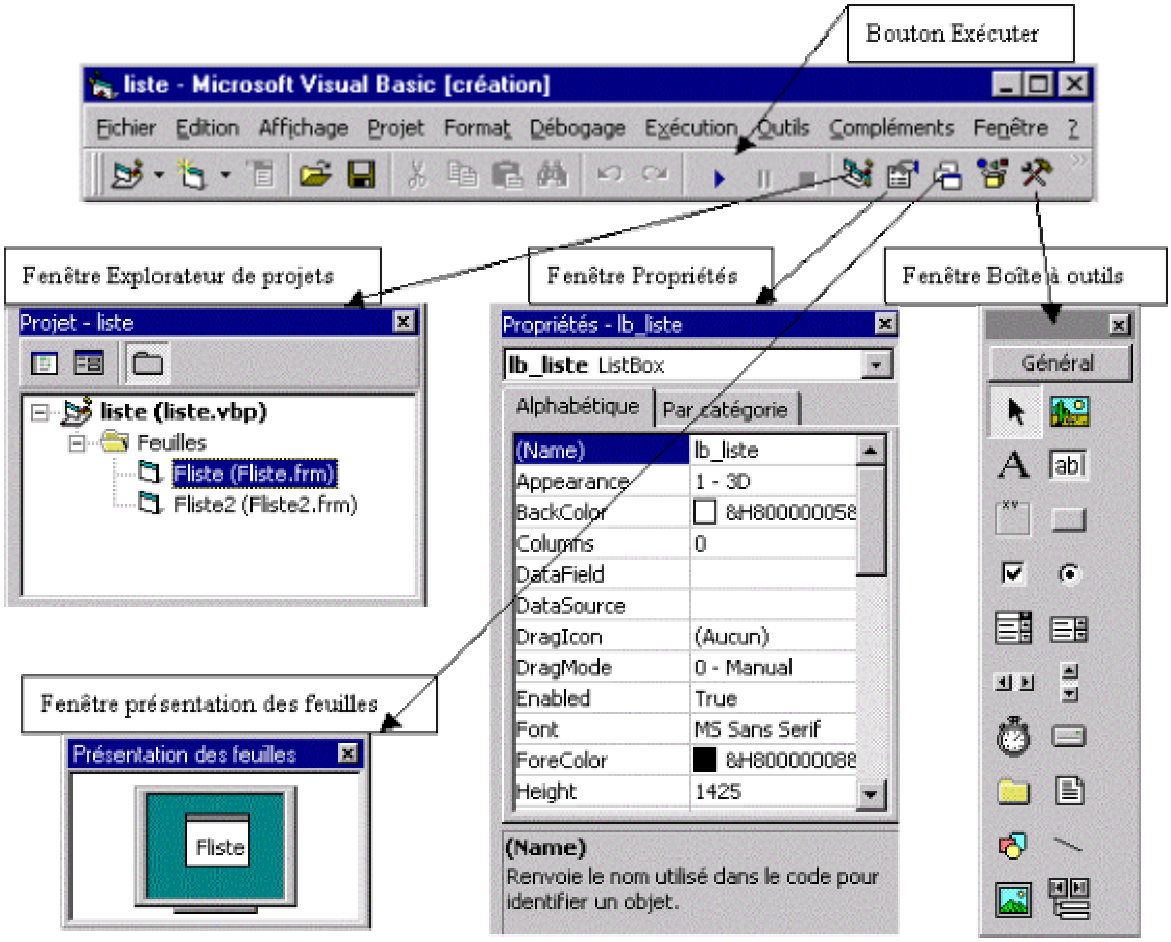
Cours 1 : Présentation de l'interface de VB

Après avoir lancé Visual Basic, vous devez voir apparaître à l'écran une boîte de dialogue vous demandant quel genre d'application voulez-vous créer? Choisissez "*EXE Standard*" si vous voulez créer un programme. Quant aux autres types d'applications, laissez-les de côté pour le moment. Vous verrez alors apparaître en face de vous une fenêtre fragmentée en plusieurs petites fenêtres. Au premier abord, cela doit certainement vous effrayer de voir une multitude de mini fenêtres. Ne vous inquiétez pas, votre comportement est tout à fait normal. N'importe quel NEWBIE aurait eu ce comportement. Moi-même, je l'ai été et j'ai pensé au début que Visual Basic était un logiciel trop compliqué pour moi et j'ai voulu laisser tomber ce fabuleux logiciel. Mais ce qui m'a encouragé à continuer à utiliser Visual Basic, c'est la possibilité de créer, créer un programme même si ce n'est qu'une simple petite application car je sais qu'après je pourrai concevoir des programmes plus élaborés au fur à mesure que j'acquiers de nouvelles connaissances.

Ca y est? Vous êtes décidé à continuer à apprendre ce langage informatique et faire partie de la communauté des programmeurs? Alors, allons-y! Commençons par décrire toutes ces multitudes de fenêtres.

- **La partie supérieure de la fenêtre est formé du *système de menus* et d'une *barre d'outils* tout comme d'autres logiciels(Office, Lotus,...).**
- **La partie de gauche est constitué de la *boîte à outils*. (Attention, ne confondez pas barre d'outils avec boîte à outils. La barre d'outils est une version simplifiée du système de menus. Elle permet d'accéder plus rapidement aux sous-commandes du système de menus.) Quant à la boîte à outils, elle permet de sélectionner les contrôles (ou si voulez "objets"), puis ensuite de les placer sur la partie centrale de la fenêtre. *Pour placer un contrôle, sélectionnez d'abord un contrôle dans la boîte à outils, puis, tracez une zone rectangulaire sur la feuille qui se trouve au milieu.* Si vous voulez ajouter de nouveaux contrôles dans la boîte à outils, appuyez sur le bouton droit de votre souris dans la boîte à outils pour faire apparaître le menu contextuel. Appuyez sur la sous-commande "Composants" pour faire apparaître la liste de tous les contrôles mis à votre disposition.**
- **Cette feuille située au centre, n'est autre que la future *interface graphique* de votre application (appelé aussi *"interface utilisateur"*).**
- **Enfin, la partie de droite est constituée de 3 boîtes de dialogue :**
    1. **La 1ère est la boîte de dialogue "*Projet*" qui donne la liste de toutes les feuilles qui constituent votre future application.**
    2. **La 2ème est la boîte de dialogue "*Propriétés*". Elle donne accès aux propriétés du contrôle sélectionné.**
    3. **Enfin, la 3ème est la boîte dialogue "*Présentation des feuilles*". Elle permet de modifier la position de la feuille de travail actuellement sélectionnée.**



Cours 2 : Structure des procédures

Une application est constituée essentiellement de l'*interface utilisateur*, formée elle-même de plusieurs *contrôles*, et de *procédures* qui génère des actions. A présent, analysons la *structure d'une procédure*.
La syntaxe d'écriture d'une procédure est la suivante :

```
[Public / Private] [Static] Sub Nom_proc (arguments)
   [Déclarations]
   [Instructions]
   [exit Sub]
   [Instructions]
   End Sub
```

Le mot Sub peut ou non être précédé des options suivantes :

1. **Les mots *Public* ou *private* définissent les limites de la procédure.**

- **Avec *Public*, la procédure peut être appelée depuis n'importe quelle instruction de l'application.**
- **Avec *private*, la procédure ne peut être appelée qu'à l'intérieur de la même feuille ou du même module.**

2. **Vous devez déclarer en début de procédures vos *variables* et *constantes* si vous ne l'avez pas fait dans la partie Général de la liste déroulante *Objet* qui se trouve en haut, à gauche de l'éditeur de code. Remarquez que si vous déclarez vos variables à l'intérieur de la procédure, sa portée sera limitée qu'à cette procédure;**
3. ***Static* signifie que toutes les variables locales déclarées dans la procédure doivent conserver leur valeur entre 2 appels.**
4. ***Exit Sub* permet de quitter la procédure avant la fin normale de celle-ci;**
5. ***End Sub* ferme la procédure.**

Pour cela, prenons un simple exemple :

```
Private Sub Form_Load()
Dim DateNaissance
Dim Message, Titre As String
Message = "Date de naissance ?"
Titre = "Entrez votre date de naissance : "
DateNaissance = InputBox(Message, Titre)
If DateNaissance <> "" Then
DateNaissance = Format(DateNaissance, "Long Date")
MsgBox DateNaissance, vbOKOnly + vbInformation, "Vous êtes né(e)
le"
End
Else
While DateNaissance = ""
MsgBox "Entrez une date", vbOKOnly + vbExclamation, "Attention!"
DateNaissance = InputBox(Message, Titre)
Wend
DateNaissance = Format(DateNaissance, "Long Date")
MsgBox DateNaissance, vbOKOnly + vbInformation, "Vous êtes né(e)
le"
End
End If
End Sub
```

Le but de ce programme est certes inutile, vous direz-vous, puisqu'il consiste juste à demander la date de naissance à quelqu'un et de l'afficher par la suite. Mais les instructions de cette procédures renferment une partie des bases de la programmation en VB6.
Ce n'est pas important si vous n'avez pas compris toutes les instructions de cette procédure. Contentez-vous pour l'instant de lire cette procédure et observez simplement de quoi peut être composée une procédure.
A présent, passons à une analyse simplifiée de cette procédure :

- **La 1ère ligne contient des infos sur le nom de la feuille principale (Form1) du projet, du type d'événement qui permet de lancer cette procédure. Ici L'événement en question est Load, ce qui veut dire que cette procédure sera exécutée au chargement de la feuille "Form1".**
- **dans les 2èmes et 3èmes lignes, sont définis des variables limitées à cette procédure uniquement.**
- **Les lignes suivantes sont formées d'instructions servant à définir le comportement de la feuille dès son chargement. (Nous verrons de plus près ces instructions dans les cours suivants.)**
- **La dernière ligne End Sub clôt la procédure.**

Cours 3 : Les variables et les constantes

---

1. Définition des variables

Les variables sont des données ou des valeurs qui peuvent changer à l'intérieur d'une application. C'est pourquoi, il est fort utile de les *nommer par un nom*, de déclarer quel genre de variables est-ce (nombre entier, nombre décimal, lettres...) et leur *affecter*, lorsque cela est nécessaire une valeur.
La longueur maximale du nom d'une variable est de 255 caractères. Ceux-ci peuvent être des chiffres, des lettres ou autres symboles. Notez que ce nom doit obligatoirement commencer par une lettre.
En effet, Visual basic classe les variables en fonction de la valeur affectée à la variable. Ainsi, une variable déclarée comme du type numérique ne peut pas recevoir une valeur chaîne de caractère, ainsi qu'à l'inverse.
Notez que si vous ne déclarez pas une variable, Visual Basic se chargera d'affecter par défaut un type de variable (*Variant*) à celle-ci. Une variable du type *Variant* peut aussi bien recevoir des données numériques que des chaînes de caractères. tout dépend de ce que vous avez affecté à cette variable.

---

2. Type de variables

A présent, observons de plus près les différents types de variables :

Type de données	Mot clé	Occupe	Limite de valeurs
Octet	Byte	1 octet	0 à 255
Logique	Boolean	2 octets	True(-1) ou False(0)
Entier	Integer	2 octets	-32 768 à 32767
Entier long	Long	4 octets	-2 147 483 648 à 2 147 483 647
Décimal simple	Single	4 octets	Nombre réel avec 7 chiffres après la virgule
Décimal double	Double	8 octets	Nombre réel avec 15 chiffres après la virgule
Monétaire	Currency	8 octets	Nombre réel avec 15 chiffres avant la virgule et 4 chiffres après la virgule
Date	Date	8 octets	1er janvier 100 au 31 décembre 9999
Objet	Object	4 octets	Toute référence à des types Object
Chaîne de caractères	String	10 octets + longueur de chaîne	Chaîne de caractère dont la longueur ne doit pas excéder 2^31 caractères
Variant (avec chiffres)	Variant	16 octets	toute valeur numérique jusqu'à l'étendue d'un double
Variant (avec lettres)	Variant	22 octets+longueur de chaîne	Même étendue que pour un String de longueur variable
Défini par l'utilisateur	Type	-	L'étendue de chaque élément est la même que son type de données

Notez que les types de variables les plus utilisées sont : String, Integer, Long, Single, Double et Currency.

---

3. Déclaration de variables

Pour utiliser des variables, il est normalement obligatoire de les prédéfinir, soit dans la *section Déclarations* de la liste déroulante Objet, soit en début de procédure ou de fonction. Un programme où les variables sont bien déclarées rend un programme plus

facile à comprendre, à lire et surtout à corriger en cas d'erreurs. Certes, il n'est pas obligatoire de les déclarer mais faites-le quand même, c'est un conseil. Si vous êtes prêt à déclarer une variable que vous voulez utilisez mais que vous êtes un étourdi, alors, utilisez simplement l'instruction *Option Explicit* (à placer dans la section Déclarations de la liste déroulante Objet) qui vous oblige à chaque fois à déclarer toutes vos variables avant de pouvoir exécuter l'application.

3.1 Déclaration explicite d'une variable

Pour déclarer une variable, on utilise l'instruction *Dim* suivi du nom de la variable puis du type de la variable. Reprenons l'exemple du cours 2 :

```
Dim DateNaissance
Dim Message, Titre As String
```

- **Remarquez que la 1ère déclaration ne contient pas d'information sur le type de variable. Si vous déclarez une variable sans donner d'information sur le type de variable que c'est, alors, cette variable (DateNaissance) aura par défaut une variable du type Variant. Si vous avez bien lu l'exemple précédent, vous aurez compris qu'il s'agit ici d'une variable de type Variant (avec chiffres) qui lui est affectée par défaut.**
- **La 2ème déclaration est par contre explicite. Vous pouvez aussi mettre 2 variables sur une même ligne à condition que le type de variable est le même pour les 2 et en les séparant par une virgule.**

3.2 Déclaration implicite d'une variable

Il existe une autre méthode de déclarer des variables. Pour cela, il suffit d'ajouter juste avant la variable, un symbole spécifique.
Voici la liste des symboles avec le type de variable auxquelles il se rapportent :

Symbole	Type de variable
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Dans l'exemple ci-après, les deux premières instructions sont équivalentes à la troisième :

```
Dim V1 as Integer
V1 = 25
```

```
V1% = 25
```

---

4. Portée des variables

En général, toute variable déclarée a une portée limitée. **Cette même variable a une valeur nulle au départ. De plus, elle ne s'applique pas forcément à toutes les procédures d'une application. Tout dépend de 2 éléments : *la manière de déclarer* et *l'emplacement* de la variable.**

- **dans une procédure, si vous déclarez une variable à l'aide de l'instruction *Dim*, sa portée se trouve limitée seulement à cette procédure. On dit que la variable est *locale*. Elle est donc initialisée à chaque appel de la procédure et détruite lorsque celle-ci se termine (à moins que vous remplacez le mot *Dim* par *Static*). Elle n'est pas accessible en dehors de la procédure. Vous pouvez remplacer l'instruction *Dim* par *Private*, les deux termes étant équivalentes s'ils sont placés à l'intérieur d'une procédure.**
- **Si vous déclarez une variable dans la section Général/Déclarations d'une feuille ou d'un module en utilisant l'instruction *Dim* (ou *Private*), la variable est dite *locale au module*. Cette variable est disponible pour toutes les procédures de la feuille ou du module, mais pas pour les autres feuilles du projet.**
- **Enfin, si vous déclarez une variable dans la section Général/Déclarations d'un module (et non d'une feuille) en utilisant l'instruction *Public* au lieu de *Dim*, elle devient accessible par toutes les feuilles et tous les modules de l'application. On dit qu'elle est globale.**

---

5. Les constantes

Contrairement aux variables dont les valeurs diffèrent souvent, les constantes ont des valeurs fixes. Mais tout comme les variables, on affecte aux constantes, un nom et une valeur qui est elle, fixe. De plus, les constantes se définissent de la même façon que les variables. Tout dépend donc de l'endroit où est défini la constante. Le mot *Const* peut être précédé de l'option *Public* pour que toutes les feuilles et modules de l'application puissent y accéder.
Attention cependant à ne pas affecter à une constante des noms identiques aux

constantes prédéfinies (VbSystemModal, VbOkOnly, VbArrow...) dans Visual Basic ! Prenons l'exemple du taux de TVA :

```
Const TVA = 20.6
TotalTTC=PrixHorsTaxe x (1 + (TVA / 100))
```

Cours 4 : Les tableaux

1ère méthode | 2ème méthode | 3ème méthode | 4ème méthode

1. Première méthode pour définir un tableau

Les tableaux de valeurs sont utiles lorsque l'on manipule des données en tout genre. Ainsi, dans un carnet d'adresses, vous pouvez stocker toutes les adresses dans un tableau structuré.
Par exemple :

```
Dim Adresses (1 to 50 ) As String
Adresses(1) = "12, avenue de la Fayette, Paris"
........
Adresses(50) = "4, rue de la Paix, Paris"
```

Comme vous pouvez le constatez, la définition d'un tableau ressemble beaucoup à celle d'une variable à la seule différence que vous devez donner une limite aux nombres de valeurs contenues dans un tableau. Pour appeler une des variables du tableau, il faut faire précéder de la variable, un numéro qui permet d'identifier la variable spécifique recherchée parmi toutes celles du tableau. ce tableau est à une dimension, Voici un exemple à 2 dimensions:

```
Dim Adresses (1 to 50, 1 to 50) As String
Adresses(1,0) = "Vincent"
Adresses(1,1) = "12, Rue de la paix, Paris"
.....
Adresses(50,0) = "Philip"
Adresses(50,50) = "7, Boulevard de la Villette"
```

Notez qu'avec les fonctions LBound et UBound, vous pouvez obtenir les limites inférieures et supérieures d'un tableau.

2. Deuxième méthode pour définir un tableau

Il existe une autre façon de définir un tableau :

```
Dim Adresses(50) As String
Adresses(0) = "12, avenue de la Fayette, Paris"
........
Adresses(49) = "4, rue de la Paix, Paris"
```

Avec cette méthode, les limites du tableau ne sont pas définies. Notez que le nombre d'adresses est de 50 mais la numérotation va de 0 à 49 et non de 1 à 50. Il existe cependant un moyen pour que la numérotation commence à 1 : c'est en utilisant l'instruction *Option Base* (par exemple: Option Base = 1) qu'il faut placer dans la section Déclarations.

3. Troisième méthode pour définir un tableau

Pour construire un tableau, on peut aussi utiliser la fonction *Array* : Exemple :

```
Dim Semaine, Jour As String
Semaine = Array("lundi", "mardi", "mercredi", "jeudi",
"vendredi", "samedi", "dimanche")
Jour = (3) 'retourne jeudi
```

4. Quatrième méthode pour définir un tableau

Une autre méthode consiste en la création de tableaux dynamiques. Pour comprendre cette méthode, prenons un exemple :

```
Dim Adresse() As String
Dim NbAdresses As Integer
'Nouvelle adresse
NbAdresses = nbAdresses + 1
```

```
Redim Adresse (NbAdresses)
Adresse (NbAdresses) = "75, Rue Lecourbe, Paris"
```

Ainsi, à chaque fois que l'on lance cette procédure, le nombre total d'adresses (NbAdresses) est incrémenté (augmenté) de 1 et la nouvelle adresse est placé à la fin du tableau. en effet, *Redim* a pour effet de reformater le tableau, et donc, tout ceux qui était dans le tableau est aussi effacé sauf la dernière valeur entrée. Cependant, il existe un moyen de conserver les données déjà présentes dans le tableau. Pour cela, on fait appel à l'instruction *Preserve* :

```
Redim Preserve Adresse (NbAdresses)
```

Enfin, pour terminer, il est possible d'effacer toutes les données d'un tableau. Pour cela, on utilise l'instruction *Erase* suivi du nom du tableau.

Cours 5 : Structure des conditions (tests, boucles)

La structure des tests | La structure des boucles

1. Structure des tests

1.1 La structure : If...Then...Else...end If

Voici un exemple de structure simple avec l'instruction *If* :

```
If condition Then
Instructions 1
else
Instructions 2
End if
```

Interprétation : Si la condition est vérifiée alors les instructions 1 sont exécutées sinon les instructions 2 sont exécutées à la place. (Notez que l'on peut utiliser des opérateurs logiques *And* (et) et *Or* (ou) pour que plusieurs conditions doivent d'abord être vérifiées avant de pouvoir exécuter les instructions suivantes.). Le mot *Else* et les instructions qui suivent ne sont pas obligatoire.

Voici un autre exemple de structure avec *If* mais un peu plus complexe:

```
If Condition 1 Then
Instruction 1
ElseIf Condition 2 Then
Instruction 2
Else
Instructions X
End If
```

Interprétation : Si la condition 1 est vérifiée alors les instructions 1 sont exécutées. Sinon si la condition 2 est vérifiée alors les instructions 2 sont exécutées. Sinon, si aucune de ces deux conditions ne sont vérifiées alors les instructions X sont exécutées.

Il existe une autre structure simplifiée de *If* mais qui moins utilisée:

```
Variable = IIf (Condition, instructions 1, instructions 2)
```

Interprétation : Si la condition est vérifiée alors les instructions 1 sont exécutées sinon les instructions 2 sont exécutées à la place.
L'instruction *IIf*(Immediate If) est utile lorsque qu'il n'y a que une ou deux instructions en fonction d'une condition. Par ailleurs, la valeur retournée par la condition est affectée à une variable.

1.2 La structure Select Case...End Select

Lorsque l'on doit effectuer toute une série de tests, il est préférable d'utiliser la structure Select Case...End Select au lieu de If...Then...End if.
Ainsi, les deux exemples suivants sont équivalents :

```
Select Case Semaine
Case 1
Jour = "Lundi"
Case 2
Jour = "Mardi"
Case 3
Jour = "Mercredi"
Case 4
Jour = "Jeudi"
Case 5
Jour = "Vendredi"
Case 6
Jour = "Samedi"
```

```
Case 7
Jour = "Dimanche"
Else
MsgBox "erreur", vbOKOnly , "Note"
End Select

If Semaine = 1 Then
Jour = "Lundi"
ElseIf Semaine = 2 Then
Jour = "Mardi"
ElseIf Semaine = 3 Then
Jour = "Mercredi"
ElseIf Semaine = 4 Then
Jour = "Jeudi"
ElseIf Semaine = 5 Then
Jour = "Vendredi"
ElseIf Semaine = 6 Then
Jour = "Samedi"
ElseIf Semaine = 7 Then
Jour = "Dimanche"
Else
MsgBox "erreur", vbOKOnly , "Note"
End If
```

Interprétation : Si la variable Semaine vaut 1, alors la variable Jour reçoit la valeur Lundi. Si la variable Semaine vaut 1, alors la variable Jour reçoit la valeur Mardi. (Etc...). Si la variable Semaine ne reçoit aucune valeur comprise entre 1 et 7, alors un message indiquant une erreur est affiché.

---

2. Structure des boucles

2.1 La structure For..Next

La structure For...Next est utile lorsqu'on doit répéter plusieurs fois la même instruction. Exemple :

```
For J = 1 To 10
T(J) = J
Next
```

Interprétation : Dans cette boucle, l'instruction "T(J)=J" est répétée 10 fois c'est à dire jusqu'à ce que J soit égale à 10. A chaque passage, l'instruction "T(J)=J" affecte à T(J) les valeurs 1 à 10.

Pour que l'instruction soit exécutée une fois sur deux, vous pouvez utiliser l'instruction *Step* :

```
For J = 1 To 10 Step 2
T(J) = J
Next
```

Interprétation : Dans cette boucle, l'instruction "T(J)=J" affecte à T(J) les valeurs 1,3,5,7,9.

2.2 La structure For Each...Next

Cette structure est moins utilisée que la précédente. Elle sert surtout à exécuter des instructions portant sur un tableau.
Exemple :

```
Dim Semaine(1 to 7) As String
Dim Jour As Variant
Semaine(1) = "Lundi"
'Etc...
For Each Jour In Semaine()
Combo1.AddItem Jour
Next
```

Interprétation : Pour chaque Jour de la Semaine, on ajoute à la liste combinée, la valeur de la variable Jour jusqu'à ce que l'indice de la semaine soit égale à 7.

2.3 Les structures Do...Loop et While...Wend

Avec ces boucles, le nombre de fois où sont répétées les instructions est indéfini.
Les deux exemples qui suivent sont équivalents :

```
i = 1
Do
T(i) = i
i = i + 1
Loop Until i > 10
```

```
i = 1
While i < 10
T(i) = i
i = i + 1
Wend
```

Interprétation : La variable i est initialisée à 1 au départ. Les instructions qui suivent sont exécutées jusqu'à ce que i soit supérieure à 10. A chaque passage, on affecte à la variable T d'indice i la valeur de i.

Il existe d'autres variantes de ces boucles et qui sont chacune équivalentes :

```
Do
Instructions
Loop While Condition
```

```
Do Until Condition
Instructions
Loop
```

```
Do While Condition
Instructions
Loop
```

Cours 6 : Les contrôles standards

 **Ne vous fiez pas aux apparences. le *Pointeur* n'est pas un contrôle. Il sert juste à déplacer et à redimensionner un contrôle placé sur une feuille.**

 **Le contrôle *PictureBox* sert à afficher une image en mode point, une icône ou un métafichier. Il sert aussi à regrouper des boutons radio. Remarquons que ce contrôle est le seul qui puisse être placé dans une feuille MDI. En effet, essayez de placer d'autres contrôles sur une feuilles MDI, vous verrez que ça ne marche pas!**

 **Le contrôle *Label* sert à placer du texte qui ne peut être modifié ou effacé lors de l'exécution de l'application.**

 **Le contrôle *Textbox* sert à placer du texte qui peut être ou non modifié par l'utilisateur lors de l'exécution de l'application.**

 **Le contrôle *Frame* sert à regrouper plusieurs contrôles(checkbox, optionbutton,...)**

dans un cadre avec un titre que vous pouvez modifier dans ses propriétés.

 **Le contrôle *CommandButton* sert à afficher un bouton de commande qui lorsque l'on clique dessus, la portion de code écrite dans sa procédure est exécutée.**

 **Le contrôle *Checkbox* sert à placer une case à cocher. En plaçant plusieurs cases à cocher dans une application, l'utilisateur aura le choix entre plusieurs options. Selon les options sélectionnées, une portion de code sera exécutée.**

 **Le contrôle *OptionButton* s'utilise comme un contrôle CheckBox. Contrairement aux cases à cocher, une seule option peut être sélectionnée parmi un groupe d'options composées de boutons radio.**

 **Le contrôle *ComboBox* laisse à l'utilisateur le choix de sélectionner un élément parmi d'autres dans une liste.**

 **Le contrôle *ListBox* s'utilise comme le contrôle ComboBox sauf que dans la liste d'option, plusieurs options sont affichées simultanément.**

 **Les *barres de défilement* permettent de lire la suite du contenu d'une fenêtre trop large ou trop longue pour être affichée en une seule fois.**

 **Le contrôle *Timer* permet de générer un événement à intervalle régulier, par exemple dans une horloge où toutes les minutes, l'aiguille des minutes bouge. Notons que le contrôle Timer n'apparaît pas pendant l'exécution d'une application.**

 **Le contrôle *DirListBox* permet à l'utilisateur de choisir un des lecteurs de disques de l'ordinateur. Les noms des lecteurs sont affichés dans une liste non modifiable.**

 **Le contrôle *DirListBox* permet d'afficher la liste de tous les répertoires d'une lecteur de disque.**

 **Le contrôle *FileListBox* permet d'afficher la liste de tous les fichiers d'un répertoire du disque.**

 **Le contrôle *Shape* permet de dessiner des figures dans une feuille (rectangle, carré, cercle). Le type de forme est choisi dans la propriété Shape du contrôle.**

 **Le contrôle *Line* permet de tracer des lignes dans une feuille. La propriété**

BorderStyle permet de choisir le type de ligne que vous désirez : continu, invisible, pointillé...

 **Le contrôle *Image* permet tout comme le contrôle PictureBox d'insérer des images en mode point, icône, ou métafichier. Cependant, il requiert moins de ressource et donc diminue la taille de votre application.**

 **Le contrôle *Data* permet d'accéder aux données provenant de bases de données.**

 **Le contrôle *OLE* permet d'incorporer ou de lier un objet provenant d'une autre application (feuille Excel, feuille Access,...).**

Cours 7 : Les boîtes de dialogues communes

Il existe 5 boîtes de dialogue communes en Visual Basic :

- **Celle de l'ouverture**
- **Celle de l'enregistrement**
- **Celle de la couleur**
- **Celle de la police de caractères**
- **Celle de l'impression**

Avant tout, pour pouvoir utiliser les boîtes de dialogue communes, il faut ajouter le contrôle *CommonDialog* dans votre boîte à outils. Pour ce faire, placer votre souris sur la boîte à outils et cliquez sur le bouton droit. Ensuite, choisissez la commande "Composants". Une liste de tous les contrôles apparaît alors. Celui qui nous intéresse ici, c'est le contrôle *CommonDialog*. Cochez la case du contrôle *Microsoft Common Dialog Control 6.0 (SP3)* et appuyez sur le bouton Appliquer. Le contrôle *CommonDialog* va alors apparaître sur votre boîte à outils. A présent, vous pouvez l'utilisez dans vos applications.
Note: commencez par modifier les propriétés du contrôle avant d'utiliser les méthodes respectives.

1. La boîte de dialogue Ouvrir

La boîte de dialogue "Ouvrir" permet d'ouvrir un fichier parmi tous ceux mémorisés sur votre disque dur, disquette ou CD-ROM. Pour faire apparaître la boîte de dialogue "Ouvrir", on fait appel à la méthode *ShowOpen*. Cependant, il est aussi nécessaire de renseigner plusieurs propriétés de la boîte de dialogue.
Voici la liste principale des propriétés à renseigner :

Propriété	*Utilisation*
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé.
DefaultExt	Elle définit l'extension par défaut du nom du fichier à ouvrir.
DialogTitle	elle définit le titre de la boîte de dialogue situé sur la barre de titre.
FileName	Elle définit le chemin d'accès et le nom du fichier sélectionné par défaut.
Filter	Elle définit le(s) filtre(s) qui sert à spécifier quel type de fichier pouvant être ouvert en lecture. Par exemple, avec l'instruction " ***CMD.Filter =" DLL (*.DLL)|*.DLL|Exécutables (*.EXE)|*.EXE|Tous (*.*)|*.*| "***, vous pouvez choisir de n'ouvrir que les fichiers DLL ou Exécutables ou bien d'ouvrir tous les fichiers.
FilterIndex	Elle spécifie le filtre à utiliser par défaut dans la boîte de dialogue. Reprenons l'instruction précédente : avec l'instruction suivante "***CMD.FilterIndex = 2***", le filtre utilisé par défaut sera donc ***Exécutables (*.EXE)|*.EXE|***.
Flags	Elle définit les options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
InitDir	Elle définit le répertoire des fichiers affichés à l'ouverture de la boîte de dialogue.

2. La boîte de dialogue Sauvegarder sous

La boîte de dialogue "Enregistrer sous" permet de sauvegarder un fichier ouvert. Pour l'afficher, on fait appel à la méthode *ShowSave*.
Pour ce qui est des propriétés à modifier, reportez-vous à ceux de la boîte de dialogue "Ouvrir".

3. La boîte de dialogue Couleur

La boîte de dialogue "Couleur" permet une couleur parmi d'autres. En l'agrandissant, vous pourrez définir une couleur par ses composantes: teinte, saturation, RVB et luminance. Pour l'afficher, on fait appel à la méthode *ShowColor*. Tout comme les boîtes de dialogue précédentes, vous devrez renseigner certaines propriétés du contrôle. Voici les principales propriétés :

Propriété	*Utilisation*
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé.
Flags	Elle définit les options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
Color	Elle sélectionne une couleur par défaut. La syntaxe de l'instruction est: "Objet.Color = QBColor(valeur)".

2. La boîte de dialogue Police de caractères

La boîte de dialogue "Police de caractères" permet de sélectionner une police et ses attributs. Pour l'afficher, on fait appel à la méthode *ShowFont*. Ensuite, renseignez les propriétés suivantes:

Propriété	*Utilisation*
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé.
Flags	Elle définit les options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
FontBold	Elle définit le style Gras par défaut.
FontItalic	Elle définit le style Italique par défaut.
FontName	Elle définit la police par défaut.

FontSize	Elle définit la taille par défaut.
FontStrikethru	Elle définit le style Barré par défaut.
FontUnderline	Elle définit le style Souligné par défaut.
Max	Elle définit la taille maximale des polices affichés.
Min	Elle définit la taille minimale des polices affichés.

2. La boîte de dialogue Imprimer

La boîte de dialogue "Imprimer" sert à définir les paramètres de l'impression. Pour l'afficher, on fait appel à la méthode *ShowPrinter*. Ensuite, renseignez les propriétés suivantes:

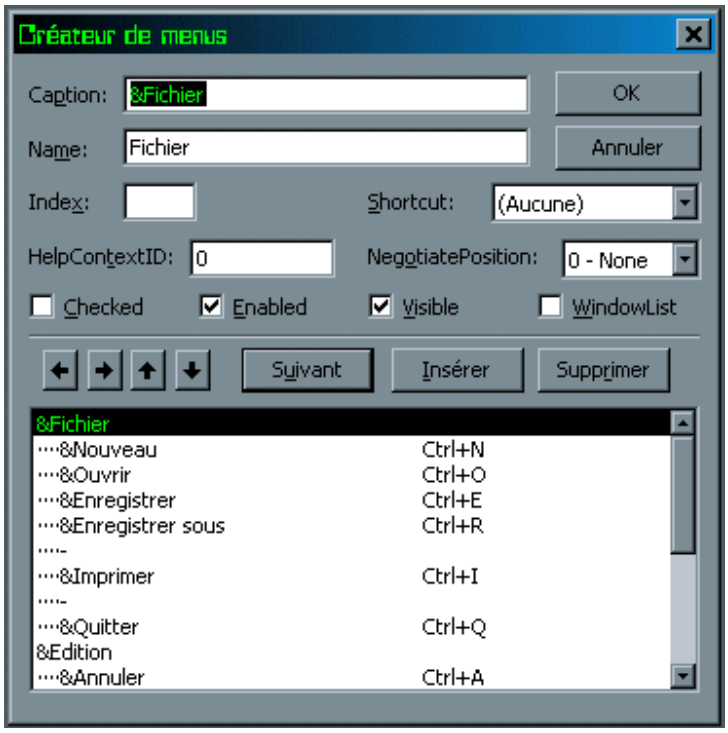
Propriété	*Utilisation*
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé.
Copies	Elle définit le nombre d'exemplaires à imprimer.
Flags	Elle définit les options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& +&H1&".
FromPage	Elle définit le numéro de la première page à imprimer.
PrinterDefault	Elle sert à indiquer si les paramètres entrés doivent devenir les nouveaux paramètres par défaut.
ToPage	Elle définit le numéro de la dernière page à imprimer.

Cours 8 : Système de menus

Un système de menus est composé de plusieurs menus qui eux-mêmes peuvent être composés de plusieurs sous-menus. La plupart des logiciels de Microsoft possèdent un système de menus pour faciliter l'utilisateur dans ses actions.

1. Le système de menus

Pour créer un système de menus, allez dans le menu "Outils" du système de menu de Visual Basic et choisissez la commande "Créateur de menus" ou pour aller plus vite, cliquez sur la 3ème image de la barre d'outils.



A première vue, ça a l'air compliqué. Mais après quelques manipulations, ce sera facile. Bon, voyons de plus près ce système de menus. L'esperlouette "&" que vous voyez dans chaque menu et commande vous permet de créer des raccourcis. Ainsi, pour ouvrir le menu "Fichier", vous pouvez tout simplement appuyer sur Alt+F. Notez que pour un menu, vous n'avez pas besoin de lui affecter une lettre pour pouvoir créer un raccourci. C'est automatiquement "Alt" + la lettre précédant l'esperlouette. Pour les commandes, allez dans la liste déroulante Shortcut et choisissez le raccourci que vous voulez pour votre commande de menu.

Pour créer une commande de menu, il faut l'indenter au menu auquel il est rattaché. Pour cela, à la suite de la création du menu "Fichier", appuyez sur la flèche de droite pour indenter la commande "Ouvrir". Visual basic permet un maximum de 5 niveaux d'indentation. Par exemple, vous pouvez après avoir créé la commande "Ouvrir", créer une sous-commande en l'indentant à ce dernier.
Vous pouvez par ailleurs créer une barre séparatrice afin de séparer plusieurs commande de menu. Pour cela, dans "Caption", mettez un "-" et donnez-lui un nom quelconque.

- **La case à cocher "Checked" permet d'afficher ou d'effacer une marque de sélection en face d'une commande de menu.**
- **La case à cocher "Enabled" permet d'empêcher l'utilisateur de sélectionner une commande. Si vous décochez cette case, l'utilisateur ne pourra pas sélectionner le menu ou la commande auquel vous avez décochez la case "Enabled".**
- **La case à cocher "Visible" permet de cacher ou d'afficher un menu ou une commande de menu.**
- **La case à cocher "Windowlist" permet d'afficher les derniers fichiers ouvert par l'application concernée.**

Pour les autres options, laissez-les de côté, pour le moment.

Voilà, c'est terminé, pour la création de menu. Il ne reste plus qu'à affecter à chaque commande de menu une action spécifique. Pour affecter une action à une commande, cliquez sur celui-ci dans la feuille où est créé le système de menus. Pour les actions de chaque commande, lisez les cours qui suivent.

---

2. Le menu contextuel

Dans une quelconque application de Microsoft, lorsque vous cliquez sur le bouton droit de votre souris, une série de commande apparaît: c'est le menu contextuel (ou Popup menu en anglais).
La création d'un popup menu se fait presque comme un système de menu. Pour cela, faites exactement comme si vous créez un système de menu. Définissez ensuite, une action pour chaque commande du menu contextuel. Par la suite, il faut définir dans quelles conditions doit apparaître le menu contextuel. Double-cliquez sur la feuille où est créé le système de menu et dans la liste déroulante "événement", sélectionnez l'événement "MouseDown". Entrez les instructions suivantes :

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer,
X As Single), Y As Single
If (Button = 2) Then
```

```
PopupMenu X
end if End Sub
```

Interprétation : Lorsque le bouton droit de la souris a été cliqué, le menu contextuel X apparaît.
Vous pouvez remplacer la valeur de la variable "Button" si vous voulez que le menu contextuel apparaisse à une autre action de l'utilisateur :

- **Pour que le menu contextuel apparaisse en cliquant sur le bouton gauche de la souris, remplacez 2 par 1 dans la condition (Button = 1);**
- **Pour que le menu contextuel apparaisse en cliquant sur le bouton central de la souris, remplacez 2 par 4 dans la condition (Button = 4);**

Enfin, revenez dans la boîte de dialogue "créateur de menu" et décochez la case "Visible" du menu que vous désirez rendre contextuel. Ne le faites pas avant car si vous le faites, vous ne pourrez pas définir une action pour chaque commande du menu contextuel.

Cours 9 : Accès aux fichiers

Il existe trois mode d'accès aux fichiers dans Visual Basic 6 :

1. **Les fichiers à accès séquentiel ne peuvent ouvrir que des fichiers avec un contenu textuel.**
2. **Les fichiers à accès direct sont formés en général de plusieurs enregistrements de longueur fixe.**
3. **Les fichiers binaires sont tous les fichiers avec un contenu graphique.**

---

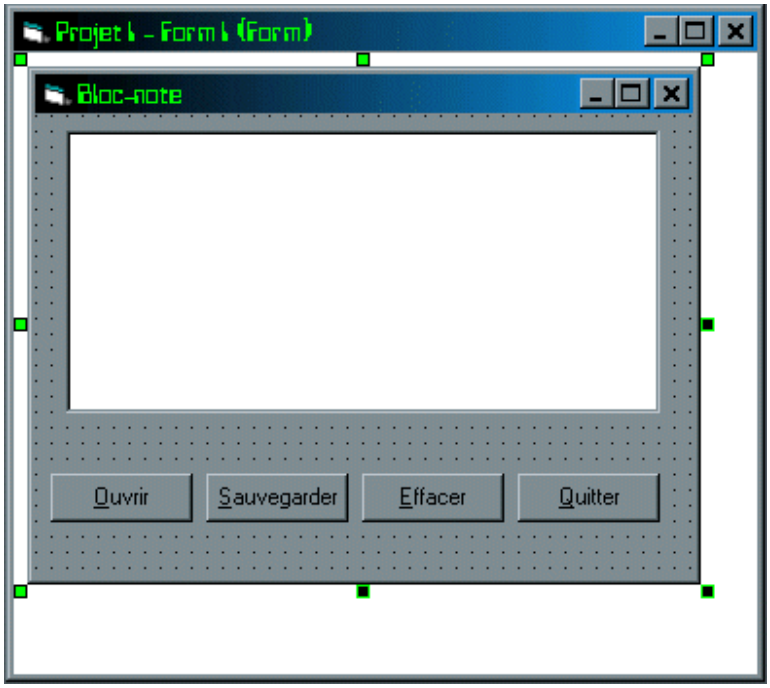
1. Les fichiers à accès séquentiel

Il existe trois manières d'ouvrir un fichier texte :

1. **La 1ère manière est l'ouverture d'un fichier uniquement en lecture à l'aide de l'instruction *Input*.**
2. **La 2ème manière est l'ouverture d'un fichier uniquement en écriture à l'aide de l'instruction *Output*.**

3. **La dernière manière est l'ouverture d'un ficher en écriture aussi mais à la seule différence de la précédente manière, les nouvelles données sont entrées à la suite de l'enregistrement précédent. Pour cela, on utilise l'instruction *Append*.**

Pour comprendre son fonctionnement, prenons un exemple. Lancez Visual Basic et ouvrez un nouveau projet. A la propriété Caption de la feuille Form1, affectez-lui la valeur "Bloc-note". Sur la feuille Form1, tracez un rectangle assez grand en utilisant le contrôle *TextBox*. Allez dans la boîte de propriété du contrôle TextBox et allez à sa propriété Text. Effacez-y le contenu. Ensuite, à sa propriété Multiline, choisissez True. Maintenant, placez sur la feuille 4 contrôles CommandButton. Pour chacun des 4 contrôles, donnez leur, les valeurs "&Ouvrir", "&Effacer", "&Sauvegarder" et "&Quitter" à la propriété Caption et les valeurs "Ouvrir","Effacer", "Sauvegarder" et "Quitter" à la propriété Name.
Voici l'allure que la feuille form1 doit avoir à présent :



Pour que ce bloc-note soit opérationnel, il faut définir des procédures qui gèrent les actions de chaque contrôle présent dans la feuille.
Double-cliquez sur le bouton de commande "Ouvrir" et écrivez la portion de code suivant :

```
Private Sub ouvrir_Click()
On Error Resume Next
Open "C:\Windows\Bureau\note.TXT" For Input As #1
retour = Chr$(13) + Chr$(10)
Line Input #1, texte
tout = texte
```

```
If Len(tout) <> 0 Then
While Not EOF(1)
Line Input #1, texte
tout = tout + retour + texte
Wend
End If
Close #1
End Sub
```

Explication :

- **L'instruction *On error Resume Next* permet d'ignorer toute erreur que peut entraîner cette procédure.**
- **L'instruction *Open "C:\Windows\bureau\note.TXT" For Input As #1* permet d'ouvrir le fichier "note" situé sur le bureau. Lorsque l'application est exécutée pour la première fois, une erreur 53 devrait apparaître car le fichier "note" n'existe pas encore. Mais grâce à l'instruction *On Error Resume Next*, cette erreur sera tout simplement ignorée.**
- **L'instruction *retour = Chr$(13) + Chr$(10)* permet d'aller à la ligne suivante. Le code ASCII 13 correspond au "retour chariot", c'est-à-dire de revenir à gauche du texte et 10 au "passage à la ligne", c'est-à-dire d'aller à la ligne suivante. Vous pouvez aussi utiliser le code *vbCrLf* à la place et c'est plus court et plus facile à retenir. Vous aurez ainsi: "tout = tout + vbCrlf + texte"**
- **L'instruction *Line Input* permet de placer dans la variable "texte" le contenu de la première ligne du fichier "note.TXT".**
- **Ensuite, le contenu de la variable "texte" est à son tour, affecté à la variable "tout".**
- **Le test avec l'instruction *If* permet de vérifier que le fichier "note.TXT" n'est pas vide à l'aide de la fonction *Len* qui compte le nombre de lettres ou tout autre symbole contenu dans la variable "tout". Si le nombre de lettres est différent de zéro alors, les instructions à l'intérieur du test sont exécutés.**
- **La boucle avec *While* permet de parcourir tout le fichier à la recherche d'éventuelles autres lignes que la première. La fonction *EOF* permet de savoir si l'on arrive à la fin du fichier ouvert. Cette boucle est donc exécutée jusqu'à ce que tout le contenu du fichier soit placé dans la variable tout. Les chaînes de caractères affectées à la variable "tout" sont concaténées (concaténées = ajoutées) aux précédentes contenues dans la variable. La variable "retour" provoque un retour à la ligne à chaque fois que cela est nécessaire.**
- **Lorsque tout le contenu du fichier sera affecté à la variable "tout", cette dernière sera à son tour affectée à la propriété *Text* du contrôle *TextBox*. Ce qui provoquera l'affichage du contenu du fichier dans le contrôle TextBox.**
- **L'instruction *Close* ferme le fichier ouvert en lecture.**

Écrivons maintenant la portion de code correspondant au bouton "Sauvegarder" :

```
Private Sub Sauvegarder_Click()
Open "C:\Windows\Bureau\note.TXT" For Output As #1
Print #1, Text1.Text
Close #1
End Sub
```

Explication :

- **L'instruction *Open "C:\Windows\Bureau\note.TXT" For Output As #1* ouvre le fichier "note.TXT" en écriture.**
- **Le contenu de la zone de texte "Text1" est ensuite copié dans le fichier à l'aide de l'instruction *Print*.**
- **Enfin, le fichier est fermé à l'aide de l'instruction *Close*.**

Passons à l'écriture de la procédure concernant le bouton "Effacer" :

```
Private Sub Effacer_Click()
Text1.Text = ""
Text1.SetFocus
End Sub
```

Explication :

- **L'instruction *Text1.Text = ""* permet d'effacer tout ce qui se trouve sur la zone de texte du contrôle TextBox.**
- **L'instruction *Text1.SetFocus* donne le focus à la zone de texte, c'est-à-dire que cette instruction provoque l'affichage d'une barre d'insertion dans la zone de texte.**

Enfin, écrivons la portion de code correspondant à l'action du bouton "Quitter" :

```
Private Sub Quitter_Click()
End
End Sub
```

Explication :

- **L'instruction *End* permet de quitter l'application en cours.**

Voilà, c'est tout. A présent, exécutez le programme et voyez ce que ça donne. Bien sûr, ce n'est qu'un simple petit programme mais en acquérant plus de connaissances par la suite, vous pourrez l'améliorer, lui ajouter de nouvelles fonctions, action,...

2. Les fichiers à accès direct

Les fichiers à accès direct contiennent des données contenues dans plusieurs enregistrements fixes. Pour déclarer un enregistrement, on utilise l'instruction *Type* pour créer un nouvel type de variable. Ensuite, On déclare une variable de ce type. Pour ouvrir un fichier à accès direct, on utilise l'instruction *Open* en indiquant la longueur de chaque enregistrement. Ainsi, un fichier à accès direct s'ouvre de la manière suivante : *Open "C:\Windows\Bureau\fichier.adr" For Random As #1 Len = Len(Adr)* où adr est le nom d'une variable de type prédéfini.
La lecture d'un enregistrement s'effectue de la manière suivante: *Get #1, Numéro, adr*.
L'écriture dans un fichier à accès direct s'effectue de la manière suivante: *Put #1, Numéro, adr*.

---

3. Les fichiers à accès binaire

Dans la mesure où le fonctionnement des fichiers à accès binaires est très proche de celui des fichiers à accès direct, reportez-vous aux fichiers à accès direct.

Cours 10 : Le presse-papier

Le presse-papier est indispensable pour toute opération de copie et de collage. Ces opérations s'effectuent grâce à l'objet *ClipBoard*.

Remarque : Si vous utilisez le contrôle *TextBox*, vous n'avez pas besoin de définir des procédures pour le presse-papier dans la mesure où il est déjà intégré dans le contrôle. Par contre, si vous utilisez le contrôle *RichtextBox*, alors il vous faudra définir vous-même le presse-papier. La différence entre ces 2 contrôles de saisie est que le contrôle *RichTextBox* possède quelques fonctions supplémentaires comme la possibilité de définir des marges autour du texte.

Passons maintenant aux méthodes utilisées pour définir le presse-papier :

- ***ClipBoard.GetText* permet de lire le contenu du presse-papier(uniquement avec des données textuelles).**

- *ClipBoard.SetText* **permet d'écrire dans le presse-papier(uniquement avec des données textuelles).**
- *ClipBoard.Clear* **permet d'effacer le contenu du presse-papier.**
- *ClipBoard.GetFormat(type)* **permet d'indiquer quel type de données est présent dans le presse-papier. Les valeurs que peut retourner Visual Basic sont :**
    1. **(vbCFText) : données de type texte.**
    2. **(vbCFBitmap) : image en mode point(bitmap).**
    3. **(vbCFMetafile) : métafichier vectoriel(wmf).**
    4. **vbCFDib) : image bitmap indépendante du périphérique.**
    5. **(vbCFPalette) : palette de couleur.**
    6. **(vbCFEMetafile) : métafichier amélioré(emf).**
    7. **(vbCFFile) : noms de fichiers copiés depuis l'Explorateur de Windows.**
    8. **&FFFFBF00 (vbCFLink) : liaisonDDE(valeur exprimée sous forme hexadécimale).**
    9. **&FFFFBF01 (vbCFRTF) : texte au format RTF.**
- *ClipBoard.GetData* **permet de lire le contenu du presse-papier(uniquement avec des données graphiques).**
- *ClipBoard.SetData* **permet d'écrire dans le presse-papier(uniquement avec des données graphiques).**

Cours 11 : La barre d'outils et la barre d'état

La barre d'outils | La barre d'état

Tout d'abord, pour pouvoir utiliser les contrôles pour placer une barre d'outils ou d'état, il faut qu'ils soient déjà présent dans votre boîte à outils. Or ce n'est pas le cas. Pour ce faire, choisissez la commande "Composants" du menu "Projet". Cochez le contrôle *Microsoft Windows Common Controls 6.0* et validez. A présent, une série de contrôle est apparue sur votre boîte à outils. Vous y trouverez le contrôle *ToolBar* et le contrôle *StatusBar*.

---

1. La barre d'outils

Commencez par sélectionner le contrôle *ToolBar* dans la boîte d'outils et sur la feuille tracez un rectangle d'une quelconque taille. Cela n'a aucune importance dans la mesure

où la barre d'outils se placera automatiquement sur le haut de votre feuille. Ensuite, sélectionnez le contrôle *ImageList* et placez-le à son tour sur la feuille. (Notez que l'emplacement de ce contrôle n'a aucune importance.) Sélectionnez à nouveau le contrôle *ImageList* mais cette fois, sur la feuille et cliquez sur le bouton droit de votre souris. Dans le menu contextuel, choisissez la commande "Propriétés". Dans l'onglet "Général", vous avez la possibilité de modifier la taille de vos images présentes dans la barre d'outils. C'est dans l'onglet "Images" que vous allez pouvoir choisir les images qui y seront placées. Appuyez sur le bouton "Insérer une image". Les images par défaut fournies avec Visual Basic sont situés dans le répertoire *C:\Program Files\Microsoft Visual Studio\Common\Graphics\Bitmaps*. sélectionnez les images que désirez mettre sur votre barre d'outils. Attribuez à chacune de ces images un nom dans la propriété "Key". Validez ensuite en appuyant sur le bouton "Appliquer".
A présent, faites un clic droit sur le contrôle *ToolBar* et sélectionnez "propriétés". Dans l'onglet "Général", sélectionnez "ImageList1" dans les propriétés "ImageList", "DisabledImageList" et "HotImageList". sélectionnez aussi "1-TbrFlat" dans la propriété "Style". Passons à présent à l'onglet "Boutons". Renseignez les principales propriétés. Écrivez dans la propriété "Key" le nom des images que vous avez choisi dans le contrôle *ImagaList*. La propriété "ToolTipText" vous permet d'afficher une bulle lorsque vous survolez l'image. Enfin, dans la propriété "Image", écrivez les numéros qui sont respectifs aux images contenues dans le contrôle *ImagaList*. N'oubliez pas que ce dernier permet d'afficher des images sur votre barre d'outils. Ok, maintenant, revenez à la feuille principale. Vous avez vu, votre barre d'outils est à présent visible.
A present, occupons-nous du code pour relier chaque icône de la barre d'outils à une action. Double-Cliquez sur la barre d'outils. Pour gérer chaque action, utilisons la structure Select Case qui convient parfaitement à ce que l'on veut faire.

```
Select Case Button.Key
Case "New"
Nouveau_Click
Case "Open"
Ouvrir_Click
End Select
```

"Nouveau_Click" correspond à la procédure que vous avez normalement attribué à la commande "Nouveau" du système de menus. Si ce n'est pas le cas, alors, il va vous falloir ici, lui attribuer une action. Par ailleurs, au lieu d'utiliser "Button.key" pour tester la condition, vous pouvez aussi utiliser "Button.Image", mais alors, au lieu d'avoir "New", vous devrez mettre à la place le numéro leur correspondant que vous avez défini dans le contrôle ToolBar, à l'onglet Boutons, dans le champs "Image".

---

2. La barre d'état

Choisissez le contrôle *StatusBar* et placez-le n'importe où. cela n'a pas d'importance car au bout du compte, il s'affichera automatiquement tout en bas de votre feuille. Faites un clic droit sur ce dernier qui est à présent placé sur la feuille et allez dans l'onglet "Propriétés". Paramétrez les différentes propriétés qui s'y trouvent.
Ca y est! A présent, votre application est comparable à toute autre logiciel. Avec quelques réglages supplémentaires, vous pouvez même rivaliser avec les applications de Microsoft! Super, non?

Cours 12 : Les tableurs et les graphes

Tout d'abord, pour pouvoir utiliser les tableurs et graphes, vous devez au préalable placer dans votre boîte d'outils, les contrôles *Microsoft Flexgrid Control 6.0* et *Microsoft Chart Control 6.0*. Pour cela, faites un clic droit sur la boîte d'outils et choisissez la commande "Composants". Ensuite, cochez les cases correspondant aux contrôles et validez.

Grâce au contrôle *MSFlexGrid* et après avoir acquis une certaine expérience, vous pouvez même créer des tableurs semblables à ceux du logiciel Excel de Microsoft. Les propriétés du contrôle *MSFlexGrid* permettent de manipuler les cellules de ce dernier.

Rows	**Elle permet de définir ou de connaître le nombre de lignes du contrôle.**
Cols	**Elle permet de définir ou de connaître le nombre de colonnes du contrôle.**
FixedRows	**Elle permet de définir ou de connaître le nombre de lignes fixes du contrôle.**
FixedCols	**Elle permet de définir ou de connaître le nombre de colonnes fixes du contrôle.**
Row	**Elle permet de définir ou de connaître les coordonnées de la cellule courante.**
Col	**Elle permet de définir ou de connaître les coordonnées de la cellule courante.**
Text	**Elle permet de définir ou de connaître le contenu de la cellule courante.**
ColWidth	**Elle permet de définir ou de connaître la largeur d'une colonne.**
RowHeight	**Elle permet de définir ou de connaître la hauteur d'une ligne.**

ColAlignment	**Elle permet de définir ou de connaître l'alignement des données d'une colonne.**
RowSel	**Elle permet de sélectionner ou de connaître les coordonnées d'une plage de cellules.**
ColSel	**Elle permet de sélectionner ou de connaître les coordonnées d'une plage de cellules.**

De même, les propriétés du contrôle *Microsoft Chart Control 6.0* permettent de définir un graphe donné.

Column	**Elle renvoie ou définit la colonne active de la grille de données.**
Data	**elle renvoie ou définit la valeur d'un point de données spécifique dans la grille de données identifiée par Row et Column.**
DrawMode	**Elle détermine quand et comment est rafraîchi l'affichage du graphique.**
CharType	**Elle renvoie ou définit le type de graphiques utilisé pour tracer les données dans la grille de données.**

Cours 13 : Les applications MDI

Tout d'abord, définissons ce que c'est qu'une application MDI. Vous connaissez sûrement le logiciel de traitement de texte WORD de Microsoft. Eh ben, c'est une application MDI et comme son nom l'indique, elle est composée d'une fenêtre principale(appelé aussi *fenêtre parent*) et d'une ou plusieurs fenêtres de documents(appelés aussi *fenêtre fille*. Les applications MDI sont en général pourvu d'un système de menu, d'une barre d'outils et d'une barre d'état. Elle permettent de visualiser plusieurs documents en même temps. Ce qui permet d'éviter d'ouvrir plusieurs fois la même application. Passons à présent à sa mise en place!

---

1. Mise en place de la fenêtre parent

Elle se déroule en 3 étapes:

1. **Pour créer la fenêtre parent en tant que tel, ouvrez un nouveau projet puis sélectionnez dans la fenêtre de projet la feuille ouverte par défaut *Form1*. Faites un clic droit dessus et choisissez la commande "Supprimer Form1". Ensuite, allez dans le menu "Projet" et sélectionnez la commande "Ajouter une feuille MDI". Voilà! Vous avez créé une *fenêtre parent*, facile non?**
2. **Passons à la création du système de menus. pour cela, reportez-vous au cours 8.**
3. **Enfin, passons à la création de la barre d'outils et celle d'état. Pour cela, reportez-vous au cours11.**

2. Mise en place de la fenêtre fille

Sa mise en place est simple. Pour la définir, placez une nouvelle feuille dans le projet. Ensuite, initialisez à True sa propriété *MDIChild*. C'est tout!

Cours 14 : Les bases de données

L'élaboration d'une base de données se fait à l'aide du contrôle *Data*. Ainsi, vous pouvez accéder à pas mal de fichiers de données divers tels que ceux de Access, Excel, FoxPro, Lotus 1-2-3, BTrieve ou tout fichier ASCII.
En clair, le contrôle *Data* est indispensable dans la réalisation d'une gestion d'une base de données. Cependant, il ne suffit pas simplement de placer le contrôle sur la feuille. Il faut impérativement renseigner quelques unes de ses propriétés sans quoi, le contrôle ne marchera pas.
Mais rassurez-vous, ce n'est pas aussi dur que vous le pensiez. D'ailleurs, j'étais comme vous auparavant. Je me disais que les bases de données, c'est pas du tout mon truc et rien que d'entendre ce mot, ça me donnait la migraine. Mais après une très courte séance d'apprentissage, je me suis vite rendu compte que ce n'était pas aussi cailloux comme le prétendent certains. Ca y est, vous avez repris confiance en vous? Allez, c'est parti!
Tout d'abord, définissez un nouveau projet et placez le contrôle *Data* sur la feuille "Form1". Maintenant, passons au renseignement de quelques unes de ses propriétés.

Connect	**Indiquez-y quel type de bases de données vous voulez utiliser: Access, Excel, Paradox, Foxpro...**
DataBaseName	**Indiquez-y où se trouve votre fichier de base de données sur votre disque.**

RecordSource	**Indiquez-y le nom de la table à éditer.**

Pour pouvoir afficher les données de votre fichier, vous aurez sans doute besoin d'un ou plusieurs champs de texte créés à l'aide du contrôle *TextBox*. Là aussi, vous aurez à modifier quelques une de ses propriétés.

DataSource	**Indiquez-y le nom du contrôle *Data* auquel il se rapporte.**
DataField	**Indiquez-y le nom du champ de valeurs auquel il se rapporte.**

Passons à présent, aux principales méthodes utilisées pour gérer une base de données dans une application.

AddNew	**Elle sert à créer un nouvel enregistrement. Cependant, Les données nouvellement entrées ne sont pas tout de suite enregistrées à moins que vous n'appuyez sur une des flèches du contrôle *Data*. Pour qu'elles soient validées, on pourrait faire appel à la méthode *Updaterecord* suivi de l'instruction suivante: *Data1.recordSet.BookMark = Data1.RecordSet.LastModified* qui permet de revenir à la dernière donnée enregistrée ou modifiée. Exemple: *Data1.RecordSet.AddNew*.**
Delete	**Elle sert à supprimer un enregistrement donné. Exemple: *Data1.RecordSet.Delete***
Movefirst, MoveLast, MovePrevious, MoveNext	**Elles servent respectivement à revenir au premier enregistrement, à aller au dernier enregistrement, à revenir à l'enregistrement précédent et à aller à l'enregistrement suivant. Exemple: *Data1.RecordSet.MoveFirst*.**
Refresh	**Elle sert à rafraîchir la base de donnée. Exemple: *Data1.Refresh*.**
UpdateRecord	**Elle sert à mettre à jour une donnée nouvellement enregistrée ou modifiée. Exemple: *Data1.UpDateRecord*.**

Pour aller un peu plus loin, nous allons ici traiter des requêtes SQL sans s'y attarder trop longtemps.
Avant tout, faites attention à la casse des mots. Si vous écrivez Select au lieu de SELECT, ça marchera pas!

Pour comprendre son fonctionnement, prenons l'exemple de l'instruction suivante: *Data1.RecordSource = "SELECT * FROM Produits WHERE [Nom de produit] = 'Riz';"*. En terme plus clair, Cette requête extrait tous les champs (*) de la table Produits (FROM Produits) pour lesquels le champ "Nom de produit" a pour valeur "Riz"(WHERE [Nom de produit]='Riz').
Dans cette requête, le champ "Nom de produit" a été mis entre crochet car il contient des espaces. Autrement dit, ces crochets ne sont nécessaires que pour les champs avec un nom ne contenant qu'un seul mot.
L'instruction suivante: *Data1.RecordSource = "SELECT * FROM Produits; "* permet à nouveau d'avoir accès à toutes les données du fichier. Tous les champs(*) de la table "Produits"(FROM Produits) sont à nouveau accessible à l'utilisateur.

Cours 15 : Le multimédia

La lecture des fichiers audio ou vidéo se fait à l'aide du contrôle *MMControl*. Pour cela, choisissez la commande "Composants" du menu "Projet". Cochez le contrôle *Microsoft Multimedia Control 6.0* et validez. A présent, le contrôle *MMControl* est dans votre boîte d'outils.
Notez qu'il existe plusieurs types d'objet multimédia comme vous pouvez vous en douter. Les plus courantes sont:

Type d'objet multimédia	Type de fichier	Extension
WaveAudio	Fichier sonore numérisé WAV	.WAV
Sequencer	Fichier musical MIDI	.MID
CDAudio	Morceau de musique sur un CD audio	-
AVIVideo	Fichier numérique au format AVI	.AVI

Bien sur, il en existe bien d'autres types d'objet multimédia mais nous nous contenterons de ceux cités dans le tableau.

Pour ouvrir un fichier multimédia, servez-vous des boîtes de dialogue communes. Pour imiter les commandes du contrôle *MMControl*, il est utile de connaître les commandes *MCI*. Elles agissent sur le comportement de ce dernier. Voici une liste des commandes MCI, les plus importantes:

Commande MCI	Fonction
Stop	Stoppe immédiatement la lecture en cours.
Play	Joue le fichier sélectionné.
Open	Ouvre le fichier sélectionné.
Close	Ferme le fichier sélectionné.
Pause	Interrompt la lecture en cours.
Eject	Éjecte le CD contenu dans le CD-ROM.

Par exemple, l'instruction *MMControl1.Command = "Stop"* permet de stopper immédiatement le fichier audio/vidéo en cours. Ainsi, pour lire un fichier ".wav", commencer par l'ouvrir avec la commande "Open", puis pour la lecture de ce fichier, utilisez la commande "Play". Si vous désirez stopper la lecture en cours, utilisez la commande "Stop". Enfin, n'oubliez pas de bien refermer le fichier à l'aide de la commande "Close" car il se peut qu'il reste encore en mémoire et donc qu'il occupe encore une partie de votre mémoire vive.

Pour les fichiers "CDAudio", cela se passe un peu différemment. Placez la portion de code qui suit dans la procédure "Form_Load()":

```
Private Sub Form_Load()
MMControl1.DeviceType = "CDAudio"
MMControl1.Command = "Open"
MMControl1.UpdateInterval = 1000
MMControl1.Timeformat = 10
End Sub
```

Explication:

- **La 1ère instruction informe au contrôle *MMControl* du type de fichier qui va ouvert. Cela est nécessaire car les fichiers "CDAudio" sont différents des autres.**
- **La 2nde instruction permet d'ouvrir le premier morceau de musique.**
- **La 3ème instruction permet de déterminer l'intervalle de temps où doit s'exécuter la procédure *MMControl1.StatusUpdate()*, c'est-à-dire, toutes les secondes (ici, le temps est défini en milliseconde), la procédure *MMControl1.StatusUpdate()*.**
- **La 4ème instruction détermine le format horaire à utiliser pour exprimer les informations relatives à la position en cours dans le CDAudio, par exemple combien de temps se sont écoulés depuis la lecture d'une musique.**

Pour éjecter le CD du CD-ROM, placez la portion de code qui suit dans la procédure "MMControl1_EjectClick(Cancel As Integer)":

```
Private Sub MMControl1_EjectClick(Cancel As Integer)
MMControl1.Command = "Eject"
MMControl1.Command = "Close"
End
End Sub
```

Explication: Comme vous vous doutez, la 1ère instruction permet d'éjecter le CD ,la 2nde de fermer le périphérique MCI (sorties audio/vidéo) et la dernière de quitter l'application.

Enfin, le plus important, placez la portion de code qui suit dans la procédure MMControl1_StatusUpdate():

```
Private Sub MMControl1_StatusUpdate()
Dim All, Seconde, Minute, Misc
All = MMControl1.Position Mod 16777216
Seconde = All / 65536
Minute = (All - Seconde * 65536) /256
Misc = All - Seconde * 65536 - Minute * 256
Piste.Caption = " Piste " + Str(Misc)
Temps.Caption = " Durée " + Minute + " : " + Seconde
End Sub
```

Explication: Les 3 premières instructions permettent d'extraire le numéro de la piste et du temps écoulé depuis le début. La 4ème instruction place le numéro de la piste en cours dans le contrôle "Piste" (il peut être un contrôle "Texbox" ou "Label"). Enfin, la dernière instruction place la durée du temps écoulé depuis la première piste dans le contrôle "Temps" il peut être un contrôle "Texbox" ou "Label").

Cours 16 : Le langage de requête structuré SQL

Tout d'abord, il faut que je vous explique ce que c'est que ce langage de requête et à quoi est-ce qu'il sert.
Eh bien, SQL("Structured Query Language" ou pour les francophones, "Langage de Requête Structuré") est un langage qui a été créé dans le but de communiquer avec une *base de données relationnelle*.
Pour ceux qui ne le savent pas, une base de données relationnelle est une base de données où les données sont stockées dans des *tables*. Ces dernières sont en interaction les unes et les autres.
ATTENTION:
veuillez noter que le langage SQL peut varier d'un constructeur à l'autre. Dans ce cas, il vous faut consulter la documentation qui lui est propre.
Le langage SQL abordé ici, est le langage standard et peut ne pas être compatible avec votre

base de données. Si vous constatez que les codes qui suivent ne fonctionnent pas, eh bien, ça veut dire qu'il n'est pas compatible avec votre base.

SQL permet d'interroger une base de données, d'insérer de nouvelles données mais aussi mettre à jour vos données existantes.
A présent voyons cela de plus près.

---

1. Recherche de données

1.1 La syntaxe

L'instruction utilisée pour effectuer une recherche dans une base de données est *SELECT*. Elle sert à interroger une base et de retourner si elles existent, les données que vous recherchez.
La syntaxe de l'instruction *SELECT* est la suivante:

```
SELECT [ *|ALL|DISTINCT ] [ TOP X [PERCENT] ] Colonne1,Colonne2
FROM Tables
WHERE (Critère de recherche)|(Critère de jointure)
AND|OR [ (Critère de recherche) ]
GROUP BY [ ALL ] Colonne1, Colonne2
HAVING (Critère de recherche)
ORDER BY Colonnes [ ASC|DESC ]
```

Je sais, à première vue d'oeil, ça peut paraître compliqué. Mais il n'en est rien en réalité. En observant de plus près, vous verrez que c'est très abordable. Bon, je vous explique en détail cette syntaxe.

- **L'instruction *SELECT* permet de sélectionner les données à partir d'une ou plusieurs colonnes d'une table.**
  - **L'argument * permet de faire une recherche dans toutes les colonnes de la table donnée.**
  - **L'argument *ALL* sert à indiquer de retourner toutes les valeurs recherchées même ceux qui sont en double. Par exemple, dans une base de données contenant une liste de tous les clients d'une société, il est parfaitement possible qu'il y ait 2 clients ayant le même nom de famille ou le même prénom. Eh bien, cet argument indique de retourner les données de ces 2 clients. Cet argument est celui par défaut. Il est facultatif.**
  - **L'argument *DISTINCT* est le contraire de *ALL*. Il sert à indiquer de ne retourner que les données uniques. Il est facultatif.**
  - **L'argument *TOP* permet de préciser le nombre d'enregistrements que vous souhaitez recevoir en réponse à votre requête à partir du premier enregistrement. *PERCENT* indique le pourcentage X d'enregistrements que vous souhaitez retournez. Il est facultatif.**
  - **Les mots *Colonne1,Colonne2* Indiquent dans quel(les) colonne(s) de la table ou des tables vous souhaitez effectuez votre recherche. N'oubliez pas la virgule si vous effectuez votre recherche dans plusieurs colonnes.**
- **La clause *FROM* sert à indiquer à partir de quel(les) table(s) les données doivent être extraites. Cette clause est indispensable pour toute requête**
  - **Tables représente le nom des différentes tables dans lesquelles vous désirez faire des recherches.**
- **La clause *WHERE* permet d'insérer des critères de recherche dans votre requête. Il est facultatif.**
  - **Le mot *Critère de recherche* représente les critères de recherche. Par exemple: "WHERE Client = 'Alphonse'" ou bien "WHERE Prix < '1500'".**
  - **Le mot *Critère de jointure* permet d'effectuer une recherche dans une colonne présente dans 2 tables différentes.**
- **Les opérateurs *AND* et *OR* permettent d'insérer un autre critère de recherche. Ils sont facultatifs.**
  - **Le mot *Critère de recherche* représente aussi des critères de recherche. Par exemple: "WHERE Client='Alphonse' OR Client='Jean'".**
- **La clause *GROUP BY* permet de regrouper des données. Il est facultatif.**
  - **Pour le mot *ALL*Voir précédemment.**
- **La clause *HAVING* permet d'insérer un critère de recherche pour les données regroupées avec la clause *GROUP BY*. Il est facultatif.**
  - **Pour le mot *Critère de recherche*, voir précédemment.**
- **Enfin, la clause *ORDER BY* permet de trier les données par colonnes. Il est facultatif.**
  - **Le mot *Colonnes* représente le nom des colonnes dans lesquelles sont triées les données.**
  - **L'argument *ASC*Permet de trier les données par ordre croissant. C'est l'ordre par défaut. Il est facultatif.**
  - **L'argument *DESC* permet de trier les données par ordre décroissant. Il est facultatif.**

C'est pas très évident mais avec de la pratique, tout vous paraîtra clair. De plus, vous n'êtes pas obligé d'utiliser tous les clauses et arguments montrés ci-dessus. Voici un exemple simple sur la façon d'utiliser les requêtes avec SQL en s'appuyant sur le contrôle *Data* qui vous permet de visualiser les données:

```
Data.RecordSource="SELECT * FROM Fichier WHERE Prix<'1500' AND Prix>'3000'
ORDER BY Prix"
```

Il vous suffit de placer ce code dans un bouton de commande pour qu'il fasse apparaître la liste de tous les prix compris entre 1500 et 3000 de la table "Fichier". La clause *ORDER BY* permet de ranger le résultat de votre recherche par ordre croissant(ordre par défaut).

1.2 Les opérateurs

Vous pouvez aussi faire appel à des opérateurs pour spécifier des conditions dans une instruction SQL ou servir de conjonction à plusieurs conditions. Vous vous rappelez, vous en avez déjà vu 2 opérateurs avant: ce sont AND et OR. Ces deux là, sont des opérateurs conjonctifs. Il existe 5 types d'opérateur en tout:

- **Les opérateurs de comparaisons servent à vérifier les conditions d'égalité, de non-égalité, les valeurs supérieures à et les valeurs inférieures à. Les voici: =,<>,<,>,<=,>=.**
- **Les opérateurs conjonctifs permettent d'effectuer plusieurs conditions à la fois. Les voici: AND et OR.**
- **Les opérateurs logiques servent à effectuer des comparaisons. Les voici: IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE, IS NULL, IS NOT NULL, EXISTS, NOT EXISTS, ALL et ANY.**
- **Les opérateurs arithmétiques servent à effectuer des opérations de calcul. Les voici: +,-,* et /.**

Le tableau suivant vous montre comment utiliser tous ces opérateurs.

Opérateur	Exemple	Interprétation
=	WHERE Prix='1500'	Retourne les valeurs dont le prix est égal à 1500
<>	WHERE Prix<>'1500'	Retourne les valeurs dont le prix est différent de 1500
<	WHERE Prix<'1500'	Retourne les valeurs dont le prix est strictement inférieur à 1500

>	WHERE Prix>'1500'	Retourne les valeurs dont le prix est strictement supérieur à 1500
<=	WHERE Prix<='1500'	Retourne les valeurs dont le prix est inférieur ou égal à 1500
>=	WHERE Prix>='1500'	Retourne les valeurs dont le prix est supérieur ou égal à 1500
AND	WHERE Prix>'1500' AND Prix<'2000'	Retourne les valeurs dont le prix est compris entre 1500 et 2000 non inclus.
OR	WHERE Prix='1500' OR Prix='2000'	Retourne les valeurs dont le prix est égal à 1500 ou 2000
IN	WHERE Prix IN('1500','2000')	Retourne les valeurs dont le prix est égal à 1500 ou 2000
NOT IN	WHERE Prix NOT IN('1500','2000')	Retourne les valeurs dont le prix est différent de 1500 ou de 2000
BETWEEN	WHERE Prix BETWEEN '1500' AND '2000'	Retourne les valeurs dont le prix est compris entre 1500 et 2000 non inclus.
NOT BETWEEN	WHERE Prix NOT BETWEEN '1500' AND '2000'	Retourne les valeurs dont le prix n'est pas compris entre 1000 et 1500 non inclus.
LIKE	WHERE Nom LIKE 'c*' or LIKE '*a'	Retourne les valeurs dont le nom du client commence par un "c" ou se terminant par un "a"
NOT LIKE	WHERE Nom NOT LIKE '*k*'	Retourne les valeurs dont le nom du client ne contient pas la lettre "k"
IS NULL	WHERE Prix IS NULL	Retourne les valeurs dont aucun prix n'a été fixé
IS NOT NULL	WHERE Prix IS NOT NULL	Retourne les valeurs dont le prix a été fixé
EXISTS	WHERE EXISTS (SELECT Prix FROM Fichier WHERE Prix='1500')	Teste pour vérifier s'il y a des prix égaux à 1500
NOT EXISTS	WHERE NOT EXISTS (SELECT Prix FROM Fichier WHERE Prix='1500')	Teste pour vérifier s'il y a des prix différent à 1500
ALL	WHERE Prix > ALL (SELECT Prix FROM Fichier_concurrent WHERE Lieu='Marseille')	teste pour vérifier s'il y a des produits concurrent se trouvant à Marseille qui coûtent moins cher que le vôtre
ANY	WHERE Prix < ANY (SELECT Prix FROM Fichier_concurrent WHERE Lieu='Marseille')	Teste le prix du produit pour savoir s'il est plus bas que celui du concurrent se trouvant dans à

		Marseille

1.3 Les fonctions mathématiques

Il est aussi possible avec le langage SQL d'effectuer des calculs directement à partir des données. Cela se fait à l'aide des fonctions mathématiques suivantes:

Fonction	Utilisation
COUNT()	Comptabilise le nombre d'enregistrements retourné
SUM()	Calcule la somme des valeurs retournées
AVG()	Calcule la moyenne des valeurs retournées
MAX()	Retourne la plus haute des valeurs trouvées
MIN()	Retourne la plus petite des valeurs trouvées

Prenons quelques exemples pour mieux comprendre:

Plaçons ce bout de code dans un champs de texte.

```
Data.RecordSource="SELECT COUNT(Produit) FROM Fichier
```

Ce code permet de voir combien est-ce qu'il y a de produits vendu dans un magasin.

Les autres fonctions fonctionnent de la même manière que cette dernière.

2. Ajout et recherche de données

2.1 Ajout de données

Pour ajouter des données dans une base de données, on utilise l'instruction *INSERT*. La syntaxe pour ajouter de nouvelles données avec SQL est la suivante:

```
INSERT INTO Table (Colonnes)
VALUES (valeurs1,valeurs2)
```

- **L'instruction *INSERT* permet d'insérer de nouvelles données.**
  - **Le mot-clé *INTO* est spécifique à une base Access. Ne la mettez que que si vous utilisez Access.**
  - **Le mot *Table* représente la table où doit être inséré ces nouvelles données.**

- Le mot ***Colonnes*** représente les colonnes dans lesquelles sont placées ces nouvelles données.

- La clause VALUES vous permet d'insérer de nouvelles données. C'est ici qu'il faut placer les nouvelles valeurs.
  - Les mots ***valeurs1, valeurs2*** représentent les données qui vont être entrées dans la base.

Par exemple:

INSERT INTO Fichier_client(Nom,Prénom,Id) VALUES ('Dupont','Jean','568')

Le client Dupont Jean, numéro d'identification 568, a été ajouté dans la base de données.

2.2 Suppression de données

La suppression de données se fait à l'aide de l'instruction *DELETE*.
La syntaxe pour supprimer des données à l'aide de SQL est la suivante:

DELETE FROM table WHERE (Critère de recherche)

- L'instruction ***DELETE*** permet d'effacer des données d'une base.
  - La clause ***FROM*** sert à indiquer à partir de quel table les données doivent être détruites.
  - Le mot ***table*** représente le nom de la table.
  - La clause ***WHERE*** permet d'introduire un critère de recherche.
  - Le mot ***Critère de recherche*** permet d'effectuer une recherche afin de déterminer quelles données doivent être effacé de la base.

Par exemple:

DELETE FROM Fichier_client WHERE Client='Dupont'

Le client Dupont a été supprimé dans la base de données.

3. Mise à jour de données

La mise à jour de vos données se fait à l'aide de l'instruction *UPDATE*.
La syntaxe est la suivante:

UPDATE table SET Colonne='valeur'
WHERE (Critère de recherche)

- **L'instruction *UPDATE* permet de mettre à jour vos données.**
  - **Le mot *table* représente le nom de la table dans lequel vont être mis à jour des données.**
  - **La clause *SET* permet d'introduire un critère de recherche.**
  - **Le mot *Colonne* représente le nom de la colonne dans lequel vont être mis à jour des données.**
  - **Le mot *valeur* représente la nouvelle valeur.**
- **La clause *WHERE* permet d'introduire un critère de recherche.**
  - **Le mot *Critère de recherche* permet d'effectuer une recherche afin de déterminer quelles données doivent être effacé de la base.**

Par exemple:

```
UPDATE Fichier SET date='20/05/01' WHERE Produit='Lait'
```

Tous les enregistrements de la colonne date vont être modifié par '20/05/01'. Mais avec la clause *WHERE*, seul les produits laitiers seront concerné.

4. Les transactions

Les *transactions* permettent de grouper une série d'instruction SQL. Ne croyez pas que ça sert à rien. Cette méthode de regroupement peut s'avérer être très utile lorsque vous manipulez des données sensibles. En effet, lorsque qu'une instruction SQL échoue, vous avez la possibilité d'annuler la transaction et de revenir en arrière c'est-à-dire avant que les données soient modifiées.
Cette transaction se fait à l'aide des commandes suivantes:

- **BEGINTRANS marque le début d'une transaction.**
- **COMMITTRANS marque la fin d'une transaction.**
- **ROLLBACK annule la transaction et vous retrouvez vos données initiales.**

Voici la syntaxe d'une transaction:

```
On Error GoTo Erreur
BeginTrans
   Série d'instruction
CommitTrans
Exit Sub
Erreur:
     MsgBox Error$
       RollBack
       exit Sub
```

Vous avez dû remarqué qu'une routine de traitement d'erreur a été placé dans ce code. C'est dans le cas où la transaction échouerait. Sinon, je pense que vous avez compris son fonctionnement en voyant la syntaxe. C'est pas dur à comprendre.

5. Recherche à partir d'un champ de texte

Voici comment faire pour effectuer une recherche à partir d'un mot tapé dans un champ de texte. Voilà, il faut que vous placiez un champ de texte nommé par exemple, "Text1" et un bouton de commande appelé par exemple, "Chercher". Placez le bout de code suivant dans la procédure du bouton:

```
valeur = "SELECT * FROM Liste WHERE Produit LIKE " + " ' " + Text1.Text + " ' "
Data.RecordSource = valeur
Data.refresh
```

Voilà, c'est tout! Si vous ne comprenez pas quelque chose, n'hésitez pas à poser votre question sur le forum.

Cours 17 : L'éditeur de ressources

Tout d'abord, faut que je vous explique ce que c'est qu'un éditeur de ressources et à quoi sert-il. C'est un utilitaire qui permet de stocker des données comme une base de données. On pourrait dire que c'est une alternative pour stocker vos données au lieu de faire appel à des bases de données.
Vous pouvez par exemple vous servir de cet utilitaire pour permettre de traduire une application en plusieurs langues. Vous pouvez y stocker 5 types de données:

- ***Les chaînes de caractère;***
- ***Les bitmaps;***
- ***Les icônes;***
- ***Les curseurs;***
- ***Les ressources de type binaires tels que les fichiers son(WAV) ou vidéo(AVI).***

Note: La taille des fichiers est limité à 64Ko.

Bon, en premier lieu, il faut le charger. pour cela, allez dans le menu "Compléments" et choisissez la commande "Gestionnaire de compléments". Ensuite, placez vous sur la

ligne "Éditeur de ressources" et cochez la case "Charger/Décharger" en bas à droite. Voilà, à présent, vous pouvez remarquer dans la barre d'outils, une icône avec des ptits carrés verts, c'est l'éditeur de ressources.

Lorsque vous avez cliqué sur cet icône, la fenêtre de l'éditeur de ressource apparaît. Vous pouvez constater qu'il est vide. Normal, vous n'y avez encore rien mis. Passons au chose sérieuse!
A droite de la barre d'outils, vous pouvez remarquer la présence de 5 icônes permettant de charger la ressource voulue. Chargeons par exemple des chaînes. Cliquez sur l'icône "Modifier les tables d'une chaîne...". Vous verrez alors apparaître un tableau. La première colonne contient le numéro d'identification pour chacune des ressources. La deuxième colonne contient les chaînes de caractère en français. Vous pouvez rajouter d'autres colonnes si vous souhaitez avoir plusieurs langues. Les numéros de la première colonne correspondent à la langue utilisée par votre ordinateur(LCID). Par exemple, supposons que vous avez placé deux colonnes contenant des chaînes en français et en anglais. Si la langue utilisé par votre ordinateur est le français, alors, ce sera les ressources en français qui sera utilisé. par contre, si la langue utilisée par votre ordinateur est l'anglais, alors, ce sera les ressources contenant les chaînes en anglais qui sera utilisé. Remarquez, si la langue utilisée par votre PC ne fait pas partie de vos ressources, alors, ce sera la première colonne qui sera utilisé.

Passons à la pratique maintenant!
Voici les fonctions qui permettent d'utiliser ces ressources dans votre projet. -Pour charger une chaîne, vous utiliserez la fonction *LoadResString(id)* où "id" correspond au numéro de la chaîne contenue dans l'éditeur de ressources.

-Pour charger un curseur, une icône ou un bitmaps, vous utiliserez la fonction *LoadResPicture(id,type)* où "id" correspond au numéro de la chaîne contenue dans l'éditeur de ressources et où "type" correspond au type de ressources que vous voulez charger.
Les différents types sont:

- **Pour les bitmaps, c'est 0 ou "vbResBitmaps"**
- **Pour les icônes, c'est 1 ou "vbResIcon"**
- **Pour les curseurs, c'est 3 ou "vbResCursor"**

-Pour charger une ressource personnalisée, vous utiliserez la fonction *LoadResData(id,type)* où "id" correspond au numéro de la chaîne contenue dans l'éditeur de ressources et où "type" correspond au type de ressources que vous voulez charger. Pour ce dernier, modifiez la propriété en écrivant par exemple "WAV" comme type pour un fichier son WAV.

Enfin, si vous voulez que l'utilisateur puisse choisir la langue à utiliser, placez toutes vos chaînes dans une seule colonne et utilisez la fonction "LoadResString" suivi de l'ID de la chaîne.

Cours 18 : Le contrôle Winsock

 **Winsock est le contrôle idéal et indispensable pour toute communication entre ordinateurs. Il s'appuie principalement sur le protocole TCP/IP(Transmission Control Protocol/Internet Protocol) qui est le langage commun utilisée par la plupart des ordinateurs pour communiquer. Winsock, diminutif de Windows Sockets, est souvent utilisée pour créer des applications client/serveur. Mais il peut très bien servir à d'autres fins comme par exemple, accéder à un serveur par FTP(File Transfer Protocol). Au cas où ceux qui ne savent pas en quoi consiste les applications client/serveur, je vais vous expliquer un peu comment ça fonctionne. De manière générale, ce sont deux programmes qui communiquent en s'échangeant des informations. D'une manière plus précise, l'application client envoie une requête à l'application serveur pour demander des informations. Dans le sens inverse, le serveur après avoir reçu la requête du client, essaie de répondre à cette requête puis renvoie la réponse de la requête au serveur.**

Bon, je vous explique à présent comment fonctionne ce contrôle. Le contrôle Winsock n'est pas présent par défaut dans la boîte à outils. Faites un clic droit sur la souris et choisissez la commande "Composants". Cochez sur la case "Microsoft Winsock Control" et validez.

---

1. Les propriétés du contrôle Winsock

Pour utiliser Winsock correctement, il est très important de bien paramétrer les propriétés de ce dernier, faute de quoi, votre application ne marchera. Notez que certaines de ses propriétés ne sont accessibles que lors de l'écriture du code. Toutes les propriétés du contrôle ne sont pas expliqués. Seuls, les plus importantes le sont et les voici:

- ***BytesReceived***

Indique en octets, les données présentes dans la mémoire tampon de réception. Cette propriété n'est disponible que lors de l'exécution de l'application et n'est pas modifiable. La valeur retournée est un nombre entier

LocalHostName

Retourne le nom du serveur local. Cette propriété n'est disponible que lors de l'exécution de l'application et n'est pas modifiable. La valeur retournée est une chaîne de caractères.

LocalIP

Retourne l'adresse IP du serveur local. Cette propriété n'est disponible que lors de l'exécution de l'application et n'est pas modifiable. La valeur retournée est une chaîne de caractères.

LocalPort

Retourne le numéro du port local. Cette propriété est disponible et modifiable aussi bien lors de la conception que lors de l'exécution de l'application. La valeur retournée est un nombre entier.

Protocol

Retourne le protocole utilisé. Vous avez le choix entre le mode TCP ou UDP.

- **Le mode TCP vous permet d'établir et maintenir une connexion à un serveur donné.**
  Pour l'application Client, il vous faut le nom du serveur auquel vous vous connectez ou bien son adresse IP ainsi que le port sur lequel vous êtes connecté. Utilisez ensuite la méthode *Connect* pour vous connecter.
  Pour l'application serveur, il vous faut connaître le port sur lequel l'application Client est connecté. Faites ensuite appel à la méthode *Listen* pour chercher à savoir si l'application Client cherche à se connecter au serveur. Si c'est le cas, l'événement *ConnectionRequest* est activé et pour que la connexion entre l'application Client et celle du serveur soit maintenue, faites appel à la méthode *Accept*.
  Enfin, après que la connexion soit maintenue, faites appel aux méthodes *SendData* et *Getdata* pour envoyer et recevoir des données.
- **Le mode UDP(User Datagram Protocol) est un protocole qui n'établit pas de connexion. Ainsi, la relation de client/serveur disparaît. L'application peut donc être aussi bien cliente que serveur. La distinction n'est plus faite avec ce mode.**

RemoteHost

Retourne le nom du serveur distant. Cette propriété est disponible et modifiable aussi bien lors de la conception que lors de l'exécution de l'application.

RemotePort

Retourne le numéro du port distant. Cette propriété est disponible et modifiable aussi bien lors de la conception que lors de l'exécution de l'application. La valeur retournée est un nombre entier.

State

Retourne l'état de la connexion. Pour finir, voici les différents états possibles de la propriété "State":

ETAT	*CONSTANTE*	*VALEUR*

Fermé(par défaut)	sckClosed	0
Ouvert	sckOpen	1
A l'écoute	sckListening	2
Connexion suspendu	sckConnectionPending	3
Recherche du serveur distant	sckResolvingHost	4
Serveur distant trouvé	sckHostResolved	5
Connexion en cours	sckConnecting	6
Connecté	sckConnected	7
Fin de connexion avec l'application client	sckOpen	8
Erreur	sckError	9

2. Les méthodes du contrôle Winsock

Passons maintenant aux méthodes les plus importantes du contrôle Winsock.

- ***Accept***

Cette méthode sert à montrer que le serveur a accepté la connexion de l'application client. Mais avant, il faut que la propriété "State" du contrôle soit paramétrée à "sckListening". Enfin, cette méthode n'est utilisée que durant l'événement "ConnectionRequest".

- ***Close***

Cette méthode met fin à la connexion TCP entre les applications client et serveur.

- ***GetData***

Cette méthode permet de récupérer les données stockées dans la mémoire tampon.

- ***Listen***

Cette méthode permet au serveur d'être à l'écoute des requêtes TCP provenant de l'application client.

- ***SendData***

Cette méthode permet d'envoyer les données soit à l'application client, soit au serveur.

- ***Connect***

Cette méthode permet de se connecter à l'ordinateur distant. Elle doit être suivie des arguments RemoteHost et Remoteport en les séparant par une virgule.

3. Les événements du contrôle Winsock

Pour finir, faut que je vous montre l'utilisation des événements. Ces événements sont très utiles pour le contrôle Winsock.

- ***Close***

Cet événement se déclenche quand l'ordinateur distant clôt la connexion.

- ***Connect***

Cet événement se déclenche quand l'ordinateur distant a réussi à établir la connexion.

- ***ConnectionRequest***

Cet événement se déclenche quand le serveur reçoit une requête provenant de l'application client.

- ***DataArrival***

Cet événement se déclenche quand l'application reçoit des données.

- ***SendComplete***

Cet événement se déclenche quand l'envoi des données est terminé.

- ***SendProgress***

Cet événement se déclenche quand les données sont en cours d'envoi.

- ***Error***

Cet événement se déclenche quand une erreur apparaît.

4. Un exemple d'applications client/serveur

Il ne suffit pas de connaître les propriétés et méthodes du contrôle Winsock pour pouvoir l'utiliser correctement. Je vais donc vous expliquer un peu comment l'utiliser à l'aide d'un exemple.

- **Le code suivant est celui d'un exemple d'application client simplifié.**

```
Option Explicit
Private Sub Connexion_Click()
If Winsock.State <> sckConnected Then
Winsock.RemoteHost = "127.0.0.1"
Winsock.RemotePort = 1007
Winsock.Connect
Else
MsgBox "Vous êtes déjà connecté"
End If
End Sub

Private Sub Quitter_Click()
Winsock.Close
End
```

```
End Sub

Private Sub Recherche_Click()
If Text1.Text <> "" Then
If Winsock.State = sckConnected Then
Winsock.SendData Text1.Text
Else
MsgBox "Non connecté au serveur"
End If
Else
MsgBox "Veuillez tapez le nom!"
End If
End Sub

Private Sub Winsock_DataArrival(ByVal bytesTotal As Long)
Dim strData
Winsock.GetData strData, vbString
Text2.Text = strData
End Sub
```

Explication du code de l'application client :

La procédure "Connexion_Click()" sert à se connecter au serveur où les données que l'on cherche à savoir, sont stockées.

- **L'instruction de test "If...Then...Else...End If" sert à savoir si l'application client est déjà connecté ou pas au serveur.**
- **Dans le cas où il n'est pas connecté au serveur, eh ben on le connecte au serveur. Mais avant, il faut lui fournir des informations sur le serveur, à savoir son adresse IP("RemoteHost") et le port("RemotePort") où sera établi la connexion. Une fois ceci faite, il ne reste plus qu'à vous connecter à l'aide de la méthode *Connect*.**
- **Dans le cas où il est déjà connecté au serveur, eh ben, on affiche tout simplement un message indiquant que vous êtes déjà connecté.**

La procédure "Quitter_Click()" sert à mettre fin à la connexion avec le serveur.

- **Pour cela, faites appel à la méthode *Close*.**
- **Enfin, utilisez l'instruction *End* pour quitter l'application.**
- **N'oubliez surtout pas de mettre fin à chaque fois à votre connexion à un serveur car sinon, cela peut provoquer des erreurs.**

Une fois connectée au serveur, la procédure "Recherche_Click()" sert à faire une requête auprès du serveur afin de chercher les données que l'on cherche.

- **La première instruction de test sert à savoir si le champ de texte(Text1) est vide.**
  - **Si ce n'est pas le cas, on fait appel à une autre instruction de test pour savoir si vous êtes déjà connecté ou pas. Si vous l'êtes, alors, on fait appel à la**

méthode *SendData* Pour envoyer au serveur ce qui est contenu dans le champ "Text1" et récupérer des infos sur le mot que vous avez entrez dans ce champ. Si c'est le cas, alors un message vous indiquant que vous n'êtes pas encore connecté au serveur, est affiché.

- **Si c'est le cas, un message vous indiquant que le champ "Text1" est vide, est affiché à votre écran.**

La procédure "Winsock_DataArrival(ByVal bytesTotal As Long)" sert à recevoir les données du serveur.

- **Pour cela, on fait appel à la méthode *GetData* suivi de l'argument dans lequel les données seront stockées et enfin suivi du type de cet argument.**
- **Enfin, on affecte au champs "Text2" les données que l'on a reçu du serveur.**

Le code suivant est celui d'un exemple d'application serveur. Les données que l'on cherche, sont stockées dans une base de données que l'on accède à l'aide du contrôle "Data". Il y aussi une chose à rajouter dans la propriété du contrôle "Winsock". Dans sa propriété "Index", mettez "0". Cela correspond au nombre de personnes connectées au serveur mais aussi au nième contrôle Winsock(groupe contrôle). En effet, à chaque fois que vous l'incrémentez, un "nouveau" contrôle Winsock est chargé à l'aide de la méthode "Load".

```
Option Explicit
Private Sub Form_Load()
Label6.Caption = Winsock(0).LocalHostName
Label4.Caption = Winsock(0).LocalIP
Winsock(0).LocalPort = 1007
List1.AddItem ("A l'écoute du port:" & Winsock(0).LocalPort)
Winsock(0).Listen
End Sub

Private Sub Winsock_ConnectionRequest(index As Integer, ByVal
requestID As Long)
Dim Message As String Message = "ID de la connexion " &
requestID & " de " & Winsock(index).RemoteHostIP
List1.AddItem (Message)
index = index + 1
Load Winsock(index)
Winsock(index).Accept requestID
Label5.Caption = index
End Sub

Private Sub Winsock_DataArrival(index As Integer, ByVal
bytesTotal As Long)
Dim InData, OutData
Dim requete As String
```

```
Winsock(index).GetData InData, vbString
requete = "select from * where Nom='" & strData & "'"
Data.RecordSource = requete
strOutData = Data.Recordset.Fields("Passe")
Winsock(index).SendData OutData
End Sub

Private Sub Winsock_Close(index As Integer)
List1.AddItem ("Déconnexion de: " & Winsock(index).RemoteHostIP)
index = index - 1
Label5.Caption = index
Winsock(index).Close
EEnd Sub
```

Explication du code de l'application serveur:

La procédure "Form_Load()" va permettre au chargement de l'application, de placer dans deux contrôles "Label", le nom du serveur ainsi que son adresse IP. Ensuite, on indique au serveur sur quel port("LocalPort"), il doit se mettre en écoute. Ensuite, il est indispensable de placer le serveur à l'écoute de requêtes de connexion. Pour cela, on fait appel à la méthode *Listen*.

La procédure "Winsock_ConnectionRequest(index As Integer, ByVal requestID As Long)" sert à traiter les requêtes de connexions. L'instruction "index = index + 1" sert à indiquer qu'une nouvelle requête de connexion est faite au serveur. On charge un "nouveau" contrôle Winsock qui gère cette requête. Pour accepter cette requête, on fait appel à la méthode *Accept* suivi de l'argument "requestID".

La procédure "Winsock_DataArrival(index As Integer, ByVal bytesTotal As Long)" sert à recevoir les données provenant de l'application client et vice-versa. Pour recevoir les données, on fait appel à la méthode *GetData*. Pour envoyer des données, on fait appel à la méthode *SendData*

Enfin, la procédure "Winsock_Close(index As Integer)" sert à mettre fin à une connexion. A chaque fois, qu'une personne se déconnecte, la propriété "Index" est décrémenté de 1. Pour fermer la connexion, utilisez la méthode *Close*.