CPR Informatique

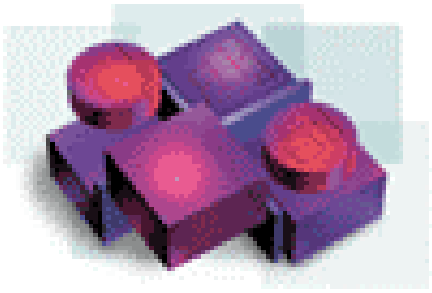
(poste 3159 ou 3164)

ECOLE NATIONALE
DES SCIENCES
GEOGRAPHIQUES

Mai 2002

www.Mcours.com
Site N°1 des Cours et Exercices Email: mymcours@gmail.com

Visual Basic v 6.0



Microsoft Visual Basic

Table des matières

1.- Les principaux concepts de Visual Basic

1.1.- présentation générale

Visual Basic est un langage de programmation « orienté objet » de Microsoft qui permet de programmer des applications indépendantes sous l'environnement Windows.

Il est intégré dans tous les logiciels de Bureautique de MicroSoft (Word, Excel, Access) sous le nom de : **VBA** (Visual Basic Application). Visual Basic est un **langage interprété**.

Pour comprendre le processus de développement d'une application, il est utile d'assimiler certains concepts sur lesquels est fondé Visual Basic. Comme il s'agit d'un langage de développement Windows, il convient également de s'être familiarisé avec l'environnement Windows.

Le fonctionnement de Windows s'articule autour de trois concepts essentiels.

- les fenêtres ;
- les événements ;
- les messages.

Considérons qu'une fenêtre est une zone rectangulaire dotée de ses propres limites. Nous connaissons tous une fenêtre de document dans un programme de traitement de texte ou une fenêtre de boite de dialogue quelconque.

S'il s'agit là des exemples les plus courants, il existe bien d'autres types de fenêtres. Un bouton de commande, les icônes, les zones de texte, les boutons d'options, les barres de menus constituent tous des fenêtres.

Windows gèrent ces nombreuses fenêtres en affectant à chacune d'elles un numéro d'identification unique (« **hWnd** »). Le système surveille en permanence chacune de ces fenêtres de façon à déceler le moindre événement. Les événement peuvent être engendrés par des actions de l'utilisateur, par un contrôle programmé, voire par des actions d'une autre fenêtre.

Chaque fois qu'un événement survient, un message est envoyé au système d'exploitation, qui traite le message et le diffuse aux fenêtres concernées.

La plupart de ces messages de bas niveau sont gérés par Visual Basic tandis que d'autres sont mis à votre disposition sous forme de procédures événementiels.

1.2- Le modèle événementiel

Dans les applications traditionnelles, dites « *procédurales* », les instructions commencent à la première ligne de code et se déroulent suivant 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 fonction des événements qui se produisent. Vous ne pouvez prévoir l'ordre des événements, en revanche vous devez prévoir les événements qui peuvent se produire et votre code doit être en mesure de les traiter.

1.3- La notion d’objet

Un objet représente un élément d'une application. Une feuille de calcul, une cellule, un graphique pour Excel, un formulaire, une table ou un état pour Access sont des objets. Dans un code Visual Basic, vous devez identifier un objet avant de pouvoir appliquer l'une des **méthodes** de l'objet ou modifier la valeur de l'une de ses **propriétés**.

Une collection est un objet contenant plusieurs autres objets, généralement, mais pas toujours, du même type. Dans Microsoft Excel, par exemple, l'objet **Workbooks** contient tous les objets Workbook. Dans Visual Basic, la collection **Forms** contient tous les objets **Form** d'une application.
Les éléments d'une collection peuvent être identifiés par numéro ou par nom.

Vous pouvez également manipuler toute une collection d'objets si les objets partagent des méthodes communes.

1.4- Les notions de méthode et de propriété

Une **méthode** est une action qu'un objet peut exécuter. Par exemple, Add est une méthode de l'objet ComboBox qui ajoute une nouvelle entrée à une liste modifiable.

Une **propriété** est un attribut d'un objet définissant l'une des **caractéristiques** de l'objet telle que la taille, la couleur ou la position à l'écran, ou un aspect de son comportement, par exemple s'il est activé ou visible. Pour changer les caractéristiques d'un objet il faut changer les valeurs de ses propriétés.

Pour définir la valeur d'une propriété, faites suivre la référence d'un objet d'un point, du nom de la propriété, d'un signe égal (=) et de la nouvelle valeur de propriété.

Certaines propriétés ne peuvent pas être définies. La rubrique d'aide de chaque propriété indique si vous pouvez la définir (lecture-écriture), seulement la lire (lecture seule) ou seulement y écrire (écriture seule). Vous pouvez extraire des informations sur un objet en renvoyant la valeur de l'une de ses propriétés.

Le chapitre 7 de ce document (« Utilisation des objets ») détaille l’ensemble de ces notions.

2.- La syntaxe de Visual Basic

Un programme écrit en Visual Basic est un ensemble de fichiers textes documentés (appelés **sources**) respectant une syntaxe précise.

Les commentaires sont des caractères, de préférence non accentués, ignorés par l'interpréteur et ne servant qu'à documenter le programme. Les lignes de commentaires débutent par une apostrophe (') ou par le mot **Rem** suivi d'un espace et peuvent être insérées n'importe où dans une procédure
Les commentaires peuvent expliquer une procédure ou une instruction particulière au lecteur de votre code. Par défaut, les commentaires s'affichent en vert.

Règles d'affectation des noms

Utilisez les règles suivantes pour nommer des procédures, des constantes, des variables et des arguments dans un module Visual Basic :

- Utilisez une lettre comme premier caractère ;
- N'utilisez pas d'espace, de point (.), de point d'exclamation (!) ou les caractères @, &, $, # dans le nom ;
- Un nom ne peut compter plus de 255 caractères ;
- Généralement, vous ne devriez pas utiliser des noms identiques aux noms de fonction, d'instruction et de méthode de Visual Basic. Vous feriez double emploi des mots clés du langage. Pour utiliser une fonction, une instruction ou une méthode du langage intrinsèque en conflit avec un nom attribué, vous devez l'identifier explicitement. Faites précéder la fonction, l'instruction ou la méthode intrinsèque du nom de la bibliothèque de types associée. Par exemple, si vous avez une variable nommée **Left**, pour invoquer la fonction **Left**, vous devez employer **VBA.Left** ;
- Vous ne pouvez pas employer deux fois le même nom au même niveau de portée. Par exemple, vous ne pouvez pas déclarer deux variables nommées **age** dans la même procédure. Cependant, vous pouvez déclarer une variable privée nommée **age** et une variable de niveau procédure nommée **age** dans le même module.

Note : Visual Basic ne différencie pas les majuscules des minuscules, mais conserve la casse dans l'instruction de déclaration du nom.

2.1.- Les types de données

2.1.1.- Les types prédéfinis

Boolean

Les variables de type « Boolean » sont stockées sous la forme de nombres de 16 bits (2 octets), mais elles ne peuvent avoir pour valeur que **True** ou **False**. Elles s'affichent sous la forme True et False (avec l'instruction Print) ou #TRUE# et #FALSE# (avec l'instruction Write #). Utilisez les mots clés True et False pour faire passer d'un état à l'autre des variables de type « Boolean ».

Lorsque d'autres types de données numériques sont convertis en valeurs de type Boolean, 0 devient False et toutes les autres valeurs deviennent True. Lorsque des valeurs de type Boolean sont converties en d'autres types de données, False devient 0 et True devient -1.

Byte

Les variables de type « Byte » sont stockées sous la forme d'un nombre de 8 bits (1 octet unique), non signé, compris entre 0 et 255.

Le type de données « Byte » est utile pour le stockage d' entiers de petite taille.

Currency

Les variables de type « Currency » sont stockées sous la forme de nombres de 64 bits (8 octets) au format entier, avec un décalage de 10 000 afin d'obtenir un nombre à virgule fixe comprenant 15 chiffres à gauche du séparateur décimal et 4 chiffres à droite. Cette représentation offre une plage comprise entre -922 337 203 685 477,5808 et 922 337 203 685 477,5807. Le caractère de déclaration de type pour les variables de type « Currency » est le signe @.

Le type de données « Currency » est utile pour les calculs monétaires et pour les calculs à virgule fixe dans lesquels la précision revêt une importance particulière.

Date

Les variables de type « Date » sont stockées sous la forme de nombres à virgule flottante de 64 bits (8 octets) IEEE représentant des dates comprises entre le 1er janvier 100 et le 31 décembre 9999, et des heures allant de 0:00:00 à 23:59:59. Toute valeur de littéral de date peut être attribuée à une variable de type « Date ». Les littéraux date doivent être délimités par le signe #, par exemple #January 1, 1993# ou #1 Jan 93#.

Les variables de type « Date » affichent les dates au format de date abrégé reconnu par votre ordinateur. Les heures s'affichent au format horaire (plage de 12 ou 24 heures) défini dans votre ordinateur.

Lorsque d'autres types de données numériques sont convertis en données de type « Date », les valeurs situées à gauche du séparateur décimal représentent la date, tandis que celles situées à droite correspondent à l'heure. Minuit est représenté par 0 et midi par 0,5. Les nombres entiers négatifs représentent des dates antérieures au 30 décembre 1899.

Decimal

Les variables de type « Decimal » sont stockées sous la forme d'entiers de 96 bits (12 octets), non signés, décalés d'une puissance de 10 variable. Le facteur de décalage (puissance de 10), qui définit le nombre de chiffres situés à droite du séparateur décimal, est compris entre 0 et 28. Avec un décalage de 0 (pas de décimales), la valeur maximale est +/- 79 228 162 514 264 337 593 543 950 335. Avec 28 décimales, la valeur maximale est +/- 7,9228162514264337593543950335 et la valeur minimale différente de zéro est +/- 0,0000000000000000000000000001.

Pour le moment, le type de données « Decimal » ne peut être utilisé qu'avec une donnée de type « Variant ». En d'autres termes, il est impossible d'attribuer à une variable le type « Decimal ».

Vous pouvez en revanche créer une variable de type « Variant » dont le sous-type est « Decimal » à l'aide de la fonction **CDec**.

Double

Les variables de type « Double » (à virgule flottante en double précision) sont stockées sous la forme de nombres à virgule flottante de 64 bits (8 octets) IEEE dont la valeur est comprise entre -1,79769313486232E308 et -4,94065645841247E-324 pour les nombres négatifs et entre 4,94065645841247E-324 et 1,79769313486232E308 pour les positifs. Le caractère de déclaration de type pour une variable de type « Double » est le signe #.

Integer

Les variables de type « Integer » sont stockées sous la forme de nombres de 16 bits (2 octets) dont la valeur est comprise entre -32 768 et 32 767. Le caractère de déclaration de type pour les variables de type « Integer » est le signe %.

Les variables de type « Integer » permettent également de représenter des valeurs énumérées. Celles-ci peuvent contenir un ensemble fini d'entiers uniques possédant tous une signification particulière dans le contexte où ils sont utilisés. Elles permettent d'opérer facilement une

sélection parmi un nombre connu de choix, du type noir = 0, blanc = 1, etc. Il est conseillé de définir des constantes pour chaque valeur énumérée via l'instruction Const.

Long

Les variables de type « Long » (entier long) sont stockées sous la forme de nombres signés de 32 bits (4 octets) dont la valeur est comprise entre -2 147 483 648 et 2 147 483 647. Le caractère de déclaration de type pour les variables de type « Long » est le signe **&.**

Object

Les variables de type « Object » sont stockées sous la forme d'adresses 32 bits (4 octets) qui font référence à des objets. L'instruction **Set** permet d'affecter une référence d'objet à une variable déclarée comme « Object ».

Une variable déclarée comme « Object » est suffisamment flexible pour contenir une référence à n'importe quel type d'objet, mais la liaison à l'objet désigné par la variable est effectuée au moment de l'exécution. Pour obtenir une liaison au moment de la compilation, attribuez la référence d'objet à une variable déclarée avec un nom de classe spécifique.

Single

Les variables de type « Single » (à virgule flottante en simple précision) sont stockées sous la forme de nombres à virgule flottante de 32 bits (4 octets) IEEE dont la valeur est comprise entre -3,402823E38 et -1,401298E-45 pour les nombres négatifs et entre 1,401298E-45 et 3,402823E38 pour les positifs. Le caractère de déclaration de type pour les variables de type « Single » est le point d'exclamation (**!**).

String

Il existe deux types de chaînes : les chaînes de longueur variable et les chaînes de longueur fixe.
Les chaînes de longueur variable peuvent contenir environ 2 milliards (2^31) de caractères.
Les chaînes de longueur fixe peuvent contenir de 1 à environ 64 Ko (2^16) de caractères.

Une chaîne de longueur fixe déclarée comme **Public** ne peut être utilisée dans les modules de classe. Les codes de caractères de type « String » sont compris entre 0 et 255. Les 128 premiers caractères (0 à 127) du jeu de caractères correspondent aux lettres et symboles d'un clavier américain standard. Ces 128 caractères sont identiques à ceux du jeu de caractères ASCII. Les 128 caractères suivants (128 à 255) représentent des caractères spéciaux, comme les lettres de certains alphabets, les accents, les symboles monétaires et les fractions. Le caractère de déclaration de type pour les variables de type String est le signe **$**.

Variant

« Variant » est le type de données attribué à toutes les variables qui ne sont pas explicitement déclarées comme étant d'un autre type (à l'aide d'instructions telles que **Dim**, **Private**, **Public** ou **Static**). Le type de données « Variant » ne possède aucun caractère de déclaration de type.

Variant est un type de données spécial pouvant contenir des données de toutes sortes, à l'exception des données de type « String » de longueur fixe et de types définis par l'utilisateur. Une variable de type « Variant » peut également contenir les valeurs **Empty**, **Error**, **Nothing** et **Null**. Vous pouvez déterminer la procédure de traitement d'une donnée de type « Variant » à l'aide des fonctions **VarType** et **TypeName**.

Les données numériques correspondent à n'importe quel nombre entier ou réel dont la valeur est comprise entre -1,797693134862315E308 et -4,94066E-324 pour les négatifs et entre 4,94066E-324 et 1,797693134862315E308 pour les positifs. En général, les données numériques de type « Variant » conservent leur type de données original au sein de la valeur de type « Variant ». Par exemple, si vous attribuez une valeur de type « Integer » à une donnée de type « Variant », cette dernière sera ultérieurement traitée comme une valeur de type « Integer ». Cependant, si une opération arithmétique est exécutée sur une donnée de type « Variant » contenant une valeur de

type « Byte », « Integer », « Long » ou « Single » et si le résultat excède la plage de valeurs du type de données, le résultat se voit automatiquement affecté le type de données « Variant » immédiatement supérieur. Une donnée de type « Byte » est transformée en « Integer », une donnée de type « Integer » est transformée en « Long » et une donnée de type « Long » ou « Single » en « Double ». Une erreur se produit lorsque des variables de type « Variant » contenant des données de type « Currency », « Decimal » ou « Double » dépassent leur plage de valeurs respective.

Le type de données « Variant » peut remplacer un autre type de données lorsque vous recherchez plus de souplesse dans le traitement de celles-ci. Si une variable de type « Variant » contient des chiffres, il peut s'agir, selon le contexte, de leur valeur réelle ou de leur représentation sous forme de chaîne. Exemple :

```
Dim MyVar As Variant
MyVar = 98052
```

Dans l'exemple précédent, **MyVar** contient une représentation numérique, c'est-à-dire la valeur réelle **98052**. Les opérateurs arithmétiques peuvent être utilisés avec des variables de type « Variant » contenant des valeurs numériques ou des chaînes de caractères pouvant être interprétées comme des nombres. Si vous utilisez l'opérateur **+** pour ajouter **MyVar** à une autre donnée de type « Variant » contenant un nombre ou à une variable de type numérique, vous obtiendrez comme résultat une somme arithmétique.
La valeur **Empty** désigne une variable de type « Variant » qui n'a pas été initialisée. Une variable de type « Variant » contenant la valeur **Empty** équivaut à **0** si elle est utilisée dans un contexte numérique et à une chaîne de longueur nulle ("") dans un contexte de chaînes.
Ne confondez pas **Empty** et **Null**. La valeur **Null** indique que la variable de type « Variant » ne contient intentionnellement aucune donnée valide.
Dans une donnée de type « Variant », la valeur **Error** permet d'indiquer qu'une condition d'erreur s'est produite dans une procédure. Cependant, aucune gestion d'erreur normale de niveau application n'a lieu dans ce cas. Le programmeur ou l'application peuvent donc appliquer un certain traitement en fonction de la valeur d'erreur. Les valeurs de type **Error** sont créées par conversion de nombres réels à l'aide de la fonction **CVErr**.

2.1.2.- Les tableaux

Savoir manipuler des données isolées est intéressant mais il est parfois nécessaire de gérer des longues listes d'informations. Les tableaux sont de simples listes d'un seul type de données (un tableau d'entiers, un tableau de nombres à virgule flottante …). Visual Basic permet de créer des tableaux de longueur fixe appelés **tableaux statiques** et des tableaux de longueur variable appelés **tableaux dynamiques**.

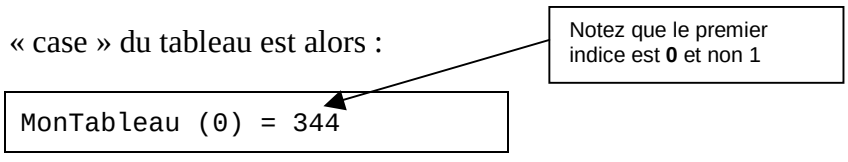
Les tableaux statiques

Il s'agit d'un tableau dont le nombre maximum d'éléments est fixé a priori.

La déclaration se fait de la manière suivante (pour un tableau « public » de 11 éléments de type « Integer ») :

```
Public MonTableau (10) As Integer
```

L'affectation de la première « case » du tableau est alors :



Les tableaux dynamiques

Par opposition aux tableaux statiques, les tableaux dynamiques n'ont pas de taille fixe. Il est possible de les « agrandir » ou de les « réduire » en intervenant sur le code et d'utiliser certaines instructions de Visual Basic pour obtenir des informations les concernant (leur taille, bornes inférieure et supérieure …). Les tableaux dynamiques doivent être déclarés « privés » dans la feuille, le module ou le module de classe où ils sont utilisés. Il ne faut pas spécifier la taille du tableau à la déclaration. Pour définir la taille du tableau il faut utiliser la fonction **ReDim**.

Un tableau dynamique se déclare comme suit :

```
Private MonTableau () As Integer
```

La taille est définie a posteriori :

```
ReDim MonTableau (10)
```

ou

```
ReDim Preserve MonTableau (10)
```

ReDim et ReDim Preserve fonctionnent différemment. ReDim définit la taille du tableau (ici 11 éléments) et supprime tout ce que contenait le tableau. ReDim Preserve définit et conserve les éléments.

2.1.3.- Les types définis par le programmeur

Ces types de données sont définis à l'aide de l'instruction **Type**. Les types de données définis par l'utilisateur peuvent contenir un ou plusieurs éléments d'un type de données, un tableau ou un type de données précédemment défini par l'utilisateur. Exemple :

```
Type MyType
      MyName As String  ' La variable de type String contient un nom
       MyBirthDate As Date    ' La variable de type Date contient une date
       de
                              naissance
      MySex As Integer  ' La variable de type Integer contient le sexe
                             (0 féminin, 1 masculin)
End Type
```

2.2.- Les instructions

Les instructions Visual Basic sont complètes. Elles contiennent des mots clés, des opérateurs, des variables, des constantes et des expressions. Chaque instruction appartient à l'une des trois catégories suivantes : déclaration, affectation, exécution.

2.2.1.- La déclaration

Les instructions de déclaration nomment une variable, une constante ou une procédure et peuvent également spécifier un type de données (Const, Dim, Private, Public, New, Static).

```
Private myVar As String
```

Ici on déclare une variable « privée » de type « String » et de nom « myVar ».

www.Mcours.com
Site N°1 des Cours et Exercices Email: mymcours@gmail.com

2.2.2.- L'affectation

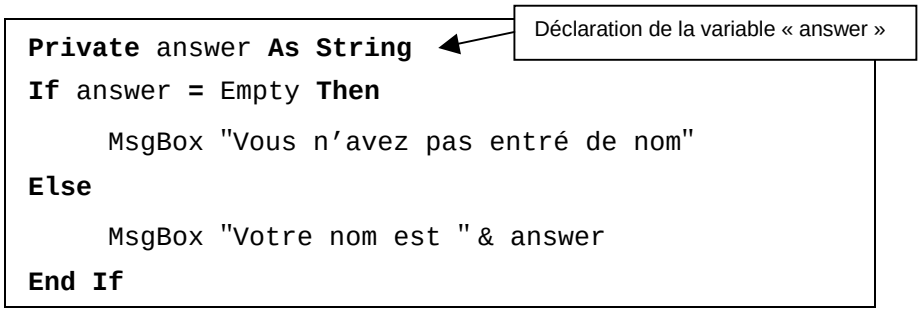
Les instructions d'affectation attribuent une valeur ou une expression à une variable ou à une constante (=, Let).

```
Title = "Zone d'accueil"
```

2.2.3.- Les instructions exécutables

Ces instructions exécutent des lignes de codes (structures alternatives, structures répétitives, débranchements, appels de modules …).

Les structures alternatives (Cf. chapitre 3.3.2)
Comme dans la plupart des langages, les mots clés correspondant à ces structures sont **If, Then, Else, End If**.




```
Private answer As String
If answer = Empty Then
     MsgBox "Vous n'avez pas entré de nom"
Else
     MsgBox "Votre nom est " & answer
End If
```

Les « If multilignes »
On utilise les mots clés « **Else If** » autant de fois que nécessaire.

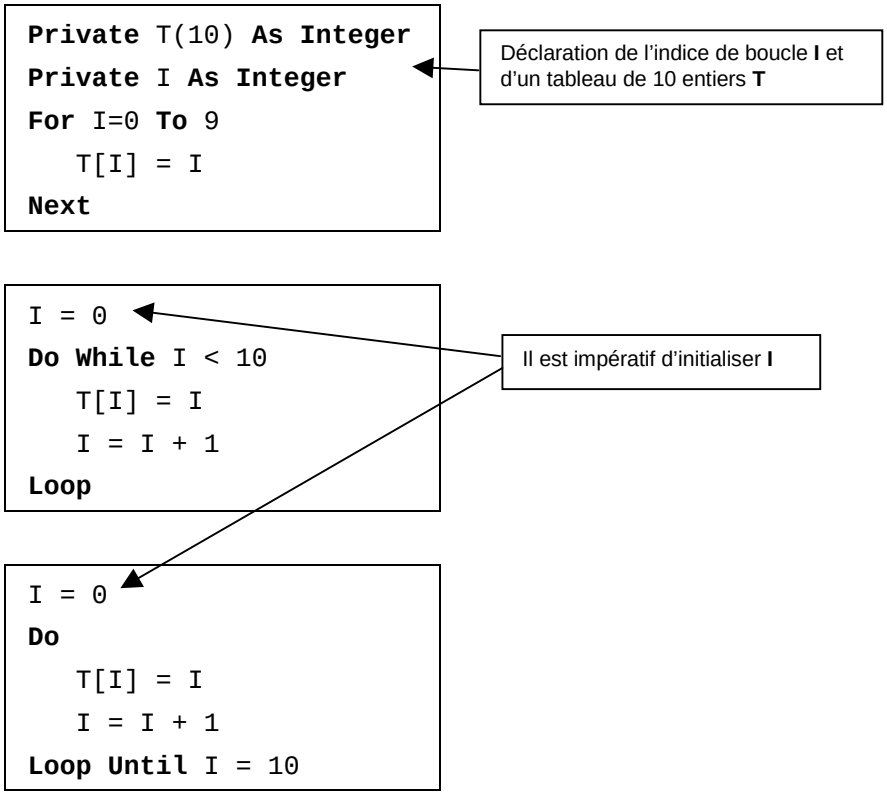
```
If answer = Empty Then
    MsgBox "Vous n'avez pas entré de nom"
Else If answer = "Toto"
    MsgBox "Votre nom est Toto"
Else
    MsgBox "Votre nom est Toto" & answer
End If
```

Le « Case »

```
Select Case answer
  Case "Toto"
     Nom = "Toto"
  Case "Titi"
     Nom = "Titi"
  Case Else
     MsgBox "Nom incohérent"
End Select
```

Les structures répétitives (Cf. chapitre 3.3.3)

Visual Basic utilise les mots clés **For, To** et **Next** pour les boucles « Pour », **Do While** et **Loop** pour la boucle « Tant que » (il existe également **While** et **Wend**), **Do** et **Loop Until** pour la boucle « Jusqu'à ce que ». Quelques exemple d'utilisation de ces boucles.



2.2.4.- Vérification des erreurs de syntaxe

Si une ligne s'affiche en rouge (avec un éventuel message d'erreur) après sa saisie et la frappe de la touche ENTRÉE, l'instruction correspondante comporte une erreur. Identifiez celle-ci et corrigez-la.

Attention : Par défaut Visual basic ne vous oblige pas à déclarer les variables, ce qui est une source d'erreur importante. Afin de remédier à cet inconvénient, il est fortement conseillé d'utiliser au début de chaque module l'instruction « **Option Explicit** » de façon à obliger VBA à détecter les variables non déclarées.

2.2.5.- Les instructions sur plusieurs lignes

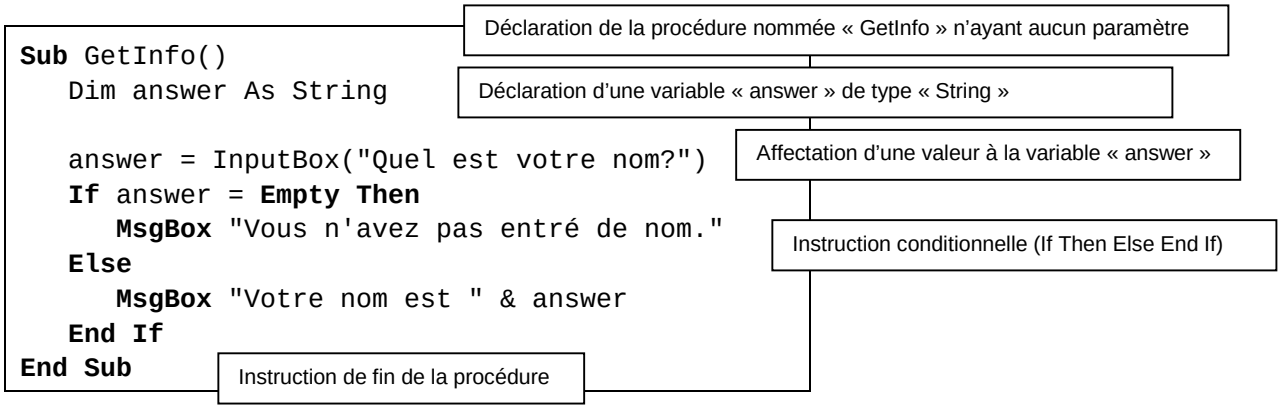
Une instruction tient généralement sur une ligne, mais il est possible de la continuer sur la ligne suivante à l'aide du caractère de continuité de ligne « _ » (blanc + blanc souligné).

```
Dim myVar As String
myVar = "John"
MsgBox Prompt:="Bonjour" & myVar, _
      Title:="Zone d'accueil", _
      Buttons:=vbExclamation
```

2.3.- Les modules

2.3.1.- Ecriture d'une procédure

Une procédure est une série d'instructions délimitée par les instructions **Sub** et **End Sub** exécutant des actions mais ne renvoyant pas de valeurs. Une procédure prend des arguments tels que des constantes, des variables ou des expressions passées par un module appelant. Si une procédure n'a pas d'argument, l'instruction **Sub** doit comporter des parenthèses vides.



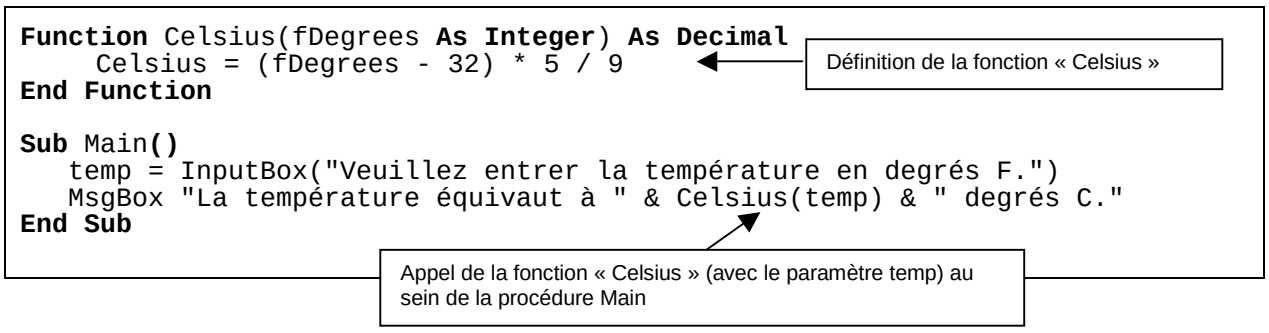

```
Sub GetInfo()
   Dim answer As String

   answer = InputBox("Quel est votre nom?")
   If answer = Empty Then
      MsgBox "Vous n'avez pas entré de nom."
   Else
      MsgBox "Votre nom est " & answer
   End If
End Sub
```

2.3.2.- Ecriture d'une fonction

Une fonction est une série d'instructions délimitée par les instructions **Function** et **End Function**. Une fonction est similaire à une procédure mais peut également renvoyer une valeur. Une fonction peut prendre des arguments, tels que les constantes, les variables ou les expressions qui lui sont passées par un module appelant. Si une fonction n'a aucun argument, son instruction **Function** doit comporter des parenthèses vides. Une fonction renvoie une valeur en affectant une valeur à son nom dans une ou plusieurs instructions de la procédure.

Dans l'exemple suivant, la fonction Celsius convertit des degrés Fahrenheit en degrés Celsius. Lorsque la fonction est appelée depuis la procédure Main, une variable contenant la valeur de l'argument est passée à la fonction. Le résultat du calcul est renvoyé à la procédure appelante et affiché dans un message.



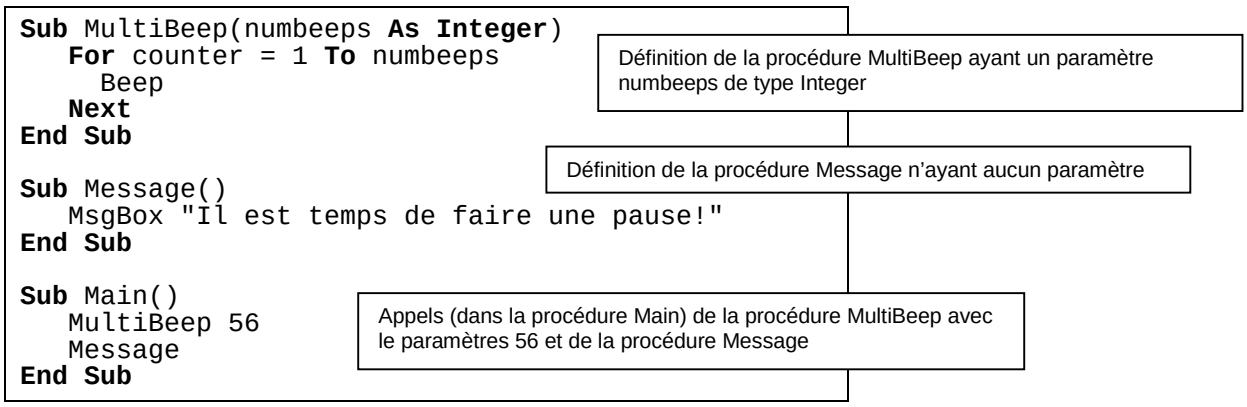

```
Function Celsius(fDegrees As Integer) As Decimal
     Celsius = (fDegrees - 32) * 5 / 9
End Function

Sub Main()
   temp = InputBox("Veuillez entrer la température en degrés F.")
   MsgBox "La température équivaut à " & Celsius(temp) & " degrés C."
End Sub
```

2.3.3.- Appel des modules

Pour appeler une procédure depuis une autre procédure, il faut simplement taper le nom de la procédure sans oublier les valeurs des arguments obligatoires.
L'instruction « **Call** » n'est pas obligatoire, mais si vous l'utilisez, vous devez mettre les éventuels arguments entre parenthèses.
Vous pouvez utiliser une procédure pour organiser d'autres procédures de manière à simplifier leur interprétation et leur débogage.

Dans l'exemple suivant, la procédure « **Main** » appelle la procédure « **MultiBeep** », passant la valeur 56 comme argument. Après l'exécution de MultiBeep, le système redonne la main à « *Main* » qui appelle la procédure « **Message** ». « Message » affiche un message ; lorsque l'utilisateur clique sur OK, la main revient à « *Main* » qui se termine.



Appel de procédures avec plusieurs arguments

L'exemple suivant illustre deux manières d'appeler une procédure avec plusieurs arguments.
Lors du second appel de HouseCalc, les arguments doivent être mis entre parenthèses parce que l'instruction « **Call** » est employée.

```
Sub Main()
     HouseCalc 99800, 43100
     Call HouseCalc(380950, 49500)
End Sub

Sub HouseCalc(price As Single, wage As Single)
     If 2.5 * wage <= 0.8 * price Then
           MsgBox "Cette maison est trop chère."
     Else
           MsgBox "Cette maison est abordable."
     End If
End Sub
```

Utilisation des parenthèses lors d'appel de fonctions

Pour utiliser la valeur renvoyée par une fonction, il faut affecter la fonction à une variable et mettre les arguments entre parenthèses, comme le montre l'exemple suivant :

```
Answer3 = MsgBox("Votre salaire vous convient-il?", 4, "Question 3")
```

Si la valeur renvoyée par une fonction ne vous intéresse pas, vous pouvez appeler une fonction de la même manière qu'une procédure, il suffit d'omettre les parenthèses, d'indiquer les arguments et de ne pas affecter la fonction à une variable, comme le montre l'exemple suivant :

```
MsgBox "Tâche terminée!", 0, "Boîte de tâche"
```

Attention **Si vous incluez des parenthèses dans l'exemple précédent, l'instruction cause une erreur de syntaxe.**

2.3.4.- Passage des paramètres

Tous les paramètres sont passés aux procédures par référence (Cf. chapitre 4.4), sauf indication contraire. Cette méthode est efficace en ce sens que le temps de passage des arguments et

www.Mcours.com
Site N°1 des Cours et Exercices Email: mymcours@gmail.com

l'espace mémoire qu'ils occupent dans une procédure (4 octets) sont les mêmes quel que soit le type de données de l'argument.
Vous pouvez également passer un argument par valeur en incluant le mot clé « **ByVal** » dans la déclaration de la procédure. Un argument passé par valeur occupe de 2 à 16 octets dans la procédure, en fonction de son type de données. Les types de données plus importants impliquent un temps de traitement supérieur à celui des types de données moins volumineux. Les types de données « **String** » et « **Variant** » ne doivent donc généralement pas être passés par valeur.

Rappel du chapitre 4.4. : lorsqu'un argument est passé par valeur, la variable d'origine est copiée dans la procédure. Les modifications apportées ensuite à l'argument au sein de la procédure ne sont pas répercutées sur la variable d'origine. Par exemple :

```
Function Factorial (ByVal MyVar As Integer) 'Déclare la fonction.
  MyVar = MyVar - 1
  If MyVar = 0 Then
     Factorial = 1
     Exit Function
  End If
  Factorial = Factorial(MyVar) * (MyVar + 1)
End Function
```

```
' Appel de la fonction Factorial avec une variable S.
S = 5
Print Factorial(S)     ' Affiche 120 (factorielle 5)
Print S          ' Affiche 5.
```

En l'absence de « **ByVal** » dans la déclaration de la fonction, les instructions « **Print** » de l'exemple précédent affichent les valeurs 1 et 0. En effet, « MyVar » fait alors référence à la variable « S », qui est décrémentée de 1 jusqu'à ce qu'elle atteigne la valeur 0.

Dans la mesure où un argument passé par la méthode « ByVal » est copié dans la procédure, vous avez la possibilité de passer un variant à la fonction Factorial de l'exemple. En effet, vous ne pouvez pas passer un variant par référence si la procédure qui déclare l'argument est d'un type de données différent.

Passage de paramètres nommés et facultatifs

Lorsque vous appelez une procédure ou une fonction, vous pouvez fournir les arguments par position, dans leur ordre d'occurrence dans la définition de la procédure, ou les fournir par nom sans respecter cette position.
Par exemple, la procédure suivante prend trois arguments :

```
Sub PassArgs (strName As String, intAge As Integer, dteBirth As Date)
   Debug.Print strName, intAge, dteBirth
End Sub
```

Vous pouvez appeler cette procédure en fournissant ses arguments à la position appropriée, chacune séparée par une virgule, comme le montre l'exemple suivant :

```
PassArgs "Mary", 26, #2-21-69#
```

Vous pouvez également appeler cette procédure en fournissant des arguments nommés séparés par des virgules.

```
PassArgs intAge:=26, dteBirth:=#2/21/69#, strName:="Mary"
```

Un argument nommé est composé d'un nom d'argument suivi des signes deux-points et égal (:=), puis de la valeur de l'argument.

Les arguments nommés sont particulièrement pratiques lors de l'appel d'une procédure comportant des arguments facultatifs. Si vous utilisez des arguments nommés, il n'est pas nécessaire d'inclure des virgules pour signaler les arguments positionnels manquants. L'utilisation d'arguments nommés permet de voir plus facilement les arguments passés et omis.

Les arguments facultatifs sont précédés du mot clé « **Optional** » dans la définition de la procédure. Vous pouvez également préciser une valeur par défaut pour l'argument facultatif dans la définition de la procédure. Par exemple :

```
Sub OptionalArgs (strState As String, Optional strCountry As String = "USA")
  ...
End Sub
```

Lorsque vous appelez une procédure avec un argument facultatif, vous pouvez préciser ou non l'argument facultatif. Si vous ne le précisez pas, la valeur par défaut, le cas échéant, est employée. Si aucune valeur par défaut n'est spécifiée, l'argument prend la valeur par défaut de la variable du type spécifié.

La procédure suivante inclut un argument facultatif, la variable « varCountry ». La fonction « IsMissing » détermine si un argument facultatif a été passé à la procédure.

```
Sub OptionalArgs(strState As String, Optional intRegion As Integer, _
                 Optional strCountry As String = "USA")
   If IsMissing(intRegion) And IsMissing(strCountry) Then
      Debug.Print strState
   ElseIf  IsMissing(strCountry) Then
      Debug.Print strState, intRegion
   ElseIf  IsMissing(intRegion) Then
      Debug.Print strState, strCountry
   Else
      Debug.Print strState, intRegion, strCountry
   End If
End Sub
```

Vous pouvez appeler cette procédure en utilisant des arguments nommés comme le montrent les exemples suivants :

```
OptionalArgs strCountry:="USA", strState:="MD"
```

```
OptionalArgs strState:= "MD", intRegion:=5
```

Tableau de paramètres

Un tableau de paramètres peut être utilisé pour passer un tableau d'arguments à une procédure. Il n'est pas nécessaire de connaître le nombre d'éléments du tableau lors de la définition de la procédure.
Vous utilisez le mot clé « **ParamArray** » pour déclarer un tableau de paramètres. Le tableau doit être déclaré comme un tableau de type « Variant », et il doit être le dernier argument de la définition de la procédure.
L'exemple suivant illustre la définition d'une procédure comportant un tableau de paramètres.

```
Sub AnyNumberArgs(strName As String, ParamArray intScores() As Variant)
   Dim intI As Integer
   Debug.Print strName; "    Résultats"
   For intI = 0 To UBound(intScores())
      Debug.Print "          "; intScores(intI)
   Next intI
End Sub
```

Utilise la fonction UBound pour déterminer la limite supérieure du tableau.

Les exemples suivants indiquent comment appeler cette procédure.

```
AnyNumberArgs "Jamie", 10, 26, 32, 15, 22, 24, 16
```

```
AnyNumberArgs "Kelly", "Haut", "Bas", "Moyen", "Haut"
```

2.3.5.- Une procédure particulière : la procédure Property

Une procédure **Property** est une série d'instructions Visual Basic permettant à un programmeur de créer et de manipuler des propriétés personnalisées.

- Les procédures Property peuvent être utilisées pour créer des propriétés en lecture seule pour des feuilles, des modules standard et des modules de classe ;
- Les procédures Property doivent être utilisées à la place des variables « Public » dans un code devant être exécuté lors de la définition de la valeur de la propriété ;
- Contrairement aux variables « Public », des chaînes d'aide peuvent être affectées aux procédures Property dans l'Explorateur d'objets.

Lorsque vous créez une procédure Property, elle devient une propriété du module contenant la procédure. Visual Basic offre les trois types de procédure Property suivantes :

Property Let	Procédure définissant la valeur d'une propriété.
Property Get	Procédure renvoyant la valeur d'une propriété.
Property Set	Procédure définissant une référence à un objet.

La syntaxe de déclaration d'une procédure Property est la suivante :

```
[Public | Private] [Static] Property {Get | Let | Set}
propertyname[(arguments)] [As type]
   traitements
End Property
```

Les procédures Property sont généralement employées par paires : Property Let avec Property Get et Property Set avec Property Get. La déclaration d'une procédure Property Get seule équivaut à la déclaration d'une propriété en lecture seule. L'utilisation simultanée des trois types de procédure Property n'est adaptée qu'aux variables de type Variant, puisque ce type de variable peut contenir un objet ou tout autre type de données. Property Set est destinée aux objets, Property Let ne l'est pas.

Les arguments requis dans les déclarations des procédures Property sont :

Property Get	Property Get propname(1, …, n) As type
Property Let	Property Let propname(1, …,,,, n, n+1)
Property Set	Property Set propname(1, …, n, n+1)

Du premier à l'avant-dernier (1, …, n), les arguments doivent tous partager les mêmes noms et les mêmes types de données dans toutes les procédures Property portant le même nom.

Une déclaration de procédure Property Get prend un argument de moins que les déclarations Property Let et Property Set associées. Le type de données de la procédure Property Get doit correspondre à celui du dernier argument (n+1) des déclarations Property Let et Property Set associées. Par exemple, si vous déclarez la procédure Property Let suivante, la déclaration Property Get doit utiliser des arguments portant le même nom et ayant le même type de données que les arguments de la procédure Property Let.

```
Property Let Names(intX As Integer, intY As Integer, varZ As Variant)
   'Instruction ici.
End Property
```

```
Property Get Names(intX As Integer, intY As Integer) As Variant
   'Instruction ici.
End Property
```

Le type de données de l'argument final d'une déclaration Property Set doit être soit un type « **Object** » soit un type « **Variant** ».

2.4.- Portée et durée de vie des variables

La portée se réfère à la disponibilité d'une variable, d'une constante ou d'une procédure en vue d'une utilisation par une autre procédure. Il existe trois niveaux de portée : niveau de procédure, niveau de module privé et niveau de module public.
Vous choisissez la portée d'une variable lors de sa déclaration. Il convient de déclarer toutes les variables explicitement pour éviter les erreurs de conflit d'affectation de nom entre variables de différentes portées.

2.4.1. - Définition d'une portée de niveau procédure

Une variable ou une constante définie dans une procédure n'est pas visible à l'extérieur de celle-ci. Seule la procédure contenant la déclaration de variable peut l'utiliser. Dans l'exemple suivant, la première procédure affiche un message contenant une chaîne. La deuxième procédure affiche un message vide puisque la variable « strMsg » est locale dans la première procédure.

```
Sub LocalVariable()
   Dim strMsg As String
   strMsg = "Cette variable ne peut pas être" & _
            "utilisée en dehors de cette procédure."
   MsgBox strMsg
End Sub
```

```
Sub OutsideScope()
   MsgBox strMsg
End Sub
```

2.4.2. - Définition d'une portée de niveau module privée

Vous pouvez définir des variables et constantes de niveau module dans la section Déclarations d'un module. Les variables de niveau module peuvent être publiques ou privées. Les variables publiques sont accessibles dans toutes les procédures de tous les modules d'un projet ; les

variables privées sont disponibles uniquement dans les procédures de ce module. Par défaut, les variables déclarées avec l'instruction Dim dans la section Déclarations ont une portée privée. Vous pouvez toutefois faire précéder la variable du mot clé « **Private** » pour améliorer la lisibilité de votre code.

Dans l'exemple suivant, la variable de chaîne « strMsg » est accessible dans toutes les procédures définies dans le module. Lorsque la deuxième procédure est appelée, elle affiche le contenu de la variable de chaîne « strMsg » dans une boîte de dialogue.

```
' Déclarations du module.
Private strMsg As String

Sub InitializePrivateVariable()
    strMsg = "Cette variable ne peut pas être" _
           & "utilisée à l'extérieur de ce module."
End Sub

Sub UsePrivateVariable()
    MsgBox strMsg
End Sub
```

Note : les procédures publiques dans un module standard ou dans un module de classe sont accessibles dans n'importe quel projet référant. Pour limiter au projet en cours la portée de toutes les procédures d'un module, ajoutez une instruction Option « **Private Module** » dans la section déclarations du module. Les variables et les procédures publiques seront toujours accessibles dans les autres procédures du projet en cours, mais pas dans les projets référents.

2.4.3. - Définition d'une portée de niveau module publique

Si vous déclarez une variable de niveau module comme publique, elle est accessible dans toutes les procédures du projet. Dans l'exemple suivant, la variable de chaîne « strMsg » peut être utilisée par n'importe quelle procédure de n'importe quel module du projet.

```
' Inclut dans la section Déclarations du module.
Public strMsg As String
```

Toutes les procédures sont publiques par défaut à l'exception des procédures d'événement. Lorsque Visual Basic crée une procédure d'événement, le mot clé **Private** est automatiquement inséré avant la déclaration de la procédure. Pour les autres procédures, vous devez explicitement déclarer la procédure « **Private** »si vous ne souhaitez pas la rendre publique.

À partir de projets référents, vous pouvez utiliser des procédures, des variables et des constantes publiques définies dans des modules standard ou des modules de classe. Cependant, vous devez d'abord définir une référence au projet dans lequel elles sont définies.

Les procédures, les variables et les constantes publiques définies dans des modules autres que des modules standard ou de classe, tels que des modules de feuilles ou des modules d'état, ne sont pas accessibles dans des projets référents, puisque ces modules sont privés dans leur projet de résidence.

2.4.4. - Durée de vie des variables

La période pendant laquelle une variable conserve sa valeur correspond à sa durée de vie. La valeur d'une variable peut changer pendant sa durée de vie, mais elle conserve une certaine valeur. Lorsqu'une variable perd sa portée, elle n'a plus de valeur.

Au début d'une procédure, toutes les variables sont initialisées. Une variable numérique est initialisée à zéro, une chaîne de longueur variable est initialisée à une chaîne de longueur nulle (""), et une chaîne de longueur fixe est remplie du caractère représentant le code de caractères ASCII 0, ou Chr(0). Les variables de type « Variant » prennent initialement la valeur « **Empty** ». Chaque élément d'une variable d'un type défini par l'utilisateur est initialisé comme s'il s'agissait d'une variable séparée.

Lorsque vous déclarez une variable objet, de l'espace est réservé en mémoire, mais elle prend la valeur « **Nothing** » jusqu'à ce que vous lui affectiez une référence d'objet à l'aide d'une instruction Set.
Si la valeur d'une variable n'est pas modifiée pendant l'exécution de votre code, elle conserve sa valeur initialisée jusqu'à ce qu'elle perde sa portée.
Une variable de niveau procédure déclarée avec l'instruction Dim conserve une valeur jusqu'à la fin de la procédure. Si la procédure appelle d'autres procédures, la variable conserve également sa valeur pendant l'exécution de ces procédures.

Si une variable de niveau procédure est déclarée avec le mot clé « **Static** », elle conserve sa valeur tant que le code est exécuté dans un module. À la fin de l'exécution du code, la variable perd sa portée et sa valeur. Sa durée de vie est identique à celle d'une variable de niveau module.
Une variable de niveau module diffère d'une variable statique. Dans un module standard ou un module de classe, elle conserve sa valeur jusqu'à ce que vous arrêtiez le code. Dans un module de classe, elle conserve sa valeur tant qu'existe une instance de la classe. Les variables de niveau module utilisant des ressources mémoire jusqu'à la réinitialisation de leurs valeurs, il convient de les utiliser uniquement lorsque cela est nécessaire.

Si vous incluez le mot clé « **Static** » avant une instruction « **Sub** » ou « **Function** », les valeurs de toutes les variables de niveau procédure dans la procédure sont conservées entre les appels.

2.5.- Les fichiers

Les fichiers sont assez peu utilisés en Visual Basic car il est très facile (via le mécanisme ODBC) d'accéder aux enregistrements d'une base de données créée avec le SGBD de Microsoft : Access. Néanmoins, Visual Basic met à la disposition du programmeur les modules nécessaires à la gestion des fichiers.

2.5.1.- Ouvrir un fichier

Un fichier disque existe indépendamment de votre application. Afin de lire ou d'écrire des fichiers, il faut établir une connexion entre votre code et le fichier situé sur le disque ce qui permet au système d'exploitation de situer physiquement le fichier. L'ouverture d'un fichier effectue ce travail grâce à le fonction « **Open** » dotée des paramètres corrects. Cette commande a besoin du nom du fichier, du mode d'utilisation du fichier (lecture, écriture ou mise à jour) et d'un numéro de référence (appelé **descripteur** du fichier). Voici un exemple d'ouverture du fichier « MonFichier.txt » en mode « lecture ». Ce fichier aura pour descripteur la variable « Fic » :

For Input est le mode lecture, **For Output** le mode écriture et **For Append** le mode mise à jour (écriture à la suite)

```
Private Fic As Integer
Open "MonFichier.txt" For Input As #Fic
```

La fonction « **FreeFile »** permet de récupérer un descripteur de fichier disponible (vous ne pouvez pas faire l'erreur d'ouvrir deux fois le même fichier).

```
Private Fic As Integer
Fic = FreeFile
Open "MonFichier.txt" For Input As #Fic
```

2.5.2.- Lire des données à partir d'un fichier

Cette opération est effectuée par la fonction Input dont il existe trois variantes :

- **Input (number, filenumber)** où « number » est le nombre de caractères lus et « filenumber » le descripteur du fichier. La lecture des données peut être placée dans une variable de type Variant.

```
Private VarInputFromFile As Variant
VarInputFromFile = Input (100, Fic)
```

  En utilisant la fonction **LOF**( ) il est possible lire l'ensemble du fichier.

```
Private VarInputFromFile As Variant
VarInputFromFile = Input (LOF(Fic), Fic)
```

- **Input #** suivi du descripteur du fichier et d'une liste des noms de variables dans lesquelles vous pouvez placer les données. Cette fonction suppose que les données sont séparées par des virgules.

```
Private sName, nPhoneNumber As String
Input #Fic, sName, nPhoneNumber
```

- **Line Input** lit une ligne de texte à la fois et la met dans une variable.

Pour lire le fichier tant que l'on a pas atteint la fin on peut utiliser la fonction **EOF.**
Exemple :

```
While Not EOF (fic)
    Call Lire (fic, Text)
WEnd
```

2.5.3.- Ecrire des données dans un fichier

Avant d'écrire dans un fichier vous devez l'ouvrir en « Output » ou en « Append ». Il existe de manières d'écrire dans un fichier :

- L'instruction Print écrit l'ensemble des données contenu dans une variable dans un fichier.

```
Private Text As String
Print #Fic, Text
```

- L'instruction Write permet de placer les données dans un endroit spécifique du fichier.

```
Private sName, nAdresse As String
Write #Fic, sName, nAdresse
```

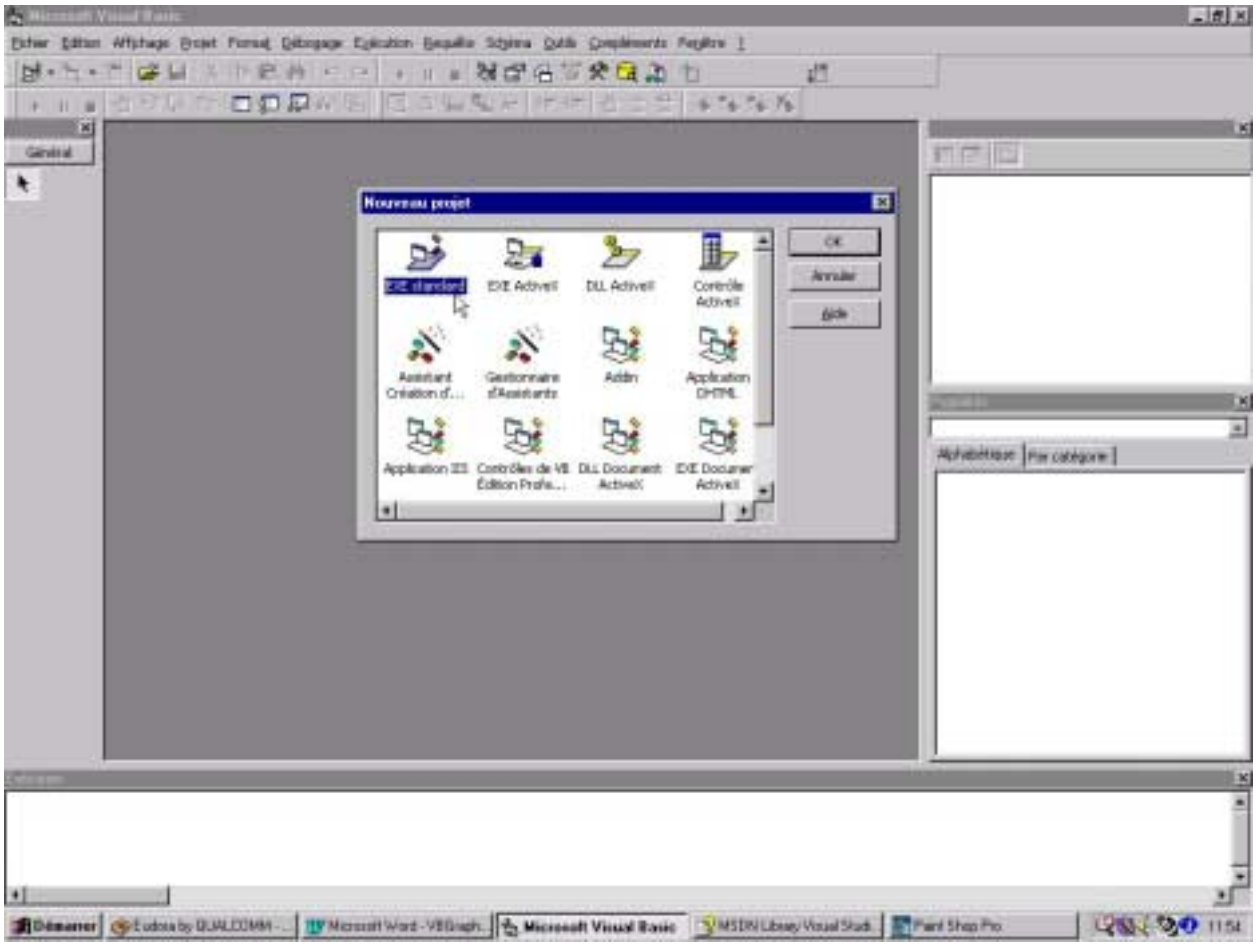
2.5.4.- Fermer un fichier

Il est possible de fermer tous les fichiers ouverts en utilisant la fonction « **Close** » ou de fermer les fichier un par un en spécifiant le descripteur « **Close #Fic** »

3.- Prise en main de l'environnement

3.1.- Démarrage de Visual Basic

Au démarrage de Visual Basic, on obtient la fenêtre suivante :

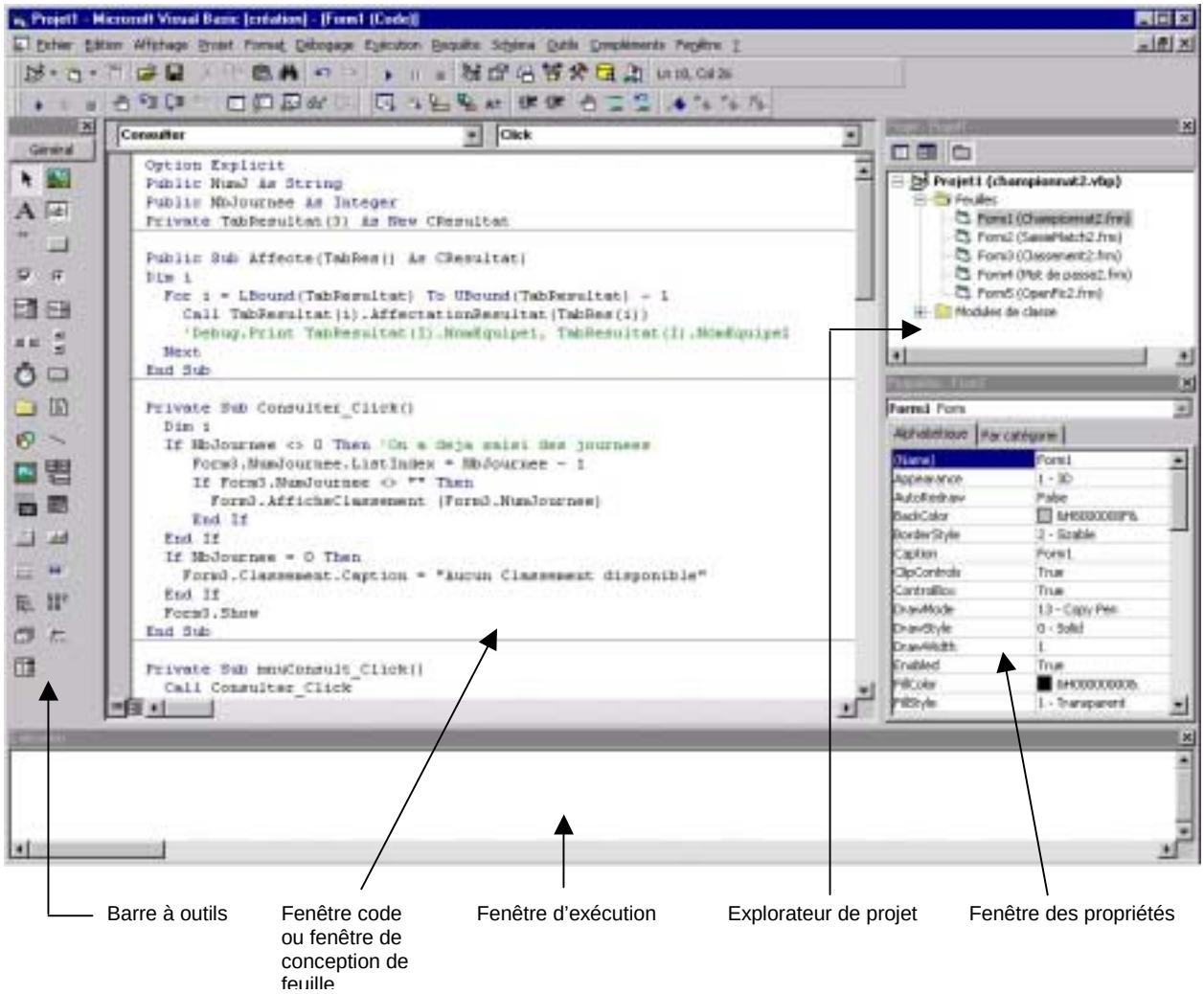


Le choix le plus couramment utilisé lors de la création d'un nouveau projet est l'application exécutable standard (EXE standard). Dans ce document, nous n'aborderons que ce type de projet.

Si vous désirez ouvrir un projet existant, cliquez sur le bouton « annuler » de la fenêtre « Nouveau projet » puis cliquez sur « Ouvrir un projet » dans le menu « Fichier ».

3.2.- Les éléments de l'environnement

L'environnement comporte sept éléments principaux : la fenêtre code, la fenêtre de conception de feuille, l'explorateur de projet, la fenêtre des propriétés, la fenêtre d'exécution, la boite à outils et l'explorateur d'objets.



Fenêtre Code

Elle fait office d'éditeur pour la saisie du code de l'application. Une fenêtre Code distincte est créée pour chaque feuille ou module de code de votre application.

Fenêtre de conception de feuille

Elle fait office de fenêtre personnalisable pour la création de l'interface de votre application. Vous ajoutez des contrôles, des graphismes et des images à une feuille de façon à ce qu'elle prenne l'apparence souhaitée. Chaque feuille de votre application possède sa propre fenêtre de conception de feuille.

Fenêtre Explorateur de projets

Cette fenêtre énumère les feuilles et les modules contenus dans votre projet en cours. Un *projet* est un ensemble de fichiers à partir desquels vous créez une application.

Fenêtre Propriétés

Cette fenêtre énumère les paramètres des propriétés de la feuille ou du contrôle sélectionné. Une *propriété* définit une caractéristique d'un objet, notamment sa taille, sa légende ou sa couleur.

Fenêtres Exécution, Variables locales et Espions

Ces fenêtres supplémentaires sont destinées au débogage de votre application. Elles ne sont disponibles que lorsque vous exécutez celle-ci dans l'environnement de développement intégré.

Boîte à outils

Elle fournit un ensemble d'outils nécessaires au moment de la création pour disposer les contrôles sur une feuille. Outre la disposition par défaut de la boîte à outils, vous pouvez créer vos propres dispositions personnalisées en cliquant sur la commande Ajouter un onglet du menu contextuel et en ajoutant ensuite les contrôles voulus à l'onglet ainsi créé.

L’explorateur d’objets

Un autre élément, l’explorateur d’objets, est très intéressant pour la conception d’application. On y accède par le menu « Affichage », « Explorateur d’objets » (touche F2).
Il énumère les objets disponibles pour votre projet et vous permet de naviguer rapidement dans votre code. Vous pouvez recourir à l'Explorateur d'objets pour examiner les objets dans Visual Basic et d'autres applications, connaître les méthodes et propriétés disponibles pour ces objets, ainsi que pour coller des procédures de code dans votre application.

Ces différents éléments seront détaillés tout au long de ce document.

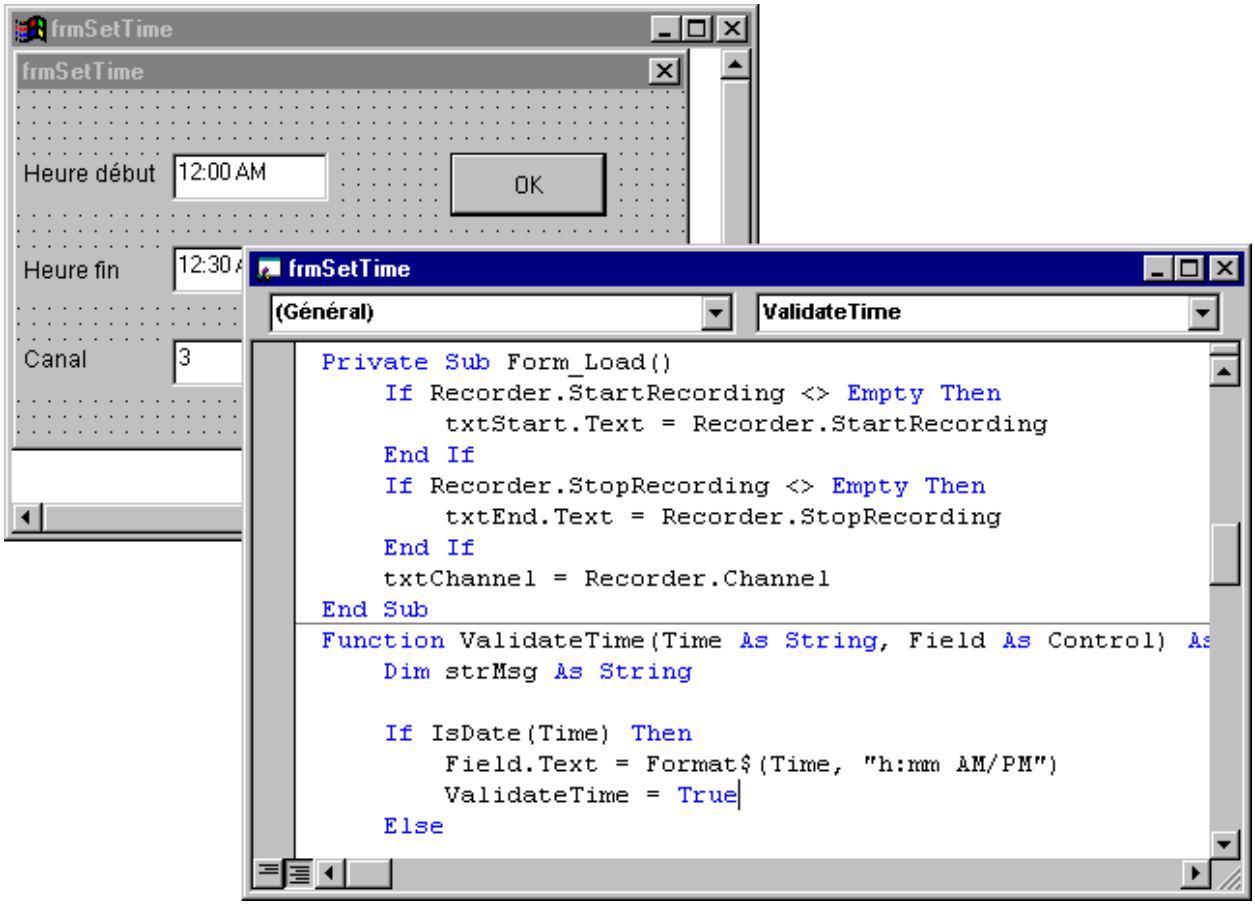
4.- Structure d'une application Visual Basic

Une application n'est rien d'autre qu'un ensemble d'instructions commandant à l'ordinateur d'exécuter une ou plusieurs tâches. La structure d'une application définit l'organisation de ces instructions, c'est-à-dire l'endroit où elles sont stockées et l'ordre dans lequel elles sont exécutées.

Les applications simples telles que l'exemple classique « Bonjour à tous » possèdent une structure rudimentaire. Quand le code ne comprend qu'une seule ligne, l'organisation n'a guère d'importance. Toutefois, plus les applications deviennent complexes, plus elles doivent être organisées et structurées. Imaginez la confusion si le code de votre application était exécuté dans un ordre aléatoire. Outre le contrôle de l'exécution d'une application, le programmeur doit aussi se soucier de sa structure. Évaluez donc la facilité avec laquelle vous retrouvez des informations spécifiques au sein de votre application !

Comme une application Visual Basic est basée sur des objets, la structure de son code se rapproche étroitement de sa représentation physique à l'écran. Par définition, les objets contiennent des données et du code. La feuille que vous voyez à l'écran correspond à une représentation des propriétés qui définissent son apparence et son comportement intrinsèque. À chaque feuille d'une application correspond un *module de feuille* connexe (dont le nom de fichier possède l'extension .FRM) **contenant son code**.

Feuille et son module de feuille (code correspondant)



Chaque module de feuille contient des *procédures d'événement*, à savoir des sections de code dans lesquelles vous placez les instructions qui sont exécutées en réponse à des événements spécifiques. Les feuilles peuvent aussi contenir des contrôles. Chaque contrôle d'une feuille est associé à un ensemble correspondant de procédures d'événement dans le module de feuille. Outre

les procédures d'événement, les modules de feuille peuvent aussi contenir des procédures générales qui sont exécutées en réponse à l'appel d'une procédure d'événement quelconque.

Le code qui n'est pas associé à une feuille ou à un contrôle spécifique peut être placé dans un autre type de module, appelé *module standard* (.bas). Une procédure susceptible d'être exécutée en réponse à des événements dans plusieurs objets différents doit être placée dans un module standard, plutôt que de dupliquer le code dans les procédures d'événement de chaque objet (règles élémentaires de la programmation structurée).

Un *module de classe* (.cls) permet de créer des objets susceptibles d'être appelés par des procédures internes à votre application. Alors qu'un module standard ne contient que du code, un module de classe contient à la fois du code et des données. Il ressemble plutôt à un contrôle dépourvu de représentation physique.

Le chapitre suivant explique, en traitant un exemple, comment écrire du code dans les différents éléments constitutifs d'une application. Par défaut, votre projet contient un module de feuille unique. Vous pouvez ajouter d'autres modules de feuille, de classe ou standard, si cela s'avère nécessaire.

5.- Apprentissage par l'exemple

La création d'une application dans Visual Basic implique trois étapes principales :

- Créer l'interface ;
- Définir les propriétés ;
- Écrire le code.

Pour mieux comprendre ces différentes étapes, les procédures suivantes vous indiquent comment créer une application simple composée d'une zone de texte et d'un bouton de commande. Lorsque vous cliquez sur celui-ci, le message « Bonjour à tous » apparaît dans la zone de texte.

5.1.- Création de l'interface

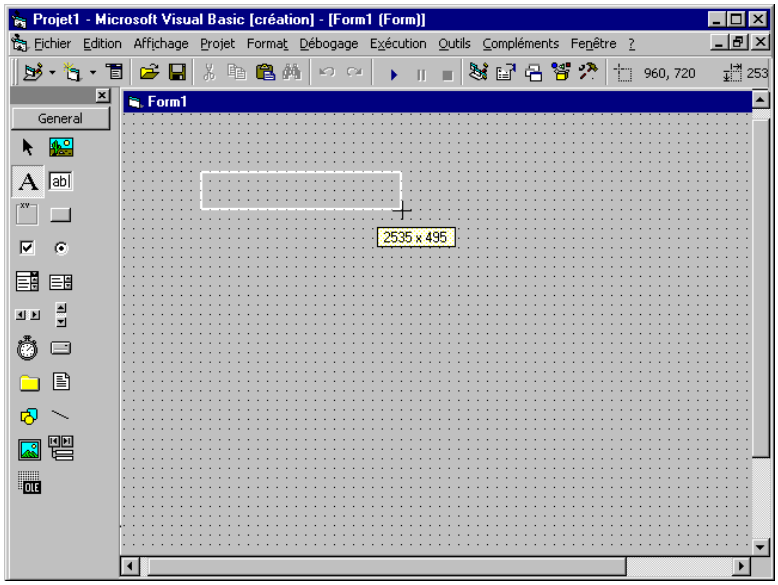
Les feuilles constituent l'élément essentiel intervenant dans la création de l'interface d'une application. Elles vous permettent d'ajouter des fenêtres et des boîtes de dialogue, mais vous pouvez également les utiliser comme conteneurs d'éléments qui ne sont pas visibles dans l'interface de l'application. Par exemple, votre application pourrait contenir une feuille qui ferait office de conteneur pour des graphiques que vous envisagez d'afficher dans d'autres feuilles.

La première étape de l'élaboration d'une application Visual Basic consiste à créer les feuilles sur lesquelles sera fondée l'interface de l'application. Vous dessinez ensuite sur les feuilles les objets qui composeront l'interface. Pour cette première application, vous utiliserez deux contrôles de la Boîte à outils.

Bouton	Contrôle
	Zone de texte (TextBox)
	Bouton de commande (CommandButton)

Dessiner un contrôle à l'aide de la boîte à outils :

Cliquez sur l'outil correspondant au contrôle que vous souhaitez dessiner, à savoir un **TextBox** et déplacer le pointeur en forme de croix jusqu'à atteindre la taille souhaitée pour le contrôle (figure ci-dessous).



Il est également possible d'ajouter facilement un contrôle à une feuille en double-cliquant sur le bouton de ce contrôle au sein de la boîte à outils. Vous créez ainsi un contrôle qui possède une taille par défaut et qui se situe au centre de la feuille. Il vous suffit ensuite de le déplacer jusqu'à l'endroit voulu.

Redimensionner un contrôle

- Cliquez sur le contrôle que vous souhaitez redimensionner ;
- Les poignées de redimensionnement apparaissent sur le contrôle ;
- Positionnez le pointeur de la souris sur une des poignées, puis faites-la glisser jusqu'à ce que le contrôle atteigne la taille souhaitée.

Déplacer un contrôle

Faites glisser le contrôle jusqu'à son nouvel emplacement sur la feuille à l'aide de la souris ou utilisez la fenêtre Propriétés pour modifier les propriétés **Top** et **Left**. Lorsqu'un contrôle est sélectionné, vous pouvez utiliser la touche CTRL avec les touches de direction pour déplacer le contrôle d'une unité de grille à la fois. Si la grille est désactivée, le contrôle est déplacé d'un pixel à la fois.

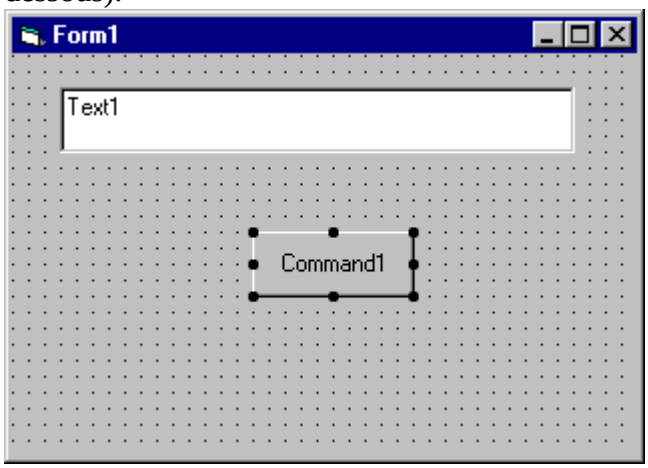
Verrouiller les positions de tous les contrôles

Dans le menu Format, cliquez sur Verrouiller les contrôles ou cliquez sur le bouton Basculer les contrôles verrouillés de la barre d'outils Fenêtre de conception de feuille. Vous verrouillez ainsi tous les contrôles de la feuille sur leur position en cours afin de ne pas risquer de les déplacer par inadvertance une fois qu'ils occupent l'emplacement désiré. Seuls les contrôles de la feuille sélectionnée sont verrouillés. Ceux des autres feuilles ne le sont pas. Comme il s'agit d'une commande à bascule, elle vous permet également de déverrouiller les positions des contrôles.

Ajuster la position des contrôles verrouillés

Vous pouvez déplacer légèrement le contrôle actif en maintenant la touche CTRL enfoncée et en appuyant sur la touche de direction appropriée ou modifiez les propriétés **Top** et **Left** du contrôle dans la fenêtre Propriétés.

Vous disposez maintenant de l'interface nécessaire à l'application « Bonjour à tous » (figure ci-dessous).



5.2. - Définition des propriétés

L'étape suivante consiste à définir les propriétés des objets que vous avez créés. La fenêtre Propriétés (ci-dessous) permet de définir facilement les propriétés de tous les objets d'une feuille. Pour ouvrir cette fenêtre, cliquez sur la commande Fenêtre Propriétés du menu Affichage, cliquez sur le bouton Fenêtre Propriétés de la barre d'outils, ou utilisez le menu contextuel du contrôle.

La fenêtre Propriétés est constituée des éléments suivants :

- **Zone Objet** : affiche le nom de l'objet dont vous pouvez définir les propriétés. Cliquez sur la flèche située à droite de la zone Objet de façon à afficher la liste des objets pour la feuille en cours ;
- **Onglets de Tri** : choisissez entre une liste alphabétique des propriétés ou une vue hiérarchique divisée en catégories logiques, notamment celles relatives à l'aspect, aux polices ou à la position ;
- **Liste Propriétés** : la colonne de gauche affiche toutes les propriétés de l'objet sélectionné. Vous pouvez modifier et afficher les valeurs dans la colonne de droite.

Définir les propriétés à partir de la fenêtre Propriétés

- Dans le menu Affichage, cliquez sur Fenêtre Propriétés, ou cliquez sur le bouton Fenêtre Propriétés de la barre d'outils ;
- La fenêtre Propriétés affiche les valeurs de la feuille ou du contrôle sélectionné ;
- Dans la liste Propriétés, sélectionnez le nom d'une propriété ;
- Dans la colonne de droite, tapez ou sélectionnez la nouvelle valeur de la propriété ;
- Les propriétés énumérées possèdent une liste de valeurs prédéfinies. Vous pouvez afficher cette liste en cliquant sur la flèche vers le bas située à droite de la zone Valeurs, à moins que vous préfériez faire défiler la liste en double-cliquant sur un de ses éléments.

Pour l'exemple d'application « Bonjour à tous », vous devez modifier les valeurs de trois propriétés et utiliser les valeurs par défaut de toutes les autres.

Objet	Propriété	Valeur
Feuille	Caption	Bonjour à tous !
Zone de texte	Text	(Vide)
Bouton de commande	Caption	OK

5.3. - Écriture du code

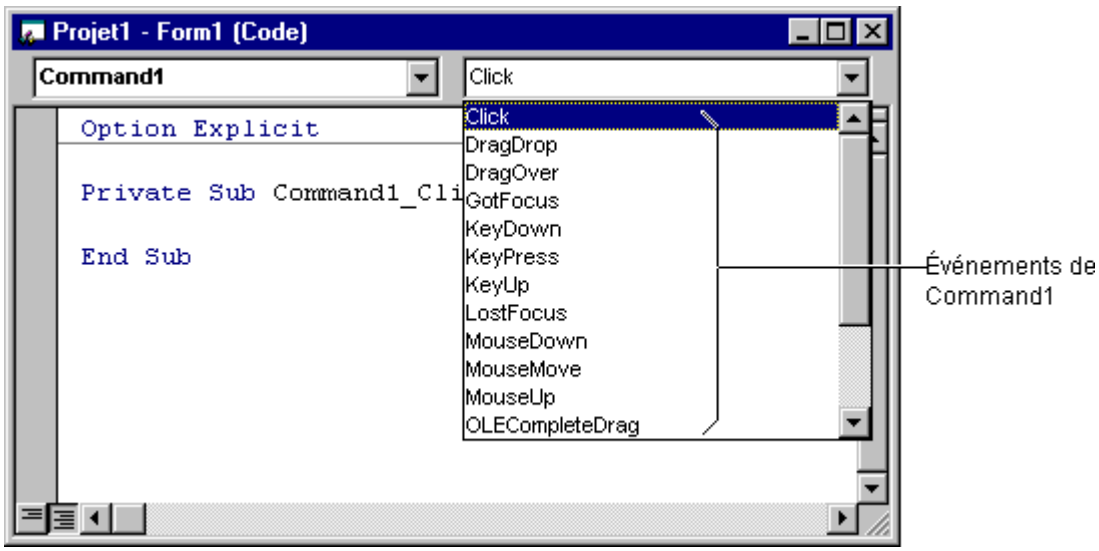
C'est dans la *fenêtre Code* que vous écrivez le code Visual Basic de votre application. Le code est composé d'instructions du langage, de constantes et de déclarations[1]. La fenêtre Code vous permet d'afficher et de modifier rapidement n'importe quelle partie du code de votre application.

Ouvrir la fenêtre Code

Double-cliquez sur la feuille ou le contrôle pour lequel vous souhaitez écrire du code ou dans la fenêtre Explorateur de projets, sélectionnez le nom d'une feuille ou d'un module, puis cliquez sur le bouton Code.

La figure ci-dessous illustre la fenêtre Code telle qu'elle apparaît lorsque vous double-cliquez sur le contrôle CommandButton, ainsi que les événements de cette commande.

Vous pouvez décider d'afficher individuellement ou simultanément les procédures dans la fenêtre Code.



Afficher toutes les procédures dans la même fenêtre Code

- Dans le menu Outils, cliquez sur Options ;
- Dans l'onglet Éditeur de la boîte de dialogue Options, activez la case à cocher située en regard de l'option « Affichage complet du module par défaut ». La case à cocher située à gauche de l'option Séparation des procédures ajoute ou supprime une ligne de séparation entre les procédures ou cliquez sur le bouton « Affichage complet du module » situé dans le coin inférieur gauche de la fenêtre Code.

[1] Pour la syntaxe du langage Visual Basic : Cf. « Initiation à la programmation », chapitre 4.6 page 26

Afficher individuellement les procédures dans la fenêtre Code

- Dans le menu Outils, cliquez sur Options ;
- Dans l'onglet Éditeur de la boîte de dialogue Options, désactivez la case à cocher située en regard de l'option « Affichage complet du module par défaut » ou cliquez sur le bouton « Affichage de procédure » situé dans le coin inférieur gauche de la fenêtre Code.

La fenêtre Code est constituée des éléments suivants :

- **Zone de liste Objet** : affiche le nom de l'objet sélectionné. Cliquez sur la flèche située sur la droite de la zone de liste pour afficher la liste de tous les objets associés à la feuille ;
- **Zone de liste Procédure** : énumère les procédures ou événements d'un objet. Cette zone affiche le nom de la procédure sélectionnée, en l'occurrence Click. Cliquez sur la flèche située à droite de la zone pour afficher toutes les procédures de l'objet.

Créer une procédure d'événement

Le code de votre application Visual Basic est divisé en blocs de petite taille appelés *procédures*. Les *procédures d'événement*, notamment celles que vous avez créées ici, contiennent du code qui est exécuté au moment où survient un événement (notamment lorsqu'un utilisateur clique sur un bouton). Une procédure d'événement d'un contrôle combine le nom effectif du contrôle (spécifié dans la propriété Name), un trait de soulignement (_) et le nom de l'événement. Par exemple, si vous souhaitez qu'une procédure d'événement soit invoquée lorsque l'utilisateur clique sur un bouton de commande nommé Command1, utilisez la procédure Command1_Click.

- Dans la zone de liste **Objet**, sélectionnez le nom d'un objet dans la feuille active. (La feuille *active* n'est autre que la feuille en cours). Pour cet exemple, cliquez sur le bouton de commande Command1 ;
- Dans la zone de liste **Procédure**, sélectionnez le nom d'un événement de l'objet sélectionné. Dans notre exemple, la procédure **Click** est déjà sélectionnée, car il s'agit de la procédure par défaut pour un bouton de commande. Notez qu'un *modèle* de la procédure d'événement est à présent affiché dans la fenêtre Code ;
- Tapez le code suivant entre les instructions **Sub** et **End Sub** :

```
Text1.Text = "Bonjour à tous !"
```

La procédure d'événement devrait se présenter comme suit :

```
Private Sub Command1_Click ()
   Text1.Text = "Bonjour à tous !"
End Sub
```

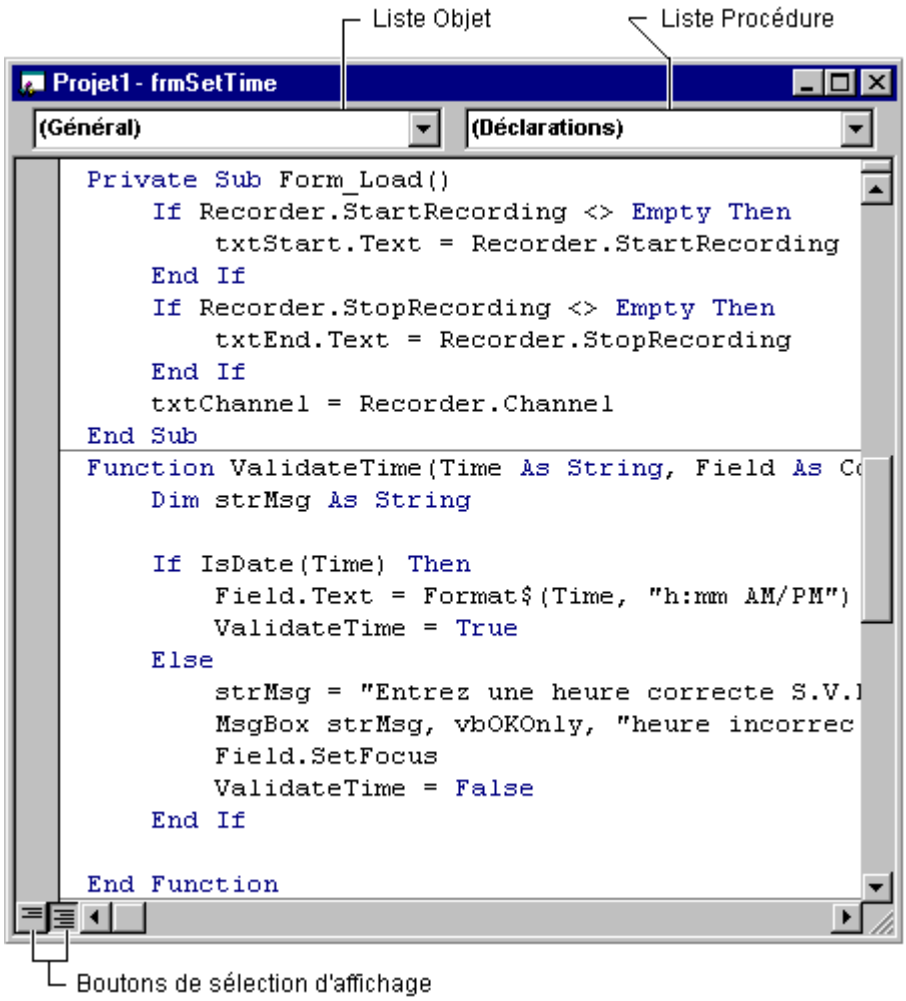
Vous remarquerez ici que le code modifie simplement la propriété Text du contrôle nommé Text1 en « Bonjour à tous ». La syntaxe de cet exemple prend la forme *object.property* dans laquelle `Text1` est l'objet et `Text` la propriété. Cette syntaxe vous permet de modifier les valeurs des propriétés de n'importe quelle feuille ou contrôle en fonction des événements qui surviennent pendant l'exécution de votre application.

5.4. - Exécution de l'application

Pour exécuter l'application, cliquez sur Exécuter dans le menu Exécution, ou cliquez sur le bouton Exécuter de la barre d'outils, ou encore appuyez sur F5. Cliquez sur le bouton de commande que vous avez créé sur la feuille et vous verrez apparaître « Bonjour à tous » dans la zone de texte.

6. - Utilisation de l'Éditeur de code

L'Éditeur de code de Visual Basic est une fenêtre dans laquelle vous écrivez la majeure partie de votre code. Semblable à un traitement de texte extrêmement sophistiqué, il offre plusieurs fonctionnalités qui facilitent considérablement l'écriture du code Visual Basic. La fenêtre de l'Éditeur de code est illustrée ci-dessous.



Comme vous manipulez le code Visual Basic dans des modules, une fenêtre de l'Éditeur de code distincte est ouverte pour chaque module que vous sélectionnez dans l'Explorateur de projets. Le code de chaque module est subdivisé en sections indépendantes pour chaque objet contenu dans le module. Vous basculez d'une section à l'autre à l'aide de la zone de liste « Objet ». Dans un module de feuille, la liste inclut une section générale, une section pour la feuille proprement dite et une section pour chaque contrôle de la feuille. Dans le cas d'un module de classe, la liste inclut une section générale et une section de classe. Enfin, le module standard ne comprend qu'une section générale.

Chaque section du code peut contenir plusieurs procédures distinctes, auxquelles vous accédez à l'aide de la liste Procédure. La liste des procédures d'un module de feuille comprend une section séparée pour chaque procédure d'événement de la feuille ou du contrôle. Par exemple, la liste des procédures d'un contrôle étiquette (Label) inclut notamment des sections pour les événements Change, Click et DblClick. Les modules de classe énumèrent uniquement les procédures d'événement de la classe proprement dite, à savoir Initialize et Terminate. Les modules standard, en revanche, ne contiennent aucune procédure d'événement, puisqu'ils ne gèrent pas d'événements.

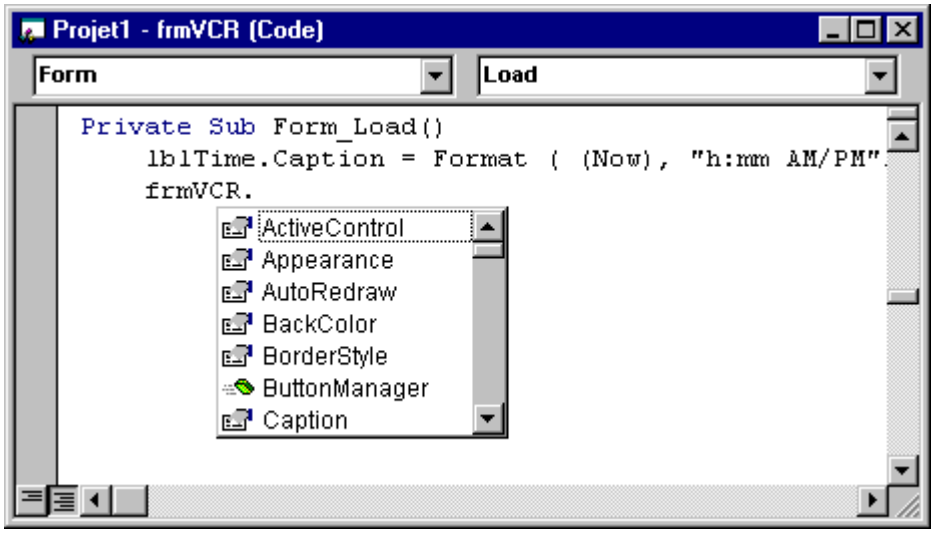
La liste des procédures de la section générale d'un module ne comprend qu'un seul élément, la section Déclarations, dans laquelle vous placez les déclarations de DLL, de variables et de constantes de niveau module. Au fur et à mesure que vous ajoutez des procédures Sub ou Function à un module, elle apparaissent dans la liste Procédure, sous la section Déclarations.

La fenêtre de l'Éditeur de code vous permet d'afficher votre code de deux manières différentes. Vous pouvez afficher une seule procédure à la fois ou afficher simultanément toutes les procédures du module, chacune d'elles étant alors séparée de la suivante par une ligne (figure ci-dessus). Pour basculer d'une vue à l'autre, utilisez les boutons de Sélection d'affichage, dans le coin inférieur gauche de la fenêtre de l'éditeur.

Achèvement automatique du code

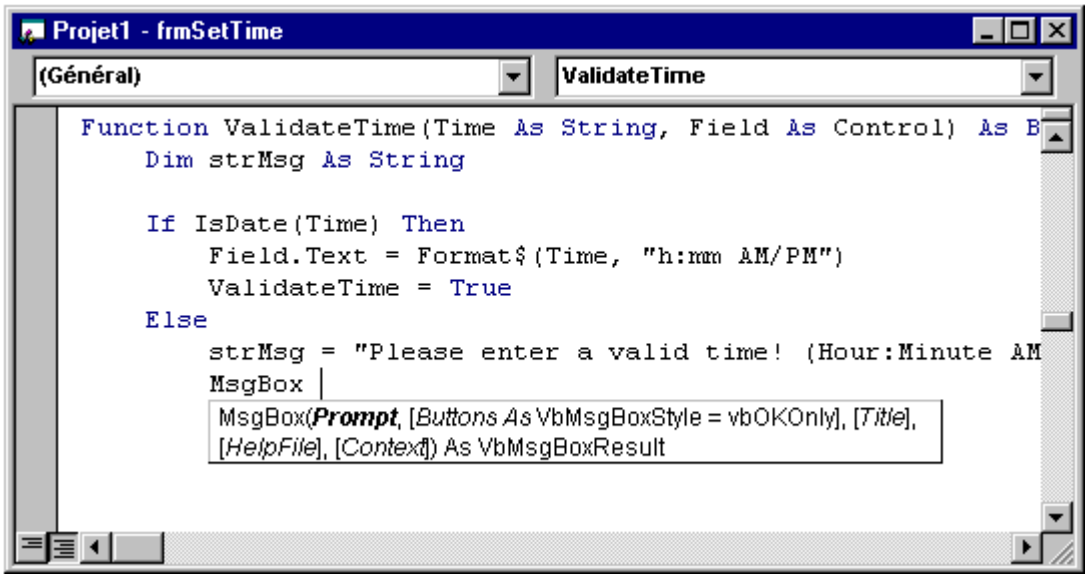
Visual Basic facilite considérablement l'écriture du code, grâce à des fonctionnalités capables de compléter automatiquement à votre place les instructions, les propriétés et les arguments. Au fur et à mesure que vous tapez le code, l'éditeur affiche les choix, les valeurs ou les prototypes de fonction ou d'instruction appropriés. Vous activez ou désactivez ces options ainsi que d'autres valeurs du code à l'aide de l'onglet Éditeur de la boîte de dialogue Options, à laquelle vous accédez à l'aide de la commande Options du menu Outils.

Quand vous spécifiez le nom d'un contrôle dans votre code, le Complément automatique des instructions affiche une liste déroulante des propriétés disponibles pour ce contrôle (figure ci-dessous). Tapez alors les premières lettres du nom de la propriété pour sélectionner automatiquement cette dernière dans la liste. La touche TAB termine automatiquement la frappe à votre place. Cette option s'avère aussi utile lorsque vous ne connaissez pas avec certitude les propriétés disponibles pour un contrôle donné. Même si vous décidez de désactiver le Complément automatique des instructions, vous pouvez toujours accéder à cette fonctionnalité à l'aide de la combinaison de touches CTRL+J.



Info express automatique

La fonctionnalité Info express automatique affiche la syntaxe des instructions et des fonctions (figure ci-dessous). Quand vous tapez le nom d'une instruction ou d'une fonction Visual Basic valide, la syntaxe apparaît immédiatement en dessous de la ligne courante tandis que le premier argument s'affiche en gras. Une fois que vous avez spécifié la valeur du premier argument, le deuxième apparaît en gras. Vous pouvez aussi accéder à la fonction Info express automatique à l'aide de la combinaison de touches CTRL+I.



Signets

Vous pouvez utiliser des signets pour marquer des lignes de code dans l'éditeur de code afin de les retrouver facilement plus tard. Les commandes de basculement des signets et de navigation avec les signets existants se trouvent dans le menu Edition, sous-menu Signets, et sur la barre d'outils Edition.

www.Mcours.com
Site N°1 des Cours et Exercices Email: mymcours@gmail.com

7. - Utilisation des objets

7.1. - Présentation des objets

Un objet est une combinaison de code et de données qui peut être traitée sous la forme d'un ensemble. Un objet peut être un élément d'une application, tel un contrôle ou une feuille. La totalité d'une application peut aussi être un objet. Le tableau suivant reprend des exemples des différents types d'objets utilisables dans Visual Basic.

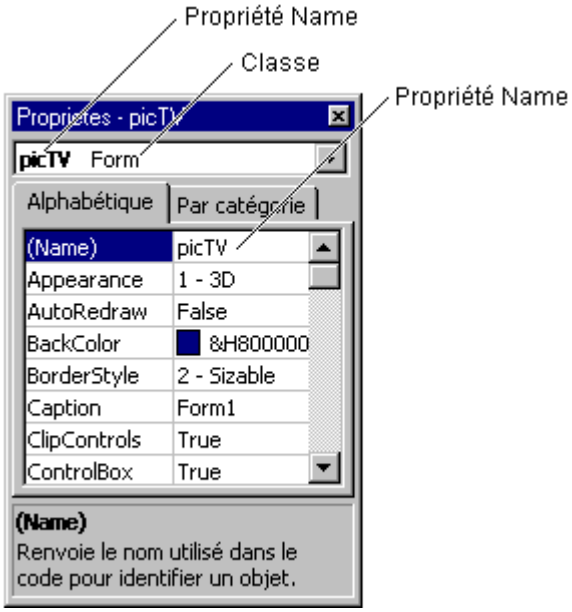
Exemple	**Description**
Bouton de commande	Les contrôles d'une feuille, notamment les boutons de commande (CommandButton) et les cadres (Frame), constituent des objets.
Feuille	Chaque feuille d'un projet Visual Basic constitue un objet distinct.
Base de données	Les bases de données (Database) sont des objets qui contiennent d'autres objets, tels que des champs et des index.
Graphique	Dans Microsoft Excel, un graphique est un objet.

7.2. - Origine des objets

Dans Visual Basic, chaque objet est défini par une *classe*. Pour mieux comprendre la relation qui existe entre un objet et sa classe, nous prendrons l'exemple des moules à gâteau et des gâteaux. Le moule à gâteau correspond à la classe. Il définit les caractéristiques de chaque gâteau, notamment sa taille et sa forme. La classe permet de créer des objets, qui sont, eux, les gâteaux.
Cette notion peut être clarifiée par deux exemples de relation entre les classes et les objets dans Visual Basic.

- Dans Visual Basic, les contrôles de la boîte à outils représentent les classes. L'objet connu comme contrôle n'existe pas avant que vous ne le dessiniez sur une feuille. Quand vous créez un contrôle, vous créez une copie ou une *occurrence* de la classe du contrôle. Cette occurrence de la classe est l'objet auquel vous faites référence dans votre application ;
- La feuille que vous utilisez au moment de la création est une classe. Au moment de l'exécution, Visual Basic crée une occurrence de la classe de la feuille.

La fenêtre Propriétés affiche la classe et la propriété Name des objets de votre application Visual Basic, comme illustré à la figure de la page suivante.



Tous les objets sont créés sous la forme de copies identiques de leur classe. Dès qu'ils existent en tant qu'objets autonomes, il est possible de modifier leurs propriétés. Par exemple, si vous dessinez trois boutons de commande sur une feuille, chaque objet bouton de commande est une occurrence de la classe CommandButton. Tous les objets ont en commun un jeu de caractéristiques et des fonctionnalités (propriétés, méthodes et événements) définies par classe. Toutefois, chaque objet possède son propre nom, peut être activé et désactivé individuellement, peut être placé en un autre endroit de la feuille, etc.

Pour simplifier les choses, il ne sera pas souvent fait référence à la classe d'un objet dans les autres chapitres. Souvenez-vous simplement que l'expression « contrôle zone de liste (ListBox) », par exemple, signifie « une occurrence de la classe ListBox ».

7.3. - Principes d'utilisation élémentaires des objets

Les objets Visual Basic prennent en charge propriétés, méthodes et événements. Dans Visual Basic, les données (valeurs ou attributs) d'un objet sont appelées propriétés, tandis que les différentes procédures qui peuvent agir sur l'objet portent le nom de méthodes. Un événement est une action qui est reconnue par un objet, notamment le clic de la souris ou l'utilisation d'une touche. Vous pouvez ainsi écrire du code qui réagit à cet événement.

Vous ne pouvez modifier les caractéristiques d'un objet en changeant ses propriétés. Prenez l'exemple d'une radio. Une de ses propriétés est son volume. Dans Visual Basic, vous pourriez dire qu'une radio possède une propriété « Volume » que vous pouvez ajuster en modifiant sa valeur. Supposez que vous régliez le volume de la radio de 0 à 10. Si vous pouviez régler une radio à partir de Visual Basic, vous pourriez écrire du code dans une procédure qui modifierait la valeur de la propriété « Volume », la faisant passer de 3 à 5, pour que le niveau sonore de la radio soit plus élevé : `Radio.Volume = 5`

Outre les propriétés, les objets possèdent des méthodes. Celles-ci font partie intégrante des objets, au même titre que les propriétés. Généralement, les méthodes correspondent à des actions que vous souhaitez exécuter, tandis que les propriétés sont les attributs que vous définissez ou que vous récupérez. Par exemple, vous composez un numéro sur le cadran d'un téléphone pour passer un appel. Vous pourriez dire que les téléphones possèdent une méthode « Composer » et vous pourriez utiliser la syntaxe suivante pour composer le numéro à sept chiffres 5551111 :

```
Phone.Dial 5551111
```

Les objets possèdent aussi des événements. Ceux-ci se déclenchent lorsque certains aspects de l'objet sont modifiés. Par exemple, une radio pourrait avoir comme événement « ChangementVolume » tandis qu'un téléphone aurait pour événement « Sonnerie ».

7.4. - Contrôle des objets au moyen de leurs propriétés

Le moment où vous pouvez définir ou renvoyer les valeurs des propriétés varie selon la propriété. Certaines peuvent être définies au moment de la création. Grâce à la fenêtre Propriétés, vous pouvez définir les valeurs de ces propriétés sans écrire le moindre code. D'autres propriétés ne sont pas disponibles au moment de la création et exigent donc que vous écriviez du code pour les définir au moment de l'exécution.

Les propriétés dont les valeurs peuvent être définies et lues au moment de l'exécution sont appelées *propriétés en lecture-écriture*. Les propriétés dont les valeurs peuvent être lues au moment de l'exécution sont appelées *propriétés en lecture seule*.

7.4.1. - Définition des valeurs de propriétés

Vous définissez la valeur d'une propriété au moment où vous modifiez l'apparence ou le comportement d'un objet. Par exemple, vous modifiez la propriété Text d'un contrôle zone de texte afin d'altérer son contenu.

Pour définir la valeur d'une propriété, utilisez la syntaxe suivante :

object.property = expression

Les instructions suivantes illustrent la manière dont vous définissez les propriétés :

```
Text1.Top = 200          ' Affecte 200 twips comme valeur de la
                         ' propriété Top.
Text1.Visible = True     ' Affiche la zone de texte.
Text1.Text = "bonjour"   ' Affiche "bonjour" dans la zone de texte.
```

7.4.2. - Renvoi des valeurs des propriétés

Vous renvoyez la valeur d'une propriété pour déterminer l'état d'un objet avant que votre code exécute d'autres actions (notamment affecter la valeur de la propriété à un autre objet). Par exemple, vous pouvez renvoyer la propriété Text d'un contrôle TextBox pour déterminer son contenu avant d'exécuter du code qui pourrait en modifier la valeur.
Généralement, pour renvoyer la valeur d'une propriété, vous utilisez la syntaxe suivante :

variable = object.property

Une valeur de propriété peut aussi être renvoyée au sein d'une expression plus complexe sans qu'il soit nécessaire d'affecter la propriété à une variable. Dans l'exemple de code suivant, la propriété Top du nouveau membre d'un groupe de contrôles est calculée comme la propriété Top du membre précédent et la valeur obtenue est majorée de 400 :

```
Private Sub cmdAdd_Click()
   ' [instructions]
   optButton(n).Top = optButton(n-1).Top + 400
   ' [instructions]
End Sub
```

7.5. - Exécution d'actions au moyen de méthodes[2]

Les méthodes peuvent affecter les valeurs des propriétés. Si nous reprenons l'exemple de la radio, nous constatons que la méthode RéglerVolume modifie la propriété Volume. De même, dans Visual Basic, les zones de liste possèdent une propriété List qui peut être modifiée au moyen des méthodes Clear et AddItem.

Quand vous utilisez une méthode dans votre code, vous écrivez l'instruction différemment selon le nombre d'arguments requis par la méthode, et selon qu'elle renvoie ou non une valeur. Quand une méthode ne prend pas d'argument, vous écrivez le code en utilisant la syntaxe suivante :

object.method

Dans cet exemple, la méthode Refresh affiche le contrôle zone d'image (PictureBox) :

```
Picture1.Refresh   ' Impose l'affichage du contrôle.
```

Certaines méthodes, notamment la méthode Refresh, ne possèdent pas d'argument et ne renvoient aucune valeur.

Si la méthode prend plusieurs arguments, vous les séparez par une virgule. Par exemple, la méthode Circle possède des arguments qui spécifient l'emplacement, le rayon et la couleur d'un cercle sur une feuille :

```
' Dessine un cercle bleu d'un rayon de 1200 twips.
Form1.Circle (1600, 1800), 1200, vbBlue
```

Si vous conservez la valeur de renvoi d'une méthode, vous devez mettre les arguments entre parenthèses. Par exemple, la méthode GetData renvoie une image provenant du Presse-papiers :

```
Picture = Clipboard.GetData (vbCFBitmap)
```

En l'absence de valeur de renvoi, les arguments apparaissent sans parenthèses. Par exemple, la méthode AddItem ne renvoie pas de valeur :

```
List1.AddItem "votre nom"     ' Ajoute le texte 'votre nom'
                              ' à une zone de liste.
```

7.6. - Relations entre les objets

Lorsque vous placez deux boutons de commande sur une feuille, il s'agit de deux objets autonomes dont la propriété Name possède des valeurs distinctes (*Command1* et *Command2*). Toutefois, ces objets possèdent la même classe, à savoir CommandButton.

Ces deux objets ont en outre en commun l'appartenance à la même feuille. Comme vous l'avez vu plus haut dans ce chapitre, un contrôle placé sur une feuille est également contenu dans la feuille. Les contrôles font ainsi partie d'une hiérarchie. Pour faire référence à un contrôle, vous devrez peut-être d'abord faire référence à la feuille, exactement comme vous composez l'indicatif téléphonique d'un pays ou un indicatif interurbain avant un numéro de téléphone proprement dit.

Les deux boutons de commande ont un autre point en commun : ce sont des contrôles. Tous les contrôles possèdent des caractéristiques communes qui les différencient des feuilles et des autres objets de l'environnement Visual Basic. Les sections qui suivent expliquent la manière dont Visual Basic utilise les collections pour regrouper des objets connexes.

[2] Ce sujet est abordé plus en détails dans le document « Introduction à la programmation » - Chapitre 4.6.5.3 - p.34

7.6.1. - Hiérarchies d'objets

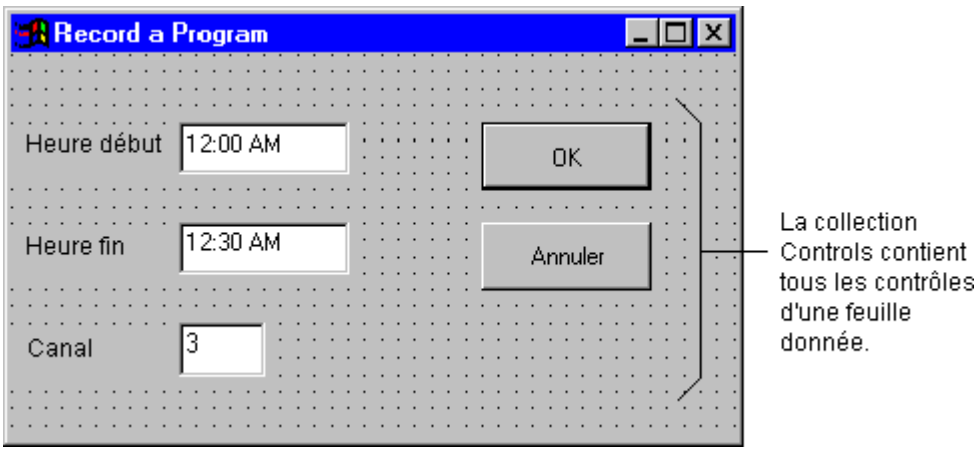
Une hiérarchie d'objets organise les objets en déterminant les relations entre eux, ainsi que leur mode d'accès. Généralement, vous ne devez pas vous préoccuper de la hiérarchie des objets Visual Basic. Toutefois :

- Quand vous manipulez des objets d'une autre application, vous devez connaître la hiérarchie de ses objets.
- Quand vous utilisez des objets d'accès aux données, vous devez également connaître la hiérarchie des objets d'accès aux données.

Dans les sections qui suivent, vous découvrirez les cas dans lesquels un objet Visual Basic en contient d'autres.

7.6.2. - Utilisation des collections d'objets

Les collections d'objets possèdent leurs propres propriétés et leurs propres méthodes. Les objets d'une collection sont appelés *membres* de la collection. Chaque membre de la collection est numéroté dans l'ordre en commençant à zéro. Ce numéro correspond au *numéro d'index* du membre. Par exemple, la collection Controls contient tous les contrôles d'une feuille donnée, comme illustré à la figure ci-dessous. Les collections vous permettent en outre de simplifier votre code si vous devez exécuter une opération qui porte sur l'ensemble des objets d'une collection.



Par exemple, le code suivant fait défiler la collection Controls et affiche le nom de chacun de ses membres dans une zone de liste.

```
Dim MyControl as Control
For Each MyControl In Form1.Controls
   ' Ajoute le nom de chaque contrôle à une zone de liste.
   List1.AddItem MyControl.Name
Next MyControl
```

7.6.3. - Application des propriétés et des méthodes aux membres d'une collection

Il existe deux moyens principaux pour désigner un membre d'un objet Collection :

- Spécifiez le nom du membre. Les expressions suivantes sont équivalentes : `Controls("List1") ou Controls!List1 ;`
- Utilisez le numéro d'index du membre : `Controls(3)`.

Quand vous pouvez désigner tous les membres collectivement et chacun des membres individuellement, vous pouvez appliquer des propriétés et des méthodes de deux manières :

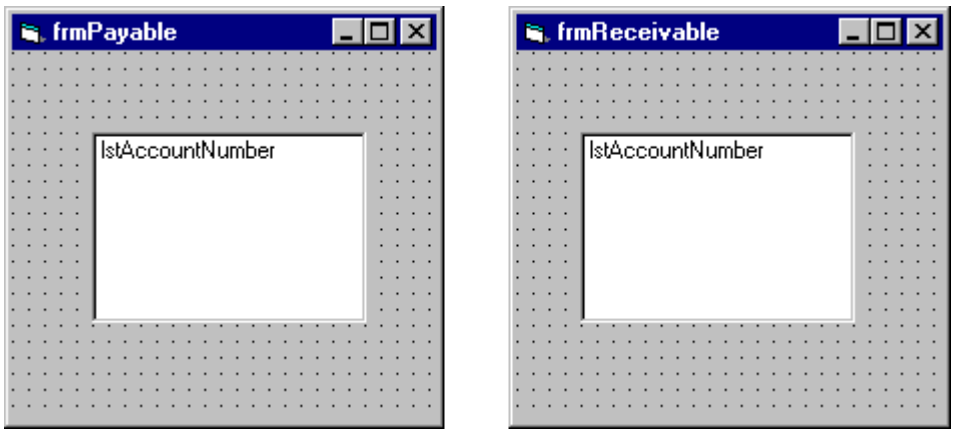
```
Controls!List1.Top = 200
```

– ou –

```
Dim MyControl as Control
For Each MyControl In Form1.Controls()
   ' Affecte la valeur 200 à la propriété Top de chaque membre.
   MyControl.Top = 200
Next MyControl
```

7.6.4. - Objets utilisés comme conteneurs d'autres objets

Dans Visual Basic, certains objets en contiennent d'autres. Par exemple, une feuille contient généralement un ou plusieurs contrôles. Les objets utilisés comme conteneurs d'autres objets offrent l'avantage que vous pouvez désigner l'objet à utiliser en faisant référence au conteneur dans votre code. Par exemple, la figure ci-dessous illustre deux feuilles différentes que vous pourriez rencontrer dans une application : une pour saisir les transactions relatives aux comptes clients et une autre pour saisir les transactions relatives aux comptes fournisseurs.



Les deux feuilles peuvent posséder une zone de liste nommée lstAcctNo. Pour spécifier exactement celle que vous souhaitez utiliser, désignez la feuille qui contient la zone de liste :

```
frmReceivable.lstAcctNo.AddItem 1201
```

– ou –

```
frmPayable.lstAcctNo.AddItem 1201
```

7.6.5. - Collections communes dans Visual Basic

Dans Visual Basic, il arrive qu'un objet en contienne d'autres. Le tableau suivant décrit brièvement les collections les plus souvent utilisées dans Visual Basic.

Collection	Description
Forms	Contient les feuilles chargées.
Controls	Contient les contrôles d'une feuille.
Printers	Contient les objets Printer disponibles.

8. - Création d'objets

Double cliquer sur un contrôle dans la boîte à outils est sans aucun doute la manière la plus simple de créer un objet. Toutefois, pour profiter pleinement de tous les objets disponibles dans Visual Basic et dans d'autres applications, vous pouvez utiliser les caractéristiques de programmation de Visual Basic et créer des objets au moment de l'exécution.

- Vous pouvez créer des références à un objet au moyen de variables objet ;
- Vous pouvez créer intégralement vos propres objets au moyen des modules de classe ;
- Vous pouvez créer vos propres collections au moyen de l'objet Collection.

8.1. - Les variables objet

8.1.1. - Utilisation des variables objet

Une variable peut non seulement stocker des valeurs, mais également faire référence à un objet. Vous affectez un objet à une variable pour les mêmes raisons que vous affectez une valeur à une variable :

- Les noms des variables sont souvent plus courts et plus faciles à mémoriser que les valeurs qu'elles contiennent (ou, dans ce cas précis, les objets auxquels elles font référence) ;
- Les variables peuvent être modifiées pour faire référence à d'autres objets pendant l'exécution de votre code ;
- La référence à une variable qui contient un objet s'avère plus efficace que de multiples références à l'objet proprement dit.

L'utilisation d'une variable objet est très similaire à celle d'une variable conventionnelle, mais nécessite en plus l'affectation d'un objet à la variable :

- Vous commencez par la déclarer : Dim *variable* As *class*
- Vous lui affectez ensuite un objet : Set *variable = object*

8.1.2. - Déclaration des variables objet

Vous déclarez une variable objet exactement comme n'importe quelle autre variable, avec Dim, ReDim, Static, Private ou Public. Les seules différences résident dans le mot clé New facultatif et dans l'argument *class* qui sont tous deux expliqués plus loin dans ce chapitre. La syntaxe est la suivante : { **Dim | ReDim | Static | Private | Public** } *variable* **As** [**New**] *class*

Par exemple, vous pouvez déclarer une variable objet qui fait référence à une feuille dans l'application appelée frmMain :

```
Dim FormVar As New frmMain   ' Déclare une variable objet
                             ' de type frmMain.
```

Vous pouvez aussi déclarer une variable objet qui fait référence à une feuille quelconque dans l'application :

```
Dim anyForm As Form          ' Variable de feuille générique.
```

De même, vous pouvez déclarer une variable objet qui fait référence à une zone de texte quelconque dans votre application :

```
Dim anyText As TextBox  ' Peut faire référence à une zone de
                        ' texte quelconque(mais uniquement
                        ' à une zone de texte).
```

Vous pouvez aussi déclarer une variable objet qui fait référence à un contrôle, quel que soit son type :

```
Dim anyControl As Control   ' Variable d'un contrôle générique.
```

Remarquez que vous pouvez déclarer une variable d'une feuille qui fait référence à une feuille donnée dans l'application, mais que vous ne pouvez pas déclarer une variable de contrôle qui fait référence à un contrôle particulier. Vous pouvez déclarer une variable de contrôle qui fait référence à un type de contrôle donné (TextBox ou ListBox, par exemple), mais pas à un contrôle déterminé de ce type (`txtEntry` ou `List1`). Vous pouvez cependant affecter un contrôle donné à une variable de ce type. Dans le cas d'une feuille dont une zone de liste est appelée lstSample, par exemple, vous pouvez écrire :

```
Dim objDemo As ListBox
Set objDemo = lstSample
```

8.1.3. - Affectation des variables objet

Vous affectez un objet à une variable objet au moyen de l'instruction Set :

Set *variable* = *object*

Utilisez l'instruction Set chaque fois que vous souhaitez qu'une variable objet fasse référence à un objet.

Vous pouvez parfois utiliser des variables objet, et plus particulièrement des variables de contrôle, simplement pour raccourcir le code que vous devez taper. Par exemple, vous pourriez écrire le code suivant :

```
If frmAccountDisplay!txtAccountBalance.Text < 0 Then
   frmAccountDisplay!txtAccountBalance.BackColor = 0
   frmAccountDisplay!txtAccountBalance.ForeColor = 255
End If
```

En recourant à une variable de contrôle, vous pourriez le raccourcir sensiblement :

```
Dim Bal As TextBox
Set Bal = frmAccountDisplay!txtAccountBalance
If Bal.Text < 0 Then
   Bal.BackColor = 0
   Bal.ForeColor = 255
End If
```

8.2. - Les modules de classes

8.3. - Les collections d’objets

9. - Boîtes de dialogue

Les boîtes de dialogue des applications Windows permettent :

- d'inviter l'utilisateur à entrer les données que l'application requiert pour poursuivre son exécution ;
- d'afficher des informations destinées à l'utilisateur.

Dans Visual Basic par exemple, la boîte de dialogue « Ouvrir un projet » permet d'afficher les projets existants. La boîte de dialogue « À propos de » illustre également la façon dont vous pouvez utiliser une boîte de dialogue pour afficher des informations. Celle-ci s'affiche chaque fois que l'utilisateur clique sur « À propos de » Microsoft Visual Basic dans le menu Aide (**?**) de la barre de menus.

9.1. - Boîtes de dialogue modales et non modales

On distingue des boîtes de dialogue modales et non modales. Une boîte de dialogue *modale* doit être fermée (masquée ou déchargée) pour que vous puissiez poursuivre votre travail. Par exemple, une boîte de dialogue est modale si elle vous demande de cliquer sur OK ou Annuler avant de passer à une autre feuille ou boîte de dialogue.

Dans Visual Basic, la boîte de dialogue « À propos de » est modale. Les boîtes de dialogue qui affichent des messages importants doivent toujours être de ce type : avant de poursuivre son travail, l'utilisateur doit toujours intervenir, soit en les fermant, soit en répondant à un message.

Les boîtes de dialogue *non modales* permettent de déplacer le focus entre la boîte de dialogue et une autre feuille sans avoir à fermer la boîte de dialogue. Vous pouvez continuer à travailler dans l'application en cours pendant que cette boîte de dialogue est affichée. Les boîtes de dialogue non modales sont assez rares. La boîte de dialogue Rechercher du menu Edition de Visual Basic en est un exemple. Utilisez ce type de boîte de dialogue pour afficher des commandes fréquemment utilisées ou des informations.

Afficher une feuille en tant que boîte de dialogue modale

Utilisez la méthode Show avec un argument *style* de vbModal (une constante de la valeur 1).

```
' Affichez frmAbout en tant que boîte de dialogue modale.
frmAbout.Show vbModal
```

Afficher une feuille en tant que boîte de dialogue non modale

Utilisez la méthode Show sans argument *style*.

```
' Affichez frmAbout en tant que boîte de dialogue non modale.
frmAbout.Show
```

Note : si une feuille est affichée en tant que boîte de dialogue modale, le code qui suit la méthode Show n'est pas exécuté tant que la boîte de dialogue n'est pas fermée. En revanche, si la feuille est une boîte de dialogue non modale, le code qui suit la méthode Show est exécuté immédiatement après l'affichage de la feuille.

La méthode Show dispose également d'un autre argument facultatif, *owner*, qui permet de spécifier pour une feuille une relation parent-enfant. Ainsi, vous pouvez passer le nom d'une feuille à cet argument pour faire en sorte que cette feuille s'approprie la nouvelle feuille.

Pour afficher une feuille en tant que fille d'une autre feuile

Utilisez la méthode Show avec les arguments *style* et *owner*.

```
' Affichez frmAbout en tant que boîte de dialogue enfant non modale de
' frmMain.
frmAbout.Show vbModeless, frmMain
```

La méthode Show utilisée conjointement à l'argument *owner* garantit la réduction simultanée de la boîte de dialogue et de son parent, ou son déchargement à la fermeture du parent.

9.2. - Utilisation de boîtes de dialogue prédéfinies

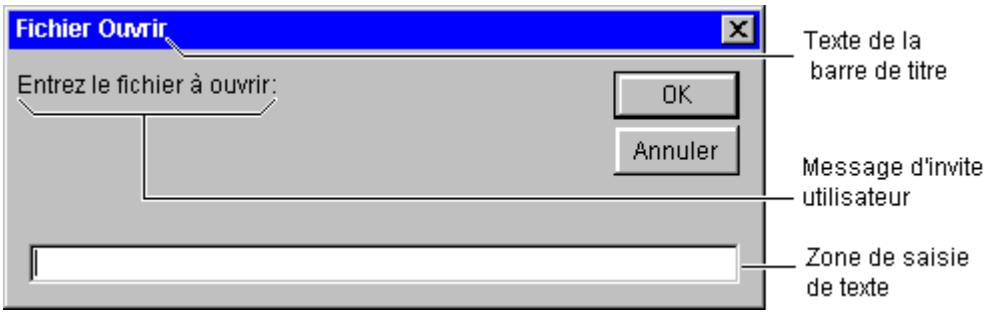
La meilleure façon d'ajouter une boîte de dialogue à votre application est d'utiliser une boîte de dialogue prédéfinie car vous n'avez pas à vous préoccuper de sa création, de son chargement ou de son affichage. Toutefois, vous disposez de peu de possibilités pour intervenir sur leur aspect. **Les boîtes de dialogue prédéfinies sont toujours modales.**

Le tableau suivant donne la liste des fonctions que vous pouvez utiliser pour ajouter des boîtes de dialogue prédéfinies dans votre application Visual Basic.

Utilisez cette fonction	Pour
InputBox	Afficher une invite dans une boîte de dialogue et renvoyer les données entrées par l'utilisateur.
MsgBox	Afficher un message dans une boîte de dialogue et renvoyer une valeur indiquant le bouton de commande sur lequel l'utilisateur a cliqué.

9.2.1. - Saisie d'une valeur à l'aide de la fonction InputBox

Utilisez la fonction InputBox pour demander à l'utilisateur de saisir certaines informations ou données par le biais d'une boîte de dialogue modale. La zone d'entrée de texte de la figure suivante invite l'utilisateur à entrer le nom du fichier qu'il veut ouvrir.



Le code suivant affiche la zone d'entrée illustrée à la figure précédente :

```
FileName = InputBox("Entrez le fichier à ouvrir :", "Fichier Ouvrir")
```

Note : si vous recourez à la fonction InputBox, n'oubliez pas que vos possibilités sont limitées quant à la modification des composants de la boîte de dialogue. Vous ne pouvez modifier que le texte de la barre de titre, l'invite destinée à l'utilisateur, la position de la boîte de dialogue sur l'écran, et afficher ou non un bouton d'aide.

9.2.2. - Affichage d'informations à l'aide de la fonction MsgBox

Utilisez la fonction MsgBox pour obtenir des réponses de type oui/non de la part des utilisateurs et afficher de courts messages dans une boîte de dialogue (messages d'erreur, d'avertissement). Après avoir lu le message, l'utilisateur clique sur un bouton pour fermer la boîte de dialogue.
Si un fichier ne peut être ouvert, l'application Éditeur de texte peut afficher la boîte de dialogue de message illustrée à la figure suivante.

Le code suivant affiche la boîte de dialogue de message de la figure précédente :

```
MsgBox "Erreur rencontrée pendant la tentative d'ouverture d'un _
   fichier, réessayez.", vbExclamation, "Éditeur de texte"
```

Note : la modalité d'une boîte de dialogue peut être limitée à l'application ou au système. Si elle est limitée à l'application (valeur par défaut), l'utilisateur ne peut pas passer à une autre partie de l'application tant que la boîte de dialogue n'est pas fermée, mais il peut passer à une autre application. Si elle est limitée au système, l'utilisateur n'est pas autorisé à passer à une autre application tant que la boîte de dialogue de message n'est pas fermée.

9.3. - Utilisation de feuilles comme boîtes de dialogue personnalisées

Une *boîte de dialogue personnalisée* est une feuille avec contrôles (boutons de commande, boutons d'option et zones de texte) que vous créez pour permettre à l'utilisateur de fournir des informations à l'application. Vous personnalisez l'aspect de la feuille en définissant des valeurs de propriétés. Vous écrivez également le code demandant l'affichage de la boîte de dialogue en mode exécution.

9.3.1. - Création

Pour créer une boîte de dialogue personnalisée, vous pouvez soit utiliser une nouvelle feuille, soit personnaliser une boîte de dialogue existante. Par la suite, vous pourrez vous constituer toute une collection de boîtes de dialogue qui vous serviront dans différentes applications.

Personnaliser une boîte de dialogue existante

- Dans le menu **Projet**, sélectionnez **Ajouter une feuille** pour ajouter une feuille existante dans votre projet ;
- Dans le menu **Fichier**, sélectionnez **Enregistrer** *nomfichier* **sous**, et entrez un nouveau nom de fichier (pour éviter de modifier la version existante de la feuille) ;
- Personnalisez l'aspect de la feuille, si nécessaire ;
- Personnalisez les procédures d'événements dans la fenêtre Code.

Créer une boîte de dialogue

- Dans le menu **Projet**, sélectionnez **Ajouter une feuille** ;
  - ou -
- Cliquez sur le bouton **Ajouter un feuille** de la barre d'outils pour créer une feuille ;

- Personnalisez l'aspect de la feuille, si nécessaire ;
- Personnalisez les procédures d'événements dans la fenêtre Code.

Vous disposez d'une grande liberté pour définir l'aspect d'une boîte de dialogue personnalisée. Elle peut être fixe ou mobile, modale ou non modale et contenir divers types de contrôles. Cependant, les boîtes de dialogue sont généralement dépourvues de barres de menus, de barres de défilement, des boutons Agrandir ou Réduire, de barres d'état et de bordures dimensionnables. La suite de cette section explique comment créer des styles de boîtes de dialogue type.

9.3.2. - Ajout d'un titre

Une boîte de dialogue doit toujours avoir un titre qui l'identifie. Pour le créer, attribuez à la propriété Caption de la feuille la chaîne de texte qui s'affichera dans la barre de titre. En général, cette opération est effectuée en mode création à l'aide de la fenêtre Propriétés, mais vous pouvez aussi définir le texte dans le code. Exemple :

```
frmAbout.Caption = "À propos de"
```

Conseil : si vous voulez supprimer complètement la barre de titre, attribuez la valeur False aux propriétés ControlBox, MinButton et MaxButton de la feuille, une valeur non dimensionnable (0, 1 ou 3) à la propriété BorderStyle et une valeur égale à une chaîne vide (" ") à la propriété Caption.

9.3.3. - Définition des propriétés d'une boîte de dialogue standard

En général, l'utilisateur fournit des informations à la boîte de dialogue puis la ferme en cliquant sur les boutons de commande OK ou Annuler. Comme une boîte de dialogue est temporaire, il n'est pas nécessaire de la déplacer, de la dimensionner, de l'agrandir ou de la réduire. Par conséquent, le style de bordure dimensionnable, la case du menu Système et les boutons Agrandir et Réduire figurant habituellement sur une nouvelle feuille sont inutiles dans la plupart des boîtes de dialogue.
Vous pouvez supprimer ces éléments en définissant les propriétés BorderStyle, ControlBox, MaxButton et MinButton. Par exemple, la boîte de dialogue À propos de utilise les valeurs suivantes :

Propriété	**Valeur**	**Effet**
BorderStyle	1	Définit une bordure simple et fixe pour empêcher le redimensionnement de la boîte de dialogue en mode exécution.
ControlBox	False	Supprime la case du menu Système.
MaxButton	False	Supprime le bouton Agrandir qui empêche l'agrandissement de la boîte de dialogue en mode exécution.
MinButton	False	Supprime le bouton Réduire qui empêche la réduction de la boîte de dialogue en mode exécution.

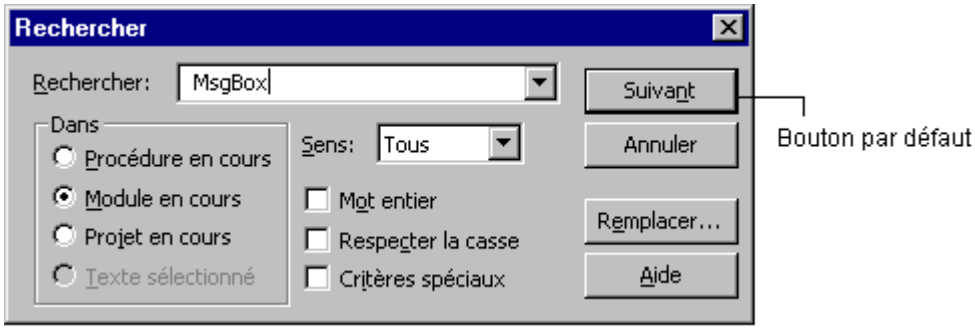
N'oubliez pas que si vous supprimez la case du menu Système (ControlBox = False), vous devrez fournir à l'utilisateur un autre moyen de quitter la boîte de dialogue. La méthode la plus courante consiste à ajouter un bouton de commande OK, Annuler ou Quitter dans la boîte de dialogue ainsi que du code dans l'événement Click du bouton qui masque ou décharge la boîte de dialogue.

9.3.4. - Ajout et positionnement des boutons de commande

Les boîtes de dialogue modales doivent contenir au moins un bouton de commande pour pouvoir être fermées. En général, deux boutons de commande sont utilisés : l'un pour permettre à l'utilisateur de lancer une action et l'autre pour lui permettre de fermer la boîte de dialogue sans apporter de modifications. OK et Annuler sont les valeurs types de la propriété Caption de ces boutons. Dans ce cas de figure, la propriété Default du bouton de commande OK et la propriété Cancel du bouton Annuler prennent la valeur True. Bien que OK et Annuler soient les boutons les plus souvent utilisés, vous pouvez utiliser d'autres combinaisons de libellés.

Les boîtes de dialogue qui affichent des messages utilisent généralement un contrôle étiquette (Label) pour afficher un message d'erreur ou une invite, et un ou deux boutons de commande pour exécuter une action. Vous pouvez par exemple affecter le message d'erreur ou l'invite à la propriété Caption de l'étiquette, et les réponses Oui et Non à la propriété Caption des deux contrôles de boutons de commande. Lorsque l'utilisateur choisit Oui, une action est exécutée et lorsqu'il choisit Non, aucune action n'est exécutée.

Les boutons de commande des boîtes de dialogue de ce type sont habituellement placés en bas ou à droite de la boîte de dialogue, le bouton du haut ou de gauche étant le bouton par défaut, comme illustré à la figure suivante.



9.3.5 - Définition des propriétés Default, Cancel et Focus

Les contrôles CommandButton offrent les propriétés suivantes : Default, Cancel, TabIndex, TabStop.

Le bouton Par défaut est sélectionné lorsque l'utilisateur appuie sur la touche ENTRÉE. La propriété Default ne peut prendre la valeur True que pour un seul bouton de commande par feuille. Appuyer sur la touche ENTRÉE appelle l'événement Click du bouton de commande par défaut. Cette caractéristique est associée à un contrôle de modification, tel qu'une zone de texte. L'utilisateur peut par exemple saisir des données dans une zone de texte puis appuyer sur ENTRÉE pour générer un événement Click au lieu de cliquer sur OK.

Le bouton Annuler est sélectionné lorsque l'utilisateur appuie sur la touche ÉCHAP. La propriété Cancel ne peut prendre la valeur True que pour un seul bouton de commande par feuille. Appuyer sur la touche ÉCHAP appelle l'événement Click du bouton de commande Annuler. Ce bouton peut aussi être le bouton de commande par défaut. Pour spécifier le bouton Annuler d'une boîte de dialogue, attribuez la valeur True à la propriété Cancel du bouton.

Conseil : en général, le bouton qui indique l'action la plus probable ou la plus sûre à exécuter doit être le bouton par défaut. Par exemple, dans une boîte de dialogue de remplacement de texte, c'est le bouton Annuler qui doit être le bouton par défaut, et non le bouton Remplacer tout.
Vous pouvez aussi indiquer le bouton qui recevra le focus pendant l'affichage de la boîte de dialogue. Le contrôle ayant la valeur TabIndex la plus faible reçoit le focus lors de l'affichage de la feuille. Appuyer sur la touche ENTRÉE appelle l'événement Click pour le bouton de

commande par défaut ou pour celui sur lequel se trouve le focus. Pour qu'un bouton de commande reçoive le focus lorsque la feuille est affichée, attribuez la valeur 0 à la propriété TabIndex du bouton de commande, et la valeur True à sa propriété TabStop. Vous pouvez aussi utiliser la méthode SetFocus pour qu'un contrôle spécifique reçoive le focus pendant l'affichage d'une feuille.

9.3.6. - Désactivation des contrôles d'une boîte de dialogue

Des contrôles doivent parfois être désactivés car leurs actions ne sont pas appropriées au contexte en cours. Par exemple, lorsque la boîte de dialogue Rechercher de Visual Basic s'affiche en premier, le bouton Rechercher suivant est désactivé, comme illustré à la figure suivante. Vous pouvez désactiver un contrôle dans une boîte de dialogue en attribuant la valeur False à sa propriété Enabled.

Désactiver un contrôle dans une boîte de dialogue

Attribuez la valeur False à la propriété Enabled de chaque contrôle.

```
cmdFindNext.Enabled = False
cmdReplace.Enabled = False
```

9.3.7. - Affichage d'une boîte de dialogue personnalisée

Vous affichez une boîte de dialogue dans une application de la même façon que toute autre feuille. La feuille de démarrage se charge automatiquement chaque fois que l'application est exécutée. Si vous voulez afficher une seconde feuille ou boîte de dialogue dans l'application, écrivez le code approprié. De même, si vous voulez que cette feuille ou boîte de dialogue ne s'affiche plus, écrivez le code qui demandera le déchargement ou le masquage.

Le code suivant affiche la boîte de dialogue À propos de, lorsque l'utilisateur sélectionne l'option À propos de du menu Aide :

```
Private Sub mnuHelpAbout_Click ()
   ' La méthode Show avec le style = vbModal est utilisée ici
   ' pour afficher la boîte de dialogue modale.
   frmAbout.Show vbModal
End Sub
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: mymcours@gmail.com

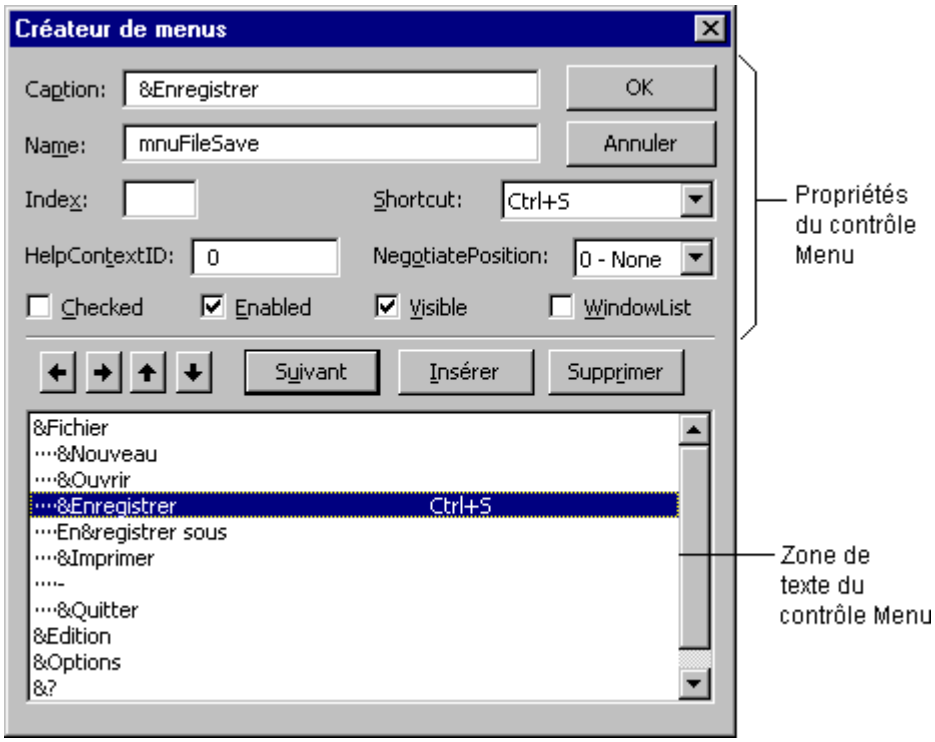
10. - Création de menus

Le Créateur de menus permet de créer des menus et des barres de menu, d'ajouter de nouvelles commandes aux menus existants, de remplacer les commandes de menu existantes par vos propres commandes, d'ajouter et de supprimer des menus et des barres de menu existants.

10.1. - Afficher le Créateur de menus

Dans le menu **Outils**, sélectionnez **Créateur de menus**.
- ou –
Cliquez sur le bouton **Créateur de menus** dans la barre d'outils.

Le Créateur de menus s'ouvre, tel qu'illustré à la figure suivante.



Alors que la plupart des propriétés des contrôles de menu peuvent être définies à l'aide du Créateur de menus, toutes les propriétés de menus s'affichent dans la fenêtre Propriétés. Les deux propriétés de contrôles de menu les plus importantes sont les suivantes :

- Name : nom utilisé pour désigner un contrôle de menu dans un code.
- Caption : texte qui s'affiche dans le contrôle.

D'autres propriétés du Créateur de menus, notamment Index, Checked et NegotiatePosition sont décrites plus bas dans ce chapitre.

10.2. - Utilisation du contrôle ListBox dans le Créateur de menus

La zone de liste des contrôles de menu (partie inférieure du Créateur de menus) contient tous les contrôles de menu de la feuille en cours. Lorsque vous saisissez un élément de menu dans la zone de texte Caption, il apparaît également dans la zone de liste des contrôles de menu. Si vous sélectionnez un contrôle de menu existant dans la zone de liste, vous pouvez modifier ses propriétés.

La figure ci-dessus illustre par exemple les contrôles d'un menu Fichier dans une application classique. La position du contrôle dans la zone de liste des contrôles de menu détermine s'il s'agit d'un titre, d'un élément de menu ou d'un élément de sous-menu :

- Un contrôle de menu qui apparaît tout à gauche de la zone de liste est un titre de menu dans la barre de menus ;
- Un contrôle de menu placé au niveau du premier retrait dans la zone de liste s'affiche dans le menu lorsque l'utilisateur clique sur le titre de menu précédent ;
- Un contrôle de menu en retrait suivi d'autres contrôles encore plus en retrait devient un titre de sous-menu. Les contrôles de menu placés en retrait sous le titre du sous-menu deviennent des éléments du sous-menu ;
- Un contrôle de menu avec un tiret (-) en tant que valeur de sa propriété Caption est une *barre séparatrice* qui sert à diviser les éléments d'un menu en groupes logiques.

Note : un contrôle de menu ne peut être une barre séparatrice s'il s'agit d'un titre de menu, s'il comprend des éléments de sous-menu, s'il est activé ou désactivé ou s'il est doté d'une touche d'accès rapide.

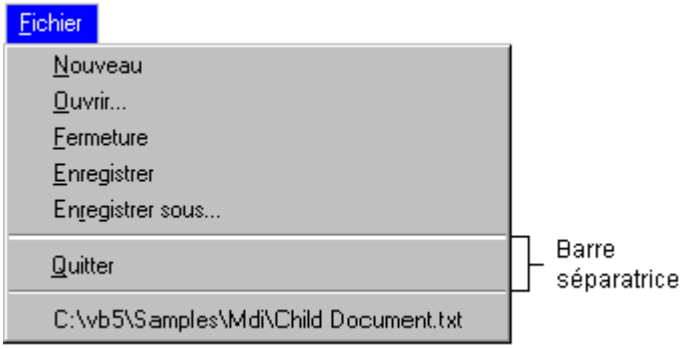
Pour créer des contrôles de menu dans le Créateur de menus

- Sélectionnez une feuille ;
- Dans le menu **Outils**, sélectionnez **Créateur de menus**.
  – ou –
  Cliquez sur le bouton **Créateur de menus** dans la barre d'outils ;
- Dans la zone de texte **Caption**, entrez le premier titre de menu que vous voulez afficher dans la barre de menus. Insérez une perluète (&) avant la lettre que vous voulez définir comme touche d'accès rapide pour cet élément de menu. La lettre est automatiquement soulignée dans le titre du menu. Le titre du menu apparaît dans la zone de liste des contrôles de menu ;
- Dans la zone de texte **Name**, entrez le nom que vous voulez utiliser pour désigner le contrôle de menu dans le code. Reportez-vous à la section « Règles des noms et titres de menus » plus bas dans ce chapitre ;
- Cliquez sur les flèches gauche ou droite pour modifier le niveau de retrait du contrôle ;
- Définissez d'autres propriétés pour le contrôle, si vous le souhaitez. Vous pouvez les définir maintenant dans le Créateur de menus ou plus tard dans la fenêtre Propriétés ;
- Sélectionnez **Suivant** pour créer un autre contrôle de menu
  - ou –
  Cliquez sur **Insérer** pour ajouter un contrôle de menu entre les contrôles existants ;
- Cliquez sur **OK** pour fermer le Créateur de menus une fois que vous avez créé tous les contrôles de menu de la feuille.

Les titres de menus que vous créez s'affichent sur la feuille. En mode création, cliquez sur un titre de menu pour dérouler les éléments de menu correspondants.

10.3. - Séparation des éléments d'un menu

Une barre séparatrice est une ligne horizontale qui apparaît entre les éléments d'un menu. Si le menu comprend beaucoup d'éléments, vous pouvez utiliser une barre séparatrice pour diviser les éléments en groupes logiques. Par exemple, dans le menu Fichier de Visual Basic, des barres séparatrices divisent les éléments de menu en trois groupes, comme illustré à la figure suivante.



Créer une barre séparatrice dans le Créateur de menus

- Si vous ajoutez une barre séparatrice dans un menu existant, cliquez sur **Insérer** pour ajouter un contrôle de menu entre les éléments que vous voulez séparer ;
- Si nécessaire, cliquez sur la flèche droite pour mettre en retrait le nouvel élément de menu au même niveau que les éléments qu'il séparera ;
- Entrez un tiret (-) dans la zone de texte **Caption** ;
- Définissez la propriété **Name** ;
- Cliquez sur **OK** pour fermer le Créateur de menus ;

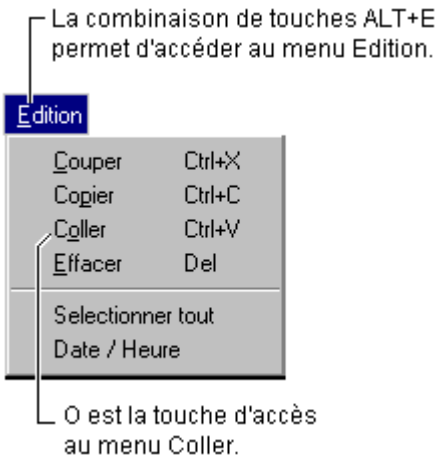
Note : bien que les barres séparatrices soient créées en tant que contrôles de menu, elles ne répondent pas à l'événement Click, et l'utilisateur ne peut les sélectionner.

10.4. - Affectation de touches d'accès rapide et de raccourci

Vous pouvez améliorer l'accès du clavier aux commandes de menu en définissant des touches d'accès rapide et de raccourci.

10.4.1. - Touches d'accès rapide

Les *touches d'accès rapide* permettent à l'utilisateur d'ouvrir un menu en appuyant sur la touche ALT et en tapant une lettre donnée. Une fois le menu ouvert, il peut choisir un contrôle en appuyant sur la touche correspondant à la lettre (touche d'accès rapide) affectée à ce contrôle. Par exemple, les touches ALT+E permettent d'ouvrir le menu Edition, et la lettre O permet de sélectionner l'élément de menu Coller. L'affectation d'une touche d'accès rapide est représentée par une lettre soulignée dans le libellé du contrôle de menu, comme illustré à la figure suivante.



Affecter une touche d'accès rapide à un contrôle de menu dans le Créateur de menus

- Sélectionnez l'élément de menu auquel vous voulez affecter une touche d'accès rapide ;
- Dans la zone **Caption**, saisissez une perluète (&) juste devant la lettre que vous voulez utiliser en tant que touche d'accès rapide.

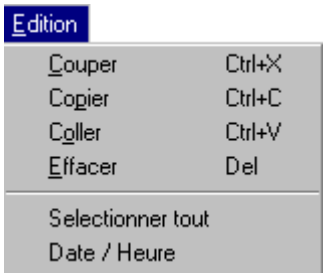
Par exemple, si le menu Edition de la figure précédente est ouvert, les valeurs suivantes de la propriété Caption répondent aux touches correspondantes.

Libellé du contrôle de menu	Propriété Caption	Touches d'accès rapide
Couper	&Couper	c
Copier	Cop&ier	I
Coller Supprimer Sélectionner tout Heure/Date	C&oller &Supprimer Sélectionner &tout Heure/&Date	o s t d

Note : n'utilisez pas de touches d'accès rapide en double dans les menus. Si vous définissez la même touche pour plusieurs éléments de menu, elle ne fonctionnera pas. Par exemple, si C est la touche d'accès rapide pour les commandes Couper et Copier et si vous sélectionnez le menu Edition et appuyez sur C, la commande Copier sera sélectionnée, mais l'application ne l'exécutera pas tant que vous n'aurez pas appuyé sur ENTRÉE. La commande Couper ne sera jamais sélectionnée.

10.4.2. - Touches de raccourci

Dès que vous appuyez sur une *touche de raccourci*, elle exécute un élément de menu. Des touches de raccourci peuvent être affectées aux éléments de menu souvent utilisés, ce qui permet de les exécuter par un seul accès au clavier, au lieu de trois (touche ALT, lettre d'accès au titre de menu et lettre d'accès à l'élément de menu). Les affectations de touches de raccourci s'effectuent à l'aide de combinaisons de touches de fonction et de contrôle, telles que CTRL+F1 ou CTRL+A. Elles apparaissent dans le menu à droite de l'élément de menu correspondant, comme illustré à la figure suivante.



Affecter une touche de raccourci à un élément de menu

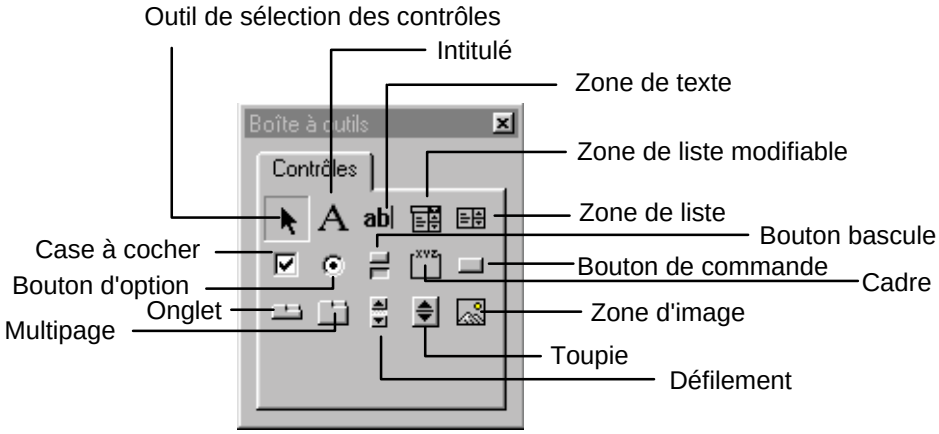
- Ouvrez le **Créateur de menus** ;
- Sélectionnez l'élément de menu ;
- Sélectionnez une touche de fonction ou une combinaison de touches dans la zone de liste modifiable **Shortcut**.

Pour supprimer une affectation de touche de raccourci, sélectionnez « (aucun) » dans le haut de la liste.

Annexes

A1. – Les contrôles les plus courants

Ci-dessous la liste des principaux contrôles utilisables dans les formulaires de Visual Basic.



Intitulé (Label)

Un contrôle Label dans une feuille affiche un texte descriptif tel que des titres, des légendes, des images ou de brèves instructions. Par exemple, les étiquettes d'un carnet d'adresses peuvent inclure un contrôle Label pour un nom, une rue ou une ville. Un contrôle Label n'affiche pas de valeurs issues de sources de données ou d'expressions ; il est toujours indépendant et ne change pas lorsque vous vous déplacez d'un enregistrement à l'autre.

La propriété par défaut d'un Label est la propriété **Caption**.

L'événement par défaut d'un Label est l'événement **Click**.

Zone de texte (TextBox)

Le contrôle zone de texte est le contrôle le plus souvent utilisé pour afficher les informations entrées par un utilisateur. Il peut également afficher un ensemble de données, tel qu'une table, une requête, une feuille de programmation ou le résultat d'un calcul. Si un contrôle zone de texte est dépendant d'une source de données, les changements apportés au contenu d'un contrôle zone de texte modifient aussi la valeur de la source de données dont il dépend.

Le format appliqué à toute partie du texte du contrôle zone de texte se répercutera à la totalité du texte du contrôle. Par exemple, si vous changez la police ou la taille d'un caractère du contrôle, la modification affecte tous les caractères du contrôle.

La propriété par défaut d'un contrôle zone de texte est la propriété **Value**.

L'événement par défaut d'un contrôle zone de texte est l'événement **Change**.

Zone de liste (ListBox)

Affiche une liste de valeurs et vous permet d'en sélectionner une ou plusieurs.

Si le contrôle Zone de liste est dépendant d'une source de données, il enregistre la valeur sélectionnée dans cette source de données.

Le contrôle Zone de liste peut apparaître sous la forme d'une liste, d'un groupe de contrôles OptionButton ou de contrôles CheckBox.

La propriété par défaut d'un contrôle Zone de liste est la propriété **Value**.

L'événement par défaut d'un contrôle Zone de liste est l'événement **Click**.

Zone de liste modifiable (ComboBox)

Allie les caractéristiques d'un contrôle Zone de liste et d'un contrôle zone de texte. L'utilisateur peut entrer une nouvelle valeur, comme dans un contrôle zone de texte ou bien sélectionner une valeur existante comme dans un contrôle Zone de liste.

Si un contrôle Zone de liste modifiable est dépendant d'une source de données, il insère dans cette source de données la valeur que l'utilisateur entre ou sélectionne. Si une liste modifiable multicolonne est dépendante, la propriété BoundColumn détermine la valeur enregistrée dans la source de données dont dépend la liste modifiable.

La liste d'un contrôle Zone de liste modifiable se compose de lignes de données. Chacune d'elle peut contenir une ou plusieurs colonnes qui peuvent apparaître avec ou sans titre. Certaines applications n'utilisent pas les titres de colonne, d'autres ne les l'utilisent que partiellement.

La propriété par défaut d'un contrôle Zone de liste modifiable est la propriété **Value**.

L'événement par défaut d'un contrôle Zone de liste modifiable est l'événement **Change**.

Case à cocher (CheckBox)

Affiche l'état de la sélection d'un élément.

Utilisez un contrôle Case à cocher pour permettre à l'utilisateur de choisir entre deux valeurs telles que Oui/Non, Vrai/Faux ou Actif/Inactif. Quand l'utilisateur sélectionne un contrôle Case à cocher, une marque spéciale (un X, par exemple) s'affiche et sa valeur courante est Oui, Vrai ou Actif ; si l'utilisateur ne sélectionne pas le contrôle Case à cocher, celui-ci est vide et sa valeur est Non, Faux ou Inactif. Selon la valeur de la propriété TripleState, un contrôle Case à cocher peut aussi avoir une valeur nulle.

Si un contrôle Case à cocher est dépendant d'une source de données, le changement de paramètre modifie la valeur de cette source. Un contrôle Case à cocher désactivé affiche sa valeur courante mais il apparaît ombré et ne permet pas à l'utilisateur de modifier sa valeur.

Vous pouvez aussi utiliser des cases à cocher à l'intérieur d'une zone de groupe pour sélectionner un ou plusieurs éléments connexes. Par exemple, vous pouvez créer un bon de commande contenant une liste des éléments disponibles, en faisant précéder chaque élément d'un contrôle Case à cocher. L'utilisateur peut sélectionner un ou plusieurs éléments particuliers en cochant le contrôle Case à cocher correspondant.

La propriété par défaut d'un contrôle Case à cocher est la propriété **Value**.

L'événement par défaut d'un contrôle Case à cocher est l'événement **Click**.

Bouton d'options (OptionButton)

Affiche l'état de la sélection d'un élément faisant partie d'un groupe.

Utilisez un contrôle Bouton d'options pour afficher si un seul élément contenu dans un groupe est sélectionné. Tous les contrôles Bouton d'options d'un contrôle Frame s'excluent l'un l'autre.

Si un contrôle Bouton d'options est dépendant d'une source de données, il peut afficher la valeur de cette source de données comme étant Oui/Non, Vrai/Faux ou Actif/Inactif. Si l'utilisateur sélectionne le contrôle Bouton d'options, la valeur courante est Oui, Vrai ou Actif ; si l'utilisateur ne sélectionne pas le contrôle Bouton d'options, la valeur est Non, Faux ou Inactif. Par exemple, dans une application de suivi des stocks, un contrôle Bouton d'options peut faire apparaître si la production d'un élément est arrêtée. Si le contrôle Bouton d'options est dépendant d'une source de données, le changement de paramètre modifie la valeur de cette source. Un contrôle Bouton d'options désactivé est indisponible et il n'affiche aucune valeur.

Selon la valeur de la propriété TripleState, un contrôle Bouton d'options peut aussi avoir une valeur nulle.

Vous pouvez aussi utiliser un contrôle Bouton d'options à l'intérieur d'une zone de groupe pour sélectionner un ou plusieurs groupes d'éléments connexes. Par exemple, vous pouvez créer un bon de commande contenant une liste des éléments disponibles, en faisant précéder chaque élément d'un contrôle Bouton d'options. L'utilisateur peut sélectionner un élément particulier en cochant le contrôle Bouton d'options correspondant.

La propriété par défaut d'un contrôle Bouton d'options est la propriété **Value**.

L'événement par défaut d'un contrôle Bouton d'options est l'événement **Click**.

Bouton bascule (ToggleButton)

Affiche l'état de la sélection d'un élément.

Utilisez un contrôle Bouton bascule pour afficher si un élément est sélectionné. Si un contrôle Bouton bascule est dépendant d'une source de données, il affiche la valeur courante de cette source de données comme étant Oui/Non, Vrai/Faux, Actif/Inactif ou tout autre choix de deux valeurs. Si l'utilisateur sélectionne le contrôle Bouton bascule, la valeur courante est Oui, Vrai ou Actif ; si l'utilisateur ne sélectionne pas le contrôle Bouton bascule, la valeur est Non, Faux ou Inactif. Si le contrôle Bouton bascule est dépendant d'une source de données, le changement de paramètre modifie la valeur de cette source. Un contrôle Bouton bascule désactivé affiche une valeur, mais il est indisponible et n'autorise pas les changements à partir de l'interface utilisateur.

Vous pouvez aussi utiliser un contrôle Bouton bascule à l'intérieur d'un contrôle Frame pour sélectionner un ou plusieurs groupes d'éléments connexes. Par exemple, vous pouvez créer un bon de commande contenant une liste des éléments disponibles, en faisant précéder chaque élément d'un contrôle Bouton bascule. L'utilisateur peut choisir un élément particulier en sélectionnant le Bouton bascule approprié.

La propriété par défaut d'un contrôle Bouton bascule est la propriété **Value**.

L'événement par défaut d'un contrôle Bouton bascule est l'événement **Click**.

Cadre (Frame)

Crée un groupe de contrôles fonctionnel et visuel.

Tous les boutons d'option d'un contrôle Cadre s'excluent l'un l'autre, de sorte que vous pouvez utiliser le contrôle Cadre pour créer un groupe d'options. Vous pouvez aussi utiliser un contrôle **Cadre** pour regrouper des contrôles dont le contenu est étroitement associé. Par exemple, dans une application qui traite les commandes clients, vous pouvez utiliser un contrôle Cadre pour regrouper noms, adresses et numéros de comptes des clients.

Vous pouvez aussi utiliser un contrôle Cadre pour créer un groupe de boutons bascules, mais ceux-ci ne s'excluent pas l'un l'autre.
L'événement par défaut d'un contrôle Cadre est l'événement **Click**.

Bouton de commande (CommandButton)

Lance, termine ou interrompt une action ou une série d'actions.

La macro ou la procédure d'événement affectée à l'événement Click du contrôle Bouton de commande détermine l'action de celui-ci. Par exemple, vous pouvez créer un contrôle Bouton de commande qui ouvre une autre feuille. Vous pouvez aussi afficher un texte, une image ou les deux sur un contrôle Bouton de commande.

La propriété par défaut d'un contrôle Bouton de commande est la propriété **Value**.

L'événement par défaut d'un contrôle Bouton de commande est l'événement **Click**.

Onglet (tabStrip)

Présente un ensemble de contrôles connexes sous forme d'un groupe visuel.

Vous pouvez utiliser un contrôle Onglet pour visualiser différents ensembles d'informations. Par exemple, les contrôles peuvent représenter des informations relatives à l'emploi du temps journalier d'un groupe de personnes, chaque ensemble d'informations correspondant à une personne différente du groupe. Définissez le titre de chaque onglet pour afficher le nom d'une seule personne. Vous pouvez ensuite écrire un code qui, quand vous cliquez sur l'onglet, met à jour les contrôles pour afficher les informations concernant la personne dont le nom figure sur l'onglet.

La propriété par défaut d'un contrôle Onglet est la propriété **SelectedItem**.

L'événement par défaut d'un contrôle Onglet est l'événement **Change**.

MultiPage (MultiPage)

Le contrôle MultiPage est un conteneur d'une collection d'objets Pages dans laquelle chaque page contient un ou plusieurs objets Page.

Un contrôle MultiPage est utile lorsque vous travaillez avec un gros volume d'informations qui peuvent être triées en plusieurs catégories. Vous pouvez notamment utiliser un contrôle MultiPage pour afficher les informations contenues dans une candidature et, par exemple, réserver une page aux renseignements personnels tels que le nom et l'adresse, une autre à la liste des précédents employeurs et une troisième à la liste des références. Le contrôle MultiPage vous permet de combiner visuellement des informations connexes tout en conservant un enregistrement complet facilement accessible.

Les nouvelles pages sont ajoutées à droite de la page sélectionnée.
La propriété par défaut d'un contrôle MultiPage est la propriété **Value** qui renvoie l'index de la Page couramment active dans la collection de Pages du contrôle MultiPage.

L'événement par défaut d'un contrôle MultiPage est l'événement **Change**.

Barre de défilement (ScrollBar)

Retourne ou définit la valeur d'un autre contrôle en fonction de la position du curseur de défilement.

Le contrôle Barre de défilement est un contrôle autonome que vous pouvez placer sur une feuille. Son aspect visuel est identique à celui de la barre de défilement présente dans certains objets tels qu'un contrôle Zone de liste ou dans la partie déroulante d'un contrôle Zone de liste modifiable. Toutefois, à l'inverse des barres de défilement citées dans ces exemples, le contrôle Barre de défilement autonome ne fait pas partie intégrante d'un autre contrôle.

Pour utiliser le contrôle Barre de défilement afin de définir ou de lire la valeur d'un autre contrôle, vous devez écrire un code pour les événements et les méthodes du contrôle Barre de défilement. Par exemple, pour utiliser le contrôle Barre de défilement en vue de mettre à jour la valeur d'un contrôle Zone de texte, vous pouvez écrire un code lisant la propriété Value du contrôle Barre de défilement, puis définir la propriété Value du contrôle Zone de texte.

La propriété par défaut d'un contrôle Barre de défilement est la propriété Value.

L'événement par défaut d'un contrôle Barre de défilement est l'événement Change.

Pour créer un contrôle Barre de défilement vertical ou horizontal, faites glisser les poignées de redimensionnement du contrôle Barre de défilement horizontalement ou verticalement sur la feuille.

Toupie (SpinButton)

Incrémente et décrémente des nombres.

Cliquer sur un contrôle Toupie ne modifie que la valeur de celui-ci. Vous pouvez écrire un code qui utilise le contrôle Toupie pour mettre à jour la valeur affichée d'un autre contrôle. Par exemple, vous pouvez utiliser un contrôle Toupie pour changer le mois, le jour ou l'année dans une date. Vous pouvez aussi utiliser un contrôle Toupie pour faire défiler une série de valeurs ou une liste d'éléments, ou bien pour modifier la valeur affichée dans une zone de texte.

Pour afficher une valeur mise à jour à l'aide d'un contrôle Toupie, vous devez affecter la valeur du contrôle Toupie à la partie affichée d'un contrôle telle que la propriété Caption d'un contrôle Label ou la propriété Text d'un contrôle Zone de texte. Pour créer un contrôle Toupie horizontal ou vertical, faites glisser les poignées de redimensionnement du contrôle Toupie horizontalement ou verticalement sur la feuille.

La propriété par défaut d'un contrôle Toupie est la propriété **Value**.

L'événement par défaut d'un contrôle Toupie est l'événement **Change**.

Zone d'Image (Image)

Le contrôle Image vous permet d'afficher une image comme partie des données contenues dans une feuille. Par exemple, vous pouvez utiliser un contrôle Image pour afficher la photographie des employés dans une feuille du personnel.

Le contrôle Image vous permet de couper, de redimensionner ou d'effectuer un zoom avant ou arrière sur l'image, mais ne vous permet pas d'en modifier le contenu. Par exemple, vous ne pouvez utiliser le contrôle Image pour modifier les couleurs d'une image, pour la rendre transparente ou pour affiner la vue de l'image. Pour effectuer ces opérations, il vous faut un logiciel de traitement d'image.

Le contrôle Image supporte les formats de fichiers suivants : *.bmp, *.cur, *.gif, *.ico, *.jpg, *.wmf

A2.- Les mots clés classés par domaines

Nous ne pouvons donner ici la syntaxe et le détails de chaque mot clé Visual Basic, la liste ci-dessous classé par domaine doit amplement suffire en sachant que vous pouvez obtenir rapidement de l'aide en utilisant l'explorateur d'objet intégré dans Visual Basic.

Création, définition et utilisation de tableaux.

Test d'un tableau.	**IsArray**
Création d'un tableau.	**Array**
Modification de la limite inférieure par défaut.	**Option Base**
Déclaration et initialisation d'un tableau.	**Dim, Private, Public, ReDim, Static**
Renvoi des limites d'un tableau.	**Lbound, Ubound**
Réinitialisation d'un tableau.	**Erase, ReDim**

Contrôle du comportement du compilateur.

Définition d'une constante de compilation.	**#Const**
Compilation sélective de certains blocs de code.	**#If...Then...#Else**

Définition de boucles et orientation du flux de procédure.

Branchement.	**GoSub...Return, GoTo, On Error, On...GoSub, On...GoTo**
Sortie ou pause du programme.	**DoEvents, End, Exit, Stop**
Boucle.	**Do...Loop, For...Next, For Each...Next, While...Wend, With**
Prise de décisions.	**Choose, If...Then...Else, Select Case, Switch**
Utilisation de procédures.	**Call, Function, Property Get, Property Let, Property Set, Sub**

Conversion des nombres et des types de données.

Code ANSI en chaîne.	**Chr**
Chaîne en minuscules ou en majuscules.	**Format, Lcase, Ucase**
Date en numéro de série.	**DateSerial, DateValue**
Nombre décimal en une autre base.	**Hex, Oct**
Nombre en chaîne.	**Format, Str**
Type de données en autre type.	**Cbool, Cbyte, Ccur, Cdate, Cdbl, Cdec, CInt, Clng, Csng, Cstr, Cvar, CVErr, Fix, Int**
Date en jours, mois, jours de semaine ou années.	**Day, Month, Weekday, Year**
Heure en heures, minutes ou secondes.	**Hour, Minute, Second**
Chaîne en code ASCII.	**Asc**
Chaîne en nombre.	**Val**
Heure en numéro de série.	**TimeSerial, TimeValue**

Types de données et sous-types des données de type Variant.

Conversion entre les types de données.	**Cbool, Cbyte, Ccur, Cdate, Cdbl, Cdec, Cint, CLng, CSng, Cstr, Cvar, CVErr, Fix, Int**
Définition des types de données intrinsèques.	**Boolean, Byte, Currency, Date, Double, Integer, Long, Object, Single, String, Variant (type par défaut)**
Test des types de données.	**IsArray, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject**

Conversion et utilisation d'expressions de date et d'heure.

Renvoi de la date ou de l'heure en cours.	**Date, Now, Time**
Calculs de date.	**DateAdd, DateDiff, DatePart**
Renvoi d'une date.	**DateSerial, DateValue**
Renvoi d'une heure.	**TimeSerial, TimeValue**
Définition de la date ou de l'heure.	**Date, Time**
Chronométrage d'un traitement.	**Timer**

Contrôle du système de fichiers et traitement des fichiers.

Changement de répertoire ou de dossier.	**ChDir**
Changement de lecteur.	**ChDrive**
Copie d'un fichier.	**FileCopy**
Création d'un répertoire ou d'un dossier.	**MkDir**
Suppression d'un répertoire ou dossier.	**RmDir**
Attribution d'un nouveau nom à un fichier, répertoire ou dossier.	**Name**
Renvoi du chemin en cours.	**CurDir**
Renvoi de l'horodatage d'un fichier.	**FileDateTime**
Renvoi d'attributs de fichier, de répertoire et de nom de volume.	**GetAttr**
Renvoi de la longueur d'un fichier.	**FileLen**
Renvoi d'un nom de fichier ou de volume.	**Dir**
Définition des attributs d'un fichier.	**SetAttr**

Interception et renvoi de codes d'erreur.

Génération d'erreurs d'exécution.	**Clear, Error, Raise**
Récupération des messages d'erreur.	**Error**
Informations sur les erreurs.	**Err**
Renvoi de la variable Error.	**CVErr**
Interception des erreurs durant l'exécution.	**On Error, Resume**
Vérification de type.	**IsError**

Exécution de calculs financiers.

Calcul d'amortissement.	**DDB, SLN, SYD**
Calcul de valeur future.	**FV**
Calcul de taux d'intérêt.	**Rate**
Calcul de taux de rendement interne.	**IRR, MIRR**
Calcul de nombre d'échéances.	**Nper**
Calcul de montant de versements.	**Ipmt, Pmt, PPmt**
Calcul de valeur actuelle.	**NPV, PV**

Réception des entrées et affichage ou impression des sorties.

Accès ou création d'un fichier.	**Open**
Fermeture de fichiers.	**Close, Reset**
Mise en forme de la sortie.	**Format, Print, Imprimer #, Spc, Tab, Width #**
Copie d'un fichier.	**FileCopy**
Récupération d'informations sur un fichier.	**EOF, FileAttr, FileDateTime, FileLen, FreeFile, GetAttr, Loc, LOF, Seek**
Gestion de fichiers.	**Dir, Kill, Lock, Unlock, Name**
Lecture d'un fichier.	**Get, Input, Input #, Line Input #**
Renvoi de la longueur d'un fichier.	**FileLen**
Définition ou lecture des attributs de fichier.	**FileAttr, GetAttr, SetAttr**
Définition de positions de lecture/écriture dans un fichier.	**Seek**
Écriture dans un fichier.	**Print #, Put, Write #**

Exécution de calculs mathématiques (trigonométrie, etc).

Fonctions trigonométriques.	**Atn, Cos, Sin, Tan**
Calculs usuels.	**Exp, Log, Sqr**
Génération de nombres aléatoires.	**Randomize, Rnd**
Renvoi de la valeur absolue.	**Abs**
Renvoi du signe d'une expression.	**Sgn**
Conversions numériques.	**Fix, Int**

Lancer d'autres applications et traitement des événements.

Traitement des événements en attente.	**DoEvents**
Exécution d'autres programmes.	**AppActivate, Shell**
Envoi de touches à une application.	**SendKeys**
Émission d'un bip par l'ordinateur.	**Beep**
Système.	**Environ**
Renvoi d'une chaîne de ligne de commande	**Command**
Macintosh.	**MacID, MacScript**
Automation.	**CreateObject, GetObject**
Couleur.	**QBColor, RGB**

Comparaison d'expressions et autres opérations.

Arithmétiques	**^, – , *, /, \, Mod, +, &**
Comparaison.	**=, <>, <, >, <=, >=, Like, Is**
Opérations logiques.	**Not, And, Or, Xor, Eqv, Imp**

Manipulation de chaînes et de données de type chaîne.

Comparaison de deux chaînes.	**StrComp**
Conversion de chaînes.	**StrConv**
Conversion en minuscules ou en majuscules.	**Format, Lcase, UCase**
Création de chaînes répétant un même caractère.	**Space, String**
Calcul de la longueur d'une chaîne.	**Len**
Mise en forme d'une chaîne.	**Format**
Alignement d'une chaîne.	**Lset, RSet**
Manipulation de chaînes.	**InStr, Left, Ltrim, Mid, Right, RTrim, Trim**
Définition des règles de comparaison de chaînes.	**Option Compare**
Utilisation des codes ASCII et ANSI.	**Asc, Chr**

Déclaration et définition de variables et de constantes.

Attribution d'une valeur.	**Let**
Déclaration de variables ou de constantes.	**Const, Dim, Private, Public, New, Static**
Déclaration d'un module privé.	**Option Private Module**
Obtention d'informations sur une variable.	**IsArray, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject, TypeName, VarType**
Référence à l'objet en cours.	**Me**
Activation de la déclaration explicite des variables.	**Option Explicit**
Définition du type de données par défaut.	**Deftype**

A3.- Constantes de code de touches

Les constantes suivantes peuvent remplacer les valeurs réelles partout dans votre code :

Constante	Valeur	Description
vbKeyLButton	0x1	Bouton gauche de la souris
vbKeyRButton	0x2	Bouton droit de la souris
vbKeyCancel	0x3	Touche ANNULER
vbKeyMButton	0x4	Bouton secondaire de la souris
vbKeyBack	0x8	Touche RET.ARR
vbKeyTab	0x9	Touche TAB
vbKeyClear	0xC	Touche EFFACER
vbKeyReturn	0xD	Touche ENTRÉE
vbKeyShift	0x10	Touche MAJ
vbKeyControl	0x11	Touche CTRL
vbKeyMenu	0x12	Touche MENU
vbKeyPause	0x13	Touche PAUSE
vbKeyCapital	0x14	Touche VERR.MAJ
vbKeyEscape	0x1B	Touche ÉCHAP
vbKeySpace	0x20	Touche ESPACE
vbKeyPageUp	0x21	Touche PG.PRÉC
vbKeyPageDown	0x22	Touche PG.SUIV
vbKeyEnd	0x23	Touche FIN
vbKeyHome	0x24	Touche ORIGINE
vbKeyLeft	0x25	Touche FLÈCHE GAUCHE
vbKeyUp	0x26	Touche FLÈCHE HAUT
vbKeyRight	0x27	Touche FLÈCHE DROITE
vbKeyDown	0x28	Touche FLÈCHE BAS
vbKeySelect	0x29	Touche SÉLECTION
vbKeyPrint	0x2A	Touche IMPR.ÉCRAN
vbKeyExecute	0x2B	Touche EXÉCUTER
vbKeySnapshot	0x2C	Touche SNAPSHOT
vbKeyInsert	0x2D	Touche INSER
vbKeyDelete	0x2E	Touche SUPPR
vbKeyHelp	0x2F	Touche AIDE
vbKeyNumlock	0x90	Touche VERR.NUM

Les touches A à Z sont les mêmes que leurs équivalents ASCII :

Constante	Valeur ascii	Description
vbKeyA	65	Touche A
vbKeyB	66	Touche B
vbKeyC	67	Touche C
vbKeyD	68	Touche D
vbKeyE	69	Touche E
vbKeyF	70	Touche F
vbKeyG	71	Touche G
vbKeyH	72	Touche H
vbKeyI	73	Touche I

vbKeyJ	74	Touche J
vbKeyK	75	Touche K
vbKeyL	76	Touche L
vbKeyM	77	Touche M
vbKeyN	78	Touche N
vbKeyO	79	Touche O
vbKeyP	80	Touche P
vbKeyQ	81	Touche Q
vbKeyR	82	Touche R
vbKeyS	83	Touche S
vbKeyT	84	Touche T
vbKeyU	85	Touche U
vbKeyV	86	Touche V
vbKeyW	87	Touche W
vbKeyX	88	Touche X
vbKeyY	89	Touche Y
vbKeyZ	90	Touche Z

Les touches 0 à 9 sont les mêmes que leurs équivalents ASCII :

Constante	Valeur	Description
vbKey0	48	Touche 0
vbKey1	49	Touche 1
vbKey2	50	Touche 2
vbKey3	51	Touche 3
vbKey4	52	Touche 4
vbKey5	53	Touche 5
vbKey6	54	Touche 6
vbKey7	55	Touche 7
vbKey8	56	Touche 8
vbKey9	57	Touche 9

Les constantes suivantes représentent les touches du pavé numérique :

Constante	Valeur	Description
vbKeyNumpad0	0x60	Touche 0
vbKeyNumpad1	0x61	Touche 1
vbKeyNumpad2	0x62	Touche 2
vbKeyNumpad3	0x63	Touche 3
vbKeyNumpad4	0x64	Touche 4
vbKeyNumpad5	0x65	Touche 5
vbKeyNumpad6	0x66	Touche 6
vbKeyNumpad7	0x67	Touche 7
vbKeyNumpad8	0x68	Touche 8
vbKeyNumpad9	0x69	Touche 9
vbKeyMultiply	0x6A	Touche MULTIPLICATION (*)
vbKeyAdd	0x6B	Touche PLUS (+)
vbKeySeparator	0x6C	Touche ENTRÉE
vbKeySubtract	0x6D	Touche MOINS (–)

www.Mcours.com
Site N°1 des Cours et Exercices Email: mymcours@gmail.com

vbKeyDecimal	0x6E	Touche POINT DÉCIMAL (.)
vbKeyDivide	0x6F	Touche DIVISION (/)

Les constantes suivantes représentent les touches de fonction :

Constante	Valeur	Description
vbKeyF1	0x70	Touche F1
vbKeyF2	0x71	Touche F2
vbKeyF3	0x72	Touche F3
vbKeyF4	0x73	Touche F4
vbKeyF5	0x74	Touche F5
vbKeyF6	0x75	Touche F6
vbKeyF7	0x76	Touche F7
vbKeyF8	0x77	Touche F8
vbKeyF9	0x78	Touche F9
vbKeyF10	0x79	Touche F10
vbKeyF11	0x7A	Touche F11
vbKeyF12	0x7B	Touche F12
vbKeyF13	0x7C	Touche F13
vbKeyF14	0x7D	Touche F14
vbKeyF15	0x7E	Touche F15
vbKeyF16	0x7F	Touche F16