrôle | Propriété | Valeur |
|---|---|---|
| *frmHTTP* | *Caption* | Utilisation du protocole HTTP |
| *lblURL* | *Caption* | URL |
| *txtURL* | *Text* | (vide) |
| *cmdGo* | *Caption* | Aller à l'URL |
| *txtOutput* | *Multiline* | *True* |
| | *ScrollBars* | 3 – Both |
| *Inet1* | | Valeurs par défaut |

**Remarques :**

- L'instruction **txtOutput.Text = Inet1.OpenURL(txtURL.Text)** utilise la méthode *OpenURL* du contrôle *Internet Transfer* pour transférer le document spécifié par l'*URL*. Le transfert se fait de manière synchrone, en effet le programme attend tant que le transfert n'est pas terminé.
- En fait, pour afficher un document *HTML*, il serait préférable d'utiliser, à la place d'un contrôle *TextBox*, un contrôle *RichTextBox*, ceci pour permettre de prendre en compte une édition sous *UNIX*.

**Conseil** : Lors de l'utilisation de la méthode *OpenURL*, ou de la méthode *Execute* (décrite par la suite), il n'est pas nécessaire de définir la propriété *Protocol*. En effet, le contrôle *Internet Transfer* active automatiquement le protocole approprié, tel qu'il est déterminé par la partie protocole de l'*URL*.

- **Réalisation d'un client *FTP* à travers la méthode *Execute***

Dans le programme suivant, on permet à l'utilisateur d'introduire des commandes *FTP* (écrites en MAJUSCULE). Au préalable, l'utilisateur introduit l'adresse URL du serveur *FTP*, le nom de l'utilisateur (*anonymous* permet un accès anonyme, minimal à des fichiers) et un mot de passe (lors d'un accès anonyme, utilisé *guest* ou son adresse de courrier électronique). Le fait de cliquer sur le bouton *Exécution de la commande FTP* provoque l'exécution de la commande *FTP*. La figure qui suit est une recopie d'écran de l'application.

Relativement à la commande *GET*, l'instruction suivante :

**GET /rep1/fic.txt C:\temp\essai.txt**

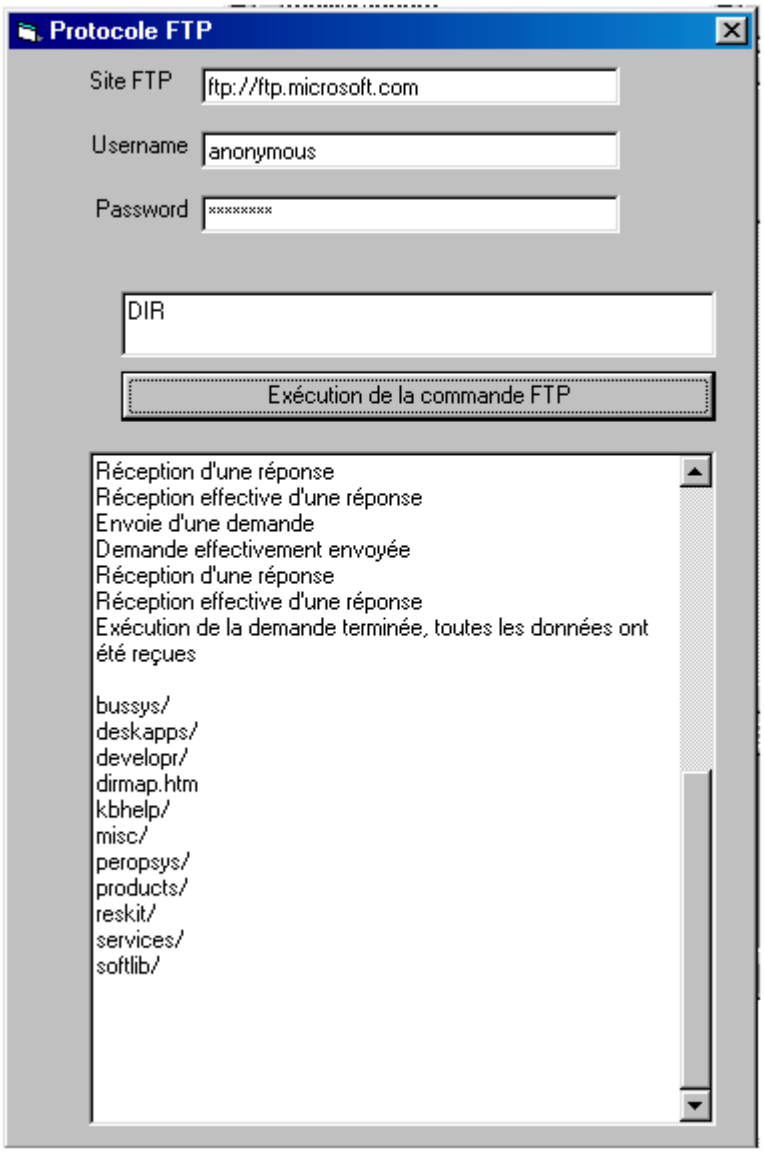
permet d'obtenir la copie du fichier distant *fic.txt*, situé dans le répertoire *rep1*. Le contenu du fichier est placé dans le fichier local ayant pour nom *essai.txt*, situé dans le répertoire (bien sûr existant) *C:\temp*.

Les propriétés, associées aux différents contrôles, sont décrites dans le tableau qui suit.

---

```
' Transfert d'un fichier texte via FTP
Option Explicit
Dim command_get As Boolean
Dim fichier As String

Private Sub cmdGo_Click()
  Inet1.URL = txtSite.Text
  Inet1.UserName = txtUserName.Text
  Inet1.Password = txtPassword.Text
  txtLog.Text = _
    txtLog.Text & vbCrLf & "COMMANDE FTP : " & txtCommand.Text & vbNewLine
  If InStr(1, UCase$(txtCommand.Text), "GET") <> 0 Then
     command_get = True          ‘ Commande FTP relative à GET
     Dim fin As Integer
     fin = InStrRev(txtCommand.Text, " ")
     fichier = Mid$(txtCommand.Text, fin + 1, Len(txtCommand.Text) - fin)
     ' Debug.Print fichier
  Else
     command_get = False         ‘ Commande FTP non relative à GET
  End If
  Call Inet1.Execute(, txtCommand.Text)
End Sub

Private Sub Inet1_StateChanged(ByVal State As Integer)
  Select Case State
    Case icResolvingHost
      txtLog.Text = txtLog.Text & "Recherche adresse IP de l'ordinateur hôte" & vbCrLf
    Case icHostResolved
      txtLog.Text = txtLog.Text & "Adresse IP de l'ordinateur hôte trouvée" & vbCrLf
    Case icConnecting
      txtLog.Text = _
        txtLog.Text & "En cours de connexion" & vbCrLf
    Case icConnected
      txtLog.Text = _
        txtLog.Text & "Connexion établie" & vbCrLf
    Case icRequesting
      txtLog.Text = txtLog.Text & "Envoie d'une demande" & vbCrLf
    Case icRequestSent
      txtLog.Text = txtLog.Text & "Demande effectivement envoyée" & vbCrLf
    Case icReceivingResponse
      txtLog.Text = _
        txtLog.Text & "Réception d'une réponse" & vbCrLf
```

```
    Case icResponseReceived
      txtLog.Text = _
        txtLog.Text & "Réception effective d'une réponse" & vbCrLf
    Case icDisconnecting
      txtLog.Text = _
        txtLog.Text & "En cours de déconnexion" & vbCrLf
    Case icDisconnected
      txtLog.Text = _
        txtLog.Text & "Déconnexion effective avec l'ordinateur hôte" & vbCrLf
    Case icError
      txtLog.Text = txtLog.Text & "Erreur : " & vbCrLf & _
      "Code : " & Inet1.ResponseCode & vbCrLf & Inet1.ResponseInfo
    Case icResponseCompleted
      Dim data As Variant
      txtLog.Text = txtLog.Text & _
        "Exécution de la demande terminée, toutes les données ont été reçues" & vbCrLf
      If command_get = False Then
        ' Extraction de données
        ' (utiliser icByteArray, plutôt que icString, lors de la
        '  réception d'un fichier binaire)
        data = Inet1.GetChunk(1024, icString)
        Do While LenB(data) > 0
           txtLog.Text = txtLog.Text & vbCrLf & data & vbCrLf
           ' Extraction de données
           data = Inet1.GetChunk(1024, icString)
        Loop
        txtLog.Text = txtLog.Text & vbCrLf & data & vbCrLf
      Else
        ' Ecriture de données dans un fichier   (voir annexe G pour plus de détails)
        ' Ouverture du fichier : Open Nom_du_fichier For Mode As Identificateur
        Open fichier For Binary Access Write As #1  ' identificateur = 1
        ' Extraction de données
        data = Inet1.GetChunk(1024, icString)
        Do While LenB(data) > 0
           ' Ecriture : Put #identificateur, numéro_d'enregistrement, variable
           Put #1, , data
           ' Extraction de données
           data = Inet1.GetChunk(1024, icString)
        Loop
        Put #1, , data
        Close #1      ' Fermeture du fichier
      End If
  End Select
  txtLog.SelStart = Len(txtLog.Text)
End Sub
```




| Contrôle | Propriété | Valeur |
|---|---|---|
| *frmFTP* | *Caption* | Protocole FTP |
| *lblSite* | *Caption* | Site FTP |
| *txtSite* | *Text* | (vide) |
| *lblUserName* | *Caption* | UserName |
| *txtUserName* | *Text* | anonymous |
| *lblPassword* | *Caption* | Password |
| *txtPassword* | *Text* | (vide) |
| | *PasswordChar* | * (permet la création d'un champ de mot de passe) |
| *txtCommand* | *Caption* | (vide) |
| *cmdGo* | *Caption* | Exécution de la commande FTP |
| *txtLog* | *Multiline* | *True* |
| *Inet1* | *Protocol* | 2 - icFTP |

**Remarques :**

- Contrairement à la méthode *OpenURL*, la méthode *Execute* fonctionne selon un mode *asynchrone*, en effet l'exécution du programme peut se poursuivre durant le service demandé au serveur.

- L'application est automatiquement prévenue d'une réponse de la part du serveur *via* la procédure événementielle *Inet1_StateChanged*. La structure *Select Case* se base sur des constantes de la forme *ic*State pour indiquer l'état courant (*State*) du contrôle. Ces changements d'état sont indiqués dans le contrôle *TextBox* intitulé *txtLog*.
- L'état *icResponseCompleted* indique que la requête faite auprès du serveur est terminée, il s'ensuit une phase de sauvegarde dans un fichier. L'instruction

**data = Inet1.GetChunk(1024, icString)**

(en français *chunk* signifie *gros morceau*) permet de récupérer des données dans un segment de taille 1024. La constante *icString* permet de récupérer une chaîne de caractères, alors que la constante *icByteArray* permettrait de récupérer les données sous forme d'un tableau d'octets (utilisé dans le cas de fichiers binaires).

- **Rappels de quelques commandes *FTP***

| Commande | Description |
|---|---|
| **CD** *file1* | Change le dossier en cours par celui spécifié dans *file1*. |
| **CDUP** | Retour au dossier parent. Équivaut à "CD..". |
| **CLOSE** | Ferme la connexion *FTP* en cours. |
| **DELETE** *file1* | Supprime le fichier spécifié dans *file1*. |
| **DIR** *file1* | Dossier. Recherche le dossier spécifié dans *file1*. Les caractères génériques sont autorisés mais l'hôte distant dicte la syntaxe. Si l'argument *file1* est omis, la liste complète du dossier de travail en cours est renvoyée.<br>Utiliser la méthode **GetChunk** pour renvoyer la liste du dossier. |
| **GET** *file1 file2* | Récupère le fichier distant spécifié dans *file1* et crée le nouveau fichier local spécifié dans *file2*. |
| **LS** *file1* | Liste. Recherche le dossier spécifié dans *file1*. Les caractères génériques sont autorisés mais l'hôte distant dicte la syntaxe. Utiliser la méthode **GetChunk** pour renvoyer les fichiers du dossier. |
| **MKDIR** *file1* | Crée le dossier spécifié dans *file1*. |
| **PUT** *file1 file2* | Copie un fichier local spécifié dans *file1* vers l'hôte distant spécifié dans *file2*. |
| **PWD** | Impression du dossier de travail. Renvoie le nom du dossier en cours. Utiliser la méthode **GetChunk** pour renvoyer les données. |
| **QUIT** | Termine la connexion en cours pour l'utilisateur. |
| **RECV** *file1 file2* | Récupère le fichier distant spécifié dans *file1* et crée le nouveau fichier local spécifié dans *file2*. Équivaut à **GET**. |
| **RENAME** *file1 file2* | Donne au fichier distant spécifié dans *file1* le nouveau nom spécifié dans *file2*. |
| **RMDIR** *file1* | Supprime le dossier distant spécifié dans *file1*. |
| **SEND** *file1 file2* | Copie d'un fichier local spécifié dans *file1*, vers l'hôte distant spécifié dans *file2*. Équivaut à **PUT**. |
| **SIZE** *file1* | Renvoie la taille du dossier spécifié dans *file1*. |

**Remarque :** Certaines des commandes listées ci-dessus ne sont pas autorisées si l'utilisateur ne possède pas les droits correspondants sur le serveur hôte. Par exemple, souvent, les sites *FTP* anonymes n'autorisent pas la suppression de fichiers ou de dossiers.

## 11.3 Le contrôle Winsock

Le contrôle *Winsock* permet à des applications client-serveur de communiquer *via* les protocoles *TCP* ou *UDP*. Nous n'utiliserons dans cette section que le protocole *TCP*.

- **Construction d'un serveur *TCP* élémentaire**

Plusieurs étapes sont à effectuer pour construire un serveur :
- L'étape 1 consiste à ajouter un contrôle *Winsock* à la feuille. Ce contrôle, non *intrinsèque*, est accessible à travers le composant *Microsoft Winsock Control 6.0*.
- On spécifie dans l'étape 2 le numéro de port, utilisé lors de connexions avec des clients, à travers la propriété *LocalPort* du contrôle *Winsock*. Chaque client fera une requête au serveur *via* ce port.
- L'étape 3 permet au serveur d'attendre une demande de connexion par un client à travers la méthode *Listen* du contrôle *Winsock*. Le serveur est informé d'une demande de connexion par un client à travers l'exécution de la procédure événementielle *ConnectionRequest*.
- L'étape 4 consiste à accepter la connexion entrante à travers la méthode *Accept* du contrôle *Winsock*. Une fois cette méthode appelée, des données peuvent être transmises entre le serveur et le client. Avant d'accepter la connexion, vérifier que l'état (*State*) du contrôle *Winsock* correspond à *sckClosed*. Dans le cas contraire, il s'agit d'appeler la méthode *Close* pour fermer la précédente connexion. Si la connexion entrante est refusée, l'homologue (client) reçoit l'événement *Close*.
- L'étape 5 est relative à la phase de communication entre le serveur et le client.
  La procédure événementielle *DataArrival* du contrôle *Winsock* est exécutée lorsque des données parviennent au serveur. L'instruction *tcpServer.GetData* (*message*) place les données dans la chaîne de caractères *message* (2 arguments optionnels permettent de spécifier le type de données reçues et la longueur maximale autorisée des données).
  Afin d'envoyer des données au client, on utilise la méthode *SendData* du contrôle *Winsock*. Par exemple, l'instruction *tcpServer.SendData* (*message*) permet de transmettre les données contenues dans *message*.
- L'étape 6 se produit lorsque la transmission des données est terminée. Le fait que le client ferme la connexion provoque l'exécution de la procédure événementielle *Close* du contrôle *Winsock*. La connexion serveur devra être fermée *via* l'instruction *tcpServer.Close*.

Un contrôle *Winsock* doit être attribué à chaque connexion avec un client. Le fait de pouvoir disposer dans Visual Basic de tableau de contrôles permet la création de serveurs capables de gérer simultanément plusieurs connexions, ceci sans créer *a priori* un ensemble suffisant de contrôles *Winsock*.

- **Construction d'un client *TCP* élémentaire**

Plusieurs étapes sont à effectuer pour construire un client :
- L'étape 1 consiste à ajouter un contrôle *Winsock* à la feuille (accessible à travers le composant *Microsoft Winsock Control 6.0*).
- Dans l'étape 2, le contrôle *Winsock* côté client doit pouvoir localiser :
  - l'ordinateur distant sur lequel un contrôle envoie, ou reçoit, des données. Vous pouvez fournir soit un nom d'hôte, par exemple "www.microsoft.com", soit une adresse IP sous forme de chaîne ponctuée, telle que "127.0.1.1". Ce nom est placé dans la propriété *RemoteHost* du contrôle *Winsock* ;
  - le numéro de port distant auquel la connexion doit être faite. Ce numéro est placé dans la propriété *RemotePort* du contrôle *Winsock*.
- Dans l'étape 3, la connexion au serveur est demandée *via* un appel de la méthode *Connect* du contrôle *Winsock*. En cas de succès, la procédure événementielle *Connect* du contrôle *Winsock* s'exécute ; en cas d'erreur, la procédure événementielle *Error* du contrôle *Winsock* s'exécute.
- L'étape 4 est relative à la phase de communication entre le serveur et le client.

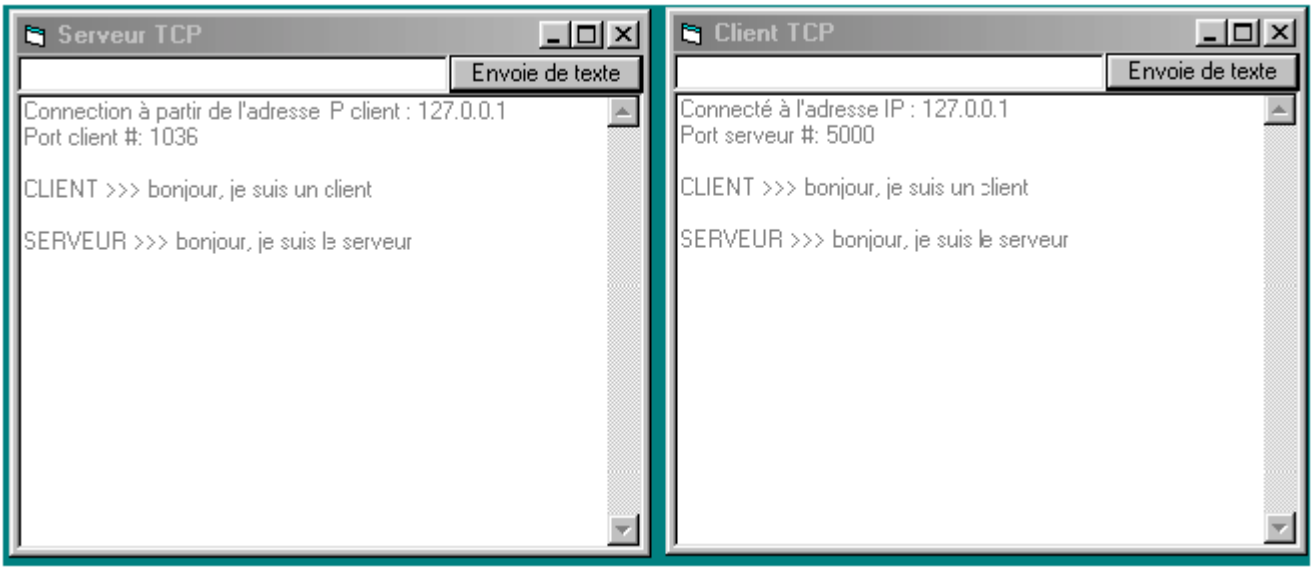
Comme du côté serveur, la procédure événementielle *DataArrival* du contrôle *Winsock* est exécutée lorsque des données parviennent au client. L'instruction *tcpClient.GetData* (*message*) place les données dans *message*.
Des données sont envoyées au serveur en utilisant la méthode *SendData* du contrôle *Winsock*. Par exemple, l'instruction *tcpClient.SendData* (*message*) envoie au serveur des données contenues dans *message*.

- L'étape 5 se produit lorsque la transmission des données est terminée. Le fait que le serveur ferme la connexion provoque l'exécution de la procédure événementielle *Close* du contrôle *Winsock*. La connexion client devra être fermée *via* l'instruction *tcpClient.Close*.

- **Exemple d'interaction client-serveur avec le contrôle *Winsock***

L'application *client-serveur*, considérée ici, utilise le protocole *TCP*. Dans cette application, l'utilisateur côté client et l'utilisateur côté serveur communiquent entre eux à travers des contrôles *TextBox*. Notons que si le client ferme la connexion, le serveur attend une autre connexion.

La figure qui suit est une recopie d'écran de l'application proposée.




Le programme qui suit correspond au serveur *TCP* ; les propriétés associées aux différents contrôles sont décrites dans le tableau qui suit.

---

```
' Serveur élémentaire basé sur le protocole TCP
Option Explicit

Private Sub Form_Load()
  ' Rend le bouton cmdSend non actif jusqu'à
  ' l'établissement d'une connexion
  cmdSend.Enabled = False
  ' Positionne le port local
  tcpServer.LocalPort = 5000
  ' Attente d'une demande de connexion
  Call tcpServer.Listen
```

```
End Sub

Private Sub Form_Resize()
  On Error Resume Next
  Call cmdSend.Move(ScaleWidth - cmdSend.Width, 0)
  Call txtSend.Move(0, 0, ScaleWidth - cmdSend.Width)
  Call txtOutput.Move(0, txtSend.Height, ScaleWidth, _
                      ScaleHeight - txtSend.Height)
End Sub

Private Sub Form_Terminate()
  ' Fermeture (si ce n'est pas déjà fait) de la connexion
  ' côté serveur (il s'ensuit du côté client un appel
  ' automatique de la procédure événementielle Close)
  Call tcpServer.Close
End Sub

Private Sub tcpServer_ConnectionRequest( _
    ByVal requestID As Long)
  ' S'assurer que la précédente connexion côté serveur
  ' a été fermée, ceci avant d'accepter une nouvelle connexion
  If tcpServer.State <> sckClosed Then
    Call tcpServer.Close
  End If

  ' Rend le bouton cmdSend actif du fait de
  ' l'établissement d'une connexion
  cmdSend.Enabled = True
  ' Connexion acceptée du côté du serveur
  Call tcpServer.Accept(requestID)
  ' Affichage de l'adresse IP et du numéro de port du client
  txtOutput.Text = _
    "Connection à partir de l'adresse IP client : " & _
    tcpServer.RemoteHostIP & vbCrLf & _
    "Port client #: " & tcpServer.RemotePort & vbCrLf & vbCrLf
End Sub

Private Sub tcpServer_DataArrival(ByVal bytesTotal As Long)
  Dim message As String
  ' Réception des données envoyées par le client
  Call tcpServer.GetData(message)
  txtOutput.Text = _
    txtOutput.Text & message & vbCrLf & vbCrLf
  txtOutput.SelStart = Len(txtOutput.Text)
  '(la propriété SelStart renvoie ou définit le point
  ' de départ du texte sélectionné, ou indique la position
  ' du point d'insertion si aucun texte n'est sélectionné)
End Sub

Private Sub tcpServer_Close()
  ' Le client a fermé la connexion
  ' Rend le bouton cmdSend non actif du fait de
```

```
  ' la fermeture de la connexion
  cmdSend.Enabled = False
  ' Fermeture de la connexion côté serveur (il s'ensuit
  ' du côté du client un appel automatique à la procédure
  ' événementielle Close)
  Call tcpServer.Close
  txtOutput.Text = txtOutput.Text & _
    "Connexion fermée côté client." & vbCrLf & vbCrLf
  txtOutput.SelStart = Len(txtOutput.Text)
  ' Attente d'une nouvelle demande de connexion
  Call tcpServer.Listen
End Sub

Private Sub tcpServer_Error(ByVal Number As Integer, _
    Description As String, ByVal Scode As Long, _
    ByVal Source As String, ByVal HelpFile As String, _
    ByVal HelpContext As Long, CancelDisplay As Boolean)
  Dim result As Integer
  result = MsgBox(Source & " : " & Description, _
    vbOKOnly, "Erreur TCP/IP")
  End
End Sub

Private Sub cmdSend_Click()
  ' Envoie de données vers le client
  Call tcpServer.SendData("SERVEUR >>> " & txtSend.Text)
  txtOutput.Text = txtOutput.Text & _
    "SERVEUR >>> " & txtSend.Text & vbCrLf & vbCrLf
  txtSend.Text = ""
  txtOutput.SelStart = Len(txtOutput.Text)
End Sub
```

---

| Contôle | Propriété | Valeur |
|---|---|---|
| *frmTCPServer* | *Caption* | Serveur TCP |
| *tcpServer* | *Protocol* | 0 – sckTCPProtocol (par défaut) |
| *txtSend* | *Text* | (vide) |
| *txtOutput* | *Multiline* | *True* |
| *cmdSend* | *Caption* | Envoie de texte |

Le programme qui suit correspond au client *TCP* ; les propriétés associées aux différents contrôles sont décrites dans le tableau qui suit.

---

```
' Client élémentaire basé sur le protocole TCP
Option Explicit
```

```
Private Sub Form_Load()
  ' Rend le bouton cmdSend non actif jusqu'à
  ' l'établissement d'une connexion
  cmdSend.Enabled = False
  ' Positionne l'adresse IP du serveur
  tcpClient.RemoteHost = _
    InputBox("Entrer l'adresse IP du serveur", _
      "Adresse IP", "localhost")
  If tcpClient.RemoteHost = "" Then
    tcpClient.RemoteHost = "localhost"
  End If
  tcpClient.RemotePort = 5000  ' Numéro de port du serveur
  ' Demande de connexion au serveur :
  '    connexion établie --> événement tcpClient_Connect
  '    erreur --> événement tcpClient_Error
  Call tcpClient.Connect
End Sub

Private Sub Form_Terminate()
  ' Fermeture (si ce n'est pas déjà fait) de la connexion côté
  ' client (il s'ensuit du côté serveur un appel automatique
  ' de la procédure événementielle Close)
  Call tcpClient.Close
End Sub

Private Sub Form_Resize()
  On Error Resume Next
  Call cmdSend.Move(ScaleWidth - cmdSend.Width, 0)
  Call txtSend.Move(0, 0, ScaleWidth - cmdSend.Width)
  Call txtOutput.Move(0, txtSend.Height, ScaleWidth, _
                      ScaleHeight - txtSend.Height)
End Sub

Private Sub tcpClient_Connect()
  ' Affichage d'un message (numéros IP et port du client) signalant
  ' l'établissement d'une connexion
  cmdSend.Enabled = True
  txtOutput.Text = "Connecté à l'adresse IP : " & _
    tcpClient.RemoteHostIP & vbCrLf & "Port serveur #: " & _
    tcpClient.RemotePort & vbCrLf & vbCrLf
End Sub

Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
  Dim message As String
  ' Réception des données envoyées par le serveur
  Call tcpClient.GetData(message)
  txtOutput.Text = txtOutput.Text & message & vbCrLf & vbCrLf
  txtOutput.SelStart = Len(txtOutput.Text)
End Sub

Private Sub tcpClient_Close()
  ' Le serveur a fermé la connexion
```

```
    ' Rend le bouton cmdSend non actif jusqu'à
    ' l'établissement d'une connexion
    cmdSend.Enabled = False
    ' Fermeture de la connexion côté client (il s'ensuit du côté
    ' serveur un appel automatique de la procédure événementielle Close)
    Call tcpClient.Close
    txtOutput.Text = _
      txtOutput.Text & "Connexion fermée côté serveur." & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
  End Sub

  Private Sub tcpClient_Error(ByVal Number As Integer, _
      Description As String, ByVal Scode As Long, _
      ByVal Source As String, ByVal HelpFile As String, _
      ByVal HelpContext As Long, CancelDisplay As Boolean)
    Dim result As Integer
    result = MsgBox(Source & ": " & Description, _
      vbOKOnly, "Erreur TCP/IP")
    End
  End Sub

  Private Sub cmdSend_Click()
    ' Envoie de données vers le serveur
    Call tcpClient.SendData("CLIENT >>> " & txtSend.Text)
    txtOutput.Text = txtOutput.Text & _
      "CLIENT >>> " & txtSend.Text & vbCrLf & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
    txtSend.Text = ""
  End Sub
```

---

| Contrôle | Propriété | Valeur |
|---|---|---|
| *frmTCPClient* | *Caption* | Client TCP |
| *tcpClient* | *Protocol* | 0 – sckTCPProtocol (par défaut) |
| *txtSend* | *Text* | (vide) |
| *txtOutput* | *Multiline* | *True* |
| *cmdSend* | *Caption* | Envoie de texte |

Liés au contrôle *Winsock*, il existe d'autres propriétés, méthodes et événements, tels que :
pour les propriétés : *BytesReceived, LocalHostName, LocalHostIP,*
pour les méthodes : *PeekData,*
pour les événements : *SendComplete, SendProgress*.

A titre d'exercice, étudier l'application multi-utilisateurs, intitulée *Serveur* et située dans le répertoire *\partage\M2AI Visual Basic\TP Reseau\Serveur TCP*. Adapter ce serveur à l'application client *TCP* précédente.

# 12 PROGRAMMATION ORIENTÉE-OBJET

La programmation en Visual Basic est dite *orientée objet*. Cette méthode de programmation, assez naturelle, consiste à décomposer une application en un ensemble (fini) d'objets ; ces objets interagissant afin de concourir à la réalisation de l'application. A titre d'exemples, tous les composants de Visual Basic, tels que les feuilles, les contrôles, sont des objets.
Les *objets* (en anglais, *objects*) *encapsulent*, à la fois, les données (attributs décrivant l'objet) et les méthodes (actions, routines permettant de manipuler l'objet) ; les données et les méthodes étant intimement liées entre-elles. Notons que la programmation *orientée objet* donne autant d'importance aux données qu'aux actions rattachées à ces données, alors que la programmation *procédurale* donnera - à travers la réalisation de procédures - plus d'importance aux actions qu'aux données proprement dites.
Notons que la notion d'objet s'intègre pleinement dans le cadre de la programmation événementielle : Le fait qu'un objet puisse provoquer un événement va permettre de signaler aux autres objets que quelque chose est survenu.

La programmation *orientée objet* (POO) repose sur la notion de *classe* : C'est à partir d'une *classe* qu'un *objet* est *instancié*, i.e., créé. Chaque classe contient des données, et l'ensemble des méthodes permettant leurs manipulations. A titre d'exemple, chaque icône de contrôle représente une classe. Quand un contrôle *TextBox* est placé sur une feuille, un objet de la classe *TextBox* (représentée par son icône) est instancié (créé), vers lequel votre programme peut envoyer des messages. Par analogie, un moule à gaufre (ou le plan d'une voiture) peut être considéré comme une classe, sachant qu'une gaufre (ou qu'une voiture) peut être considéré comme une instances de cette classe.

Les objets ont la propriété de *cacher l'information* - en effet, ils communiquent entre-eux sans connaître en détail leurs fonctionnements internes -, ce qui facilite la réalisation de logiciels. En effet, un programmeur pourra manipuler un objet "uniquement" à travers la connaissance de ses différents comportements, sans être concerné par la manière dont ces comportements sont exécutés (codés) : Il est question d'*abstraction de données* (en anglais, *data abstraction*). Par analogie, il est tout à fait possible de conduire une voiture sans pour autant connaître le détail de son mécanisme.
A titre d'exemple, considérons une pile d'assiettes. Une assiette est toujours placée au sommet (en anglais, *push*) de la pile ; de même, c'est toujours du sommet qu'elle en est ôtée (en anglais, *pop*). Ainsi, la dernière assiette placée sur la pile sera toujours la première ôtée : Un tel fonctionnement s'assimile à une structure de données, appelée pile *LIFO* (*Last-In, First-Out*). Par exemple, si les assiettes numéro 1, 2, 3 sont respectivement placées (*push*) sur la pile, les 3 prochaines opérations consistant à ôter (*pop*) les assiettes de la pile retourneront respectivement les assiettes numéro 3, 2 et 1. Un client de la classe *pile* (client : portion de programme utilisant la classe) n'aura pas besoin de connaître la manière dont a été réalisée cette classe, mais seulement son fonctionnement "*dernier entré, premier sorti*" à travers les opérations *push* et *pop*.
En outre, une classe peut être remplacée par une autre version, sans affecter le reste de l'application, pour autant que l'*interface publique* de la classe soit inchangée. L'*interface* d'une classe combine ses propriétés, méthodes et événements, i.e., tout ce qu'il est utile de savoir pour utiliser pleinement la classe.

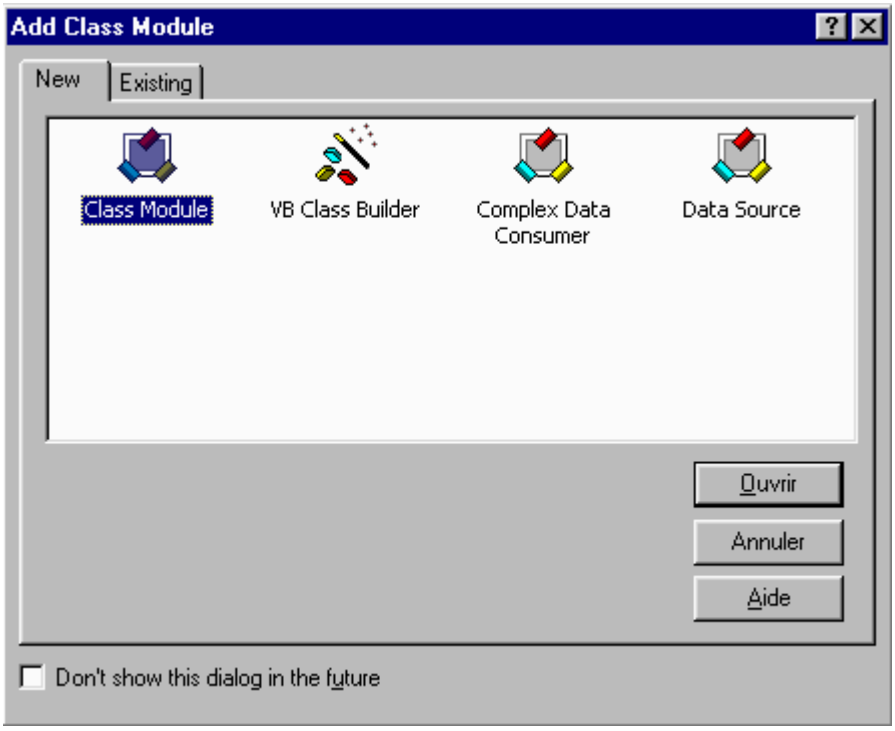
La programmation *orientée objet* requiert trois concepts clés – l'*encapsulation*, l'*héritage* et le *polymorphisme*[9]. A la différence des langages C++ et Java, Visual Basic ne permet pas d'utiliser la notion d'*héritage*, cependant, il fournit une alternative à travers l'utilisation des *interfaces* et une technique appelée *délégation*.

## 12.1 Les modules de classe

[9] Ces notions seront étudiées en détail dans d'autres enseignements de la formation.

La création de *classes* se fait *via* le mécanisme des modules de *classe*. Un module *de classe* contient uniquement du code, il ne peut pas contenir d'*IUG*. Comme les modules *feuilles* et *standards*, les modules de *classes* contiennent une partie *déclaration générale* ; seulement deux événements leurs sont associés : *Initialize* (événement déclenché lorsqu'une instance de la classe est créée) et *Terminate* (événement déclenché avant la destruction d'une instance de la classe).

Pour ajouter un module de *classe* à un projet, sélectionner *Add Class Module* à partir du menu *Project* : Il en résulte la boîte de dialogue suivante, qui permet d'ouvrir un module de *classe*, intitulé (par défaut) *Class1*, et modifiable *via* la propriété *Name*.




**Remarque** : Notons qu'il est possible d'ajouter un module de classe déjà existant (extension *.cls*) dans un projet en cliquant sur l'onglet *Existing* de la précédente boîte de dialogue, ce qui permet de retrouver la classe désirée.

Le mot-clé *New* est utilisé pour créer un objet. Par exemple, l'instruction

**Dim heure As New CTemps**

permet de créer un objet *heure* comme une nouvelle instance de la classe *CTemps* (en fait, Visual Basic ne crée l'objet qu'au moment correspondant au premier accès à une propriété, ou la première fois qu'une méthode est appelée). Une fois l'objet créé, il est possible de l'utiliser - *via* son *interface* (*publique*) - en lui assignant ses propriétés et en invoquant ses méthodes.

Le programme qui suit contient une (première) définition de la classe *CTemps*. Cette classe contient 3 variables *entières* (*Integer*) – *mHeure, mMinute* et *mSeconde*, appelées *variables d'instance* de classe. Chacune de ces 3 variables est déclarée *Private*, ce qui la rend accessible uniquement à travers les méthodes de la classe. En principe, les *variables d'instance* sont déclarées *Private* et les méthodes sont déclarées *Public*.
La classe *CTemps* a une interface publique composée de 3 méthodes *Public* :

- *Fixer_heure* règle l'heure à partir de 3 entiers, chaque entier étant préalablement vérifié ;

- *Mettre_sous_forme_universelle* retourne une chaîne de caractères donnant l'heure au format universel, *i.e.,* sous la forme : *hh:mm:ss* (par exemple : 02:24:35, ou 17:14:45) ;
- *Mettre_sous_forme_standard* retourne une chaîne de caractères donnant l'heure au format standard, *i.e.,* sous la forme : *hh:mm:ss AM* (ou *PM*) (par exemple : 2:24:35 AM, ou 5:14:45 PM).

La procédure *Form_Load*, du programme écrit pour tester la classe, déclare une variable locale de la classe *CTemps* appelée *t*. Le module *Form* peut utiliser la classe *CTemps* car la classe fait partie du projet.
L'objet *t* est instancié, *i.e.*, créé, *via* l'instruction

```
Dim t As New CTemps
```

Lorsque l'objet *t* est instancié, chacune des 3 variables d'instance (privées) est initialisée à 0.
Les méthodes *Mettre_sous_forme_standard*, *Mettre_sous_forme_universelle* de l'objet sont appelées de manière à afficher l'heure sous une forme standard, respectivement universelle.
La méthode *Fixer_heure* permet de régler l'heure, une vérification des données introduites est faite au préalable. Une valeur non valide est remplacée par 0 : C'est le cas - dans le programme de test de la classe - de la variable *mSeconde* lors du deuxième appel de la méthode *Fixer_heure*.

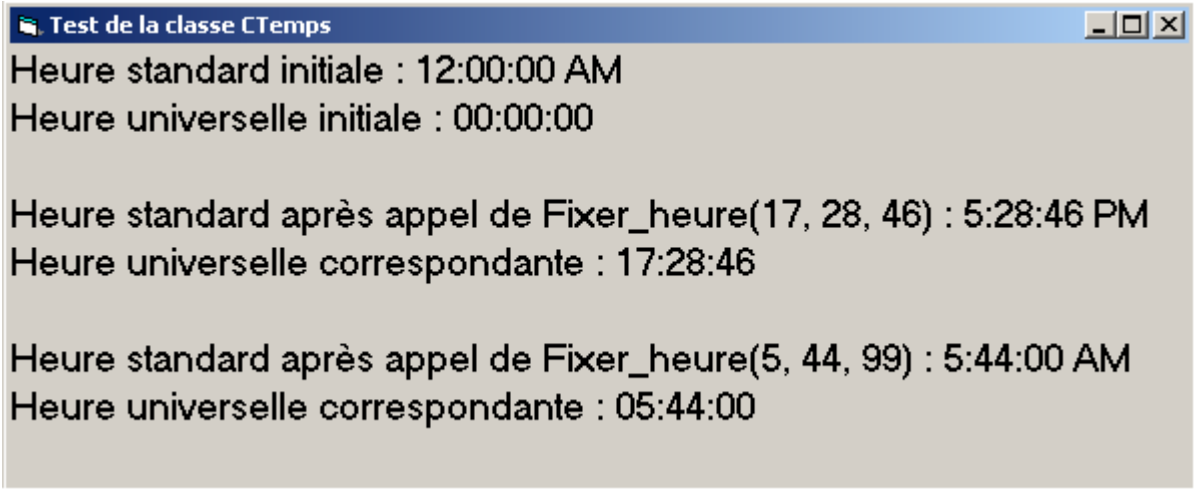
**_____ module de classe Ctemps.cls _____**

```
' Définition de la classe CTemps
Option Explicit

Private mHeure As Integer
Private mMinute As Integer
Private mSeconde As Integer

Public Sub Fixer_heure(ByVal h As Integer, ByVal m As Integer, _
                       ByVal s As Integer)
  mHeure = IIf((h >= 0 And h < 24), h, 0)
  mMinute = IIf((m >= 0 And m < 60), m, 0)
  mSeconde = IIf((s >= 0 And s < 60), s, 0)
End Sub

Public Function Mettre_sous_forme_universelle() As String
  Mettre_sous_forme_universelle = Format$(mHeure, "00") & ":" & _
                                  Format$(mMinute, "00") & ":" & _
                                  Format$(mSeconde, "00")
End Function

Public Function Mettre_sous_forme_standard() As String
  Dim h As Integer
      ' IIf(expr, true part, false part)
  h = IIf((mHeure = 12 Or mHeure = 0), 12, mHeure Mod 12)
  Mettre_sous_forme_standard = h & ":" & _
                               Format$(mMinute, "00") & ":" & _
                               Format$(mSeconde, "00") & " " & _
                               IIf(mHeure < 12, "AM", "PM")
  ' Format$(expr, format) renvoie une valeur de type Variant (String) contenant
  ' une expression formatée selon l'expression de mise en forme indiquée dans format.
```

```
End Function
```

**_____ programme de test de la classe CTemps _____**

```
' Programme permettant de tester la classe CTemps
Option Explicit

Private Sub Form_Load()
  Dim t As New Ctemps
  frmCTempsTest.Show          ' frmCTempsTest : nom de la feuille
  Print "Heure standard initiale : " & t.Mettre_sous_forme_standard()
  Print "Heure universelle initiale : " & t.Mettre_sous_forme_universelle()

  Print

  Call t.Fixer_heure(17, 28, 46)
  Print "Heure standard après appel de Fixer_heure(17, 28, 46) : " & _
        t.Mettre_sous_forme_standard()
  Print "Heure universelle correspondante : " & t.Mettre_sous_forme_universelle()

  Print

  Call t.Fixer_heure(5, 44, 99)
  Print "Heure standard après appel de Fixer_heure(5, 44, 99) : " & _
        t.Mettre_sous_forme_standard()
  Print "Heure universelle correspondante : " & t.Mettre_sous_forme_universelle()
End Sub
```




## 12.2 Accès aux membres d'une classe : *Public, Private*

Les mots-clés *Public* ou *Private* permettent de spécifier l'accès des membres[10] d'une classe (variables d'instance, ou méthodes). Une variable d'instance, ou une méthode, ayant un accès *public* est accessible par chacun des modules du projet. Au contraire, une variable d'instance *privée* peut seulement être manipulée à travers les méthodes de la classe. Les méthodes *privées*, appelées aussi méthodes *utilitaires*,

[10] Cette notion explique la raison pour laquelle les variables d'instance sont préfixées par la lettre *m*.

peuvent seulement être utilisées par d'autres méthodes de la classe. Par défaut, les procédures *Property, Sub* ou *Function* sont *publiques*, alors que les variables d'instance sont *privées*.

Par exemple, considérons le programme suivant pour tester la classe *CTemps*, précédemment définie (cf. § 12.1) :

**_____ programme de test de la classe CTemps _____**

```
Option Explicit

Private Sub Form_Load()
  Dim t As New CTemps
  t.mHeure = 8
  Print t.Mettre_sous_forme_standard()
End Sub
```

Un tel programme déclenche une erreur de compilation. En effet, la variable d'instance *mHeure* de l'objet *t* étant *privée*, elle n'est pas accessible en dehors de la classe *CTemps*.
En fait, l'accès à des données *privées* est rendu possible à travers des méthodes particulières de la classe, appelées *propriétés* (notion introduite au § 3.1). Pour permettre la lecture d'une donnée *privée* en dehors de son module de classe, il est possible de déclarer - dans le module de classe - une méthode *get*. De même, pour permettre la modification d'une donnée *privée* en dehors de son module de classe, il est possible de déclarer - dans le module de classe - une méthode *let*. Ces accès semblent transgresser la notion de données *privées* d'une classe. Toutefois, une méthode *let* a la possibilité de vérifier la cohérence d'une donnée, de même, une méthode *get* permet une "visualisation" différente, éventuellement partielle, d'une donnée "brute". Ces méthodes particulières dans Visual Basic sont appelées *Property Let, Property Set* et *Property Get*. Par exemple, la méthode permettant de modifier la variable d'instance *privée* nommée *mVar* pourrait s'appeler *Property Let Var*, la méthode permettant de lire cette variable pourrait s'appeler *Property Get Var*.

**Exemple :**

Relativement à la classe *CTemps*, la portion de programme suivante définit une propriété *Heure* permettant de stocker l'heure de 0 à 23 :

```
Private mHeure As Integer

Public Property Let Heure(ByVal hr As Integer)        ' -- > modification
  mHeure = IIf((hr >= 0 And hr < 24), hr, 0)
End Property
```

Supposons l'objet *réveil* de la classe *CTemps* défini, considérons l'instruction suivante :

```
réveil.Heure = -6
```

En réponse à cette instruction, la procédure *Property Let* va rejeter cette valeur, non valide, et positionner *mHeure* à 0.

A présent, considérons la procédure *Property Get Heure* suivante :

```
Public Property Get Heure() As Integer ' -- > visualisation
  Heure = mHeure
```

```
End Property
```

L'instruction suivante permet de stocker la valeur de la variable *Heure* de l'objet *réveil* dans la variable *Valeur_heure_alarme* (supposée entière et définie) :

```
Valeur_heure_alarme = réveil.Heure
```

**Emploi de la procédure *Property Set* :**

Supposons définie une classe *CEmployé* qui contient un objet *mDate_anniversaire* de la classe *CDate1*. La procédure *Property Let* ne permet pas d'assigner une valeur à un objet (en l'occurrence *mDate_anniversaire*). Il faut utiliser, à la place, la procédure *Property Set* comme dans les procédures *Property* suivantes :

```
Public Property Set Date_anniversaire(ByVal ani As CDate1)
  Set mDate_anniversaire = ani
End Property

Public Property Get Date_anniversaire() As CDate1
  Set Date_anniversaire = mDate_anniversaire
End Property
```

Remarque : L'instruction *Exit Property* provoque la sortie immédiate d'une procédure *Property* (*Property Get, Property Let* ou *Property Set*).

Le programme qui suit contient une définition accrue de la classe *CTemps* (définie précédemment, cf. § 12.1). Cette nouvelle classe, appelée *CTemps1*, inclut les procédures *Property Let* et *Property Get* pour les variables d'instance *mHeure, mMinute* et *mSeconde*.
Les propriétés *Property Let* vont permettre le contrôle et l'assignation des variables d'instance. La validation des variables *mHeure, mMinute* et *mSeconde* est implémentée dans des méthodes *privées* (*i.e.*, à usage interne à la classe), appelées respectivement *Valider_heure, Valider_minute* et *Valider_seconde*.
Dans cet exemple, chaque procédure *Property Get* retourne la valeur de la variable d'instance souhaitée.

**_____ module de classe Ctemps1.cls _____**

```
' Définition de la classe CTemps1
' Cette classe étend la classe CTemps en fournissant des procédures
' Property pour les propriétés Heure, Minute et Seconde

Option Explicit

' On définit 3 variables d'instances privées (mHeure, mMinute, mSeconde)
Private mHeure As Integer
Private mMinute As Integer
Private mSeconde As Integer

' On définit 3 méthodes publiques (Fixer_Heure, Mettre_sous_forme_universelle,
'                                  Mettre_sous_forme_standard)
Public Sub Fixer_heure(ByVal h As Integer, ByVal m As Integer, _
                       ByVal s As Integer)
  mHeure = Valider_heure(h)
  mMinute = Valider_minute(m)
```

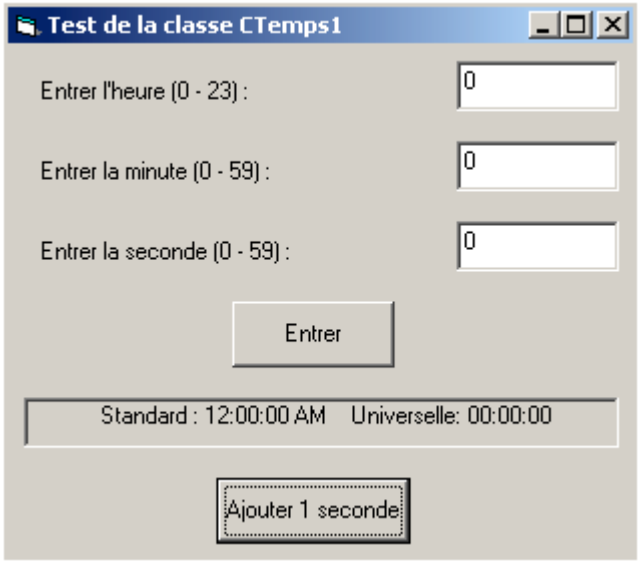
```
  mSeconde = Valider_seconde(s)
End Sub

Public Function Mettre_sous_forme_universelle() As String
  Mettre_sous_forme_universelle = Format$(mHeure, "00") & ":" & _
                                  Format$(mMinute, "00") & ":" & _
                                  Format$(mSeconde, "00")
End Function

Public Function Mettre_sous_forme_standard() As String
  Dim h As Integer
  h = IIf((mHeure = 12 Or mHeure = 0), 12, mHeure Mod 12)

  Mettre_sous_forme_standard = h & ":" & _
                               Format$(mMinute, "00") & ":" & _
                               Format$(mSeconde, "00") & " " & _
                               IIf(mHeure < 12, "AM", "PM")
End Function

' On définit 3 propriétés (Heure, Minute, Seconde)
Public Property Get Heure() As Integer
  Heure = mHeure
End Property

Public Property Let Heure(ByVal h As Integer)
  mHeure = Valider_heure(h)
End Property

Public Property Get Minute() As Integer
  Minute = mMinute
End Property

Public Property Let Minute(ByVal m As Integer)
  mMinute = Valider_minute(m)
End Property

Public Property Get Seconde() As Integer
  Seconde = mSeconde
End Property

Public Property Let Seconde(ByVal s As Integer)
  mSeconde = Valider_seconde(s)
End Property

' On définit 3 méthodes privées (Valider_heure, Valider_minute, Valider_seconde)
Private Function Valider_heure(ByVal h As Integer) As Integer
  Valider_heure = IIf((h >= 0 And h < 24), h, 0)
End Function

Private Function Valider_minute(ByVal m As Integer) As Integer
  Valider_minute = IIf((m >= 0 And m < 60), m, 0)
End Function
```

```
Private Function Valider_seconde(ByVal s As Integer) As Integer
  Valider_seconde = IIf((s >= 0 And s < 60), s, 0)
End Function
```

**_____ programme de test de la classe CTemps1 _____**

```
' Programme permettant de tester la classe CTemps1
Option Explicit

Private mTemps As New CTemps1

Private Sub Form_Load()
  Call mTemps.Fixer_heure(txtHeure.Text, txtMinute.Text, _
                          txtSeconde.Text)
  Call Reactualiser_affichage
End Sub

Private Sub cmdEntrer_Click()
  mTemps.Heure = txtHeure.Text
  mTemps.Minute = txtMinute.Text
  mTemps.Seconde = txtSeconde.Text
  Call Reactualiser_affichage
End Sub

Private Sub cmdAjouter_Click()
  mTemps.Seconde = (mTemps.Seconde + 1) Mod 60
  If mTemps.Seconde = 0 Then
    mTemps.Minute = (mTemps.Minute + 1) Mod 60
    If mTemps.Minute = 0 Then
      mTemps.Heure = (mTemps.Heure + 1) Mod 24
    End If
  End If
  Call Reactualiser_affichage
End Sub

Private Sub Reactualiser_affichage()
  lblAffichage.Caption = Space$(12) & "Standard : " & _
                         mTemps.Mettre_sous_forme_standard() & _
                         "    Universelle: " & _
                         mTemps.Mettre_sous_forme_universelle()
End Sub
```

**N.B. :** Penser à mettre 0 dans les propriétés *Text* des contrôles *TextBox txtHeure, txtMinute, txtSeconde*.




Le programme de test de la classe *CTemps1* permet, *via* le bouton de commande "Entrer" et les contrôles *TextBox*, de fixer des valeurs aux variables *mHeure, mMinute* et *mSeconde*. L'utilisateur peut aussi incrémenter d'une seconde l'heure courante en appuyant sur le bouton de commande "Ajouter 1 seconde". Après chacune de ces opérations, l'heure courante est affichée.

## 12.3 Composition : Objets en tant que variables d'instance d'autres classes

La *composition*, appelée aussi *agrégation*, consiste à former de nouvelles classes incluant en tant que membres des objets d'autres classes existantes. Afin de comprendre cette notion, définissons ce qu'est une *référence* dans Visual Basic : Une *référence* est un nom qui se réfère à un objet, ou non (*i.e.*, *Nothing*). Soit par exemple :

```
Dim t1 As New CTemps1
Dim t2 As CTemps1
```

Du fait du mot-clé *New*, la référence *t1* se réfère à un objet de la classe *CTemps1* ; le programmeur peut appeler ses méthodes et accéder à ses propriétés. Ceci n'est pas le cas de la référence *t2*, qui se réfère à *Nothing*, tant que *New* n'est pas utilisé ; l'instruction

```
Set t2 = New CTemps1
```

permet à *t2* de se référer à une nouvelle instance (objet) de la classe *CTemps1*.

A titre d'exemple, considérons le programme qui suit où sont définies les classes *CEmployé* et *CDate1*. La classe *CEmployé* contient les variables d'instance *mPrénom, mNom, mDate_naissance* et *mDate_embauche*. Les membres *mDate_naissance* et *mDate_embauche* utilisent des objets de la classe *CDate1*, cette dernière contient les variables d'instance *mJour, mMois* et *mAnnée*.
Le programme de test instancie 2 objets *CEmployé*, les initialise et affiche leurs variables d'instance.

**_____ module de la classe CDate1.cls ____ _____**

```
' Définition de la classe CDate1
Option Explicit
```

```
' On définit 3 variables d'instances privées (mJour, mMois, mAnnée)
Private mJour As Integer
Private mMois As Integer
Private mAnnée As Integer

Private Sub Class_Initialize()
  mJour = 1
  mMois = 1
  mAnnée = 1900
End Sub

' On définit 3 propriétés (Jour, Mois, Année)
Public Property Get Jour() As Integer
  Jour = mJour
End Property

Public Property Let Jour(ByVal jr As Integer)
  mJour = Valider_jour(jr)
End Property

Public Property Get Mois() As Integer
  Mois = mMois
End Property

Public Property Let Mois(ByVal ms As Integer)
  mMois = Valider_mois(ms)
End Property

Public Property Get Année() As Integer
  Année = mAnnée
End Property

Public Property Let Année(ByVal an As Integer)
  mAnnée = an   ' Pourrait aussi être validée
End Property

' On définit 2 méthodes publiques (Afficher, Fixer_Date)
Public Function Afficher() As String
  Afficher = mJour & "/" & mMois & "/" & mAnnée
End Function

Public Sub Fixer_date(ByVal jr As Integer, ByVal ms As Integer, _
                      ByVal an As Integer)
  mMois = Valider_mois(ms)
  mAnnée = an
  mJour = Valider_jour(jr)
End Sub

' On définit 2 méthodes privées (Valider_jour, Valider_mois)
Private Function Valider_jour(ByVal jr As Integer) As Integer
  Dim Jours_par_mois()
  Jours_par_mois = Array(0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
```

```
  If jr > 0 And jr <= Jours_par_mois(mMois) Then
    Valider_jour = jr
    Exit Function
  End If
  If mMois = 2 And jr = 29 And (mAnnée Mod 400 = 0 Or mAnnée Mod 4 = 0 And _
                                mAnnée Mod 100 <> 0) Then
    Valider_jour = jr
    Exit Function
  End If
  ' Une entrée non valide est remplacée par la valeur 1
  Valider_jour = 1
End Function

Private Function Valider_mois(ByVal ms As Integer) As Integer
  Valider_mois = IIf((ms > 0 And ms <= 12), ms, 1)
End Function
```

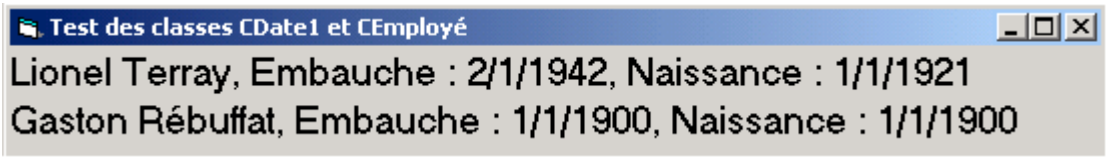
**_____ module de la classe CEmployé.cls _____**

```
' Définition de la classe CEmployé
Option Explicit

' On définit 4 variables d'instances privées (mPrénom, mNom, mDate_naissance,
'                                             mDate_embauche)
Private mPrénom As String
Private mNom As String
Private mDate_naissance As CDate1
Private mDate_embauche As CDate1

Private Sub Class_Initialize()
  Set mDate_naissance = New CDate1
  Set mDate_embauche = New CDate1
End Sub

' On définit 1 méthodes publique (Afficher)
Public Function Afficher() As String
  Afficher = mPrénom & " " & mNom & _
             ", Embauche : " & mDate_embauche.Afficher() & _
             ", Naissance : " & mDate_naissance.Afficher()
End Function

' On définit 4 propriétés (Prénom, Nom, Date_naissance, Date_embauche)
Public Property Get Prénom() As String
  Prénom = mPrénom
End Property

Public Property Let Prénom(ByVal pn As String)
  mPrénom = pn
End Property

Public Property Get Nom() As String
```

```
  Nom = mNom
End Property

Public Property Let Nom(ByVal nm As String)
  mNom = nm
End Property

Public Property Get Date_naissance () As CDate1
  Set Date_naissance = mDate_naissance
End Property

Public Property Set Date_naissance (ByVal ani As CDate1)
  Set mDate_naissance = ani
End Property

Public Property Get Date_embauche() As CDate1
  Set Date_embauche = mDate_embauche
End Property

Public Property Set Date_embauche(ByVal emb As CDate1)
  Set mDate_embauche = emb
End Property

Private Sub Class_Terminate()
  Set mDate_naissance = Nothing        ' Libération mémoire de l'objet
  Set mDate_embauche = Nothing          ' Libération mémoire de l'objet
End Sub
```

**_____ programme de test _____**

```
' Programme de test des classes CDate et CEmployé
Option Explicit

Private Sub Form_Load()
  Dim employé1 As New CEmployé
  Dim employé2 As New CEmployé

  With employé1
    .Prénom = "Lionel"
    .Nom = "Terray"
    Call .Date_naissance.Fixer_date(1, 1, 1921)
    Call .Date_embauche.Fixer_date(2, 17, 1942)
  End With

  employé2.Prénom = "Gaston"
  employé2.Nom = "Rébuffat"

  Print employé1.Afficher()
  Print employé2.Afficher()
End Sub
```




**Relativement à la classe *CDate1* :**

La procédure événementielle *Class_Initialize* fixe une valeur (non nulle) par défaut aux 3 variables d'instance *entières* de la classe, à savoir *mJour* = 1, *mMois* = 1 et *mAnnée* = 1900. Cet événement est déclenché (uniquement) lorsqu'une instance de la classe est créée.
La classe *Cdate1* permet, *via* des procédures *Property Get* et *Property Let*, l'accès aux 3 variables d'instance (*privées*). *Property Let Jour* et *Property Let Mois* appellent chacune une méthode *privée* (respectivement *Valider_jour* et *Valider_mois*) afin d'assurer la cohérence des variables d'instance *mJour* et *mMois* (par choix, la variable d'instance *mAnnée* n'est pas vérifiée).
Deux méthodes sont associées à cette classe. La méthode *Afficher* retourne une chaîne de caractères représentant la date. La méthode *Fixer_date* permet à l'utilisateur de fixer le jour, le mois et l'année (une autre possibilité consisterait à utiliser les *propriétés Jour, Mois* et *Année*). Notons que cette méthode assure aussi la cohérence des variables d'instance *mJour* et *mMois*, à travers les méthodes *privées Valider_Jour* et *Valider_mois*.

**Relativement à la classe *CEmployé* :**

Quatre variables d'instance *privées* sont associées à cette classe : *mPrénom, mNom, mDate_naissance* et *mDate_embauche*. Notons que *mDate_naissance* et *mDate_embauche* se réfèrent à des objets de la classe *CDate1*, il en résulte une relation de *composition*. Chacun de ces objets est construit, *via New*, dans la procédure événementielle *Class_Initialize*.
La classe *CEmployé* permet, *via* des procédures *Property Get, Property Let* et *Property Set*, l'accès aux 4 variables d'instance (*privées*) de la classe. Quand *Property Get*, ou *Property Set*, est appelée pour un objet *CDate1*, on note l'emploi du mot-clé *Set* dans le corps de la procédure *Property*.
La méthode *Afficher*, associée à la classe *CEmployé*, affiche le prénom, le nom, la date d'embauche et la date de naissance de l'employé.
La procédure événementielle *Class_Terminate* permet de "nettoyer" un objet avant sa destruction.

**Relativement au programme de test :**

Deux objets de la classe *CEmployé* sont créés (*employé1* et *employé2*) dans la procédure événementielle *Form_Load*.
Relativement à *employé1*, les *propriétés Prénom* et *Nom* sont fixées, les dates d'embauche et de naissance sont fixées à travers la méthode *Fixer_date.* En effet, l'appel de la procédure *Property Date_naissance Get* retourne un objet de la classe *CDate1*, à partir duquel on peut appeler la méthode *Fixer_date*.
Seules sont fixées les propriétés *Prénom* et *Nom* pour *employé2*. Aussi les dates d'embauche et de naissance sont mises par défaut *via* la procédure événementielle *Class_Initialize* de la clase *CDate1*.

## 12.4 Evénements et classes

Il est possible de déclarer des événements au sein d'une classe. Le fait de provoquer un événement permet à un objet de signaler aux autres objets que quelque chose est survenu. Le mot-clé *RaiseEvent* est utilisé pour provoquer un événement.

Un événement est déclaré, *via* le mot-clé *Event*, de type *Public* dans la partie *déclaration générale* : Les paramètres associés à *Event* permettent l'émission, ou la réception, de données avec le programme.

A titre d'exemple, est définie dans la figure suivante une classe appelée *CEvénement*, laquelle peut déclencher un événement.

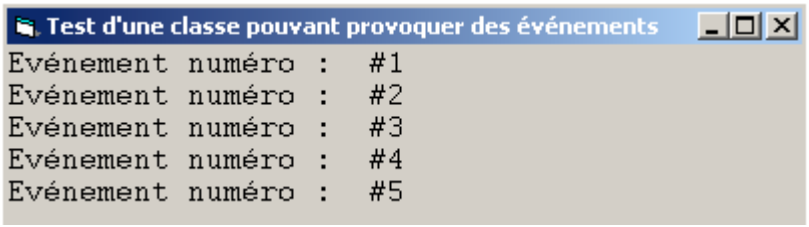
**_____ module de la classe CEvénement.cls _____**

```
' Définition de la classe pouvant déclencher un événement
Option Explicit

' On définit 1 variable d'instance privée (mNombre)
Private mNombre As Long

' On définit 1 événement (NomEvénement)
Public Event NomEvénement(s As String, n As Long)

' On définit 1 méthodes publique (ProvoqueEvénement)
Public Sub ProvoqueEvénement()
  mNombre = mNombre + 1
  ' Déclencher l'événement NomEvénement
  RaiseEvent NomEvénement("Evénement numéro : ", mNombre)
End Sub
```

**_____ programme de test _____**

```
' Programme de test de la classe CEvénement
Option Explicit

Private WithEvents mEvt As CEvénement

' mEvt est 1 objet de la classe CEvénement dans laquelle sont associés des événements
Private Sub Form_Initialize()
  Set mEvt = New CEvénement
End Sub

Private Sub Form_Load()
  Dim x As Integer
  frmTest.Show       ' frmTest : nom de la feuille
  For x = 1 To 5
    ' Appel de la méthode ProvoqueEvénement de l'objet mEvt
    Call mEvt.ProvoqueEvénement
  Next x

End Sub

Private Sub mEvt_NomEvénement(s As String, n As Long)
  Print s & " #" & n
End Sub

Private Sub Form_Terminate()
  Set mEvt = Nothing
```

**End Sub**




Relativement à la classe *CEvénement*, la ligne de code

**Public Event NomEvénement(s As String, n As Long)**

déclare un événement, appelé *NomEvénement*, lequel comprend un paramètre de type *String* et un paramètre de type *Long*. Cette classe définit également une *méthode*, appelée *ProvoqueEvénement*, qui ;

- incrémente de 1 la variable d'instance appelée *mNombre* (de type *Long*),
- puis, déclenche l'événement, appelé *NomEvénement*, *via* la ligne de code

**RaiseEvent NomEvénement("Evénement numéro : ", mNombre)**

La chaîne de caractères "Evénement numéro : " et la valeur de *mNombre* sont les 2 paramètres de *NomEvénement*. Retenons que la méthode appelée *ProvoqueEvénement* doit être exécutée afin de déclencher l'événement appelé *NomEvénement*.

Relativement au programme de test, la ligne de code

**Private WithEvents mEvt As CEvénement**

crée une référence, appelée *mEvt*, à un objet de la classe *CEvénement*. Le mot-clé *WithEvents* indique que des événements sont associés à la classe.
La ligne de code

**Call mEvt.ProvoqueEvénement**

déclenche l'événement appelé *NomEvénement*. La procédure événementielle *mEvt_NomEvénement* permet alors la prise en compte de l'événement *NomEvénement*. Dans le cas présent, on se contente d'afficher les paramètres *s* et *n*, associés à cette procédure.

- **ANNEXE A (Code de compilation)**

Les programmes Visual Basic sont, par défaut, compilés dans un code intermédiaire, appelé *p-code*. Lors de l'exécution, les instructions *p-code* sont interprétées en *code machine* par un *moteur d'exécution* **msvbvm60.dll** (fichier présent dans le répertoire WINDOWS\SYSTEM). Les versions *Professional* et *Enterprise* de Visual Basic permettent la compilation directement en *code machine*, il en résulte, souvent, une augmentation de la vitesse d'exécution des programmes, mais aussi de leurs tailles en mémoire.
Pour passer d'une compilation en *p-code* à une compilation en *code machine*, ouvrer le menu *Projet | Propriétés de projet*, cliquer l'onglet nommé *Compilation* et sélectionner la case à options *Compiler en code natif*.

- **ANNEXE B (Types de projet)**

| Type | Description |
|---|---|
| EXE Standard | Programme Windows ordinaire, fichier exécutable. |
| EXE ActiveX | Fonction d'automatisation fondée sur les modules de classe ; fichier ActiveX exécutable. |
| DLL ActiveX | Fonction d'automatisation fondée sur les modules de classe ; fichier ActiveX non exécutable. |
| Contrôle ActiveX | Contrôle destiné à la boîte à outils, ou à l'utilisation d'Internet (fichier OCX). |
| Assistant Création d'Applications | Assistant de création de fichier EXE Standard. |
| Add-In | Extension de l'environnement Visual Basic. |
| DLL Document ActiveX | Document Visual Basic destiné à l'affichage, par exemple dans Internet Explorer. |
| EXE Document ActiveX | Document ActiveX destiné à l'affichage, par exemple dans Internet Explorer. |

- **ANNEXE C (Éléments clés de Windows : Fenêtres, événements et messages)**

Considérons qu'une fenêtre est simplement une zone rectangulaire dotée de ses propres limites. Sans doute, connaissez-vous déjà plusieurs types de fenêtres : Une fenêtre *Explorateur* dans Windows, une fenêtre de document dans un programme de traitement de texte, ou encore une boîte de dialogue. S'il s'agit là des exemples les plus courants, il existe bien d'autres types de fenêtres. Un bouton de commande est une fenêtre. Les icônes, les zones de texte, les boutons d'option et les barres de menus constituent tous des fenêtres.
Le système d'exploitation Microsoft Windows gère ces nombreuses fenêtres en affectant à chacune d'elles un numéro d'identification unique (descripteur de fenêtre ou *hWnd*). Le système surveille en permanence chacune de ces fenêtres de façon à déceler le moindre événement ou signe d'activité. Les événements peuvent être engendrés par des actions de l'utilisateur (notamment, lorsque celui-ci clique un bouton de la souris ou appuie sur une touche), par un contrôle programmé, voire même par des actions d'une autre fenêtre.
Chaque fois qu'un événement survient, un message est envoyé au système d'exploitation. Celui-ci traite le message et le diffuse aux autres fenêtres. Chacune d'elles peut alors exécuter l'action appropriée de la manière prévue pour ce type de message (notamment, se redessiner si elle n'est plus recouverte par une autre fenêtre).

Comme vous pouvez l'imaginer, il n'est pas simple de faire face à toutes les combinaisons possibles de fenêtres, d'événements et de messages. Heureusement, Visual Basic vous épargne la gestion de tous les messages de bas niveau. La plupart de ceux-ci sont automatiquement gérés par Visual Basic tandis que d'autres sont mis à disposition sous forme de procédures *Event* afin de vous faciliter la tâche. Vous pouvez aussi écrire rapidement des applications puissantes sans vous préoccuper de détails inutiles.

- **ANNEXE D (Description du modèle événementiel)**

Dans les applications traditionnelles ou " procédurales ", c'est l'application elle-même, et non un événement, qui contrôle les parties du code qui sont exécutées, ainsi que leur ordre d'exécution. Celle-ci commence à la première ligne de code et suit un chemin défini dans l'application, appelant les procédures au fur et à mesure des besoins.
Dans une application événementielle, le code ne suit pas un chemin prédéterminé. Différentes sections du code sont exécutées en réaction aux événements. Ceux-ci peuvent être déclenchés par des actions de l'utilisateur, par des messages provenant du système ou d'autres applications, voire même par l'application proprement dite. L'ordre de ces événements détermine l'ordre d'exécution du code. Le chemin parcouru dans le code de l'application est donc différent à chaque exécution du programme.
Comme il est impossible de prévoir l'ordre des événements, votre code doit émettre certaines hypothèses quant à " l'état du système " au moment de son exécution. Lorsque vous élaborez des hypothèses (par exemple quand vous supposez qu'un champ de saisie doit contenir une valeur avant l'exécution de la procédure chargée de traiter cette valeur), vous devez structurer votre application de telle sorte que cette hypothèse soit toujours vérifiée (par exemple en désactivant le bouton de commande qui démarre la procédure aussi longtemps que le champ de saisie ne contient pas une valeur).
Votre code peut également déclencher des événements pendant l'exécution. Par exemple, la modification par programmation du contenu d'une zone de texte déclenche l'événement *Change* qui lui est associé et donc l'exécution du code éventuellement contenu dans cet événement. Si vous avez estimé que cet événement ne serait déclenché que par dialogue avec l'utilisateur, vous risquez d'être confronté à des résultats inattendus. C'est pour cette raison qu'il est important de bien comprendre le modèle événementiel et de le garder toujours à l'esprit tout au long de la phase de création d'une application.

- **ANNEXE E (Description (notamment) des paramètres de *MoveComplete*)**

**Méthodes WillMove et MoveComplete**
La méthode **WillMove** est appelée *avant que* l'opération en attente change la position actuelle dans le **Jeu d'enregistrements**. La méthode **MoveComplete** est appelée *après* modification de la position actuelle dans le **Recordset**.

**Syntaxe**
**WillMove** ***adReason, adStatus, pRecordset***
**MoveComplete** ***adReason, pError, adStatus, pRecordset***

**Paramètres**
***adReason*** Une valeur **EventReasonEnum**. Spécifie le motif de cet événement. Il peut prendre comme valeur **adRsnMoveFirst**, **adRsnMoveLast**, **adRsnMoveNext**, **adRsnMovePrevious**, **adRsnMove** ou **adRsnRequery**.
***pError*** Un objet **Error**. Il décrit l'erreur survenue pour autant que la valeur de ***adStatus*** soit **adStatusErrorsOccurred** ; dans le cas contraire, il n'est pas défini.
**adStatus** Une valeur d'état ***EventStatusEnum***.

Lors de l'appel de **WillMove**, ce paramètre est défini à **adStatusOK** si l'opération qui a provoqué l'événement a réussi. Il est défini à **adStatusCantDeny** si cette méthode ne peut demander l'annulation de l'opération en attente.
Lors de l'appel de **MoveComplete**, ce paramètre est défini à **adStatusOK** si l'opération qui a provoqué l'événement a réussi, ou à **adStatusErrorsOccurred** si l'opération a échoué.
Avant le renvoi d'une valeur de la méthode **WillMove**, définisser ce paramètre à **adStatusCancel** pour demander l'annulation de l'opération en attente. Avant le renvoi d'une valeur de la méthode **MoveComplete**, définisser ce paramètre à **adStatusUnwantedEvent** pour empêcher des notifications ultérieures.
***pRecordset*** Un objet **Recordset**. Le **Recordset** pour lequel cet événement s'est produit.

**Remarque :** Un événement **WillMove**, ou **MoveComplete**, peut survenir à la suite des opérations **Recordset** suivantes : **Open**, **Move**, **MoveFirst**, **MoveLast**, **MoveNext**, **MovePrevious**, **Bookmark**, **AddNew**, **Delete**, **Requery** et **Resync**.

- **ANNEXE F (Les types de variables *objet*)**

Les variables *objet*, comme toute autre variable, peuvent se déclarer explicitement en spécifiant le type.
Par exemple, pour déclarer une variable *objet*, nommée *txtContrôle*, comme zone de texte, on écrit :

**Dim txtContrôle As TextBox**

Il correspond un type pour chaque contrôle (*objet*) standard, à savoir :

| | | | |
|---|---|---|---|
| *CheckBox* | *ComboBox* | *CommandButton* | *Data* |
| *DirListBox* | *DriveListBox* | *FileListBox* | *Form* |
| *Frame* | *Grid* | *HscrollBar* | *Image* |
| *Label* | *Line* | *ListBox* | *MDIForm* |
| *Menu* | *OptionButton* | *OLE* | *PictureBox* |
| *Shape* | *TextBox* | *Timer* | *VscrollBar* |

Des types existent également pour les contrôles non *intrinsèques.*

En fait, Visual Basic rappelle le type d'un contrôle dans la zone de liste modifiable en haut de la fenêtre de propriétés.

- **ANNEXE G (Traitement de fichiers avec d'anciennes instructions et fonctions d'E/S de fichiers)**

Depuis la première version de Visual Basic, les fichiers ont été traités à l'aide de l'instruction *Open* et d'autres instructions et fonctions associées (dont la liste est présentée ci-dessous). Ces mécanismes deviendront éventuellement périmés en faveur du modèle d'objet *FSO* (*File System Object*), mais ils sont entièrement pris en charge dans Visual Basic 6.0.
Bien que vous puissiez créer votre application de sorte qu'elle utilise des fichiers de base de données, il n'est pas nécessaire de prévoir l'accès direct aux fichiers dans votre application. Le contrôle *Data* et les contrôles dépendants vous permettent de lire et d'écrire des données dans une base de données, ce qui est beaucoup plus simple que de recourir aux techniques d'accès direct aux fichiers.
Toutefois, il se peut que vous deviez parfois lire et écrire dans des fichiers autres que des bases de données. Vous devez alors traiter directement les fichiers en créant, manipulant et stockant du texte et d'autres données.

**Types d'accès aux fichiers**

Pris isolément, un *fichier* n'est autre qu'une suite d'octets connexes enregistrés sur un disque. Lorsque votre application accède à un fichier, elle doit imaginer ce que représente ces octets (des caractères, des enregistrements de données, des entiers, des chaînes, etc.).
Le type d'accès aux fichiers varie selon le type de données contenues dans le fichier. Dans Visual Basic, il existe trois types d'accès aux fichiers :

- *Séquentiel* : Pour la lecture et l'écriture de fichiers texte dans des blocs continus.
- *Aléatoire* : Pour la lecture et l'écriture de fichiers texte ou binaires structurés sous la forme d'enregistrements de longueur fixe.
- *Binaire* : Pour la lecture et l'enregistrement de fichiers possédant une structure arbitraire.

L'*accès séquentiel* a été conçu pour les fichiers texte sans mise en forme. Chaque caractère du fichier est censé représenter un caractère de texte ou une séquence de mise en forme du texte, par exemple un caractère de nouvelle ligne (NL). Les données sont stockées sous forme de caractères *ANSI*.

Un fichier ouvert dans le cadre d'un *accès aléatoire* est censé être constitué d'un ensemble d'*enregistrements* de même longueur. Grâce aux types définis par l'utilisateur, vous pouvez créer des enregistrements constitués de nombreux champs pouvant même posséder chacun un type de données différent. Les données sont stockées sous forme d'informations binaires.

L'*accès binaire* vous permet de stocker des données dans un ordre quelconque au sein de fichiers. Ce mode d'accès est semblable à l'accès aléatoire, mais s'en différencie par le fait qu'il n'y a aucune estimation du type de données ou de la longueur des enregistrements. Toutefois, vous devez connaître avec précision la manière dont les données ont été écrites dans le fichier si vous souhaitez les extraire correctement.

**Pour plus d'informations** sur les types d'accès aux fichiers, reportez-vous dans *MSDN* aux sections " *Utilisation de l'accès séquentiel aux fichiers* ", " *Utilisation de l'accès aléatoire aux fichiers* " et " *Utilisation de l'accès binaire aux fichiers* ".

**Fonctions et instructions d'accès aux fichiers**

Les fonctions suivantes sont utilisées avec les trois types d'accès aux fichiers.

| | | |
|---|---|---|
| *Dir* | *FileLen* | *LOF* |
| *EOF* | *FreeFile* | *Seek* |
| *FileCopy* | *GetAttr* | *SetAttr* |
| *FileDateTime* | *Loc* | |

Le tableau suivant énumère toutes les instructions et fonctions d'accès aux fichiers disponibles pour chacun des trois types d'accès direct aux fichiers.

| Instruction et fonction | Accès séquentiel | Accès aléatoire | Accès binaire |
|---|---|---|---|
| *Close* | X | X | X |
| *Get* | | X | X |
| *Input( )* | X | | X |
| *Input #* | X | | |
| *Line Input #* | X | | |
| *Open* | X | X | X |
| *Print #* | X | | |

| | | | |
|---|---|---|---|
| *Put* | | X | X |
| *Type...End Type* | | X | |
| *Write #* | X | | |