

UNIVERSITE CATHOLIQUE DE LOUVAIN
Institut d'Administration et de Gestion

VISUAL BASIC .NET : TUTORIAL

O. Moursli et N. Souchon

Table des matières

Section 1 : Introduction à Visual Basic .NET - Structures de base.....	6
1.1 Variable et opérations arithmétiques.....	6
1.1.1 Notion de Variable	6
1.1.2 Opérateurs arithmétiques	7
1.2 Instructions conditionnelles	8
1.2.1 If ... Then ... Else ... End If	8
1.2.2 If (Condition, ValeurSiVrai, ValeurSiFaux)	10
1.2.3 Select case ... Case ... Case ... Else Case ... End Select	10
1.3 Tableaux	11
1.4 Instructions répétitives.....	12
1.4.1 For ... To ... Next	12
1.4.2 Do While ... Loop / Do ... Loop While	13
1.4.3 Do Until ... Loop / Do ... Loop Until	14
1.4.4 For ... Each ... Next	15
1.4.5 Conclusion.....	15
1.5 Procédures et Fonctions	15
1.5.1 Procédure (Transmission par valeur : ByVal)	16
1.5.2 Procédure (Transmission par référence : ByRef)	16
1.5.3 Fonction.....	17
1.5.5 Portée des variables, procédures et fonctions	18
1.5.6 Quelques fonctions globales.....	19
1.5.7 Interruption de séquences	20
Section 2. Introduction à l'environnement de développement VB	21
2.1 Environnement VB.....	21
2.1.1 Formulaires (Forms).....	21
2.1.2 L'explorateur des solutions	24
2.1.3 La fenêtre Properties.....	25
2.1.4 La boîte à outils et les contrôles standards	27
2.2 Programmation par événements	29
2.3 Exercices	33
Section 3. Les contrôles	34
3.1 Concept d'objet.....	34
3.2 Contrôles standards.....	35
3.2.1 La propriété "Name"	35
3.2.2 Label.....	36
3.2.3 TextBox	36
3.2.4 RadioButton.....	36
3.2.5 CheckButton.....	38
3.2.7 GroupBox	39
3.2.8 Exercices	41
3.2.9 ListBox	42
3.2.10 ComboBox.....	42
3.2.11 La propriété Items.....	42
3.2.12 Exercices	45
3.2.13 Solution	47
3.2.14 L'éditeur de menus	49
3.2.14 L'éditeur de menus	49

4. <i>Les Bases de Données</i>.....	50
4.1 Notion de Table.....	50
4.2 Notion de Base de Données	51
4.3 Création d'une base de données Access.....	51

0. Introduction

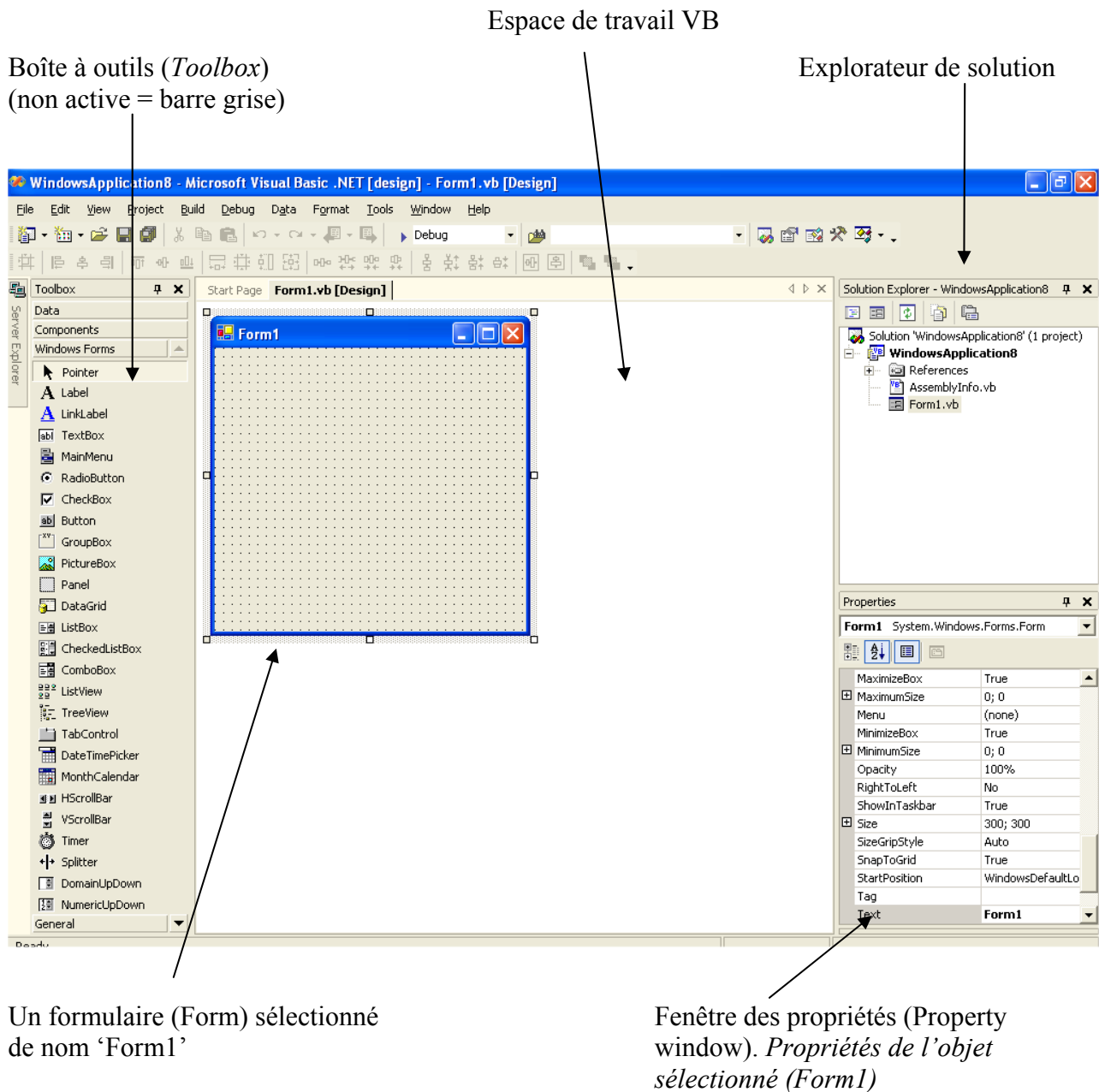


Figure 1: Environnement de développement Visual Basic .NET

Pré-requis :

Ce tutorial suppose que vous soyez familier avec un langage structuré de programmation (comme le Pascal). Certains concepts seront cependant, brièvement rappelés.

Contenu de ce tutorial:

Ce tutorial vous présente Visual Basic (VB) .NET de manière succincte. Il vous introduit à la programmation multi-fenêtrée sous *MS Windows* en vous exposant les outils de base de VB. Il vous introduit à la programmation de bases de donnée.

Limites de ce tutorial:

Ce tutorial n'est pas un manuel de référence VB .NET. Pour des fonctionnalités plus avancées, référez vous à d'autres ressources, notamment à des **sites Web** sur internet traitant de VB .NET.

Plan de travail

Le tutorial est composé six parties qui peuvent être étudiées en six (semaines) séances. A chacune de ces parties correspond une série d'exercices, qu'il faut réaliser et bien assimiler avant de passer à la partie suivante.

Chaque partie nécessite *approximativement* deux heures de travail sur un ordinateur, pendant lesquelles vous devez réaliser les exercices correspondants.

Pour chaque partie, avant de travailler sur un ordinateur, vous devez bien lire les sections correspondantes et essayer de bien comprendre les programmes qui y sont présentés.

- **Partie 1** : *Sections 1 et 2. Introduction à VB .NET et à l'environnement de développement VB .NET*

Vous introduit à l'environnement de travail de VB .NET et à la programmation avec VB .NET.

- **Partie 2** : *Section 3. Les contrôles (toutes les sous-sections jusqu'à la sous-section 3.2.7)*

Vous introduit à la programmation des contrôles *Label*, *TextBox*, *RadioButton*, *CheckBouton* et *GroupBox*.

- **Partie 3** : *Section 3. Les contrôles (toutes les sous-sections jusqu'à la sous-section 3.2.14)*

Vous introduit à la programmation des contrôles utilisant des listes : *ListBox*, *ComboBox* et *Menu*.

- **Partie 4** : *Section 4. Création de bases de données*

Vous introduit aux bases de données avec *Microsoft Access*

SECTION 1 : INTRODUCTION À VISUAL BASIC .NET - STRUCTURES DE BASE

Dans cette section nous allons présenter les structures de base de la programmation en VB. Nous allons d'abord présenter la notion de variable, les différents types standards, les opérateurs arithmétiques, les structures usuelles: structures de contrôles (Instructions conditionnelles et répétitives), les structures de données élaborées (vecteurs et matrices) et finalement, les procédures et les fonctions.

Quelques remarques préliminaires :

- Cette première section présente les structures théoriques de VB. Il est conseillé de la parcourir une première fois (*sans trop insister*) pour avoir une idée des structures de base de VB. Au fur et à mesure que vous avancerez dans le cours, vous devez y revenir pour approfondir la partie qui vous intéresse.
- Votre application VB sera composée d'un ou plusieurs fichiers (*formulaires et modules*) contenant du code VB.
- Il n'existe pas de séparateurs d'instructions en VB (comme le ' ; ' en *Pascal* ou en *C* et le ' . ' en *Cobol*).
- VB ne fait pas de distinction entre les minuscules et les majuscules. Ainsi, *ValeurVariable* et *vALEURvARIABLE* représentent la même variable. En fait, VB est '*très intelligent*', en ce sens qu'il vous réécrira (automatiquement) la variable dans le format de caractères que vous avez utilisé lors de sa déclaration.

1.1 Variable et opérations arithmétiques

1.1.1 Notion de Variable

Les variables sont nécessaires pour **stocker** (conserver) une valeur dynamique et réutilisable. C'est en fait une simple **zone mémoire** qui porte un nom choisi par le programmeur. Le nom de la variable est une **adresse mémoire**. Si l'on veut une programmation cohérente, il faut déclarer chaque variable en précisant le type de celle-ci. La déclaration se fait avec le mot réservé **Dim**.

Syntaxe

```
Dim NomVariable As Type
```

Pour la lisibilité du code, on peut ajouter un commentaire après une apostrophe (')

Exemple

```
Dim Taux As Single      ' Ceci est un commentaire
                        ' Taux de la TVA
Dim Réponse As String   ' Mot proposé par le joueur
```

Par défaut, le compilateur VB considère que toute variable qui apparaît doit avoir été déclarée. Toutefois, si vous ajoutez dans votre code la ligne

```
Option Explicit Off
```

VB sera *permissif* et vous autorisera à utiliser des variables sans les déclarer. Prenez *la très bonne habitude* de toujours déclarer vos variables.

Pour éviter tout problème il est préférable d'initialiser les variables déclarées.

Exemples

```
Compteur = 0      ' = est le symbole d'affectation en VB
Taux = 21
```

Le langage VB utilise **plusieurs types** de données dont les plus utilisés sont le type **String** (chaîne de caractères), le type **Integer** (entier) et le type **Single** (décimal). Les types standards de données en VB sont résumés dans le tableau ci-dessous.

Types standards de données

Opérateur	Plage de valeurs	Déclaration et affectation
Integer	Nombres entiers de -32 768 à +32 767	Dim Nb As Integer Nb = 100
Single	Nombres réels avec précision de sept décimales Valeurs négatives : de -3,402823 ^E 38 à -1,401298 ^E -45 Valeurs positives: de 1,401298 ^E -45 à 3,402823E38	Dim Mt As Single Mt = 45.11
String	Chaîne de caractères pouvant aller jusqu'à 65535 caractères (environ 2 milliards si la longueur est variable)	Dim as String Prénom = "Jean"
Long	Nombres entiers de -2 147 483 648 à +2 147 483 647	Dim Profit As Long Profit = 123 465 789
Double	Nombres réels avec précision de seize décimales Valeurs négatives : De -1,79769313486232 ^E 308 à -4,94065641247 ^E -324 Valeurs positives: De 4,94065641247 ^E -324 à 1,79769313486232 ^E 308	Dim DblPrec As Double Mt = 1.23456789012
Byte	Nombres entiers de 0 à 255	Dim BitPattern As Byte BitPattern = 128
Boolean	Vrai ou faux (valeur logique)	Dim Test As Boolean Trouvé = True
Date	De 1/1/100 à 31/12/9999	Dim JourPlus As Date JourPlus = "06/06/44"
Currency	Nombres entiers de - 922337203685477,5808 à 922337203685477,5808	Dim Valeur As Currency

1.1.2 Opérateurs arithmétiques

VB reconnaît les opérateurs arithmétiques usuels qui sont résumés dans le tableau suivant :

Opérateurs arithmétiques

Opérateur	Description	Exemples
+, -	Addition et soustraction	12 + 34; 87.56 – 387.222
*	Multiplication	45.87 * 4
/	Division décimale	36 / 25 = 1.44
^	Puissance	5 ^ 3 = 125
\	Division entière	36 \ 25 = 1
MOD	Modulo (reste de la division entière)	36 MOD 25 = 11

Si, dans une expression arithmétique plusieurs opérateurs sont utilisés, les priorités sont résolues comme indiqué dans le tableau qui suit :

Priorité des opérateurs arithmétiques

Opérateur	Description	Priorité
()	Parenthèses	1
^	Puissance	2
-	Négation	3
*, /	Multiplication et division	4
\	Division entière	5
MOD	Modulo	6
+, -	Addition et soustraction	7

1.2 Instructions conditionnelles

Les deux instructions conditionnelles le plus utilisées en VB sont *If* et *Select Case*.

1.2.1 If ... Then ... Else ... End If

Si la condition se trouvant après le mot réservé *If* est *vraie*, les instructions qui suivent le mot réservé *Then* sont exécutées sinon, ce sont celles qui suivent le mot réservé *Else* qui sont exécutées. L'instruction *If* se termine (obligatoirement) avec les mots réservés *End If*.

Forme simple :

Syntaxe

```
If condition(s) Then
    Instruction11
    Instruction12
    ...
Else
    Instruction21
    Instruction22
    ...
End If
```

Exemple

```

If Moyenne >= 12 Then
    Admis = Admis + 1
    MsgBox(" Candidat admis ")      ' affiche une fenêtre avec le message indiqué
Else
    Ajournés = Ajournés + 1
    MsgBox(" Candidat ajourné ")
End If

```

Forme imbriquée**Syntaxe**

```

If condition(s) Then
    Instruction11
    If condition Then
        Instruction12
    Else if condition Then
        Instruction13
    Else
        Instruction14
    End If
    ...
Else
    Instruction21
    Instruction22
    ...
End If

```

Exemple

```

If NombreProposé > NombreATrouver Then
    MsgBox("Votre nombre est trop grand !")
ElseIf NombreProposé < NombreATrouver Then
    MsgBox("Votre nombre est trop petit !")
Else
    MsgBox("Gagné !")
End If

```

Opérateurs de comparaison

Opérateur	Signification	Exemple	Résultat
=	Egal à	15 = 11 + 4	True
>	Supérieur à	17 > 11	True
<	Inférieur à	17 < 11	False
<>	Différent de	23 <> 23.1	True
>=	Supérieur ou égale à	23 >= 23.1	False
<=	Inférieur ou égal à	23 <= 23.1	True

Si plusieurs conditions doivent être testées, celles-ci doivent être combinées avec des opérateurs logiques. VB accepte les opérateurs logiques suivants: *AND*, *OR*, *NOT* et *XOR*. La signification de chacun d'eux est présentée dans le tableau qui suit:

Opérateurs logiques

Opérateur	Signification	Exemple	Résultat
AND	Connexion ET. Il faut que les conditions soient vraies pour que le résultat soit vrai	(1 = 1) AND (2 < 4) (1 > 2) AND (2 = 4)	True False

OR	Connexion OU. Il faut que l'une des deux conditions soit vraie pour que le résultat soit vrai	(1 = 2) OR (3 < 2) (1 > 2) OR (2 > 1)	False True
NOT	Connexion NON. La valeur logique est inversée	EstCeVrai = True NOT EstCeVrai	False
XOR	Connexion OU exclusif. Une seule des deux conditions doit être vraie pour que le résultat soit vrai	(1 = 1) XOR (2 = 2) (2 > 1) XOR (3 < 1)	False True

1.2.2 IIf (Condition, ValeurSiVrai, ValeurSiFaux)

Cette instruction (IIF) fonctionne comme le **IF** d'EXCEL.

Syntaxe

```
IIf (Condition, ValeurSiVrai, ValeurSiFaux)
```

Exemple

```
Dim Note As Single
Dim Réponse As String
Note = InputBox (" Tapez votre note ")
Réponse = IIf (Note >= 10, " Admis ", " Ajourné ")
MsgBox (Réponse)
```

1.2.3 Select case ... Case ... Case ...Else Case ... End Select

L'instruction **Select Case** est une instruction conditionnelle *alternative*, c'est-à-dire qu'une *expression* peut être testée par rapport à plusieurs valeurs possibles.

Syntaxe

```
Select Case expression
    Case Liste_Valeurs_1
        Instruction11
        Instruction12
    ...
    Case Liste_Valeurs_2
        Instruction21
    ...
    ...
Else Case
    InstructionElse1
    InstructionElse2
    ...
End Select
```

Les instructions se trouvant après '*Case Liste_Valeurs_i*' seront exécutées si '*expression = à l'un des éléments de Liste_Valeurs_i*', *i* = 1, 2, 3, Sinon, les instructions se trouvant après '*Else Case*' seront exécutées. *Liste_Valeurs_i* peut être :

- une suite de valeurs : 1, 3, 5, 7, 9
- une fourchette de valeur : 0 To 9
- une plage de valeur : **Is** >= 10 (Is est un mot réservé)

Exemple

```

Select Case CodeASCIICaractère
    Case 65, 69, 73, 79, 85
        MsgBox(" C'est une voyelle ")
    Case 66 To 90
        MsgBox(" C'est une consonne ")
    Case Else
        MsgBox(" Ce n'est pas une lettre ")
End Select

```

Notez que '*Liste_Valeurs_i*' peut être une combinaison de listes de valeurs comme dans le cas des exemples suivants :

```

Case 1 To 4, 7 To 9, 11, 13, Is > NombreMAX
Case "Lundi", "Mercredi", "Dimanche", VariableJour

```

1.3 Tableaux

Un tableau permet de stocker une suite d'éléments de même type. L'accès à un élément précis se fait à l'aide d'un indice (valeur ou variable entière). En VB, pour un vecteur déclaré avec une dimension (N), le premier élément a l'indice 0, le deuxième a l'indice 1, le troisième a l'indice 2, ..., le dernier a l'indice N¹.

Syntaxe

```
Dim NomVecteur(N) As TypeVecteur
```

Cette instruction déclare un vecteur *NomVecteur* de taille *N+1*. Pour accéder au i^{ème} élément du vecteur, il faut préciser l'indice entre parenthèses comme suit : *NomVecteur(i-1)*, i doit être compris dans l'intervalle [0, N].

TypeVecteur est un type standard (Boolean, Integer, String, etc.) ou tout autre type (*type d'objet*) définie dans VB ou dans votre application.

Exemple

```
Dim TabTemp(12) As Single
```

Numéro	1	2	3	4	5	...
Température	6	5,5	7	11,5	15	...

L'accès à la case numéro 3 se fait par TabTemp(3) qui vaut 7.

Syntaxe

```
Dim NomVecteur(1 To N) As TypeVecteur ' déclare un vecteur de N éléments
```

Exemple

```
Dim TabMajuscules(65 to 90) As String
```

¹ Pour éviter toute confusion (et garder vos bonnes habitudes Pascal), déclarez toujours le vecteur avec une taille (N) et ignorez l'élément à l'indice 0. Le premier élément ne sera jamais utilisé.

Numéro	65	66	67	...	89	90
Majuscule	A	B	C	...	Y	Z

VB permet de travailler avec des tableaux de deux, trois, quatre, dimensions ou plus

Exemple d'un tableau à deux dimensions:

```
Dim ExempleMatrice(10, 10) As Single
```

ExempleMatrice est une matrice (de nombres réels) de 11 lignes et 11 colonnes et où *ExempleMatrice(1, 9)* est l'élément se trouvant à l'intersection de la première ligne et de la dixième colonne².

Exemple de déclaration d'un tableau à trois dimensions:

```
Dim ExempleMatrice(10, 10, 10) As Single ' matrice à trois dimensions
```

1.4 Instructions répétitives

Les instructions répétitives sont utilisées pour boucler sur une suite d'instructions.

1.4.1 For ... To ... Next

Si le nombre de boucles est connu à l'avance, on utilise l'instruction *For ... To ... Next*.

Syntaxe

```
For Compteur = Début To Fin [Step Incrément]
    Instructions
    [ ... Exit For]           ' pour une interruption préalable de la boucle
    [Instructions]
Next [Compteur]              ' le mot Compteur est facultatif
```

Le test (*Compteur* = *Début*) est effectué au début de la boucle. La variable numérique *Compteur* est incrémentée à chaque fin de boucle du nombre indiqué par l'incrément. Si l'*Incrément* (le pas par lequel *Compteur* augmente à chaque boucle) n'est pas spécifié, il est fixé par défaut à 1.

Si la valeur de *Fin* est inférieure à la valeur de *Début*, l'incrément est négatif. La valeur de *Compteur* peut être utilisée (par exemple, pour numéroter le passage dans la boucle) mais **ne doit pas** être modifiée dans le corps de la boucle.

Exemple

```
Dim i As Integer
Dim Chaîne As String
Dim TabInitial(1 To 12) As Single
For i = 1 To 12
    Chaîne = InputBox("Température N° " & Compteur)
```

² Aussi, pour éviter toute confusion (et garder vos bonnes habitudes Pascal), déclarez toujours la matrice avec une dimension N*N et ignorez la ligne et la colonne à l'indice 0.

```

        TabInitial(i) = Chaîne
Next i
'le i n'est pas obligatoire

```

1.4.2 Do While ... Loop / Do ... Loop While ...

Test antérieur

Syntaxe

```

Do While Condition
    Instructions
    [... Exit Do]
    [Instructions]
Loop

```

La condition est ici testée au début, c'est-à-dire à l'entrée de la boucle. Avec *While* (tant que), la boucle est répétée tant que la condition est vraie. Si la condition n'est pas vraie au départ, les instructions de la boucle ne sont pas exécutées.

Exemple

```

Do While MotProposé <> MotDePasse

    MotProposé = InputBox("Donnez votre mot de passe")

Loop

```

Cela présuppose que MotProposé soit initialisé par une valeur autre que MotDePasse (par exemple, la valeur par défaut "").

Test postérieur

Syntaxe

```

Do
    Instructions
    [... Exit Do]
    [Instructions]
Loop While Condition

```

La condition est alors testée à la fin de la boucle. Avec *While* (tant que), la boucle est répétée tant que la condition est vraie. Les instructions de la boucle sont donc exécutées au moins une fois.

Exemple

```

Do
    MotProposé = InputBox("Donnez votre mot de passe")
Loop While MotProposé <> MotDePasse

```

Cet exemple ne présuppose aucune initialisation de MotProposé.

1.4.3 Do Until ... Loop / Do ... Loop Until ...

Test antérieur

Syntaxe

```
Do Until Condition
    Instructions
    [... Exit Do]
    [Instructions]
Loop
```

La condition est ici testée au début, c'est-à-dire à l'entrée de la boucle. Avec *Until* (jusqu'à), la boucle est répétée jusqu'à ce que la condition soit vraie. Si la condition est vraie au départ, les instructions de la boucle ne sont pas exécutées.

Exemple

```
Do Until MotProposé = MotDePasse
    MotProposé = InputBox("Donnez votre mot de passe")
Loop
```

Cela présuppose que `MotProposé` soit initialisé par une valeur autre que `MotDePasse` (par exemple, la valeur par défaut "").

Test postérieur

Syntaxe

```
Do
    Instructions
    [... Exit Do]
    [Instructions]
Loop Until Condition
```

La condition est alors testée à la fin de la boucle. Les instructions de la boucle sont donc exécutées au moins une fois. Avec *Until* (jusqu'à), la boucle est répétée jusqu'à ce que la condition soit vraie.

Exemple

```
Do
    MotProposé = InputBox("Donnez votre mot de passe")
Loop Until MotProposé = MotDePasse
```

Cet exemple ne présuppose aucune initialisation de `MotProposé`.

1.4.4 For ... Each ... Next

C'est **une extension** de la boucle For ... To ... Next.

Syntaxe

```
For Each Elément In Ensemble
    Instructions
    [ ... Exit For]
    [Instructions]
Next [Elément]
```

Ensemble est le plus souvent un tableau.

Exemple

```
Dim TabHasard(100) As Integer
Dim Cellule As Integer
Dim Réponse As String
Randomize 'initialise le générateur de nombres au hasard
For Each Cellule In TabHasard
    Cellule = Rnd * 100 + 1 'génère un nombre au hasard entre 1 et 100
Next
For Each Cellule In TabHasard
    Réponse = Réponse & Cellule & " " 'Concaténation de chaînes de caractères
Next
MsgBox (Réponse)
```

1.4.5 Conclusion

Selon le problème à traiter, vous aurez **le choix** entre ces différentes structures de contrôle. Il s'agira de choisir **la plus élégante** ou du moins, celle qui ne provoquera pas de dysfonctionnement de votre programme.

Trouvez les erreurs dans les exemples suivants :

Exemple 1:

```
Dim VotreRéponse As String
Réponse = "LaRéponse"
Do
    VotreRéponse = InputBox("Donnez votre réponse")
Loop While VotreMot = Réponse
```

Exemple 2

```
Dim Cote As Single
Do Until Cote >= 0 And Cote <= 20
    Cote = InputBox("Taper une note entre 0 et 20")
Loop
```

1.5 Procédures et Fonctions

Comme dans le cas du langage *Pascal*, VB .NET permet l'utilisation des procédures et des fonctions avec ou sans paramètres. Rappelez vous que la grande différence entre la procédure et la fonction est que cette dernière retourne une valeur lorsqu'elle est appelée.

Lors de l'appel de la procédure, un paramètre peut être transmis soit par valeur, soit par **référence** (variable).

1.5.1 Procédure (Transmission par valeur : ByVal)

Pour transmettre un paramètre par valeur, celui-ci doit être **obligatoirement** précédé par le mot réservé **ByVal**. Sinon, il est considéré de passer par référence.

Syntaxe

```
Private Sub NomProcédure ( ByVal argument As Type, ... )
    Instruction1
    Instruction2
    ...
End Sub
```

Exemple

```
Private Sub Affectation( ByVal valeur1,valeur2 As integer)
    Dim Chaîne As String
    Chaîne = "La somme de " & valeur1 & " et " & valeur2 & " = "
    valeur1 = valeur1 + valeur2
    Chaîne = Chaîne & valeur1
    MsgBox (Chaîne)
End Sub
```

L'appel de la procédure se fait soit en inscrivant **call** suivi du nom de la procédure, et des paramètres à lui transmettre, soit en écrivant uniquement le nom de la procédure, suivi des paramètres à lui transmettre.

```
Dim X As integer
Dim Y As integer
Call Affectation (X, Y) ' avec les parenthèses
MsgBox (" Et X = " & X & " n'a pas changé ")
```

1.5.2 Procédure (Transmission par référence : ByRef)

Si **ByVal** n'est pas précisé ou si le paramètre est précédé par le mot réservé **ByRef**, la variable est transmise par référence (c'est-à-dire transmise en tant que **variable**). Ainsi, toute modification de la variable locale correspondante dans la procédure se répercute sur la variable utilisée lors de l'appel. VB suppose que la transmission se fait par référence si le mot réservé **ByVal** est omis.

Exemple

```
Private Sub Transvase ( valeur1 As Integer, valeur2 As Integer )
    Dim variable As Integer
    variable = valeur1
    valeur1 = valeur2
    valeur2 = variable
End Sub
```

L'appel suivant transvase le contenu de X dans Y et inversement.

Exemple

```
Dim X As Integer, Y As Integer
X = 100
Y = 200
MsgBox (" X = " & X & " et Y = " & Y)
Transvase(X, Y)
MsgBox (" Alors que maintenant X = " & X & " et Y = " & Y)
```

1.5.3 Fonction

Lors de la déclaration d'une fonction, la valeur qui doit être retournée par celle-ci doit être affectée *au nom de la fonction*. La déclaration de la fonction se termine par les mots réservés *"End function"*.

Syntaxe

```
Private function NomFonction( Argument As Type, ... ) As Type
    Instruction1
    Instruction2
    ...
    NomFonction = RésultatDeLaFonction
End function
```

Exemple

```
Private function Somme( valeur1 As Integer, valeur2 As Integer ) As integer
    Somme = Valeur1 + valeur2
End function
```

L'appel suivant retourne la somme de X et Y et affecte le résultat à la variable Z.

Exemple

```
Dim X As Integer, Y As Integer, Z As Integer
X = 10
Y = 20
Z = somme(X, Y)
```

1.5.4 Transmission d'un tableau comme argument d'une procédure ou d'une fonction

Pour transmettre un tableau comme argument d'une fonction ou d'une procédure, il suffit de déclarer (à l'intérieur des parenthèses) une variable (le nom local du tableau) *sans type, ni dimension*. Lors de l'appel de la fonction ou de la procédure, VB donne à cette variable le type et la taille du tableau envoyé. On peut aussi utiliser comme type de la variable locale, le type *Variant*. Comme tout variable, un tableau peut être envoyé par valeur ou par référence.

Ci-après vous trouvez un exemple de déclaration d'une procédure qui reçoit un vecteur (passation par référence: par défaut).

Exemple

```
Private Sub Init(vec)                                ' ou Private Sub init(vec As Variant)
    Dim i As Integer
    For i = 1 To 10
        vec(i) = 0
    Next
End Sub
```

L'appel de la procédure avec un vecteur comme argument se fait comme pour toute variable.

Exemple

```
Dim vecteur(10) As Integer
Call Init(vecteur)
```

1.5.5 Portée des variables, procédures et fonctions

Une application VB peut être composée d'un ou de plusieurs formulaires et d'un ou de plusieurs modules. Dans chaque module ou formulaire, des variables, des procédures et/ou des fonctions peuvent être déclarées. Dans chaque procédure et fonction, des variables locales peuvent être déclarées.

Une fonction ou une procédure peut être déclarée soit *Privée (Private)*, soit *Publique (Public)*. Le sens de *Privé ou Public* se comprend par rapport au formulaire ou au module dans lesquelles elles sont déclarées.

Se pose alors le problème de la portée des variables, des fonctions et des procédures.

Si une variable est déclarée au début de la procédure (fonction) qui la manipule (**Dim** ou **Private**), elle n'est alors valide que pour cette procédure (fonction). L'existence et la valeur de la variable disparaissent avec l'instruction **End Sub** (End Function). Toute référence à cette variable en dehors de cette procédure (fonction) provoquera une erreur de compilation. Si une variable est déclarée dans la section des déclarations d'un module (formulaire), elle est valide dans toutes les procédures (fonctions) du module (formulaire).

Une variable peut aussi être déclarée **Public** ou **Global** et sera alors valide pour toute l'application.

Exemple :

```
Global MotInitial As String ' premier mot à traiter
```

Le tableau qui suit résume la portée des variables, des procédures et des fonctions en fonction du type de déclaration (*Dim*, *Private* ou *Public*) et de l'endroit où la déclaration a eu lieu.

Portée des variables, procédures et fonctions

Type	Déclaré dans	Mot clé	Portée
Variable	Procédure événementielle	Dim	Procédure événementielle
Variable	Procédure / fonction générale du formulaire	Dim	Procédure / fonction générale
Variable	Procédure / fonction générale de module	Dim	Procédure / fonction générale
Variable	Partie générale d'un formulaire	Dim/private	Formulaire
Variable	Partie générale d'un module	Dim/private	Module
Variable	Procédure événementielle	Private/Public	Interdit
Variable	Procédure générale d'un formulaire	Private/Public	Interdit
Variable	Procédure générale de module	Private/Public	Interdit
Variable	Partie générale d'un formulaire	Public	Formulaire
Variable	Partie générale d'un module	Public	Projet
Procédure / fonction	Partie générale d'un formulaire	Private	Formulaire

Procédure / fonction	Partie générale d'un module	Private	Module
Procédure / fonction	Partie générale d'un formulaire	Public	Formulaire
Procédure / fonction	Partie générale d'un module	Public	Projet

1.5.6 Quelques fonctions globales

Les deux tableaux suivant résument quelques fonctions mathématiques et quelques fonctions pour la manipulation des chaînes de caractères.

Fonctions mathématiques

Fonction	Utilité	Exemple
Abs(Nb)	Donne la valeur absolue du nombre	Abs (- 89) = 89
Atn(Angle)	Donne l'arc tangente de l'angle	Atn(0) = 0
Cos(Angle)	Donne le cosinus de l'angle	Cos(0) = 1
Exp(Nb)	Donne l'exponentielle du nombre	Exp(1) = 2.71828
Fix(Nb)	Tronque les décimales du nombre	Fix(-4.6) = -4 Fix(4.6) = 4
Int(Nb)	Donne la partie entière du nombre Int et Fix ne diffèrent que pour les valeurs supérieures à 0	Int(-4.6) = -5 Int(4.6) = 4
Log(Nb)	Donne le logarithme naturel (base e)	Log(1) = 0
Sgn(Nb)	Donne le signe du nombre : 1, 0 ou -1	Sgn(- 89) = -1
Sin(Angle)	Donne le sinus du nombre	Sin(0) = 0
Sqr(Nb)	Donne la racine carrée du nombre	Sqr(4) = 2
Tan(Angle)	Donne la tangente de l'angle	Tan(0) = 0
Round(Nb)	Arrondi à la valeur supérieure si (Nb - Int(Nb)) > 5 inférieure si (Nb - Int(Nb)) <= 5	Round(4.5) = 4 Round(4.51) = 5

Fonctions de chaîne de caractères

Fonction	Utilité	Exemple
Asc(Car)	Donne le code ASCII d'un caractère	Asc("A") = 65
Chr(N)	Donne le caractère correspondant au code ASCII	Chr(65) = "A"
Len(Chaîne)	Donne la longueur d'une chaîne	Len ("Orange") = 6
Lcase(Chaîne)	Transforme la chaîne en minuscules	Lcase("ABC") = "abc"
Ucase(Chaîne)	Transforme la chaîne en majuscules	Ucase("abc") = "ABC"
LTrim(Chaîne)	Supprime les espaces de tête	LTrim(" Hello") = "Hello"
RTrim(Chaîne)	Supprime les espaces de fin	RTrim("Hello ") = "Hello"
Trim(Chaîne)	Supprime les espaces de tête et de fin	Trim(" Hello ") = "Hello"
Left(Chaîne, N)	Renvoie les N caractères de gauche	Left("Auto", 2) = "Au "
Right(Chaîne, N)	Renvoie les N caractères de droite	Right("Auto", 2) = "to "
Mid(Chaîne, Pos, N)	Renvoie N caractères à partir de la position Pos	Mid("Locom", 3, 2) = "co"
InStr(Chaîne, Car)	Renvoie la position de la première occurrence du caractère dans la chaîne ou la valeur 0 si la chaîne ne contient pas le caractère	InStr ("Locom", "o") = 2 InStr ("Locom", "a") = 0
Str(N)	Convertit N en chaîne de caractères	Str(123) = "123"

String(N, Car)	Génère N fois le caractère spécifié	String(5, "A") = "AAAAA"
Space(N)	Génère des espaces	Space(4) = " "
Val(Chaîne)	Convertit en nombre les chiffres d'une chaîne (la conversion s'arrête au premier caractère qui n'est pas un chiffre, ou à 0 s'il n'y a pas de chiffre en tête)	Val("123") = 123 Val("123abcd") = 123 Val("abcd123") = 0

1.5.7 Interruption de séquences

Pour interrompre l'exécution d'une séquence d'instructions (dans une fonction, procédure ou boucle For), on utilise l'instruction *Exit*. Le tableau suivant résume son utilisation.

Les possibilités d'interruption de séquences

<i>Instruction</i>	<i>Porté</i>	<i>Description</i>
Exit function	Limitée à la fonction	Interruption de la fonction, sans exécution des instructions restantes
Exit Sub	Limitée à la procédure	Interruption de la procédure, sans exécution des instructions restantes
Exit For	Limitée à la boucle For	Interruption de la boucle, sans exécution des instructions restantes
Exit Do	Limitée à la boucle Do	Interruption de la boucle, sans exécution des instructions restantes

SECTION 2. INTRODUCTION A L'ENVIRONNEMENT DE DEVELOPPEMENT VB

La programmation en Visual Basic (VB) se fait principalement (comme le nom du langage l'indique) de manière visuelle. Vous serez cependant souvent amenés à (mettre la main à la pâte et) programmer en écrivant du code VB.

VB est avant tout un environnement de développement d'applications informatiques. VB (entre autres) offre:

- un environnement graphique de développement permettant de développer *visuellement* une grande partie de votre application. Voir figure 1
- un langage de programmation orienté objet (voir plus loin le concept d'objet)
- des composants logiciels (ActiveX) ou des bibliothèques (pré-programmées) très puissants et prêts à être intégrés et utilisés dans votre application. *On verra par la suite qu'une grande partie de votre apprentissage de VB consistera à apprendre l'utilisation de ces composants logiciels (contrôles)*
- la possibilité d'intégrer aisément de nouveaux composants ActiveX développés dans le commerce
- une grande facilité pour développer de nouveaux composants ActiveX

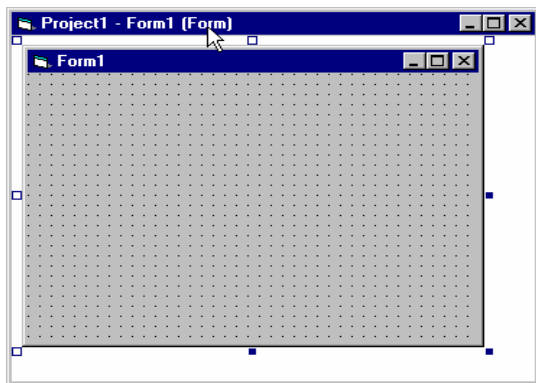
VB est devenu grâce à ces atouts un langage de programmation très utilisé de par le monde, supplantant un grand nombre de langages de programmation.

Pour rentrer dans le vif du sujet, nous allons regarder de quoi sera composée votre application (programme) VB. Celle-ci sera composée, entre autres, de deux parties essentielles: un ou plusieurs **formulaires** (la partie visuelle ou graphique) et le **code VB** (*des formulaires et modules*).

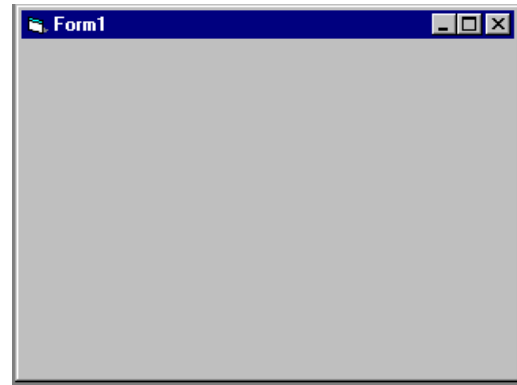
2.1 Environnement VB

2.1.1 Formulaires (Forms)

La partie visuelle de votre application ou projet est composée principalement d'*UN* ou de *PLUSIEURS* formulaires (Forms). Un formulaire n'est rien d'autre qu'une fenêtre (Window). Les figures 1 et 2 présentent des exemples de formulaires.



(a) Lors de la conception

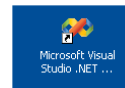


(b) Lors de l'exécution

Figure 2 : Exemple d'un formulaire vierge

Pratique.

Pour démarrer VB, double cliquez sur l'icône *VB6.exe*



La fenêtre qui apparaît alors à l'écran vous propose soit d'ouvrir un projet existant, soit de créer un nouveau projet (new project) (figure 3).

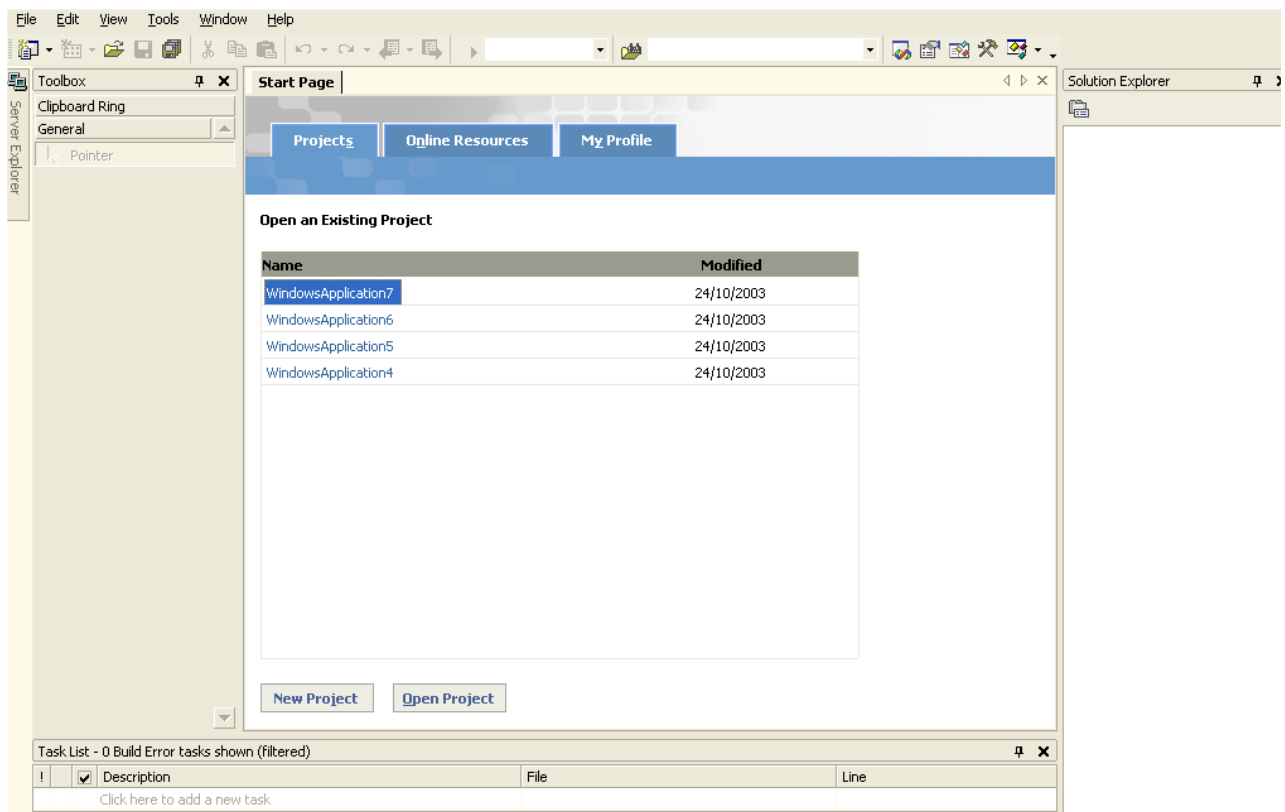


Figure 3 : Fenêtre de choix d'ouverture ou de création de projet

Lorsque vous faites un nouveau projet, il vous reste à choisir le type de projet que vous allez créer (Visual Basic Projects – sur la gauche de l'écran), le modèle utilisé (Application Windows) ainsi que le nom associé à ce projet et l'emplacement sur le disque où seront stockées ces informations (attention, sur les ordinateurs de l'IAG, vous ne pouvez enregistrer que sur le disque L)

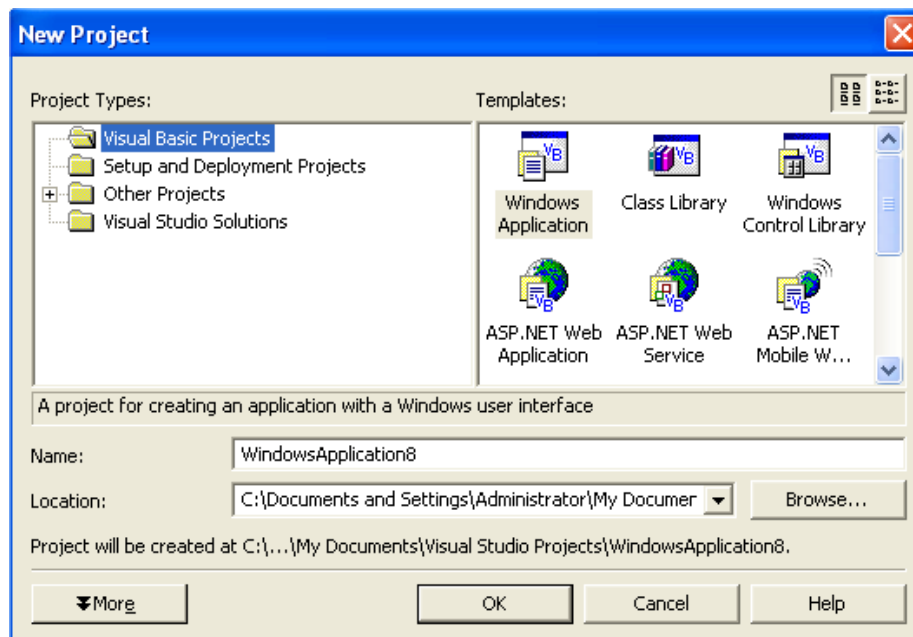


Figure 4 : Fenêtre de création d'un nouveau Projet

Il est également toujours possible de créer un nouveau projet en sélectionnant dans la barre du menu VB, *File, New Project*.

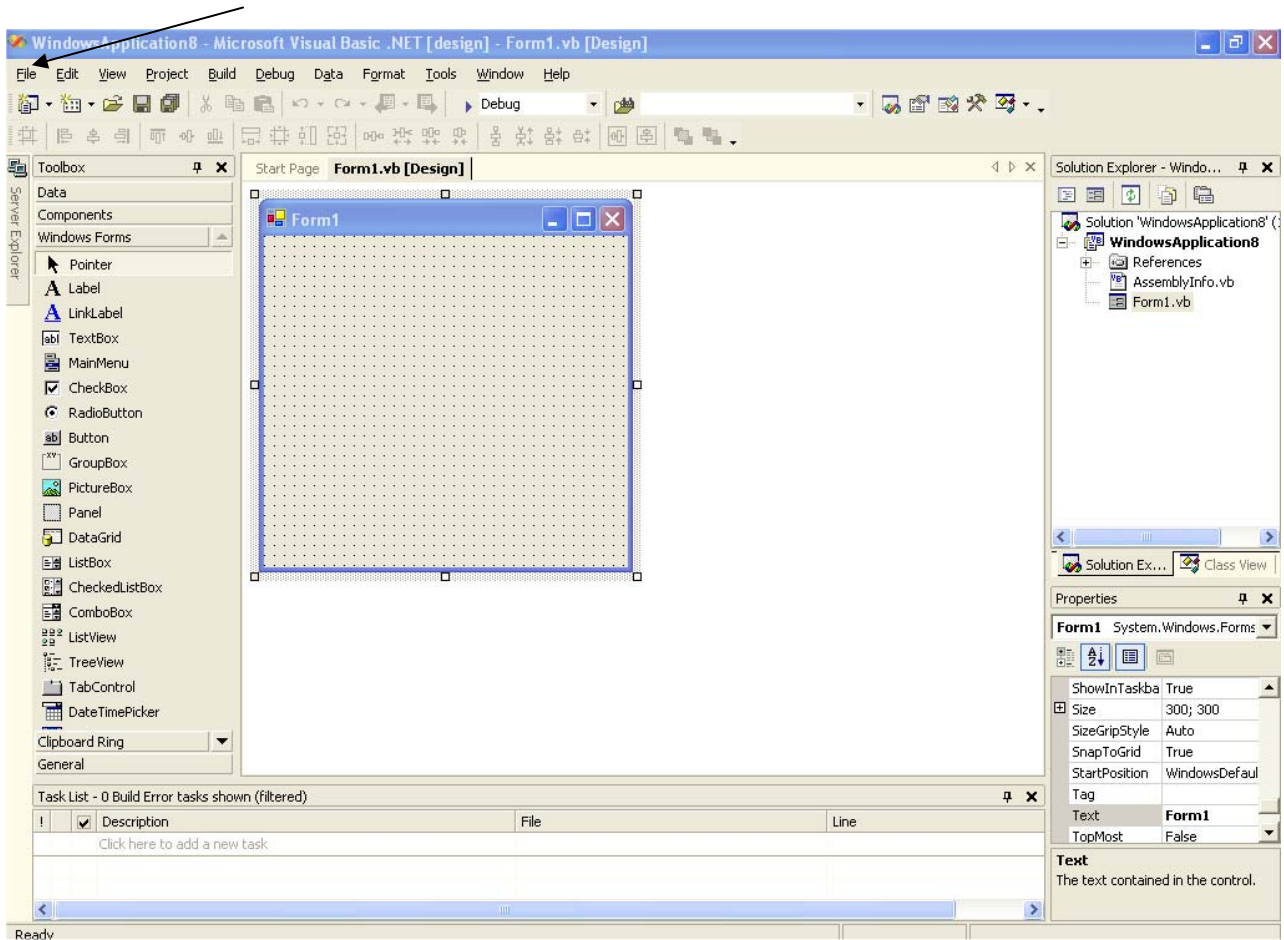


Figure 5 : Projet nouvellement créé, composé d'un seul formulaire vierge (Form1)

VB va créer un projet composé d'un formulaire portant un nom généré automatiquement : *Form1*, voir figure 5.

L'environnement VB est composé de trois types d'éléments :

- une zone de barre de menus et de barres d'outils,
- une zone de travail central
- une multitude de fenêtres qui gravitent autour, constituant les différents outils mis à votre disposition pour travailler

2.1.2 L'explorateur des solutions

L'explorateur des solutions présente de manière arborescence et visuelle les *objets* composant l'application chargée. La figure 6(a) montre que le projet de nom '*WindowsApplication8*' est composé d'un seul formulaire de nom '*Form1*'.

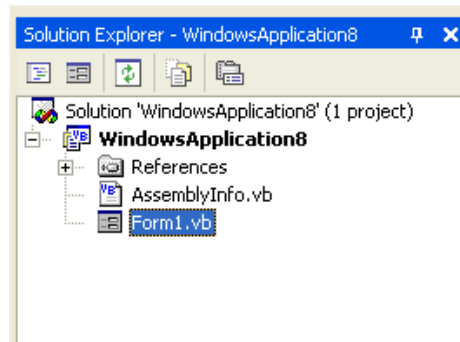


Figure 6(a): L'explorateur des solutions

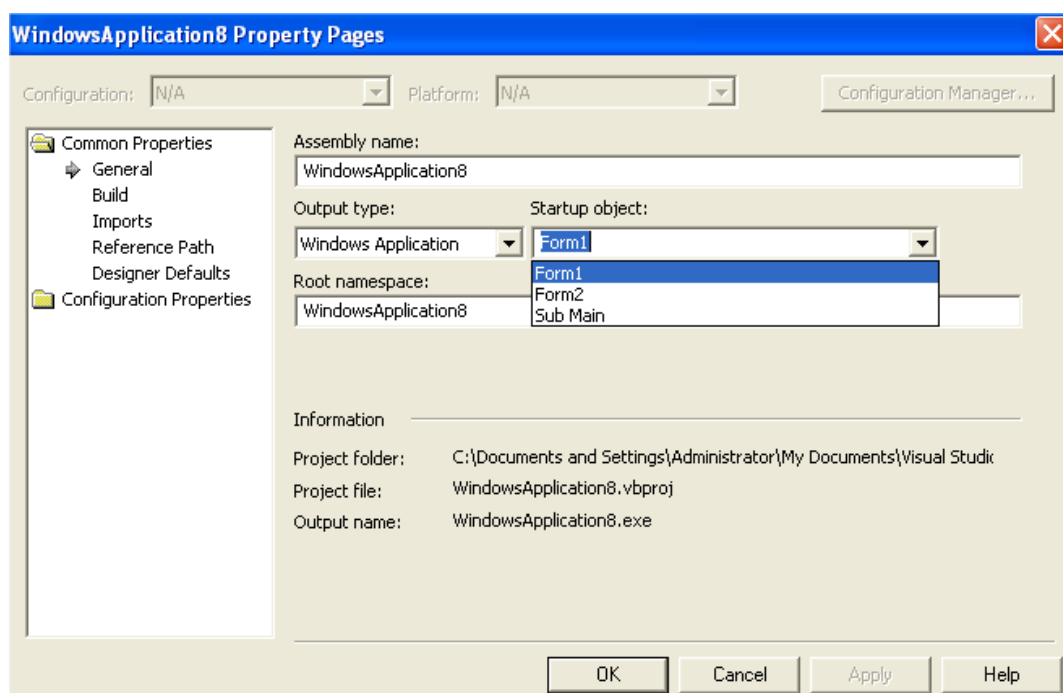


Figure 6(b): Fenêtre des propriétés du projet

Les '**Forms**' constitueront les *objets* principaux qui composeront vos projets dans ce cours. Notez qu'un projet peut être composé d'un ou de plusieurs formulaires. Dans ce dernier cas, il faut préciser à VB le formulaire qui sera chargé en premier, en cliquant sur le nom du projet avec le bouton droit de la souris et ensuite sur *Properties*, voir figure 6(b) où le projet est composé de deux formulaires, *Form1* et *Form2*.

2.1.3 La fenêtre Properties

La fenêtre **Properties** présente les propriétés (ou attributs) de l'objet sélectionné. La figure 6 présente les propriétés de l'objet (sélectionné) *Form1*.

On peut citer quelques propriétés de '*Form1*' et leurs valeurs respectives:

- *Name* = *Form1*, nom logique utilisé pour référencer l'objet dans du code VB.
- *BackColor* = *&H800000*, couleur de fond de l'objet *Form1*.
- *Text* = *Form1*, nom qui apparaît visuellement sur l'objet, celui-ci peut être différent de la propriété *Name*.

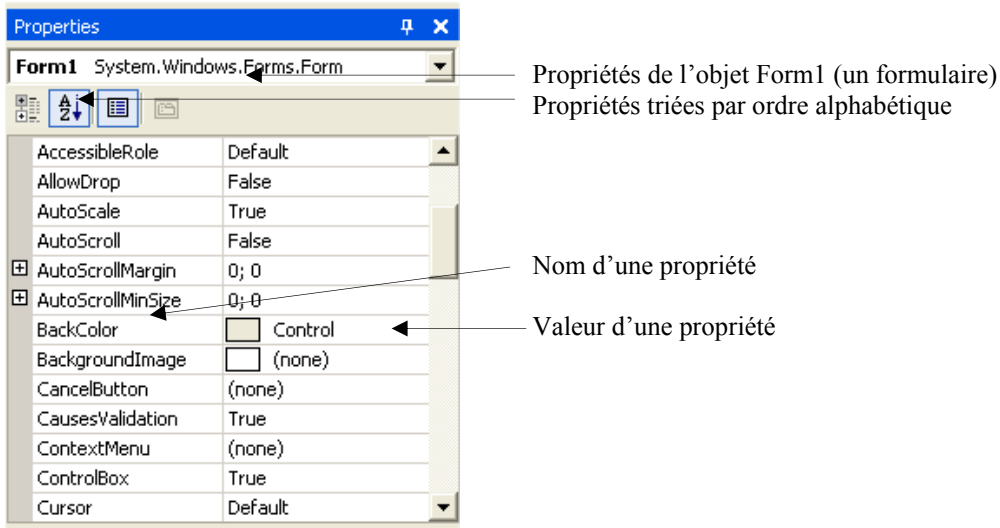


Figure 7: Fenêtre **Properties**

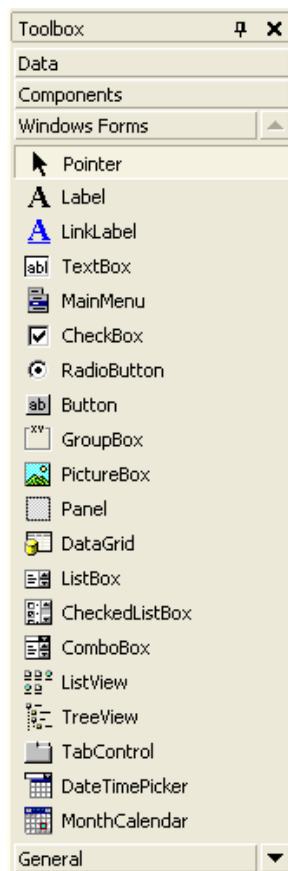
Notez que la valeur de chaque propriété peut être modifiée en cliquant sur la colonne de droite de la fenêtre **Properties**.

Pratique. Modifiez la valeur des propriétés suivantes :

- *Text* = *Convertisseur FB en Euro*,
- *BackColor* = à votre guise,
- Etc.

N.B. Ne modifiez que les propriétés dont vous comprenez le sens

2.1.4 La boîte à outils et les contrôles standards



La partie graphique de votre application va contenir un (ou plusieurs) formulaire(s). Sur un formulaire, on peut placer un ou plusieurs *objets graphiques* ou ce qu'on appellera des **contrôles** (Bouton à cliquer, Champ libellé (texte statique), Champ texte à saisir au clavier, Menu, etc.).

Ces contrôles sont des objets pré-programmés dont l'utilité principale est de faciliter l'*interaction* avec l'utilisateur. Chacun de ces objets graphiques a une fonctionnalité bien précise. Le tableau suivant résume les contrôles standards de base les plus utilisés:

Figure 8 : Toolbox

Contrôle	Nom du contrôle	Utilité
	Label	Afficher un texte statique : un libellé
	Text Box	Afficher et rentrer une valeur au clavier
	Button	Lancer l'exécution une procédure événementielle
	ListBox	Afficher une liste statique de valeur
	ComboBox	Combiner l'utilité des contrôles TextBox et ListBox
	PictureBox	Afficher une image dans un cadre. Celui-ci peut être redimensionné en fonction de l'image (<i>Autosize = True</i>)
	RadioButton	Sélectionner une option. Si utilisé en plusieurs instances (<i>Option Button</i>), une seule peut être choisie
	Check Box	Sélectionner une option. Si utilisé en plusieurs instances (<i>Check Box</i>), une ou plusieurs peuvent être choisies
	GroupBox	Créer une fenêtre au sein d'un formulaire et créer un groupe de contrôles.

Les contrôles standards dans VB se trouvent dans la Boîte à outils (*ToolBox*), voir figure 8. D'autres contrôles plus élaborés (*Components*) peuvent être ajoutés dans la boîte à outils, en sélectionnant dans la barre du menu : *Project, Add Components*.

Comment placer un contrôle sur un formulaire ?

Sélectionnez dans la boîte à outils le contrôle désiré. Dessinez sur le formulaire le rectangle dans lequel vous voulez placer le dit contrôle. Pour ce faire, cliquez (sans relâcher) sur le bouton gauche de la souris, sur le coin haut gauche du rectangle et déplacez la souris vers le coin bas droit du rectangle puis relâchez le bouton de la souris. Le contrôle apparaît par magie sur le formulaire.

Comment déplacer un contrôle ou le redimensionner ?

*Sélectionnez d'abord (en cliquant dessus) le contrôle placé sur le formulaire. Glissez le vers l'endroit désiré ou cliquez et tirez sur l'un des huit petits carrés bleus délimitant l'objet sélectionné (le contrôle **Label** Euro est sélectionné dans la figure 8).*

Chaque contrôle peut être vu comme un objet défini par un ensemble de propriétés. Quand un contrôle, placé sur un formulaire, et sélectionné, ses propriétés apparaissent dans la fenêtre **Propriétés**.

De manière générale, une fois qu'un *objet* est sélectionné, ses propriétés apparaissent dans la fenêtre **Propriétés**. Notez que certains objets (contrôles, formulaires, etc.) ont les mêmes propriétés, par exemple *Name* et *Text*, mais ont **bien évidemment** des valeurs différentes qui leurs sont propres.

Pratique. Placez les contrôles suivants sur le formulaire qui a été créé:

Contrôle	Propriété	Valeur
Label	Text	FB
TextBox	Name	Franc_Belge
Label	Text	Euro :
CommandButton	Text	Convertir
CommandButton	Text	Sortir

Modifiez leurs positions et leurs propriétés pour que le formulaire apparaisse comme suit :



Figure 9 : Exemple de conception d'un formulaire

Nous venons de créer un programme (graphique et statique) sans écrire aucune ligne de *code* VB. Celui-ci peut d'être exécuté. Ce programme ne fait qu'afficher une boîte de dialogue ayant la forme de la figure 9. Pour lui donner vie (lui faire faire ce qui est désiré), c'est-à-dire convertir du *Franc Belge* en *Euro*, il faudra mettre la main à la pâte et écrire du code VB.

Comment exécuter votre programme ?

*Pour exécuter un programme, appuyez sur la touche **F5** ou sélectionnez dans la barre de menu, **Run**, **Start**, ou cliquez sur le bouton **Start**.*

Comment arrêter l'exécution d'un programme ?

Pour arrêter l'exécution de votre programme, cliquez sur la petite croix située en haut à droite du formulaire ou cliquez sur le bouton End. 

2.2 Programmation par événements

A la différence de la programmation séquentielle, où les instructions s'exécutent de manière séquentielle, VB est un langage qui permet de réaliser de la programmation par événements, c'est-à-dire programmer des procédures qui s'exécutent quand un événement est déclenché. La plupart du temps, l'événement est déclenché par l'utilisateur du programme.

Quand on travail dans un environnement multifenêtrés (Windows) chaque fois, qu'on *clique sur la souris*, qu'on *ouvre ou ferme une fenêtre*, qu'on *appuie sur une touche du clavier*, on déclenche un événement auquel le programme utilisé réagit. La programmation par événements consiste à programmer ce que le programme doit faire quand un événement particulier survient.

A chaque objet VB (*contrôle, formulaire, etc.*) peut être associé *une ou plusieurs procédures événementielles* écrites avec le langage de programmation VB.

Procédures événementielles (*Private Sub NomObjet_NomEvénement... End Sub*)

Une procédure *événementielle* n'est rien d'autre qu'une procédure classique mais qui s'exécute quand un *EVENEMENT particulier* se produit ³.

La déclaration de l'événement *NomObjet_NomEvénement()* se fait comme suit (voir syntaxe), où *NomObjet* est le nom de l'objet auquel est rattaché l'événement *NomEvénement*. Comme dans une procédure classique, *aucun, un ou plusieurs paramètres et leurs types respectifs* peuvent être déclarés entre parenthèses.

Pour attacher une procédure événementielle à un objet, il suffit de « double cliquer » sur celui-ci. VB inscrit alors la déclaration de la procédure avec des paramètres par défaut (ne pas modifier ces paramètres).

Syntaxe

```
Private Sub NomObjet_NomEvénement ( Argument As Type, ... )
    Instruction1
    Instruction2
    ...
End Sub
```

Un ensemble d'événements peut être rattaché à chaque type d'objet. Ci-dessous quelques exemples d'événements :

Événement	Se produit quand
<i>Click</i>	<i>On clique sur le bouton gauche de la souris</i>
<i>DbClick</i>	<i>On double clique sur le bouton gauche de la souris</i>
<i>Load</i>	<i>L'objet NomObjet est chargé</i>
<i>Change</i>	<i>La valeur stockée par l'objet Nomobjet change</i>
<i>MouseDown</i>	<i>On clique sur la souris sans relâcher le bouton</i>

³ Notez qu'une procédure événementielle peut être aussi appelée dans du code comme une procédure classique

<i>MouseUp</i>	<i>On a relâché le bouton de la souris</i>
<i>MouseMove</i>	<i>On a bougé la souris</i>
<i>KeyDown</i>	<i>On a appuyé sur une touche du clavier sans la relâcher</i>
<i>MouseUp</i>	<i>On a relâché une touche du clavier</i>
<i>MousePress</i>	<i>On a appuyé sur une touche du clavier et on l'a relâché</i>

A chaque formulaire sera associé un fichier logique portant le nom '*Nom Formulaire*', voir figure 10. Celui-ci contiendra le **code VB** des différentes procédures relatives aux événements associés au formulaire en question ainsi qu'aux différents objets qui lui sont rattachés.

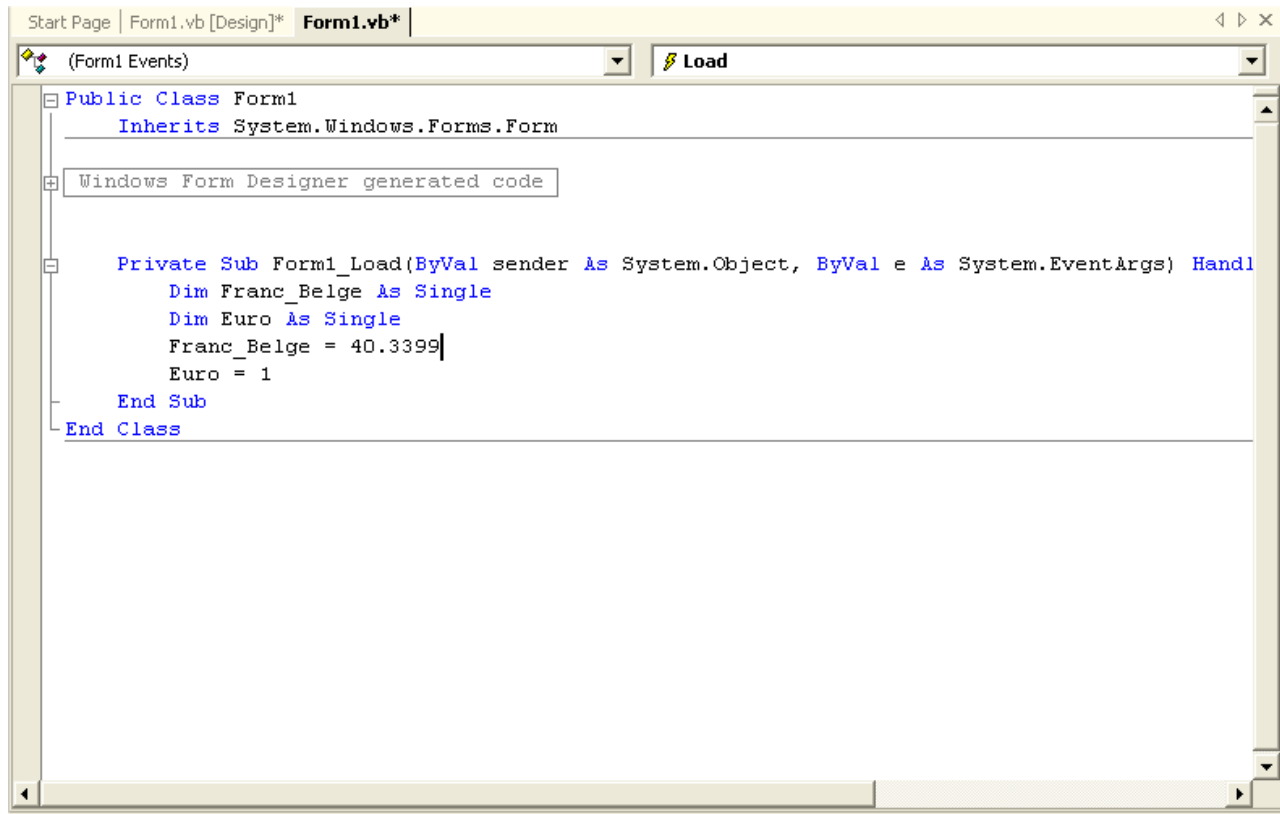


Figure 10 : Fenêtre du code VB relative au formulaire *Convertisseur*

Comment attacher une procédure événementielle « Load » à un formulaire ?

*Pour attacher une procédure événementielle à un formulaire, double cliquez sur celui-ci (et non pas sur un des contrôles qui le composent). VB ouvre alors une fenêtre textuelle et place le curseur dans le cadre d'une procédure événementielle particulière : **Form_Load()**.*

Form_Load()

La procédure de nom *Form_Load()* s'exécute lors du chargement du formulaire correspondant, c'est-à-dire **avant que** le formulaire n'apparaisse à l'écran.

Pratique. Placez les deux lignes de codes comme indiqué à la figure 10 (entre les deux lignes *Private Sub Form_Load()* et *End Sub*). Ainsi, avant que le formulaire n'apparaisse à l'utilisateur, *Franc_Belge.Text* et *Euro.Text* seront initialisés à 40.3399 et à "Euro : 1" (voir figure 9).

Exécutez votre programme pour noter l'effet de l'initialisation. Y-a-t il moyen d'initialiser sans écrire du code ?

Pratique.

- On désire que, lorsque l'utilisateur clique sur le bouton *Convertir* (figure 9), une procédure s'exécute et convertisse le montant dans la zone *Franc_Belge* et donne le montant équivalent en Euro (dans la zone label).
- On désire que, lorsque l'utilisateur clique sur le bouton *Quitter* (figure 9), une procédure s'exécute et ferme la fenêtre. L'instruction *End* ferme une fenêtre.

Exécutez votre programme, introduisez un montant en Franc Belge et appuyez sur *Convertir*.

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
    Franc_Belge.Text = 40.3399
    Euro.Text = "Euro = 1"

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
    Dim valEuro As Double
    valEuro = Franc_Belge.Text / 40.3399
    Euro.Text = "Euro = " & valEuro

End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
    End
End Sub
End Class

```

Figure 11 : Fenêtre du code VB relative au formulaire *Convertisseur* (suite)

Comment Sauver votre travail ?

Sélectionnez dans la barre du menu : *File, Save Project as*. VB vous demandera de donner un nom à votre projet et, à chaque formulaire et module, le composant.

L'intérêt de donner un nom à chaque *formulaire et module* réside dans le fait qu'un formulaire ou un module peut être **réutilisé** dans des projets différents.

Comment ajouter un nouveau formulaire dans un projet ?

Cliquez avec le bouton droit de la souris sur le mon du *Projet* se trouvant dans la fenêtre *Projet*, sélectionnez dans le menu proposé : *Add, Windows form*.

Comment ajouter un formulaire existant dans un projet ?

Cliquez avec le bouton droit de la souris sur le mon du *Projet* se trouvant dans la fenêtre *Projet*, sélectionnez dans le menu proposé : *Add, Add Existing Items*. Sélectionnez le nom du fichier correspondant au formulaire recherché, puis appuyez sur *Open*.

Comment retrouvez les différentes fenêtres (ToolBox, Project, Properties) ?

Si ces fenêtres sont fermées vous pouvez toujours les ouvrir en sélectionnant dans le barre du menu, View (Toolbox, Project Explorer, Properties Windows).

En résumé

- Les objets manipulés sont appelés des **contrôles** (bouton de commande, boîte de dialogue, zone de texte, zone d'image, etc.)
- L'interface utilisateur créée est multifenêtrée. Une fenêtre est appelée un **formulaire** (Form). Un **formulaire** est lui-même un contrôle.
- Chaque contrôle peut réagir à des **événements** qui lancent des procédures (dédiées) codées en VB.
- Des **modules généraux** de code peuvent porter sur tout le programme. Ces modules sont réutilisables.

2.3 Exercices

- a. Réalisez les parties pratiques des sous-sections précédentes
- b. Modifiez le programme de conversion pour qu'il puisse convertir dans les sens FB->Euro et Euro->FB. Votre formulaire doit apparaître comme suit :



Figure 12 : Exercice à programmer

- c. Modifiez votre programme et placez avant l'instruction, *End*, l'instruction suivante :
`MsgBox("Je termine et je sors, confirmez avec OK")`
- d. Après cette introduction, vous êtes censés être capable de répondre aux questions suivantes :
 1. Qu'est ce qu'un formulaire ?
 2. De quoi sera composé votre application VB ?
 3. Qu'est ce qu'un contrôle ? Donnez des exemples.
 4. Comment placer un contrôle sur un formulaire ?
 5. Qu'est-ce qu'une procédure événementielle ? Donnez quelques exemples de procédures événementielles.
 6. Comment associer une procédure événementielle à un contrôle ?
 7. Quant la procédure événementielle *Click()* s'exécute-t-elle ?
 8. Quant la procédure événementielle *Load()* s'exécute-t-elle ?
 9. Qu'est ce qu'un objet ? Donnez des exemples d'objets en VB.
 10. Qu'est ce qu'une propriété ? Donnez des exemples de propriétés.
 11. Comment changer la propriété d'un objet ?
 12. Quelle est la différence entre la propriété *Name* et la propriété *Text* ?
 13. Quel est le lien entre la propriété *Name* du contrôle **TextBox** et la propriété *Name* du contrôle **Label** ?
 14. Quel est le lien entre la propriété *Name* du contrôle **TextBox** et la propriété *Name* du formulaire dans lequel il se trouve ?
 15. Comment exécuter et fermer votre application ?
 16. Quel est l'intérêt de sauvegarder chaque formulaire sous un nom particulier ?
 17. Comment ajouter un nouveau formulaire (vierge) à un votre projet ?
 18. Comment ajouter un formulaire existant à un votre projet ?
 19. Comment sauvegarder votre projet et le recharger à nouveau ?
 20. Que fait l'instruction `MsgBox("Chaîne de caractères")` ?
 21. Quelle est la différence majeure entre une procédure et une fonction ?
 22. Comment effectuer une transmission de paramètre(s) par référence ?

SECTION 3. LES CONTROLES

3.1 Concept d'objet

Le concept d'objet

Comme vous l'avez et vous allez encore le constater, le terme **OBJET** est souvent cité dans ce texte, et ceci est loin d'être le fruit du hasard. En effet, VB .NET est un *langage orienté objet*, c'est-à-dire que toute **CHOSE** que vous aurez à manipuler et à utiliser n'est rien d'autre qu'un *OBJET indépendant*. Un objet est défini par un **nom** et un certain nombre de **propriétés**. Il est aussi défini par un ensemble de méthodes (procédures ou fonctions).

Notez bien que les propriétés et les méthodes qui définissent l'objet ne peuvent être invoquées qu'en spécifiant le nom de celui-ci.

Le concept de propriété d'un objet

Une propriété d'un objet est un attribut ou une caractéristique de celui-ci. Chaque propriété porte un nom (attribut ou variable) et a une valeur qui lui est associée. La figure 7 montre une partie des propriétés de l'objet portant le nom 'Form1'. Comme propriété d'un objet, on peut citer: nom, forme graphique, dimension, couleur, structure de données associée, etc.

Pour accéder à la propriété d'un objet avec du code VB, il faut **obligatoirement** préciser le nom de l'objet suivi d'un point suivi du nom de la propriété en question. On peut ainsi distinguer et utiliser les mêmes priorités appartenant à des objets différents. Pour accéder et modifier les propriétés d'un objet, on procède comme dans l'exemple suivant :

Exemple

```
Form1.Text = " Convertisseur FB en Euro "
Form1.BackColor = &H800000
Label1.Text = " Entrez votre texte "
Button.Text = " Franc Belge "
```

Pour les objets existants lors du développement, leurs propriétés peuvent **aussi** être modifiées à l'aide de la fenêtre de propriétés : **Properties**, voir figure 7.

Le concept de méthode d'un objet

Une méthode d'un objet est une fonction ou une procédure rattachée à l'objet en question. Pour être appelée, comme dans le cas de la propriété, elle doit être précédée par le nom de l'objet correspondant suivi d'un point.

Syntaxe

```
NomObjet.NomProcedure( paramètre1, paramètre2, ...)
Variable = NomObjet.NomFonction (paramètre1, paramètre2, ...)
```

*Exemple**Form1.ShowDialog**' fait apparaître la fenêtre Form1*

La figure 13 montre l'éditeur des classes (objets) VB. La colonne de droite présente les méthodes et les propriétés de la classe sélectionnée. La fenêtre du bas donne une explication succincte de la classe, propriété ou méthode sélectionnée. Pour ouvrir l'éditeur des classes VB, il suffit de cliquer sur le bouton *Object Browser*.

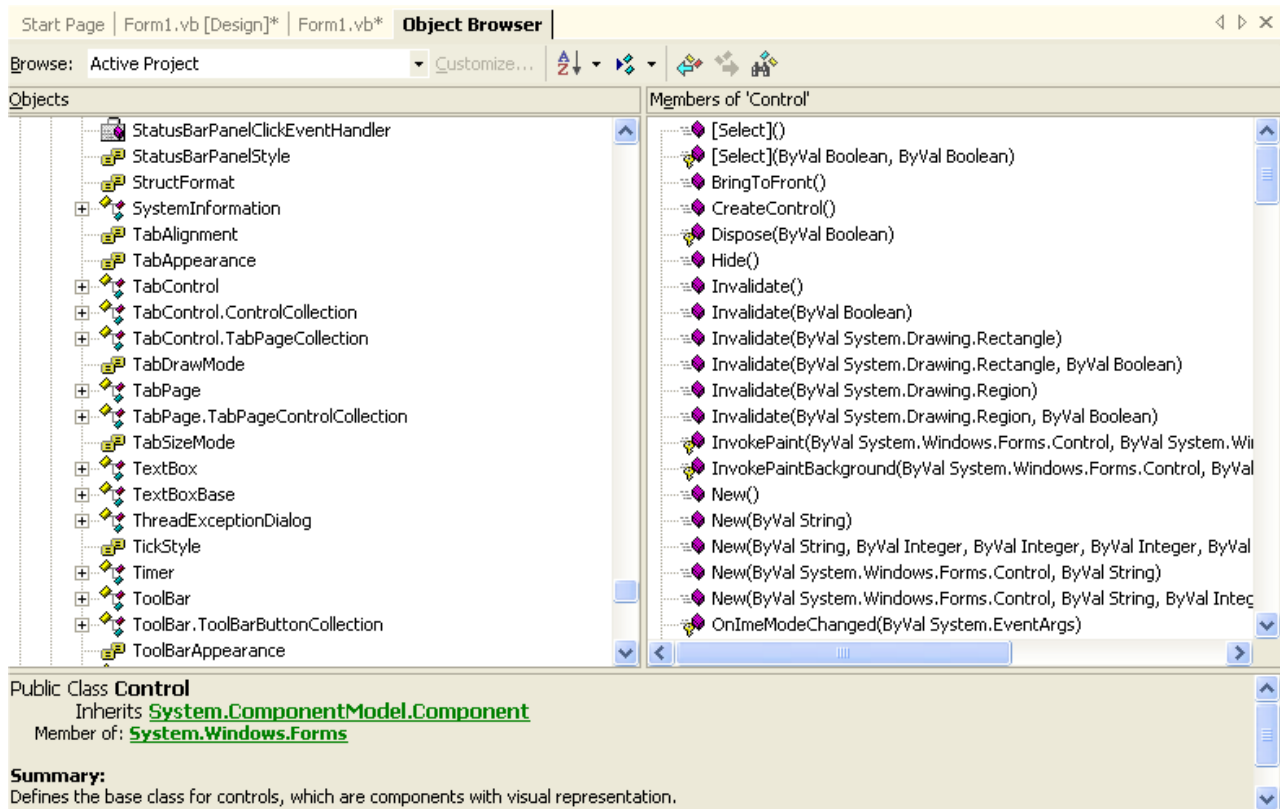


Figure 11 : Editeur des objets VB

3.2 Contrôles standards

3.2.1 La propriété "Name"

Dans tous les contrôles, la propriété *Name* permet de référencer le contrôle correspondant dans du code VB. A l'intérieur d'un même formulaire, la propriété *Name* doit être unique. Comme indiqué ci-dessus, *Name* permettra aussi d'accéder aux différentes propriétés et d'appeler les différentes méthodes de l'objet.

Lorsqu'un contrôle est placé sur un formulaire, VB lui affecte un nom, généré automatiquement, composé du nom du contrôle, suivi d'un chiffre correspondant au nombre de contrôles de même type déjà intégrés dans le formulaire.

3.2.2 Label

Le contrôle *Label* permet d'afficher un texte statique. La propriété (de type String) chargée de stocker ce texte (une chaîne de caractères) est la propriété *Text*. Celui-ci sera affiché lors de l'affichage du formulaire dans lequel il est placé. L'instruction qui suit modifie le texte correspondant au contrôle *Label* de nom *Label1*.

Exemple

```
Label1.Text = "l'équivalent en Euro = 15.689,89"
```

3.2.3 TextBox

Le contrôle *Textbox* permet d'afficher et de saisir un texte au clavier. La propriété (de type String) chargée de stocker ce texte (une chaîne de caractères) est la propriété *Text*. Celui-ci sera affiché lors de l'affichage du formulaire dans lequel il est placé et modifiable par l'utilisateur. L'instruction qui suit modifie le texte correspondant au contrôle *TextBox* de nom *Text1*.

Exemple

```
Text1.Text = "Entrez votre texte ici"
```

La figure 9 montre des exemples des contrôles *TextBox* et *Label*.

3.2.4 RadioButton

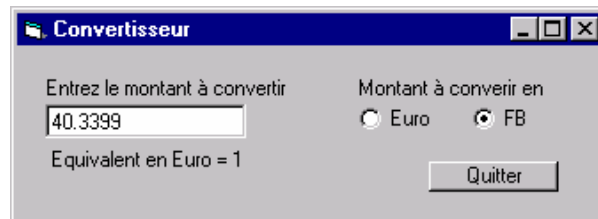
Le contrôle *RadioButton* combine deux fonctionnalités. Il permet de sélectionner une option présentée par un texte statique (un contrôle *Label*). Le contrôle *RadioButton*, utilisé sur un formulaire en au moins deux instances, permet de faire une seule sélection parmi les différents choix proposés (parmi les différents *RadioButton* affichés). La propriété du contrôle qui stock l'état de celui-ci est la propriété *Enabled*, de type *Boolean*. La valeur *True* veut dire que l'option est choisie. Notez bien que VB se charge de mettre à jour la propriété *Enabled* une fois qu'une sélection est faite (mettre *True* à la propriété *Enabled* du bouton sélectionné et *False* aux autres).

Exemple

```
OptionButton1.Enabled = True
OptionButton2.Enabled = False
```

Pratique. Dans l'exemple de la figure 14, l'utilisateur peut soit convertir du FB vers l'euro ou inversement. Les contrôles ont été déclarés comme suit :

Contrôle	Name	Caption
TextBox	Montant	
Label	Label1	Equivalent en Euro = 1
Label	Label2	Entrez le montant à convertir
Label	Label3	Montant à convertir en
OptionButton	OptionEuro	Euro
OptionButton	OptionFB	FB
CommandButton	Quitter	Quitter

Figure 14 : Exemple d'utilisation du contrôle *RadioButton*

```

| Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
|     entree.Text = 40.3399
|     sortie.Text = "equivalent en euro = 1"
| End Sub
|
| Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
|     MsgBox("je termine et je sors")
| End Sub
| End Sub
|
| Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
|     Dim Val_euro As Single
|     Val_euro = entree.Text / 40.3399
|     sortie.Text = "L'equivalent en euro est " & Val_euro
| End Sub
|
| Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
|     Dim Val_franc As Single
|     Val_franc = entree.Text * 40.3399
|     sortie.Text = "L'equivalent en FB est " & Val_franc
| End Sub
| End Class

```

Figure 15 : Code VB de l'exemple de la figure 14 – 1 ère possibilité

```

Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim Montanten As Char

    Private Sub Montant_Change()
        Dim valeuro, valeFB, value As Double
        value = Double.Parse(entree.Text)
        If entree.Text <> " " Then
            Select Case Montanten
                Case "E"
                    valeuro = value / 40.3399
                    sortie.Text = "Equivalent en Euro: " & valeuro

                Case "F"
                    valeFB = value * 40.3399
                    sortie.Text = "Equivalent en FB: " & valeFB
            End Select
        End If
    End Sub
End Class

```

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    MsgBox("Quitter?")
End
End Sub

Private Sub euro_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles euro.CheckedChanged
    Montanten = "E"
    Montant_Change()
End Sub

Private Sub FB_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FB.CheckedChanged
    Montanten = "F"
    Montant_Change()
End Sub

End Class

```

Figure 16 : Code VB de l'exemple de la figure 14 – 2 ème possibilité

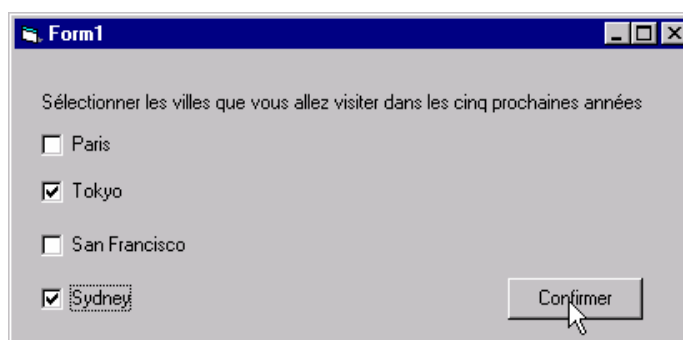
Dans l'exemple ci-dessus, la conversion se fait de manière *dynamique* chaque fois que le montant change (*Montant_Change*). La procédure événementielle *Montant_Change* est aussi appelée comme simple procédure dans les deux autres procédures événementielles : *OptionEuro_Click* et *OptionFB_Click*

Nom_objet_Change()

La procédure événementielle de nom *Nom_objet_Change()* s'exécute quand l'objet portant le nom *Nom_objet* change.

3.2.5 CheckButton

Ce contrôle ressemble de très près au contrôle *RadioButton*. Il combine aussi deux fonctionnalités. Il permet de *sélectionner* une option présentée par un *texte statique* (un contrôle *Label*). La différence majeure réside dans le fait que l'utilisateur peut faire de multiples sélections parmi les différents contrôles de même type. La propriété du contrôle qui stocke l'état de celui-ci est la propriété *Checked* qui prend l'une des deux valeur suivantes : *false* = choix non sélectionné, *true* = choix sélectionné.

Figure 17 : Exemple d'utilisation du contrôle *CheckButton*

Dans l'exemple de la figure 17, l'utilisateur peut choisir jusqu'à 4 villes.

Notez que c'est à l'utilisateur de gérer la relation entre les différents choix en programmant la *procédure événementielle Click()*.

Exemple

```
Private Sub Check1_CheckedChanged()
    If Check1.Checked = true Then
        Check3.Enabled = false
    Else
        Check3.Checked = true
    End If
End Sub
```

\ on clique sur le 1^{er} choix
 \ si le 1^{er} choix vient d'être sélectionné
 \ le 3^{ème} devient indisponible
 \ sinon le 3^{ème} devient disponible

3.2.7 GroupBox

Un *GroupBox* est une fenêtre. C'est un contrôle qui peut être placé sur un formulaire pour créer un groupe de contrôles. Tout contrôle placé sur le *GroupBox* (lui-même placé sur un formulaire) appartiendra à ce groupe.

On a vu dans le cas du contrôle *RadioButton* qu'une seule option peut être choisie. Cependant, si on veut présenter à l'utilisateur deux groupes de choix dans lesquels il peut sélectionner deux choix non exclusifs, un dans le premier groupe et un dans le second, ceci n'est pas possible. Pour résoudre ce problème, il suffit de placer l'un des groupes d'options dans un *GroupBox*, l'autre appartiendra au formulaire.

Notez que les *GroupBox* peuvent être utilisés pour créer un groupe de contrôles de différents types, *TextBox*, *RadioButton*, *Label*, etc.

La propriété *Text* permet de donner un titre au *GroupBox*.

Pratique.

Dans l'exemple de la figure 18, deux *GroupBox* sont utilisés : *Destination* et *Moyen de transport*. Chaque *GroupBox* intègre un groupe d'options. Ainsi, l'utilisateur peut choisir une seule ville et un seul moyen de transport.

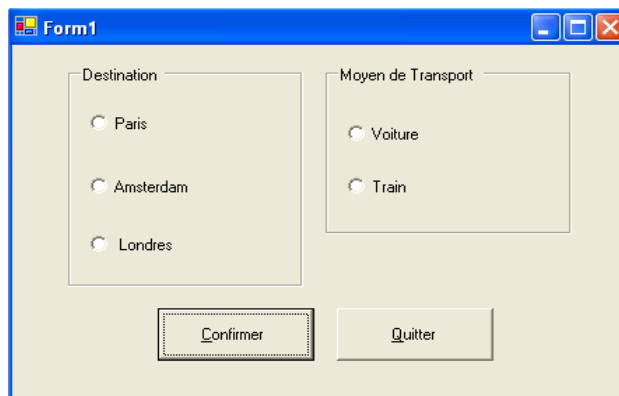


Figure 18 : Exemple d'utilisation du contrôle *GroupBox*

Le code correspondant à ce programme est présenté dans la figure 19. Les contrôles ont été déclarés comme suit :

Contrôle	Name	Text
Label	Label1	Choisissez les villes que vous allez visiter
CheckBox	C_Paris	Paris
CheckBox	C_Tokyo	Tokyo
CheckBox	C_SanFrancisco	San Francisco
CheckBox	C_Sydney	Sydney
CommandButton	Confirmer	Confirmer
CommandButton	Quitter	Quitter

```

Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim villeChoisie As String
    Dim MoyenChoisi As String

    Windows Form Designer generated code

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        If R_Paris.Checked = True Then
            villeChoisie = "Paris "
        End If
        If R_Amsterdam.Checked = True Then
            villeChoisie = "Amsterdam "
        End If
        If R_Londres.Checked = True Then
            villeChoisie = "Londres "
        End If
        If R_Voiture.Checked = True Then
            MoyenChoisi = "Voiture "
        End If
        If R_Train.Checked = True Then
            MoyenChoisi = "Train "
        End If
        MsgBox("La destination est " & villeChoisie & "avec comme moyen de transport le " & MoyenChoisi)
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        End
    End Sub
End Class

```

Figure 19 : Code VB relatif à l'exemple de la figure 18

3.2.8 Exercices

- a. Programmez l'application qui permet de réaliser l'application présentée à figure 14. Exemple d'utilisation du contrôle *RadioButton*.
- b. Sachant qu'1 US\$ coûte 38.5168 FB et qu'1 FF coûte 6.1498 FB, modifiez le code de l'exemple pour qu'on puisse aussi avoir la conversion en US\$ et en FF. Votre application doit avoir l'allure de la figure 20.

Figure 20 : Exemple de formulaire de conversion de devises

- c. Programmez l'application qui permet de réaliser la figure 17 (Exemple d'utilisation du contrôle *CheckBox*). Le bouton *Confirmer* doit ouvrir (avec la procédure *Show*) un autre formulaire contenant quatre contrôles *Label* qui affichent *uniquement* les choix faits par l'utilisateur.
- d. Programmez l'application qui permet de réaliser la figure 18 (Exemple d'utilisation du contrôle *GroupBox*). Utilisez deux vecteurs de contrôles pour les différents *RadioButton*. Le bouton *Confirmer* doit ouvrir (avec la procédure *Show*) un autre formulaire résumant les choix faits par l'utilisateur.
- e. Vous êtes à présent capable de répondre aux questions suivantes
1. Expliquez le concept d'objet
 2. Expliquez le concept de propriété d'un objet
 3. Expliquez le concept de méthode d'un objet
 4. Comment peut-on accéder à une propriété d'un objet ?
 5. Comment fait-on appel à une méthode d'un objet ?
 6. Donnez des exemples d'objets en VB.
 7. Donnez des exemples de propriétés en VB.
 8. Donnez des exemples de méthodes en VB.
 9. Comment trouver la liste des classes d'objets VB ?
 10. Quel est l'intérêt de la propriété *Name*?
 11. Pourquoi à l'intérieur d'un formulaire la propriété *Name* d'un objet doit être unique ?
 12. Définir le rôle du contrôle *Label* et donnez des exemples de propriétés
 13. Définir le rôle du contrôle *TextBox* et donnez des exemples de propriétés
 14. Définir le rôle du contrôle *RadioButton* et donnez des exemples de propriétés
 15. Définir le rôle du contrôle *CheckBox* et donnez des exemples de propriétés
 16. Quelle est la différence entre les contrôles *CheckBox* et *RadioButton*?
 17. Définir le rôle du contrôle *GroupBox* et donnez des exemples de propriétés

3.2.9 ListBox

Un *ListBox* est un contrôle qui permet de proposer une liste de valeurs parmi lesquelles l'utilisateur ne peut en choisir qu'une seule. La dite liste est stockée dans la propriété *Items*. La figure 21 montre un exemple d'utilisation du contrôle *ListBot*.



Figure 21 : Exemple utilisant le contrôle *ListBox*

3.2.10 ComboBox

Le contrôle *ComboBox* combine les fonctionnalités des contrôles *TextBox* et *ListBox*. La propriété *Text* stocke l'élément à chercher, à sélectionner ou à ajouter et la propriété *Items* stocke la liste des valeurs possibles, comme dans le cas du contrôle *ListBox*. La figure 22 montre un exemple d'utilisation du contrôle *ComboBox*.

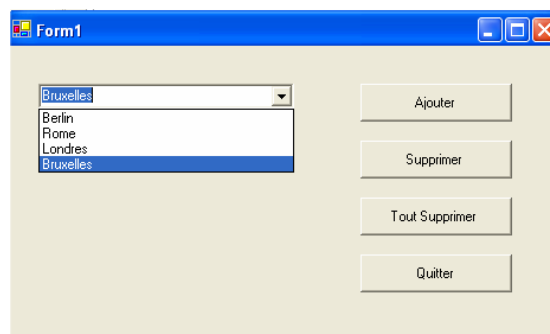


Figure 22 : Exemple utilisant le contrôle *ComboBox*

3.2.11 La propriété Items

La propriété *Items* se trouve dans plusieurs contrôles (*ListBox*, *ComboBox*, ...). Elle peut être remplie lors de la conception dans la fenêtre *Propriétés* comme monter dans la figure 23⁴. Cette liste peut aussi être mise à jour (ajout, suppression) de manière dynamique durant l'exécution du programme.

⁴ Pour ajouter une ligne (un élément) dans la liste, il faut cliquer sur les 3 petits points

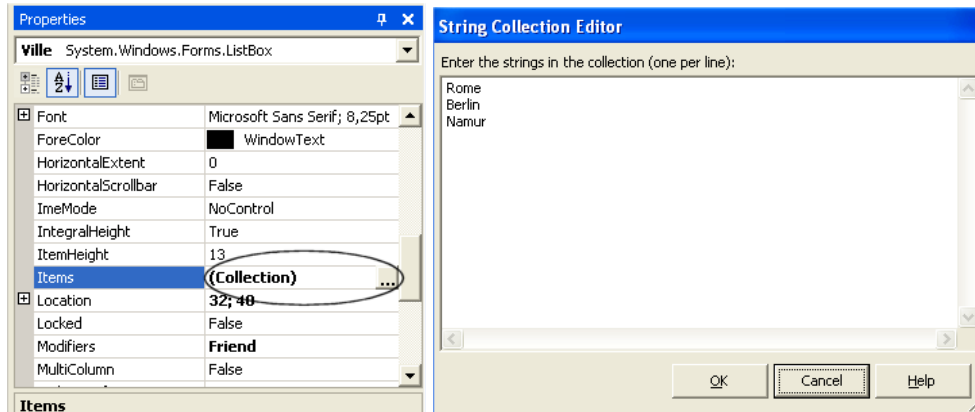


Figure 23: La propriété *Items* d'un contrôle *ComboBox* ou d'un contrôle *ListBox*.

Quelques propriétés et méthodes pour gérer les **ListBox** ou **ComboBox**

Syntaxe

<code>Nom_List_Box.Items.Add(valeur_i)</code>	: ajoute la valeur <i>valeur_i</i> dans la propriété <i>List</i>
<code>Nom_List_Box.Items.Remove(valeur_i)</code>	: enlève l'élément <i>valeur_i</i> de <i>List</i>
<code>Nom_List_Box.Items.RemoveAt(i)</code>	: enlève l'élément d'indice <i>i</i> de <i>List</i>
<code>Nom_List_Box.Items.Clear</code>	: enlève tous les éléments de <i>List</i>
<code>Nom_List_Box.Items.Count</code>	: donne le nombre d'éléments dans <i>List</i>
<code>Nom_List_Box.items(i)</code>	: retourne l'item d'indice <i>i</i> de la <i>List</i>
	<i>ListIndex</i> 0 = 1 ^{er} élément, 1 = 2 ^{er} élément, ...
<code>Nom_List_Box.SelectedItem</code>	: retourne l'élément sélectionné
<code>Nom_List_Box.Sorted</code>	: = <i>True</i> , maintient la liste triée par ordre alphabétique croissant

Exemples

<code>ListBox1.Items.Add ("Paris")</code>	'Ajoute <i>Paris</i> à la liste de <i>ListBox</i>
<code>List1. Items.Add(text1.text)</code>	'Ajoute <i>Text1.Text</i> à la liste de <i>ListBox</i>
<code>Text1.Text = List1.SelectedItem</code>	'affecte l'élément sélectionné à <i>Text1.Text</i>
<code>List1.Items.Remove(List1.SelectedItem)</code>	'enlève l'élément sélectionné de <i>list</i>

La procédure événementielle suivante supprime la ville introduite par l'utilisateur.

```
Private Sub Supprimer_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Supprimer.Click
```

```
    Dim Ville As String
    Dim i As Short
    Ville = InputBox("Enter la ville à supprimer : ")
    For i = 0 To (villes.Items.Count - 1)
        If villes.Items(i) = Ville Then
            villes.Items.RemoveAt(i)
            Exit For
        End If
    Next
```

```
End Sub
```

Pratique. L'exemple de la figure 24 utilise un *ComboList* pour stocker une liste de villes. L'utilisateur peut manipuler dynamiquement la dite liste. Le code correspondant à ce programme est présenté dans la figure 25.

La fonction *Existe(Ville)* vérifie si *Ville* se trouve dans la liste des villes. Une ville ne peut être ajoutée que si elle n'existe pas. Une ville ne peut être supprimée qu'après confirmation de l'utilisateur, c'est-à-dire, celui-ci a appuyé sur OK. La fonction *MsgBox*, utilisée avec l'option *OkCancel*, affiche les boutons *Ok* et *Cancel*. Elle retourne 1 si on appuie sur *OK* et 2 si on appuie sur *Cancel*.

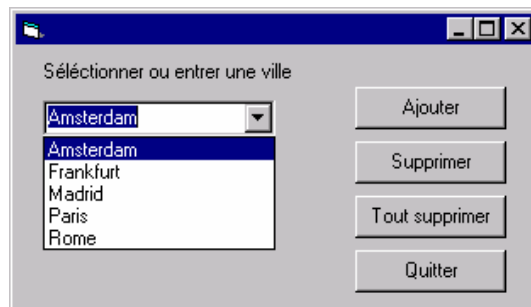


Figure 24: Exemple manipulant la propriété *List* du contrôle *ComboList*

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Function existe(ByVal ville As String) As Boolean
        Dim var As Boolean
        Dim i As Short
        var = False
        For i = 0 To (villes.Items.Count - 1)
            If villes.Items(i) = ville Then
                var = True
            End If
        Next
        existe = var
    End Function

    Private Sub Ajouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Ajouter.Click
        If Not existe(villes.Text) Then
            villes.Items.Add(villes.Text)
            MsgBox("ajout de " & villes.Text & " réussi")
        Else
            MsgBox(villes.Text & " est déjà dans la liste")
        End If
    End Sub

    Private Sub Quitter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Quitter.Click
        End
    End Sub

    Private Sub Supprimer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Supprimer.Click
        If MsgBox("Etes-vous bien sur de vouloir supprimer " & villes.Text & " ?", MessageBoxButtons.OKCancel) = 1 Then
            villes.Items.Remove(villes.SelectedItem)
        End If
    End Sub

    Private Sub SupprimerTout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SupprimerTout.Click
        If MsgBox("Etes-vous bien sur de vouloir tout supprimer ?", MessageBoxButtons.OKCancel) = 1 Then
            villes.Items.Clear()
        End If
    End Sub
End Class
```

Figure 25 : Code VB relatif à l'exemple de la figure 24

3.2.12 Exercices

- a. Programmez le programme qui permet de réaliser la figure 24: Exemple manipulant la propriété *List* du contrôle *ComboBox*
- b. Dans le même esprit que l'exercice précédent, créez un programme qui permet de convertir un montant en devise vers une autre devise. Votre application doit avoir l'allure de la figure 26. Utilisez trois contrôles *ListBox* : *DeviseB*, *DeviseC* et *TauxC*. *DeviseB* et *DeviseC* serviront, respectivement, à sélectionner la devise de base et la devise de conversion. *TauxC* servira à stocker les taux de change des différentes devises par rapport à une devise de référence (l'Euro). Notez bien que les trois contrôles ne doivent pas être triés, ainsi la devise d'indice *i* du contrôle *DeviseC* correspond au taux *i* du contrôle *TauxC*.

Figure 26 : Application à programmer

L'exemple ci-dessous montre comment convertir un montant (*Montant.Text*) d'une devise de base (*DeviseB*) vers une devise de conversion (*DeviseC*) en utilisant le taux de conversion d'une devise de référence (ici l'Euro). Par exemple, si on désire convertir un montant en CHF (*Devise de base*) en BEF (*Devise de conversion*), il faut d'abord trouver le montant équivalent en Euro puis multiplier ce montant par le taux du FB d'un Euro.

$$100 \text{ CHF} = ((100 / 1.6014) * 40.3399) \text{ BEF} = 2519.04 \text{ BEF}.$$

Exemple

```
Private Sub Convertir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Convertir.Click
    Dim MontantConverti, mont As Double
    mont = Double.Parse(montant.Text)
    MontantConverti = (mont / TauxC.Items(DeviseB.SelectedIndex)) * TauxC.Items(DeviseC.SelectedIndex)
    Label1.Text = "le montant en " & DeviseB.SelectedItem
    Label2.Text = "vaut " & MontantConverti & " " & DeviseC.SelectedItem
End Sub
```

Figure 27 : Code associé au bouton « convertir »

c. Modifiez votre programme pour que les différentes listes soient dynamiques, permettant à l'utilisateur d'ajouter ou de supprimer une devise.

3.2.13 Solution

```

Public Class Form1

Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    DeviseB.SelectedItem = DeviseB.Items(0)
    DeviseC.SelectedItem = DeviseC.Items(1)
    TauxC.SelectedItem = TauxC.Items(1)
    Label1.Text = "le montant en " & DeviseB.SelectedItem
    Label2.Text = "vaut 40.3399 " & DeviseC.SelectedItem
    montant.Text = 1
End Sub

Private Sub Convertir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Convertir.Click
    Dim MontantConverti, mont As Double
    mont = Double.Parse(montant.Text)
    MontantConverti = (mont / TauxC.Items(DeviseB.SelectedIndex)) *
        TauxC.Items(DeviseC.SelectedIndex)
    Label1.Text = "le montant en " & DeviseB.SelectedItem
    Label2.Text = "vaut " & MontantConverti & " " & DeviseC.SelectedItem
End Sub

Private Function existe(ByVal Devise As String) As Boolean
    Dim var As Boolean
    Dim i As Short
    var = False
    For i = 0 To (DeviseB.Items.Count - 1)
        If DeviseB.Items(i) = Devise Then
            var = True
            Exit For
        End If
    Next
    existe = var
End Function

Private Sub Ajouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Ajouter.Click
    Dim NouvelleDevise As String
    Dim NouveauTaux As Short
    NouvelleDevise = InputBox("Entrez la nouvelle devise")
    If NouvelleDevise <> "" And Not existe(NouvelleDevise) Then
        NouveauTaux = InputBox("Entrez le nouveai taux")
        If NouveauTaux > 0 Then
            TauxC.Items.Add(NouveauTaux)
            DeviseC.Items.Add(NouvelleDevise)
            DeviseB.Items.Add(NouvelleDevise)
        End If
    Else
        MsgBox("Entrez une autre devise, celle-ci est déjà répertoriée")
    End If
End Sub

Private Sub Supprimer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Supprimer.Click
    Dim Indice As Integer
    If MsgBox("Etes-vous bien sur de vouloir supprimer " & DeviseB.SelectedItem & "
        ?", MessageBoxButtons.OKCancel) = 1 Then
        Indice = DeviseB.SelectedIndex
        TauxC.Items.RemoveAt(Indice)
        DeviseB.Items.Remove(DeviseB.SelectedItem)
    End If
End Sub

```

```
        DeviseC.Items.RemoveAt(Indice)
    End If
End Sub

Private Sub Quitter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Quitter.Click
    End
End Sub

End Class
```

3.2.14 L'éditeur de menus

L'éditeur de menus permet de rattacher à un formulaire un menu comme celui utilisé par exemple dans les applications sous Windows. La figure 28 montre le menu de VB.

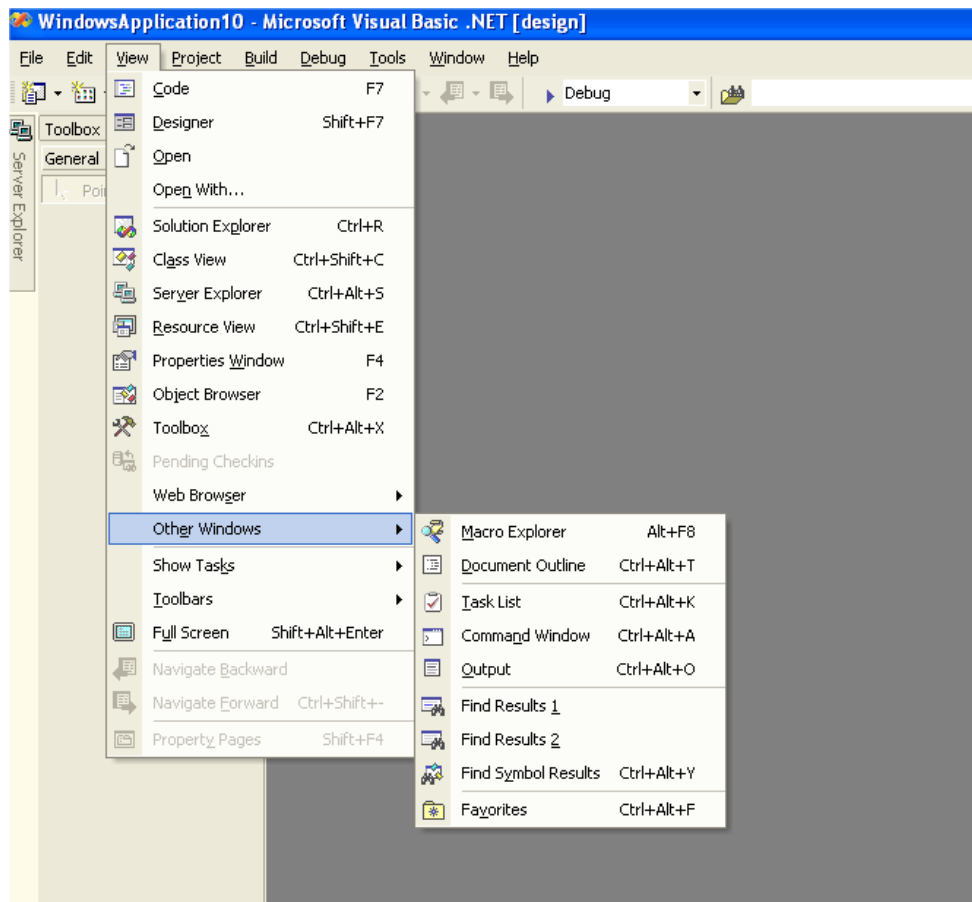


Figure 28: Exemple de menu

Un menu n'est rien d'autre qu'un ensemble de contrôles (commandes) qui sont accessibles par une représentation sous forme de menus. Une commande peut soit ouvrir un sous-menu soit lancer une procédure événementielle (comme celles vues précédemment).

Pour ouvrir l'éditeur de menu, il suffit de cliquer sur le bouton *Menu Editor*.



4. LES BASES DE DONNEES

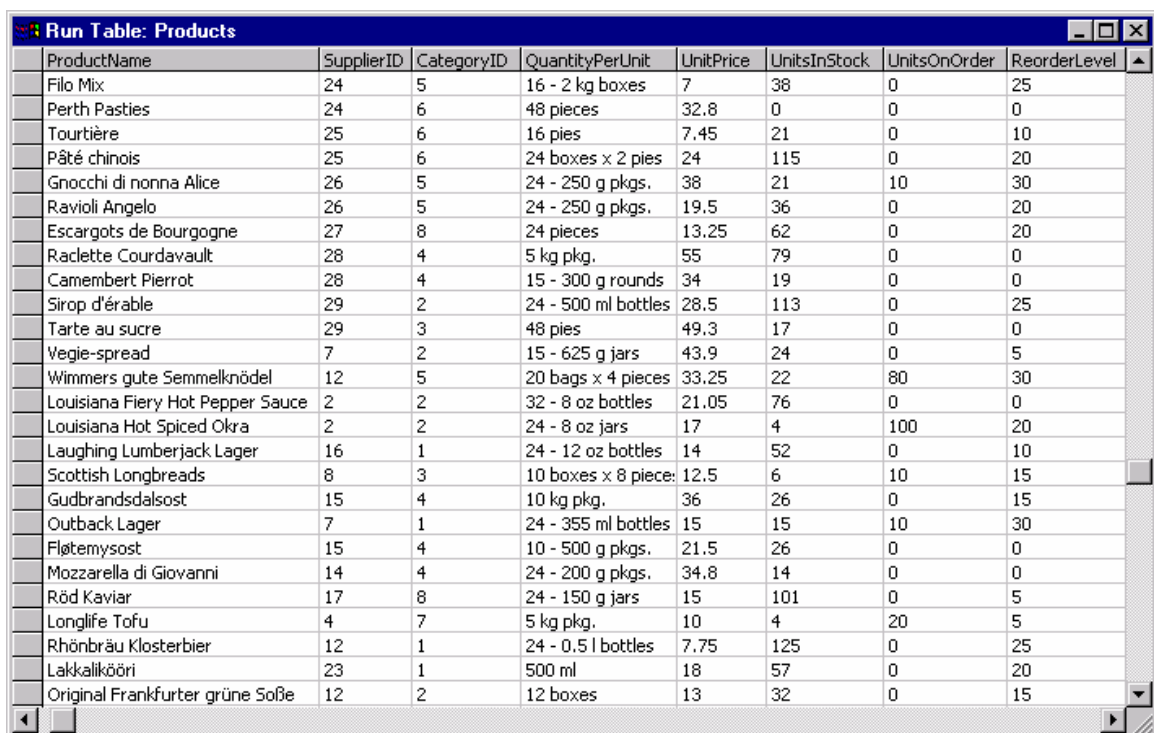
4.1 Notion de Table

Une table est *un fichier structuré* contenant des informations sur un ensemble de sujets de même *type*. On peut citer comme exemples de tables: *les clients, les produits, les commandes, les lignes de commandes, les étudiants, les transactions effectuées à des distributeurs automatiques de billets*.

Une table peut être vue comme un tableau à deux dimensions où chaque ligne représente les informations relatives *à un seul sujet*. Cette ligne, appelée aussi *enregistrement (record)*, est composée d'une suite structurée d'informations (*champs*) sur le dit sujet. Par exemple, pour le cas d'une table client, chaque ligne contiendra pour un seul client (*les champs*) les informations suivantes:

Nom, Adresse, Téléphone, Fax, Email, Chiffre d'affaires, personne de contact,

On dit qu'un *enregistrement* (une ligne d'une table) est composé d'un ensemble de champs ou d'attributs. Chaque champ peut être d'un type standard de données comme ceux de VB (*Integer, String, Date, ...*). La figure 30 montre un exemple d'une table de produits. Les noms des champs sont donnés à la première ligne.



ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel
Filo Mix	24	5	16 - 2 kg boxes	7	38	0	25
Perth Pasties	24	6	48 pieces	32.8	0	0	0
Tourtière	25	6	16 pies	7.45	21	0	10
Pâté chinois	25	6	24 boxes x 2 pies	24	115	0	20
Gnocchi di nonna Alice	26	5	24 - 250 g pkgs.	38	21	10	30
Ravioli Angelo	26	5	24 - 250 g pkgs.	19.5	36	0	20
Escargots de Bourgogne	27	8	24 pieces	13.25	62	0	20
Raclette Courdavault	28	4	5 kg pkg.	55	79	0	0
Camembert Pierrot	28	4	15 - 300 g rounds	34	19	0	0
Sirop d'érable	29	2	24 - 500 ml bottles	28.5	113	0	25
Tarte au sucre	29	3	48 pies	49.3	17	0	0
Vegie-spread	7	2	15 - 625 g jars	43.9	24	0	5
Wimmers gute Semmelknödel	12	5	20 bags x 4 pieces	33.25	22	80	30
Louisiana Fiery Hot Pepper Sauce	2	2	32 - 8 oz bottles	21.05	76	0	0
Louisiana Hot Spiced Okra	2	2	24 - 8 oz jars	17	4	100	20
Laughing Lumberjack Lager	16	1	24 - 12 oz bottles	14	52	0	10
Scottish Longbreads	8	3	10 boxes x 8 pieces	12.5	6	10	15
Gudbrandsdalsost	15	4	10 kg pkg.	36	26	0	15
Outback Lager	7	1	24 - 355 ml bottles	15	15	10	30
Fløtemysost	15	4	10 - 500 g pkgs.	21.5	26	0	0
Mozzarella di Giovanni	14	4	24 - 200 g pkgs.	34.8	14	0	0
Röd Kaviar	17	8	24 - 150 g jars	15	101	0	5
Longlife Tofu	4	7	5 kg pkg.	10	4	20	5
Rhönbräu Klosterbier	12	1	24 - 0.5 l bottles	7.75	125	0	25
Lakkalikööri	23	1	500 ml	18	57	0	20
Original Frankfurter grüne Soße	12	2	12 boxes	13	32	0	15

Figure 29 : Exemple d'une table de produits

4.2 Notion de Base de Données

Une base de données est un ensemble de tables conçues de manière cohérente. Une base de données doit contenir l'information nécessaire et suffisante sur les sujets représentés. On peut citer comme exemple de base de données celles conçues pour gérer : *un magasin commercial, le système de prêts de livre d'une bibliothèque, les inscriptions des étudiants d'une université, les réservations d'une compagnie aérienne, le système de production d'une entreprise manufacturière, etc.*

La figure 30 montre un exemple d'une base de données pour la gestion de livres dans une bibliothèque. Elle est composée de quatre tables, *Authors*, *Title Author*, *Titles* et *Publishers*. Cette figure montre uniquement les structures (les différents champs qui composent les enregistrements) des différentes tables.

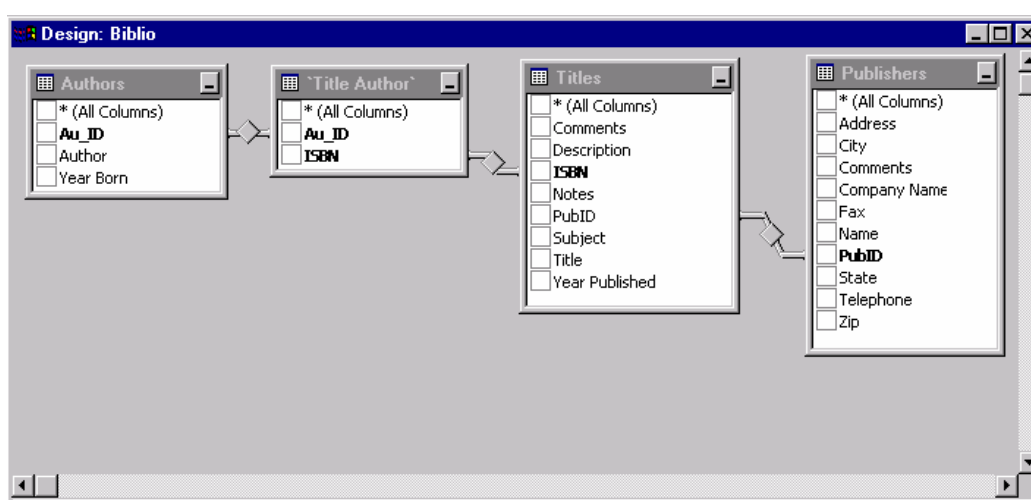


Figure 30: Exemple d'une base de données

Il existe dans le commerce plusieurs systèmes de gestion de base de données (SGBD), tels que *Oracle, Sybase, SQL server, DB2, Informix, Dbase, MS Access, etc.* Dans ce cours, on utilisera *Microsoft Access*. VB .NET permet de travailler avec une base de données de type *MS Access* sans disposer du logiciel *MS Access*⁵. Dans la suite de ce syllabus, on va voir comment **créer** et **interroger** une base de données.

4.3 Création d'une base de données Access

Pour créer une base de données, on utilise l'utilitaire *Microsoft Access*, voir figure 31. Pour lancer *Microsoft Access*, sélectionnez, dans vos programmes, Microsoft Office, Microsoft Access.

Pour créer une base de données, sélectionnez Blank Database. *MDB* est l'extension du fichier contenant une base de données de type *Access*. *Microsoft Access* vous invite à introduire le nom de la base de données à créer. Après confirmation de votre part, il ouvre une fenêtre : *Nom_Database : Database*.

⁵ Notez que VB permet de travailler avec d'autres types de base de données en utilisant des outils plus avancés, mais qui sortent du cadre de ce cours.

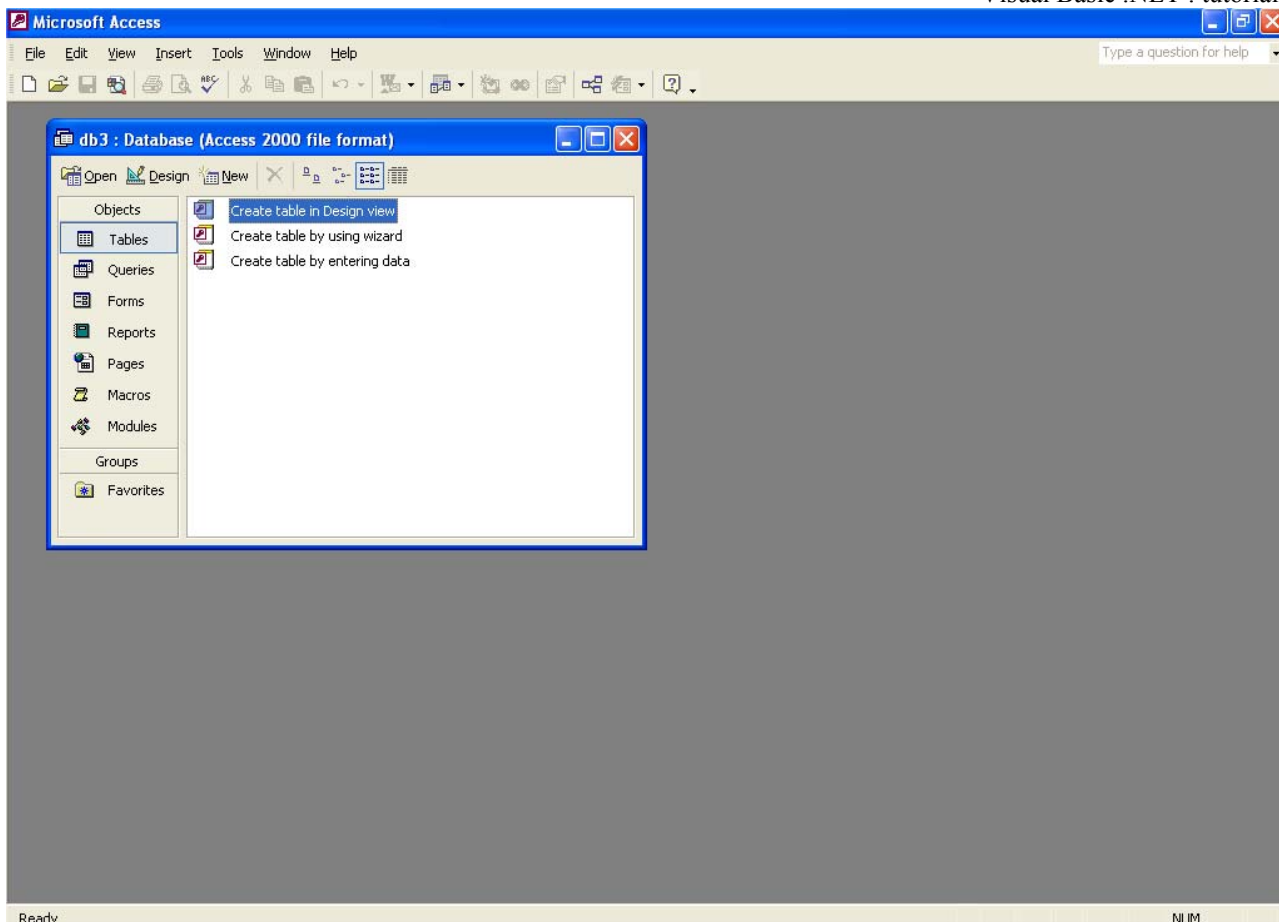


Figure 31 : Microsoft Access

Pour ajouter une table à la base de données cliquez sur *Create Table in Design View*. Microsoft Access ouvre la fenêtre de définition de la structure d'une table, figure 32. Pour ajouter des champs à la table, cliquez dans *Field Name* et insérez le type de donnée contenu dans ce champ (colonne *Data Type*). Si vous voulez indiquer que l'un des champs est la clé primaire, sélectionnez le champ (mettre le curseur sur la bonne ligne) et appuyez sur l'icône avec une clé (dans la barre des outils).

Lorsque tous les champs sont complétés, fermez la fenêtre et n'oubliez pas de donner un nom à votre table.

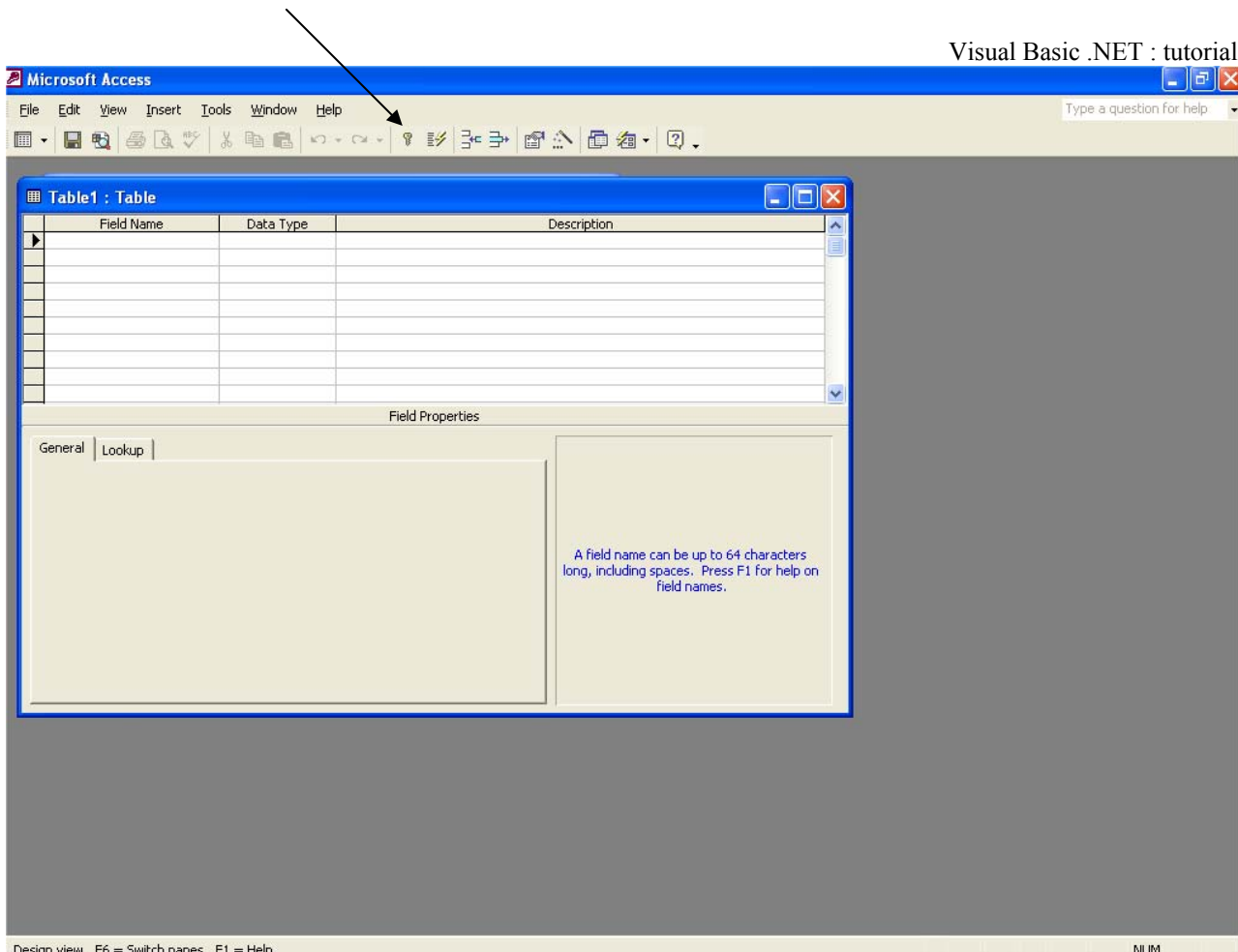


Figure 32 : Fenêtre de définition de la structure d'une table.

Pour ajouter une nouvelle table, il faut recommencer le même processus. Pour modifier la définition de la structure, cliquez sur le nom de la table avec le bouton droit de la souris et sélectionnez, *Design View*.

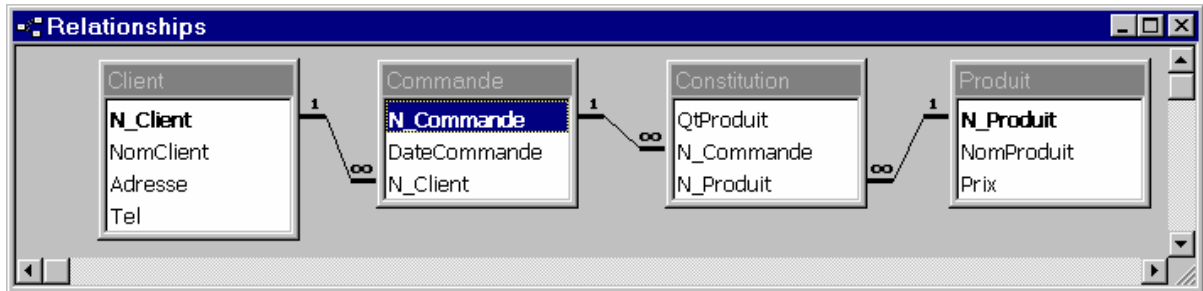
Lorsque vous aurez créé toutes vos tables, il s'agira d'indiquer les relations existant entre-elles. Pour cela, cliquez sur l'icône  (*Relationships*). Sélectionnez d'abord les tables à insérer dans le schéma et reliez-les les unes avec les autres.

Une fois que vous aurez créé toutes les tables et leurs structures, fermez votre base de données, (*File, Close*).

Annexes

Requêtes SQL

Soit le schéma relationnel suivant:



Requête Statique:

Quels sont les produits (*nomProduit*, *Prix*) achetés par le client numéro 3

```
SELECT      Produit.NomProduit, Produit.Prix
FROM        Client, Commande, Constitution, Produit
WHERE       (Client.N_Client = Commande.N_Client) And
            (Commande.N_Commande = Constitution.N_Commande) And
            (Constitution.N_Produit = Produit.N_Produit) And
            (Client.NomClient = 3);
```

➔ Ceci est une requête statique, c'est-à-dire que cette requête n'admet pas de paramètre qui changerait la valeur du résultat

Requête Dynamique:

Quels sont les produits (*nomProduit*, *Prix*) achetés par le client dont le numéro est donné en paramètre

```
SELECT      Produit.NomProduit, Produit.Prix
FROM        Client, Commande, Constitution, Produit
WHERE       (Client.N_Client = Commande.N_Client) And
            (Commande.N_Commande = Constitution.N_Commande) And
            (Constitution.N_Produit = Produit.N_Produit) And
            (Client.NomClient = ?);
```

➔ Ceci est une requête dynamique, car le numéro du client est donné en paramètre à la requête (le paramètre est représenté par le symbole ' ? ' dans la requête SQL)

Exemple de Requête dynamique :

- Supposons que l'application Visual Basic ci-dessous permette de rechercher les informations d'un client dont le nom est donné en paramètre et qui affichera le résultat de cette requête à l'écran.

	Adresse	N_Client	NomClient
*			

- La fenêtre ci-dessous montre comment construire cette requête avec le 'Query Builder' de Visual Basic .Net.

Column	Alias	Table	Output	Sort Type	Sort Order	Criteria
N_Client		Client	✓			
Adresse		Client	✓			
NomClient		Client	✓			= ?

```
SELECT N_Client, Adresse, NomClient
FROM Client
WHERE (NomClient = ?)
```

- Le code Visual Basic permettant de donner le paramètre à la requête SQL et d'afficher le résultat à l'écran.

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        OleDbDataAdapter1.SelectCommand.Parameters("NomClient").Value = nom.Text
        DataSet11.Clear()
        OleDbDataAdapter1.Fill(DataSet11)
    End Sub

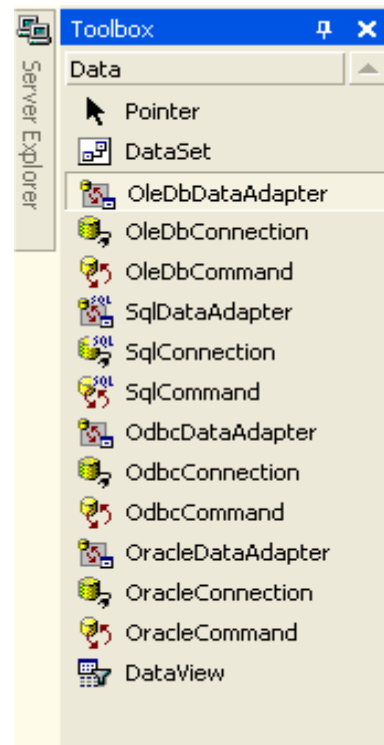
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        OleDbDataAdapter1.Update(DataSet11)
        MessageBox.Show("Database updated!")
    End Sub

End Class
```

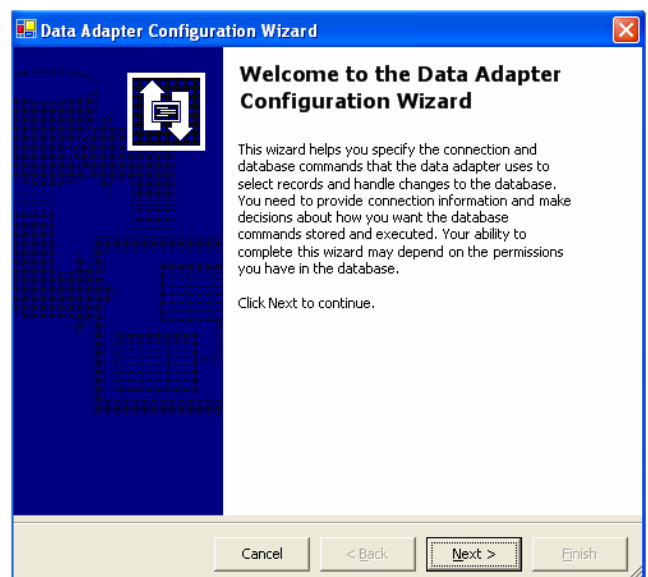
Travailler avec une BD dans VB .NET

1. Accéder aux informations enregistrées grâce à l'objet **OleDbDataAdapter**

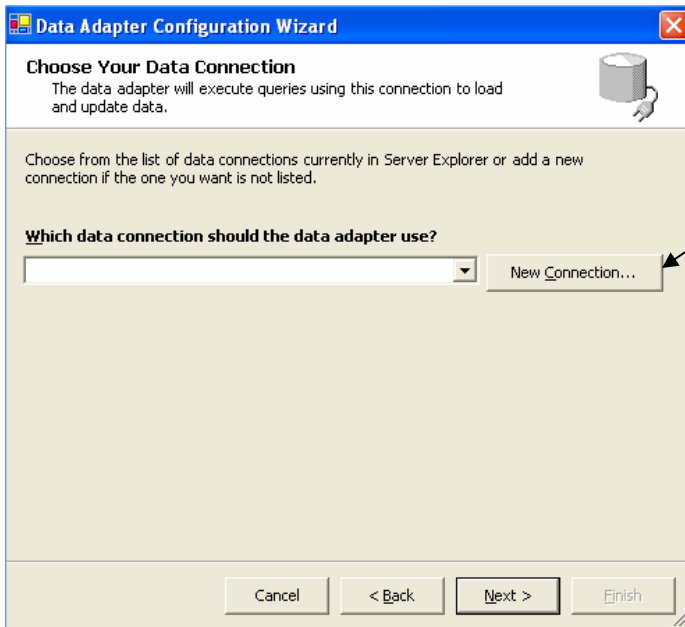
- Cliquer sur l'objet et le faire glisser sur le formulaire
- Suivre les instructions de l'assistant
Configuration d'adaptateur de données



- Une fenêtre de dialogue apparaît suite à cette action
- Cliquer sur le bouton « Next »

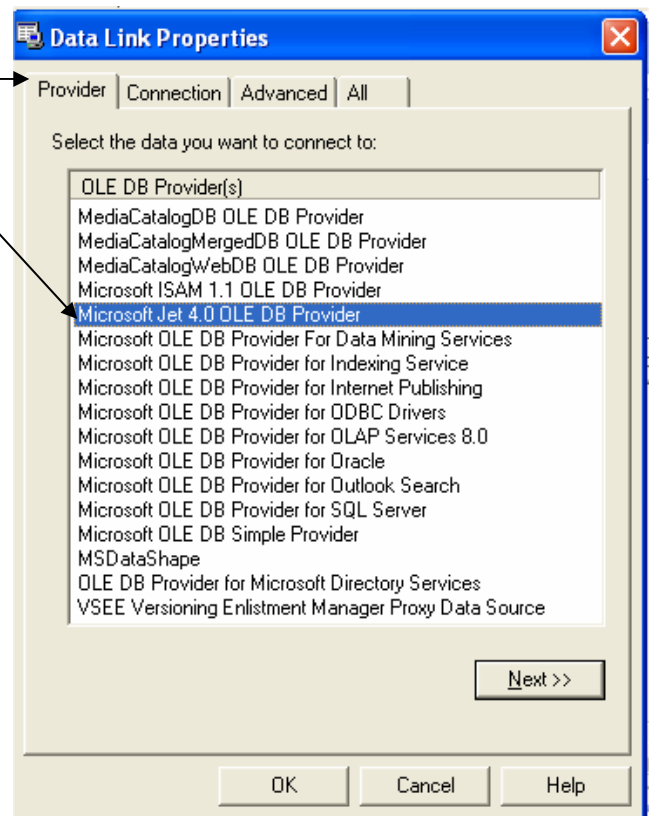


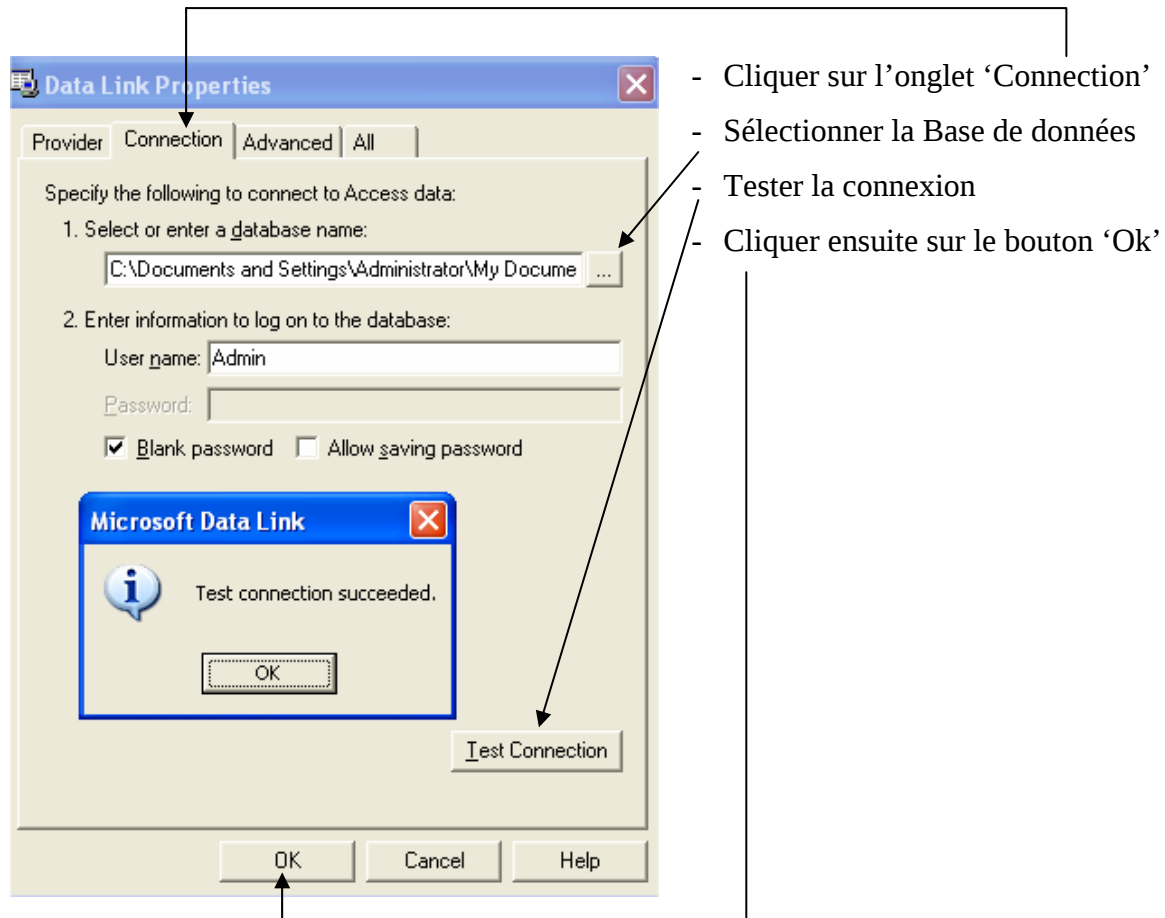
Base de données avec Visual Basic .Net et le modèle relationnel



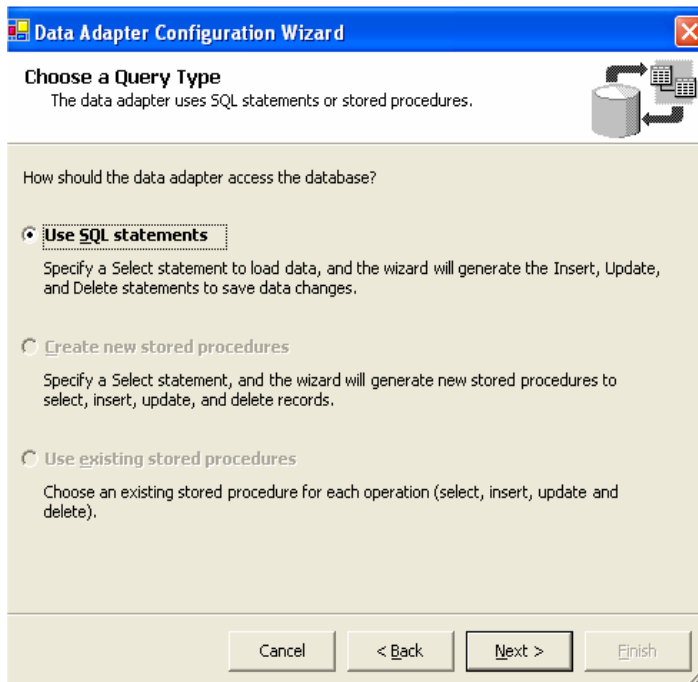
- Créer une nouvelle connexion (si la connexion n'existe pas encore) à Appuyer sur le bouton

- Cliquer sur l'onglet 'Provider'
- Cliquer sur Microsoft Jet 4.0 OLE DB Provider, le fournisseur de BD Microsoft Access

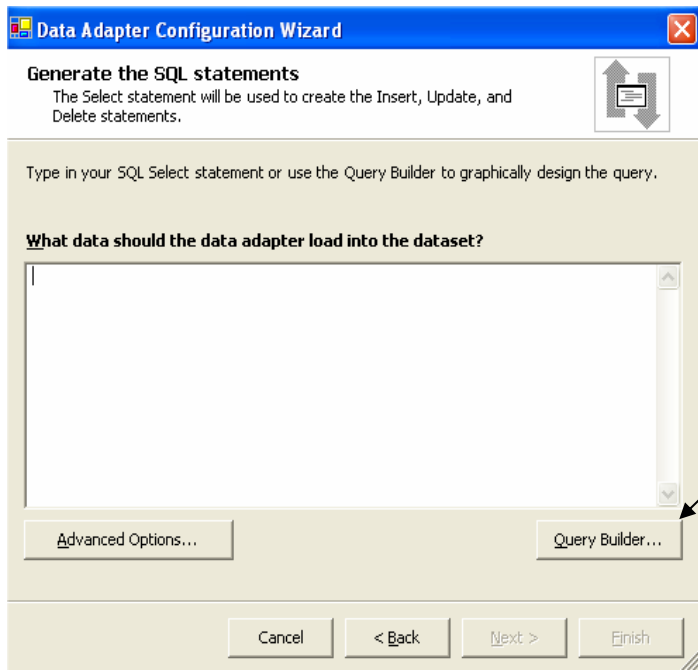




Base de données avec Visual Basic .Net et le modèle relationnel

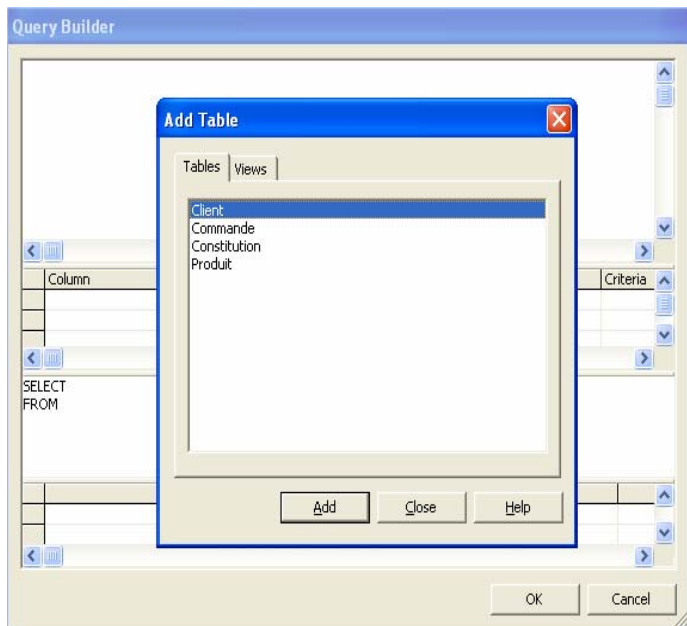


- Le type de requête sélectionné est 'Instructions SQL'. Cela spécifie la manière dont l'objet OleDbDataAdapter s'adresse à la BD
- Appuyez sur le bouton 'Next'



- Ecrire les requêtes SQL. Vous pouvez soit les écrire vous-mêmes, soit laisser le Générateur de requêtes les créer à votre place à Bouton « Query Builder ».

Base de données avec Visual Basic .Net et le modèle relationnel



Le query Builder :

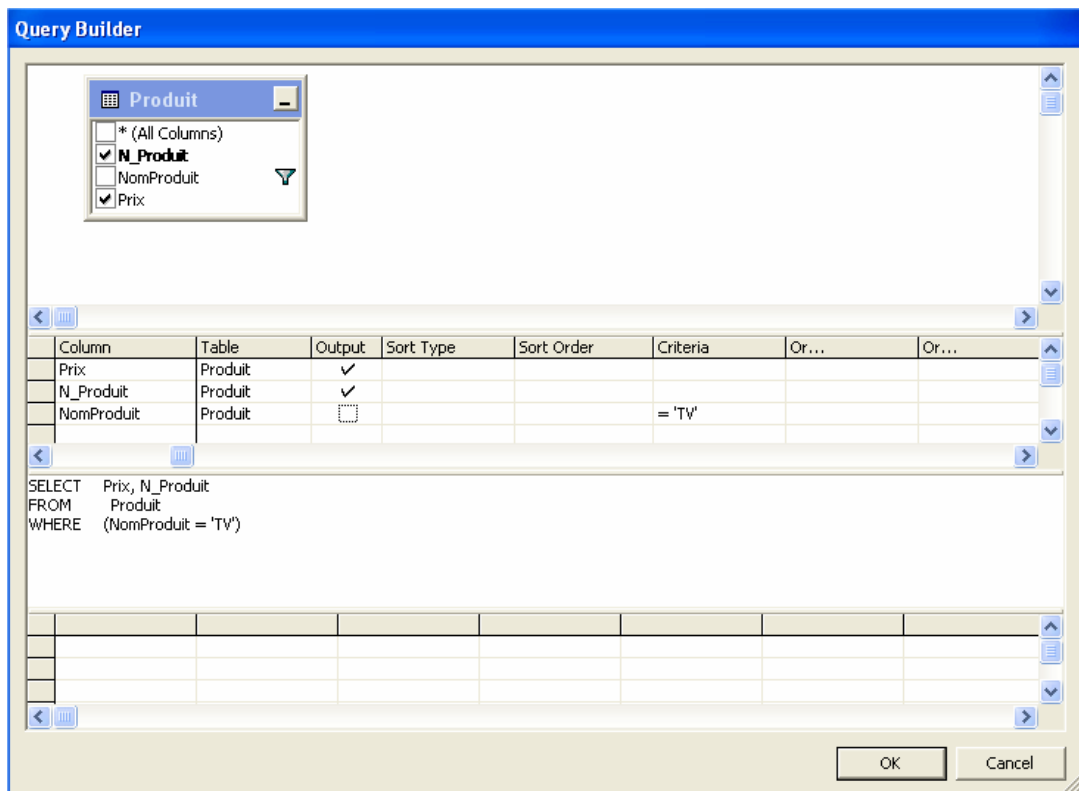
- Sélectionnez les tables sur lesquelles vous voulez travailler

- Ensuite construisez votre requête graphiquement.

Exemple : Nous avons spécifié la requête suivante : **Quel est le prix et numéro du produit 'TV'**

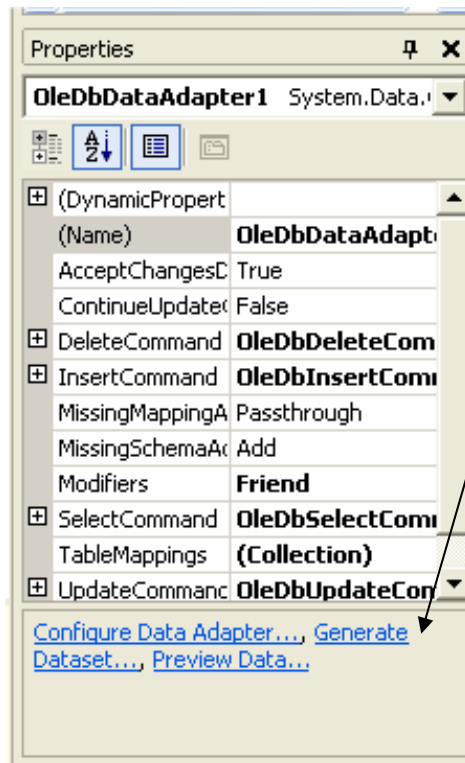
Cela se traduit, en Langage SQL, par:

```
SELECT Produit.Prix, Produit.N_Produit  
FROM      Produit  
WHERE  Produit.NomProduit =  "TV"
```

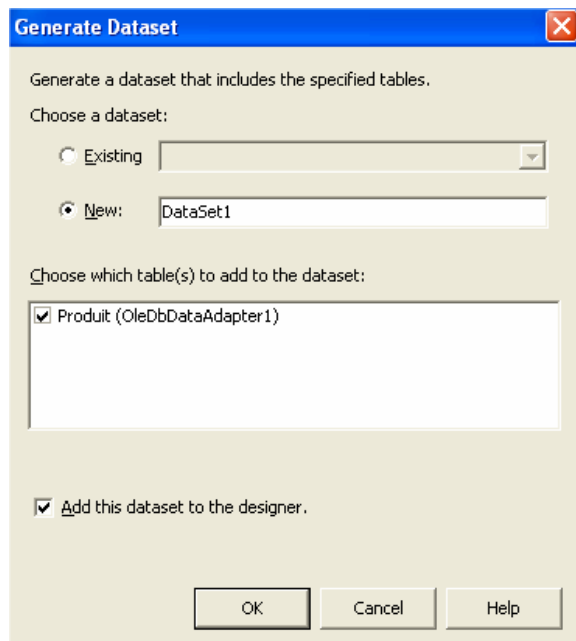


- Après cela quittez le 'Query Builder' en cliquant sur le bouton 'OK'
- Ensuite quittez la fenêtre de configuration du 'Data Adapter'

2. Créer un *groupe de données* ou *DataSet* (représente les données auxquelles on souhaite accéder).



- Cliquez Sur le lien 'generate' pour faire apparaître une fenêtre de dialogue



- Validez en appuyant sur le bouton 'Ok'

3. Faire appel à la méthode *Fill* de l'objet *OleDbDataAdapter* pour peupler l'objet *DataSet* avec les données adéquates

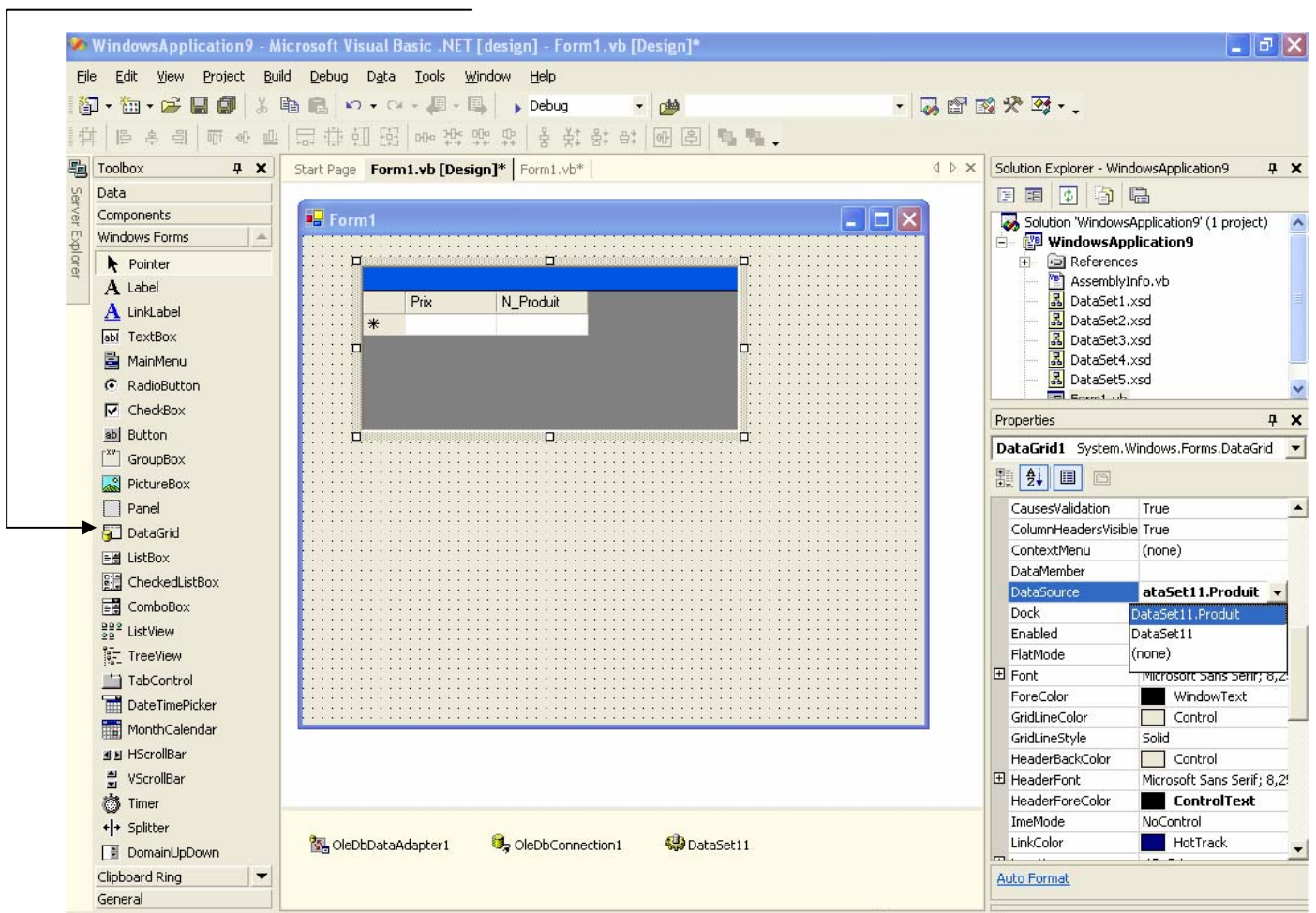
```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
        OleDbDataAdapter1.Fill(DataSet11)
    End Sub

End Class
```

4. Ajouter un contrôle DataGrid au Formulaire

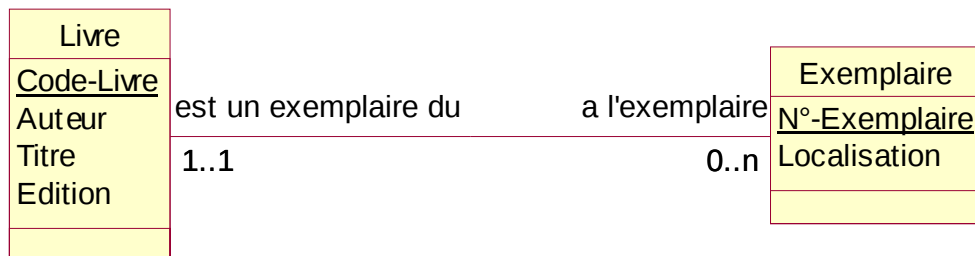


Passage d'un diagramme de classe à un schéma relationnel

Définition :

- *Clé primaire*: champs ou ensemble de champs qui identifie, de manière unique, chaque enregistrement stocké dans la table.
- *Clé étrangère*: champs ou ensemble de champs qui correspond à une clé primaire dans une autre table

Règle 1: présence de la cardinalité (?..1) d'un côté de l'association



- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champ de table
- L'identifiant de la classe qui est associée à la cardinalité (?..1) (ex: Livre) devient le clé étrangère de l'autre classe (ex: Exemplaire)

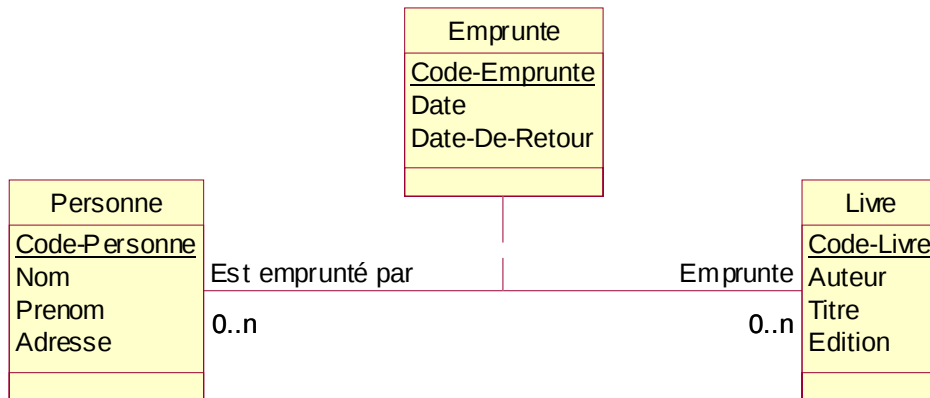


Contrainte d'intégrité référentielle:

CléEtrangère $\subseteq$ CléPrimaire

Ex: Exemplaire.Code-Livre $\subseteq$ Livre.Code-Livre

Règle 2: présence de (?..N) des deux côtés de l'association



- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champ de table
- L'association se transforme en une table. Cette table a comme champs l'identifiant de chacune des deux classes, plus d'éventuels autres attributs.

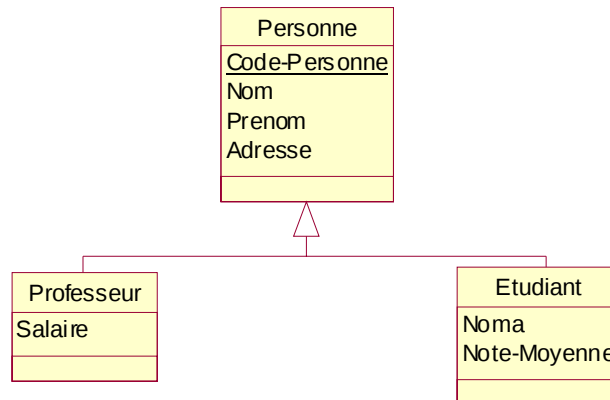


$\text{Emprunte.Code-Personne} \subseteq \text{Personne.Code-Personne}$

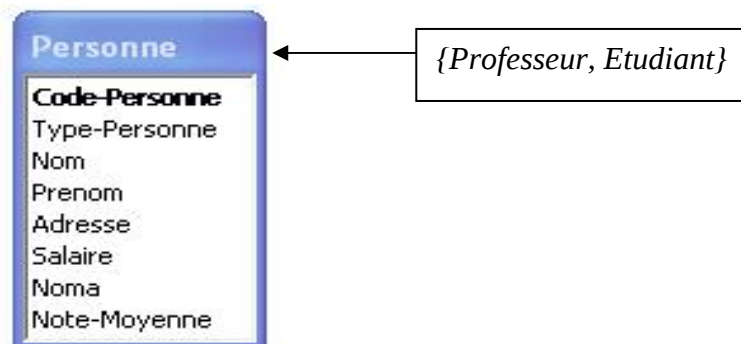
$\text{Emprunte.Code-Livre} \subseteq \text{Livre.Code-Livre}$

Règle 3: présence d'une généralisation

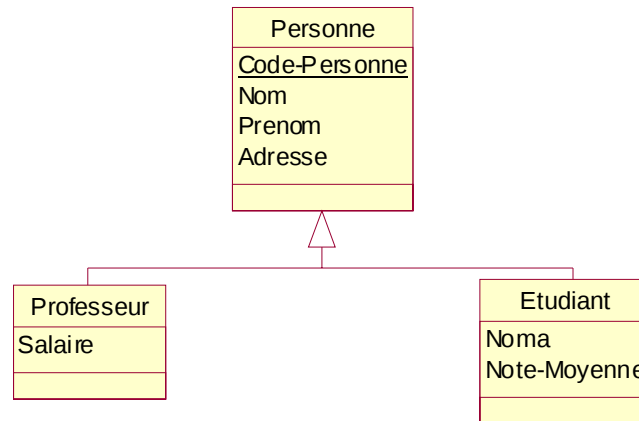
Méthode 1:



- Créer une table avec tous les attributs des classes
- Ajouter un attribut pour distinguer les types des objets



Méthode 2:



- Créer une table pour chaque sous type, chaque table se compose des attributs génériques et d'attributs spécifiques



Règle 4: présence d'une agrégation



- Créer une table pour les classes 'Voiture' et 'Roue', et lier les deux tables par une association 1 à N

