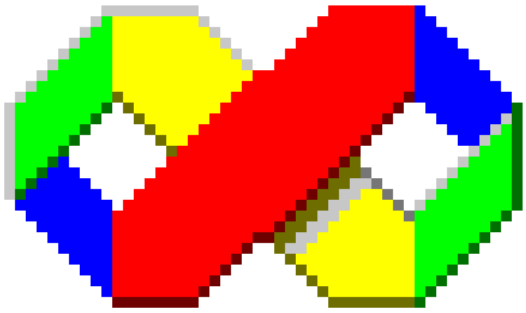
# Microsoft
# Visual Basic.Net



www.tri.on.ma

Olivier Zimmermann
Victor Formation
Août 2003
Fiche sur VBFrance : **Olixelle** http://www.vbfrance.com/auteurdetail.aspx?ID= 315759

# Sommaire

# 1. Présentation

Depuis son apparition, le langage Visual Basic ainsi que ses divers environnements de développement ont su s'imposer comme les standards en matière d'outils de réalisation d'applications Windows.

## 1.1 Historique

| Version | Nouveautés |
|---|---|
| 1.0 | Sortie en 91, Visual Basic innove en particulier grâce à son environnement de développement permettant de masquer les tâches fastidieuse |
| 3.0 | Evolution du langage, apparition de nouvelles fonction et de structures de contrôle (select case) |
| 4.0 | Sorti en 96, la version 4.0 marque une étape importante dans l'évolution du langage :<br>- Création d'application 32 bits<br>- Création et utilisation de DLL<br>- Apparition des fonctionnalités Objet (Classes) |
| 5.0 | Disparition des applications 16 bits et stabilité accrue du langage |
| 6.0 | Peu d'évolution sur le langage mais apparition de la technologie ADO (remplaçante de DAO et RDO) et des outils de connexion aux sources de données (DataEnvironment) |

## 1.2 Nouveautés

La version .Net (ou version 7.0) de Visual Basic et plus généralement de l'IDE Visual Studio marquent une étape importante dans l'évolution du langage. L'élément principal de l'infrastructure .NET est le CLR (Common Language Runtime), langage offrant un ensemble de classe permettant à l'utilisateur d'interagir avec le système. Ces classes peuvent être utilisée avec n'importe quel langage .Net (Vb, C++, C#) car elle font partie d'une norme commune : le CLS (Common Language Specification).

Une autre des révolutions du .Net réside dans le MSIL (Microsoft Intermediate Language) : les applications réalisées avec Vb .Net sont d'abord compilés en pseudo code, le MSIL, et c'est lors de l'exécution que le compilateur (JIT pour Just In Time) compile le code MSIL afin de le rendre exécutable par le processeur utilisé. L'avantage de ce système est double :

- Gain de ressource lors de l'exécution car seules celle devant être utilisées seront chargées et de ce fait, leur installation préalable n'est pas obligatoire
- Portabilité des applications sur différentes plateformes à la manière de la machine virtuelle Java.

## 1.3 Installation

A l'instar de la version 6.0, développer en VB .Net requiert une configuration machine beaucoup plus importante.

Ci dessous figurent les spécifications Microsoft :

| | Config. Minimum | Config Optimale |
|---|---|---|
| **Processeur** | P2 450 Mhz | P3 733 Mhz |
| **Mémoire Vive (Ram)** | 128 Mo | 256 Mo |
| **Espace Disque** | 3 Gb | 3Gb |
| **Carte Vidéo** | 800x600, 256 Couleurs | 1024x768, 65536 Couleurs |
| **Lecteur CD Rom** | Obligatoire | Obligatoire |
| **Système d'exploitation** | Windows 2000<br>Windows NT 4.0<br>Windows Me<br>Windows 98 | |

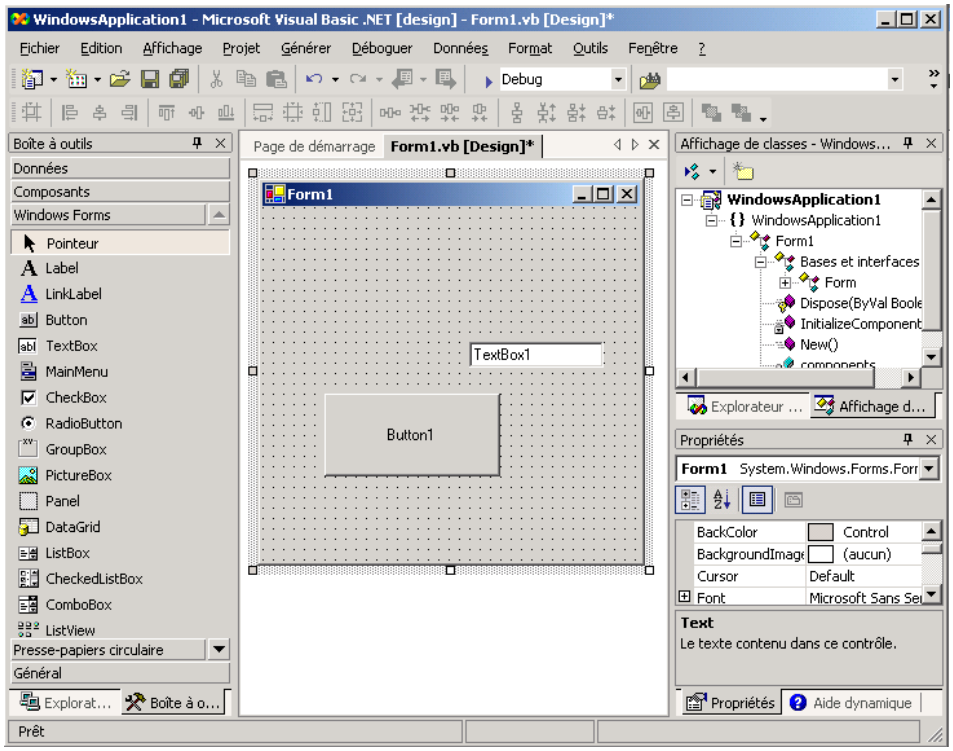
Ces spécifications concernent l'utilisation du FrameWork .Net ainsi que de l'IDE, pas l'exécution des programmes MSIL.

# 2. Environnement Visual Studio

## 2.1 L'interface

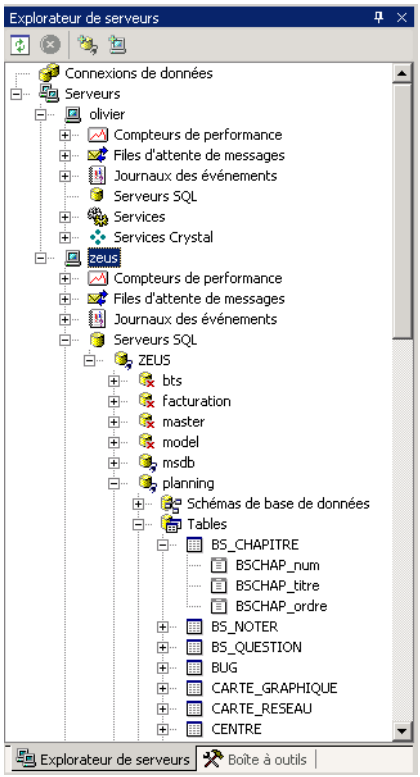
Les habitués de Visual Studio 6.0 ne seront pas déroutés : l'interface de Visual Basic .net reprend la plupart des palettes standards avec quelques outils en plus :

- la barre d'outils regroupe les différents contrôles par catégories
- La zone centrale permet tour à tour d'écrire le code et de définir les interfaces graphiques utilisateurs
- A droite, l'explorateur de solutions et la fenêtre de propriétés



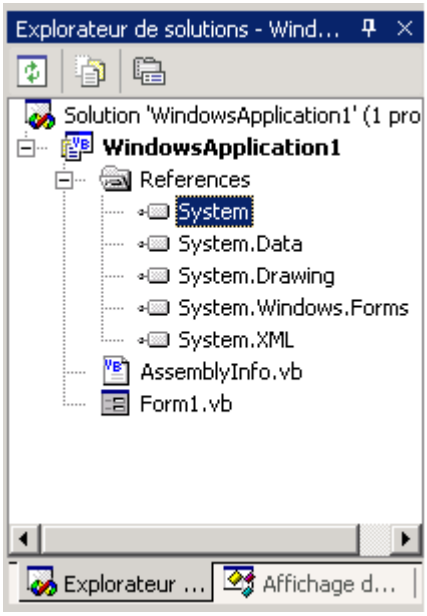

## 2.2 Fenêtres d'exploration

### 2.2.1 Explorateur de serveur




L'explorateur de serveur permet de recenser sous forme hiérarchique l'ensemble des objets d'un serveur (Serveurs SQL, journaux, Services …). Cette fenêtre est principalement utilisées afin d'accéder au schéma d'une base de données utilisée dans une application.

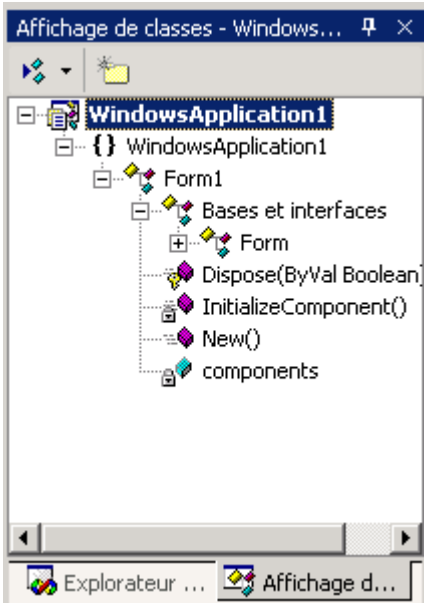
### 2.2.2 Explorateur de Solutions




L'explorateur de solutions (Ex explorateur de projet sous VB 6.0) référence l'ensemble des éléments du projets (Fichier de configuration pour l'assemblage, super classes hérités, Feuilles….)

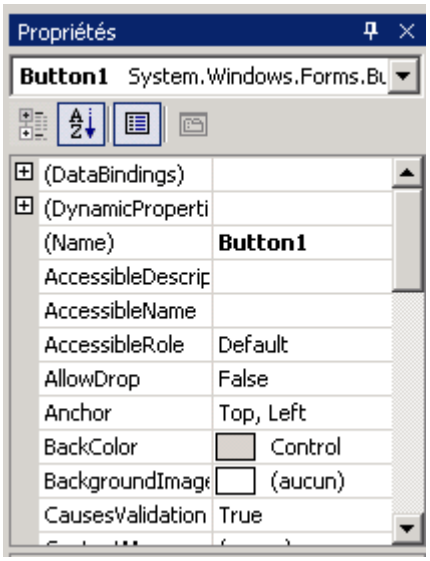
Une solution contient les fichiers suivants :
.sln : fichier de configuration de la solution
.vbproj : fichier projet, (ancien .vbp)
.vb : fichiers contenant du code (anciens .bas, .frm, .cls)
.resx : associé à une feuille, contient les ressources

### 2.2.3 L'affichage de classes




L'affichage de classes liste de manière hierarchique les différentes classes du projet ainsi que leurs méthodes, propriétés, évènements et autre relations d'héritage.

### 2.2.4 Fenêtre de propriétés




Déjà présente dans la version 6.0, cette fenêtre recense toutes les propriétés relatives à l'objet selectionné.

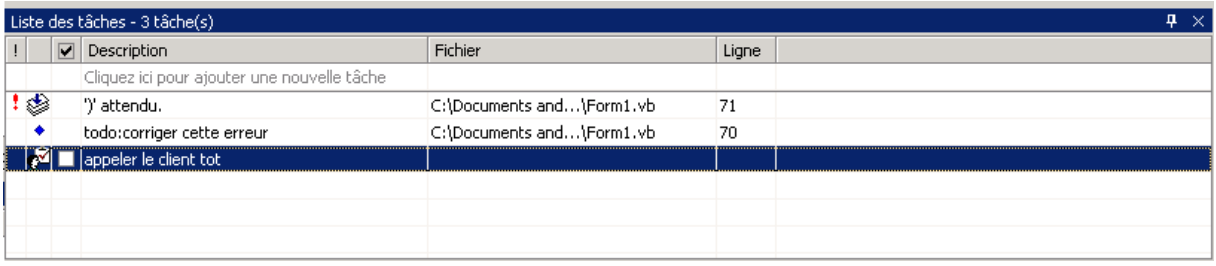
### 2.2.5 L'aide dynamique

L'aide dynamique propose à tous les moments de la conception des rubriques d'aide utiles en fonction de ce que vous faîtes. Par exemple, la rubrique « créer une collection de contrôles » sera affichée lorsque vous ajouterez un bouton radio à votre application.

### 2.2.6 La liste des tâches

La fenêtre liste des tâche permet de recenser l'ensemble des tâche à réaliser sur votre projet. Cette liste peut être remplie de plusieurs façons :

- Une tâche que vous aurez vous même définie (ex : appeler le client à 11h)
- Une tâche issue des commentaires de votre code : tous commentaires de votre code commencant par « todo: » sera automatiquement ajouté
- Lorsqu'une erreur de syntaxe est détectée par Visual Studio, elle est automatiquement ajoutée dans la liste
-



# 3. Gestion de projets

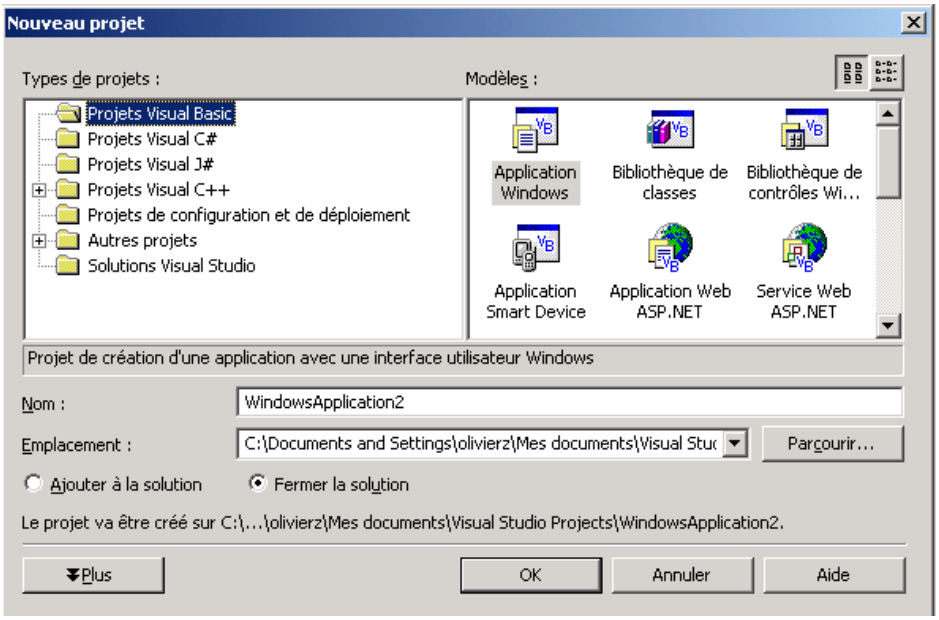
## 3.1 Les solutions

Une solution le plus haut conteneur logique d'éléments (projets, fichiers, feuilles, classes). Une solution peut contenir plusieurs projets.

## 3.2 Création d'un projet

Il est possible de créer un projet soit directement dans une solution ouverte soit dans une nouvelle solution.

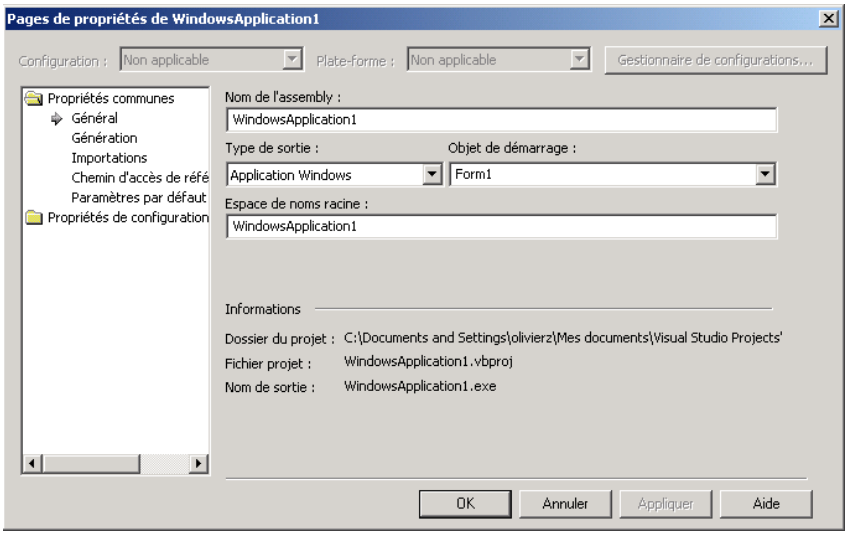
- Menu Fichier > Nouveau > Projet




| Principaux projets Visual Basic | |
|---|---|
| **Type** | **Description** |
| Application Windows | |
| Bibliothèque de classe | |
| Bibliothèque de contrôle Windows | |
| Application Smart Device | |
| Application Web ASP.Net | |
| Service Web ASP.Net | |
| Application console | |
| Service Windows | |

## 3.3 Configuration d'un projet

Pour chaque projet de votre solution, un ensemble de propriétés sont configurables. Pour accéder aux propriétés d'un projet, clic droit sur le projet dans l'explorateur de solution et Propriétés.




| Propriétés communes | |
|---|---|
| **Propriété** | **Description** |
| Nom de l'assembly | Nom du fichier généré après compilation (MSIL) |
| Type de sortie | Type d'application a générer |
| Objet de démarrage | Feuille ou procédure servant de point de départ au programme |
| Espace de nom racine | Permet de définir un préfixe pour accéder à l'ensemble des classes disponibles dans le projet |
| Icône de l'application | Fichier .ico servant d'icône au fichier de sortie |
| Option explicit | Interdit l'utilisation d'une variable non déclarée |
| Option Strict | Oblige l'écriture de explicite de la conversion de données |
| Option compare | Distinction de la casse en mode binaire (pas en mode texte) |
| Espaces de noms | Permet de définir les espaces de noms qui devront être automatiquement importés dans le projet (ex : permet d'écrire « form » à la place de « system.windows.forms.form » |
| Chemin d'accès de référence | Définit les dossiers dans lesquels se trouvent les références utilisées dans le projet |
| Présentation page | Mode de positionnement des contrôles : en mode Grid, le placement est libre, en mode Flow, le placement se fait dans l'ordre de création. |
| Schéma cible | Navigateur pour lequel le code HTML doit être compatible |
| Langage de script Client | Type de langage à utiliser |

| Propriétés de configuration | |
|---|---|
| **Propriété** | **Description** |
| Action de démarrage | Action à réaliser par l'environnement lors de la demande |

| | |
|---|---|
| | d'exécution |
| Argument de la ligne de commande | Permet de passer des paramètres lors de l'exécution du programme |
| Répertoire de travail | Répertoire actif pour l'application |
| Débogueurs | Débogueurs à activer lors de l'exécution |
| Optimisation | Définit un ensemble de contrôles que le compilateur ne fera pas lors de la compilation du programme |
| Chemin de sortie | Répertoire où sera généré l'exécutable ou la bibliothèque |
| Activer les avertissements de génération | Si coché, le compilateur ajoutera ses remarques à la liste des tâches |
| Considérer les avertissements du compilateur comme des erreurs | Lors de la compilation, les avertissements (Warning) fait par le compilateur stopperont la compilation (aucun fichier généré) |
| Constantes de compilation conditionnelles | Cette option permet de définir des constantes qui influeront sur les lignes compilées |

# 4. Bases du langage

## 4.1 Structure du code

Un exemple valant mieux qu'un long discours, nous allons commencer par décortiquer un programme de base avant de parcourir le langage en lui même.

### 4.1.1 Formulaire

Lors de la création d'un formulaire, le code suivant est automatiquement généré :

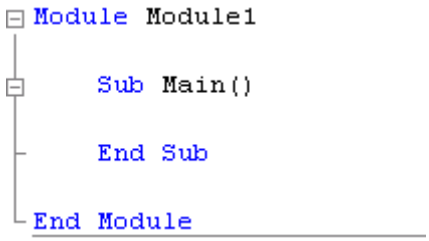
```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Code généré par le Concepteur Windows Form

End Class
```

- Le code est en fait constitué de la définition d'une nouvelle classe portant le nom du formulaire.
- L'instruction « inherits System.windows.forms.form » indique que la classe que nous créons doit hériter (récupérer) de tous les éléments inscrits dans la classe « form »
- Enfin, la région « Code généré par le concepteur Windows Form » contient l'appel aux méthodes de base de classe form ainsi que l'initialisation des contrôles.

Tout ajout d'éléments au formulaire (Variables, procédures, contrôles) seront ensuite perçus (selon leur visibilité) comme membres de la classe « form1 ».

### 4.1.2 Module

Lors de la création d'un nouveau module, le code suivant est automatiquement généré :




Tout le code contenu dans le module sera inséré entre « module » et « end module »

### 4.1.3 Les commentaires

Les commentaires permettent d'ajouter des annotations sur le code qui, une fois la compilation lancée, ne seront pas traitées par le compilateur et ainsi exclue du programme final.

Pour ajouter un commentaire, utiliser le caractère apostrophe « ' ». lorsqu'un commentaire est ajouté, tout le reste de la ligne sera un commentaire.

```
'ceci est un commentaire
Dim mavar as string 'ceci est un autre commentaire
```

Il n'est pas possible de créer des commentaires multilignes. Dans ce cas, vous êtes obligé de répéter le caractère apostrophe au début de chaque ligne

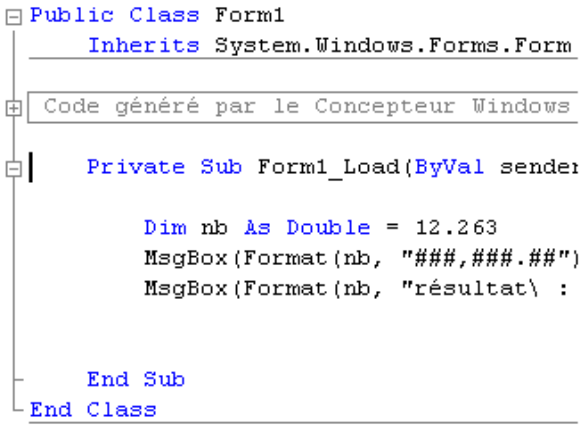
A noter que lorsqu'un commentaire commence par le mot clé todo:, il sera automatiquement ajouté dans la liste des tâches

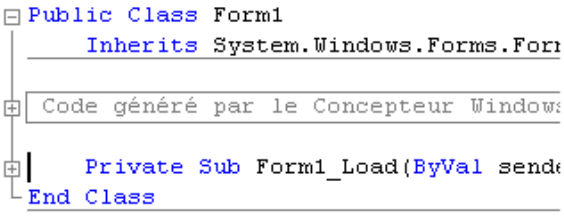
```
'todo: commentaire automatiquement ajouté dans la liste des tâches
```

### 4.1.4 Les régions

Les régions sont une nouvelle fonctionnalité de Visual Basic permettant de masquer une partie du code pour gagner en lisibilité.

La fonctionnalité appliquée aux région se retrouve sur chaque procédure Visual Basic : le signe + situé à gauche du nom des procédures permet de ne garder que leur déclaration.




Il est possible de créer vos propres régions avec la syntaxe suivante :

```
#Region ‘’libellé de la région‘’
        …
#End Region
```

```
#Region ‘’Déclaration des variables’’
        dim i, j as integer =0
        dim nom as string
#End Region
```

## 4.2 Les variables

Les variables permettent de stocker des informations lors de l’exécution du programme.
Une variable est caractérisée par les informations suivantes

- Un nom : le nom d’une variable commence obligatoirement par une lettre, peut contenir des lettres, chiffres et le signe « _ » (underscore) avec un maximum de 255 caractères. Visual Basic ne tient pas compte de la casse (Majuscules / Minuscules)
- Un type : le type d’un variable précise le type de la valeur stockées par la mémoire (numérique, chaîne de caractère, date …)
- Une portée : la portée d’une variable correspond à sa durée de vie

### 4.2.1 Types de variables

#### *4.2.1.1 Types numériques*

| Nom | Min | Max | Taille |
|---|---|---|---|
| Byte | 0 | 255 | 8 bits |
| Short | -32768 | 32767 | 16 bits |
| Integer | -2 147 483 648 | 2 147 483 647 | 32 bits |

| | | | |
|---|---|---|---|
| Long | -9223372036854775808 | 9223372036854775807 | 64 bits |
| Single | -3.402823 E 38 | 3.402823 E 38 | 32 bits |
| Double | -1.79769313486231 E 308 | -1.79769313486232 E 308 | 64 bits |
| Decimal | -79228162514264337593543950335 | 79228162514264337593543950335 | 16 Octets |

#### *4.2.1.2 Types chaîne de caractère*

Depuis la version .Net, les chaînes de caractère sont enregistrées au format Unicode, c'est à dire que chaque caractère utilise 2 octets.

| Nom | Description | Exemple |
|---|---|---|
| Char | Utilisé pour stocker un seul caractère | ''a'' |
| String | Chaîne de caractère à longueur variable d'une taille de 0 à environ 1 milliard de caractères | ''Sandrine Desayes'' |

#### *4.2.1.3 Autres types*

| Nom | Description | Exemple |
|---|---|---|
| Boolean | Variable dont le contenu peut être False (0) ou True (1) | True |
| Date | Stocke les informations de date et heure. Si la date est omise, le 1er janvier de l'année en cours sera utilisé. Si l'heure est omise, minuit sera l'heure par défaut. | #12/07/02 15:33:17#<br>#12/07/02#<br>#15:33:17# |
| Object | Le type object est un type universel (anciennement nommé Variant). Ce type ne stocke en fait pas la donnée elle même mais il contient un pointeur vers l'adresse mémoire contenant la donnée. De ce fait, une variable de type object peut pointer vers (contenir) n'importe quel type de données | !Erreur !<br>L'affectation des adresses mémoire se fait de manière implicite |

### 4.2.2 Déclaration de variables

La déclaration des variables permet, lors de l'exécution d'un programme de réserver l'espace mémoire nécessaire au stockage des informations. Par défaut, la déclaration des variables est obligatoire mais il est possible de la désactiver (voir les options communes de projet). Il est cependant déconseiller de la désactiver car une simple erreur de syntaxe au niveau du nom d'une variable ne sera pas interprété comme une erreur par le compilateur et de ce fait, le programme travaillera sur deux emplacements mémoire bien distincts.

La déclaration des variables se fait de la façon suivante

```
Dim nomvariable1, nomvariable2, nomvariable3 as type_variables = valeur_par_defaut
```

L'affectation d'une valeur par défaut est facultative.

```
Dim i as integer = 0
Dim moncar as char = ''i''
Dim aujourdui as date = #12/07/02#
Dim nom as string
```

### 4.2.3 Portée et visibilité des variables

#### *4.2.3.1 Portée des variables*

La portée d'une variable est équivalent à sa durée de vie, c'est à dire tant qu'elle est accessible. La portée est définie en fonction de l'endroit où est placée sa déclaration. Quatre niveaux de portée existent :

- Niveau Bloc : si la variable est déclarée dans un bloc (boucle while, condition …), la variable ne sera valable que à l'intérieur du bloc

- Niveau Procédure : si la variable est déclarée à l'intérieur d'une procédure ou d'une fonction, elle ne sera accessible qu'à l'intérieur de cette procédure / fonction

- Niveau Module : la variable déclarée dans un module sera accessible uniquement à partir des éléments du modules (procédures, fonctions)

- Niveau projet : la variable est accessible à partir de tous les éléments du projet. Dans ce cas, il faut utiliser le mot clé « friend » à la place du mot « dim ».

#### *4.2.3.1 Visibilité des variables*

En dehors de l'emplacement où est définie la variable, plusieurs mot clés sont disponibles pour agir sur la visibilité :

- Public : tous les blocs de code peuvent accéder à la variable
- Private : seul les éléments membres de la classe ou du module peuvent y accéder

```
Dim i as integer
Public y as string
Private z as integer
```

### 4.2.4 Les tableaux

Un tableau est un regroupement de variables accessibles par le même nom et différentiables par leurs indices.

- Un tableau peut avoir jusqu'à 32 dimensions.
- Les indices de tableaux commencent toujours à 0.

- Lors de la déclaration d'un tableau, l'indice maximum est précisé : le tableau comportera donc (indice_max + 1) valeurs.

```
Dim montableau(10) as integer
Dim tableau2(5) as string
```

Pour accéder à un élément du tableau, il faut préciser le nom du tableau avec entre parenthèses l'indice de l'élément désiré.

```
Montableau(2) = 123
Tableau2(5) = ''toto''
```

En Vb, les tableaux sont dynamiques, c'est à dire qu'il est possible de modifier leur taille. Pour cela, il faut utiliser l'instruction « Redim » suivi du nom du tableau et de sa nouvelle taille.

```
Redim montableau(15)
```

Cependant, l'instruction « Redim » ne conserve pas les données déjà présentes dans le tableau. Pour conserver son contenu, utiliser le mot clé « preserve »

```
Redim preserve montableau(15)
```

### 4.2.5 Les constantes

Les constantes permettent de stocker une valeur en s'assurant qu'elle ne sera jamais modifiée. Elles sont généralement utilisées pour améliorer la modularité du code en ne définissant qu'une seule fois une valeur utilisée plusieurs fois.

Il n'est pas obligatoire de préciser le type de la constante sauf lorsque l'option « Strict » du compilateur est activée.

```
Dim tx_tva = 19.6
Dim voyelles as string = ''aeiouy''
```

### 4.2.6 Les énumérations

Les énumérations permettent de définir un type de variable dont les valeurs appartiennent à un domaine précis. Chacune des valeurs d'une énumération correspond à un entier : le premier élément de l'énumération est initialisé à 0, le second à 1 etc…

```
Enum civilite
        Monsieur
        Madame
        Mademoiselle
End enum
```

Il est cependant possible de personnaliser les indices utilisés :

```
Enum jour
        Jeudi = 4
        Lundi = 1
        Dimanche = 7
        …
End enum
```

Une fois l'énumération déclarée, il est possible de déclarer une variable du type de l'énumération.

```
Dim macivilite as civilite
```

De cette façon, la variable macivilite pourra uniquement prendre les valeurs définies dans l'énumération :

```
Macivilite = monsieur
Macivilite = 1              'correspond à madame
Macivilite = 4              'invalide, l'indice max est 2
```

De manière générale, les énumérations sont utilisées pour le passage de paramètres, de façon à utiliser des constantes « parlantes » plutôt que des valeurs numériques pas toujours compréhensibles.

## 4.3 Les opérateurs

il existe 5 types d'opérateurs

### 4.3.1 Opérateur d'affectation

Un seul opérateur d'affectation existe et ce quelque soit le type de données concerné : le signe égal « = ». Il permet d'affecter une valeur à une variable.

```
Mavar = 123
```

Attention, le signe égal est aussi utilisé pour la comparaison.

Il est également possible d'utiliser les opérateurs arithmétique lors d'une affectation.

```
Dim i as integer = 5
I += 3          'équivalent à i = i + 3
```

### 4.3.2 Opérateurs Arithmétiques

les opérateurs arithmétique permettent d'effectuer des calculs sur des variables, constantes …

| Opérateur | Description | Exemple | Résultat |
|---|---|---|---|
| + | Addition | 3 + 2 | 5 |
| - | Soustraction | 8 – 6 | 2 |
| * | Multiplication | 3 * 4 | 12 |
| / | Division | 8 / 2 | 4 |
| \ | Division entière | 9 \ 2 | 4 |
| Mod | Modulo (Reste de la division entière) | 9 mod 2 | 1 |
| ^ | Puissance | 3 ^ 2 | 9 |

```
Dim nombre as double
Nombre = 50 + 7
Nombre = 8 mod 2
```

### 4.3.3 Opérateurs de comparaison

Il permettent de comparer deux membres et, en fonction du résultat, retournent True ou False

| Opérateur | Description | Exemple | Résultat |
|---|---|---|---|
| = | Egalité | 2 = 5 | False |
| < > | Inégalité | 3 < > 6 | True |
| < | Inférieur | -2 < 12 | True |
| > | Supérieur | 8 > 9 | False |
| < = | Inférieur ou égal | 9 <= 9 | True |
| > = | Supérieur ou égal | 13 >= 7 | True |
| Like | Egalité de chaîne de caractères | ‘’maison’’ like ‘’m*’’ | True |

### 4.3.4 Opérateurs de concaténation

La concaténation est l’union de deux chaînes. Deux opérateurs existent :

- Le plus « + » : dans ce cas, il faut que les deux membres de la concaténation soient de type chaîne de caractères.

- L’éperluette « & » : dans ce cas, l’opérateur effectue une conversion implicite lorsque les deux membres ne sont pas des chaînes de caractères.

```
Dim chaine as string
Dim nombre as integer
Chaine = ‘’nous sommes le ‘’
Nombre = 23
Msgbox (chaine & nombre)        ‘affiche nous sommes le 23
Msgbox (chaine + nombre)        ‘plantage !
```

#### 4.3.5 Opérateurs logiques

Les opérateurs logiques permettent de combiner des expressions logique (renvoyant True ou False) à l’intérieur de conditions.
Par rapport à la version 6 du langag, deux opérateurs font leur apparition : AndAlso et OrElse. Ces opérateurs permettent une optimisation des conditions dans le sens où il n’effectueront le second test que si le premier ne répond pas à la condition.

| Opérateur | Description | Exemple | Résultat |
|---|---|---|---|
| And | Et logique | True and False | False |
| Or | Ou logique | True or False | True |
| Xor | Ou exclusif | True Xor True | False |
| Not | Négation | Not true | False |
| AndAlso | Et logique optimisé | Test1 AndAlso test2 | Test2 sera évalué que si Test1 est True |
| OrElse | Ou logique optimisé | Test1 OrElse Test2 | Test2 sera évalué que si Test1 est faux |

## 4.4 Les structures de contrôle

Les structures de contrôle permettent de modifier le nombre d’exécution d’une instruction.

#### 4.4.1 Les conditions

Une condition est une comparaison entre deux membres dont le résultat retourne True (Vrai) ou False (Faux). Une condition est composée d’au moins 2 membres (variables, constantes, appel de fonction) et éventuellement d’opérateurs logiques permettant de lier les sous conditions. Les conditions sont utilisées dans les structures de contrôle afin d’en définir leur fonctionnement.

#### 4.4.2 Structures conditionnelles

Egalement nommées structures de décision, les structures conditionnelles aiguillent l’exécution de tel ou tel bloc de code en fonction d’une condition.

##### *4.4.2.1 Structure If*

Plusieurs syntaxes sont possibles :

- Forme simple

A utiliser dans le cas où vous ne souhaitez réaliser qu'une seule instruction lorsque la condition est vérifiée. Dans ce cas, la condition et l'instruction à réaliser doivent se trouver sur le même ligne.

```
If qte_stock < 10 then nouvelle_commande()
```

- Forme normale

La forme normale permet d'exécuter plusieurs instructions lorsque la condition est vérifiée.

```
If qte_stock < 10 then
        nouvelle_commande()
        prevenir_service_reception()
End if
```

- Forme évoluée

La forme évoluée permet d'exécuter plusieurs instructions lorsque la condition est vérifiée et d'exécuter d'autres instructions.

```
If note  < 10 then
        Msgbox(''Examen échoué'')
Else
        Msgbox(''Examen réussi'')
End if
```

#### *4.4.2.2 Structure Select Case*

La structure de contrôle Select Case permet d'effectuer un ensemble de test sur une seule valeur. Cette valeur peut-être le contenu d'une variable, le résultat d'un calcul ou d'une fonction. Le principal intérêt de cette structure est de clarifier le code : en effet, toute utilisation de cette structure peut être remplacée par un ensemble de structures If.

Les différentes possibilités de test sont les suivantes :

- Case constante : test valide si valeur  = constante
- Case min to max : pour les valeurs numérique, permet de définir un interval
- Case is > constante : pour les valeurs numériques, définition d'un interval non fermé
- Case Else : cette condition sera validée si tous les tests définis dans le select case sont faux

```
Dim note as integer
Note = inputbox(‘’Veuillez saisir une note’’)
Select case Note
        Case is <= 10
                Msgbox(‘’Examen échoué’’)
        Case 11 to 19
                Msgbox(‘’Examen réussi’’)
        Case 20
                Msgbox(‘’Excellent, tout est juste’’)
        Case else
                Msgbox(‘’Note invalide’’)
End Select
```

#### *4.4.2.3 Instruction conditionnelles*

3 autres instructions existent pour effectuer des tests :

- Iif

L’instruction IIF permet d’évaluer une condition et de retourner des valeurs différentes en fonction du résultat de la condition.

```
IIF (condition, valeur_retournée_si_vrai, valeur_retournée_si_faux)
```

```
Appreciation = iif(note < 10, ‘’Echoué’’, ‘’Reçu’’)
```

- Switch

L’instruction Switch regroupe plusieurs conditions et retourne le résultat correspondant pour la première condition vraie.

```
Switch (condition1, valeur1, condition2, valeur2 ….)
```

```
Lib_famille = Switch (code_famille=1, ‘’Info’’, code_famille=2, ‘’Consommable’’)
```

- Choose

L’instruction Choose permet de choisir une valeur dans une liste en fonction d’un Index. Attention, les Index commencent à 1 !

```
Choose(index, valeur1, valeur2 ….)
Lib_famille = Choose(code_famille, ‘’Info’’, ‘’Consommable’’)
```

### 4.4.3 Structures répétitives

Les structures répétitives ou structures de boule permettent d'exécuter un bloc d'instructions un certain nombre de fois.

#### *4.4.3.1 Structure While*

La structure while répète les instructions contenues entre While et End while tant que la condition est vérifiée.

```
While condition
        …
end while
```

```
dim i as integer = 0
while i < 10
        msgbox (i)
        i++
end while
```

#### *4.4.3.2 Structure Do loop*

La structure do loop possède 4 variantes. Elle peut être paramétrée avec les mots clés while (tant que la condition est vérifiée) et Until (Jusqu'à ce que la condition soit vérifiée).

```
Do while condition
        …
loop
```

```
Do until condition
        …
loop
```

Dans ces deux cas, la condition est évaluée une première fois avant l'entrée dans la boucle, c'est à dire qu'il est tout à fait possible que les instructions de la boucle ne soient exécutées aucune fois.

Une seconde variant de cette structure permet d'exécuter au moins une fois les instructions de la boucle et d'évaluer la condition ensuite :

```
Do
        …
loop while condition
```

```
Do
        …
loop until condition
```

#### *4.4.3.3 Structure For*

La structure For est utilisée pour exécutée un bloc d'instruction un nombre de fois déterminé tout en gérant un compteur qui sera automatiquement incrémenté. Le nombre d'exécution est fonction de l'intervalle définit pour le compteur :

```
For compteur = valeur_depart to valeur_fin step valeur_increment
        …
next
```

L'exemple suivant affiche les nombres 1,3,5,7,9

```
dim i as integer
for i = 1 to 10 step 2
        msgbox (i)
next
```

#### *4.4.3.4 Structure For each*

La structure For Each permet de parcourir un tableau ou une collection (tableau d'objets) sans se préoccuper de leurs indices. L'utilisation d'une telle structure est généralement plus rapide lors de l'exécution qu'un parcourt par indices.

```
For each element in collection_ou_tableau
        …
next
```

## 4.5 Procédures et fonctions

Dans une application Visual Basic, les instructions doivent obligatoirement figurer dans une procédure ou une fonction. Ce sont des unités logique de code qui seront ensuite appelées. Le découpage d'un programme en procédures et fonctions présente plusieurs intérêts :

- Lisibilité du code : lors de débogage, il est plus facile d'analyser le comportement de plusieurs blocs de code de 10 lignes qu'un pavé de 200 lignes
- Modularité : l'écriture de procédures et fonctions permet ensuite d'effectuer des appels à partir de plusieurs endroits dans le code au lieu de réécrire les même instructions
- Evolutivité : lorsqu'une procédure est utilisée dans plusieurs partie du programme, modifier la procédure permettra de répercuter les modifications pour tous les appels de la procédure

Enfin, il existe plusieurs cas dans lequel l'utilisation de procédures est imposé par Vb.

Il existe en Vb 4 catégories de procédures :

- Les procédures sub qui exécutent du code à la demande

- Les procédures évènementielle déclenchées automatiquement lorsqu'un événement se produit (clic sur un bouton par exemple)
- Les fonctions qui exécutent un bloc de code et retournent une valeur
- Les procédures de propriétés obligatoires lors de la manipulation d'objets (leur utilisation est spécifié dans le chapitre consacré aux objets)

### 4.5.1 Création de procédure

L'utilisation d'une procédure comporte deux étapes : la déclaration définissant le comportement de celle ci et l'appel demandant l'exécution.

Attention, la déclaration d'une procédure ne l'exécute en aucun cas !!!

#### *4.5.1.1 Déclaration*

```
public sub nom_procedure(paramètres)
        …
end sub
```

- Toutes les instructions situées entre sub et end sub feront partie de la procédure
- Le nom d'une procédure suit les même règles que pour les variables
- Il possible de préciser la visibilité de la procédure en remplaçant public par private ou friend

#### *4.5.1.2 Appel*

Pour appeler une procédure, il suffit de placer son nom avec le mot clé call (facultatif)

```
Call nom_procedure(valeurs_parametres)
nom_procedure(valeurs_parametres)
```

### 4.5.2 Création de fonction

Les fonction sont des procédures retournant un résultat.

#### *4.5.2.1 Déclaration*

Lors de la déclaration d'une fonction, il faut reprendre les même éléments que pour les procédures avec en plus le type de retour de la fonction (String, integer…).
La fonction doit également comporter l'instruction Return qui permet de définir la valeur retournée par la fonction.

```
Public function nom_fonction(paramètres) as type_retour
        …
        return valeur_retour
end function
```

#### *4.5.2.1 Appel*

Nous l'avons dit plus haut, un fonction est une procédure retournant une valeur. Cette valeur peut être issue d'un calcul ou être un code erreur. De ce fait, lors de l'appel d'un fonction, il faut récupérer le résultat afin de le traiter. Le résultat peut donc être stocké directement dans une variable ou encore être passé en paramètre à une autre procédure ou fonction.

```
Variable = nom_fonction(valeurs_parametres)
Msgbox (nom_fonction(valeurs_parametres))
```

### 4.5.3 Passage de paramètres

Le passage de paramètres permet de personnaliser le fonctionnement d'une procédure ou d'une fonction en lui précisant lors de l'appel les valeurs sur lesquelles elle devra travailler.

#### *4.5.3.1 Déclaration*

C'est lors de la déclaration de la procédure ou de la fonction que vous devez préciser les paramètres attendus. Ces paramètres seront ensuite utilisables comme des variables locales à la procédure.

```
Public sub nom_procedure (parametre1 as type_parametre1, parametre2 as type_parametre2
…)
```

```
Public sub feliciter (nom as string)
        Msgbox (''bravo '' & nom)
End sub
```

Dans certains cas, un paramètre peut être facultatif. Dans ce cas, il faut placer le mot clé optionnal devant le paramètre et, éventuellement, spécifier la valeur par défaut avec l'opérateur d'affectation :

```
Public sub feliciter (nom as string, optional prenom as string = ''inconnnu'')
        Msgbox (''bravo '' & nom & '' '' & prenom)
End sub
```

#### *4.5.3.2 Appel*

Lors de l'appel d'une procédure ou d'une fonction, il faut donner les valeurs des différents paramètres utilisés par la fonction.

```
Feliciter(''Dupont'')
Feliciter(''Dupont'', ''gerard'')
```

#### *4.5.3.3 Passage par valeur et par référence*

Lors du passage de paramètres à une procédure ou à une fonction, deux méthodes sont disponibles :

- Passage par valeur

C'est le mode de passage de paramètre par défaut : lors du passage par référence, seule la valeur du paramètre est passé à la procédure. Cette dernière travaille donc sur une variable locale.

Pour définir un passage de paramètre par valeur, il suffit d'ajouter le mot clé « byval » devant le nom du paramètre dans la déclaration de la procédure.

Dans l'exemple suivant, les 2 variables nom et enom font référence à des emplacements mémoires distincts.

```
Sub main()
        Dim nom as string = ''toto''
        Afficher(nom)
End sub

Sub afficher(byval enom as string)
        Msgbox(enom)
End sub
```

- Passage par référence

Dans le cas d'un passage par référence, ce n'est pas la valeur du paramètre qui est passé mais l'adresse de la variable contenant la valeur : dans ce cas, la procédure appelée et la procédure appelante travaille sur la même variable, même si le nom utilisé pour le paramètre est différent du nom de la variable initiale. De ce fait, il est possible de modifier le contenu de la variable passée en paramètre à partir de la procédure appelée.

Pour définir un passage de paramètre par valeur, il suffit d'ajouter le mot clé « byref » devant le nom du paramètre dans la déclaration de la procédure.

```
    Sub main()
        Dim nom as string = ''toto''
        Modifier(nom)
        Msgbox(nom)          'affiche titi
    End sub

    Sub modifier(byref enom as string)
        Enom = ''titi''
    End sub
```

#### *4.5.3.4 Passer un nombre quelconque de paramètre*

Visual basic .Net permet le passage d'un nombre quelconque de paramètre grâce au mot clé « ParamArray ». Il suffit de définir un paramètre de type tableau correspondant au type des paramètres qui devront être passé. Chaque élément du tableau correspond ensuite à un paramètre.

```
Sub nom_procedure(ParamArray tableau() as type_parametre)
```

```
Sub somme (ParamArray nombres() as integer)
        Dim nombre as integer
        Dim total as integer
        For each nombre in nombres
                Total += nombre
        Next
        Msgbox (total)
End sub
```

Pour la procédure somme, il sera donc possible d'effectuer les appels suivants :

```
Somme (12,2)
Somme(15,7,6,9,2,4)
```

## 4.6 Fonctions intégrées

Visual Basic comporte un ensemble de fonctions déjà intégrées permettant de réaliser les opérations de base sur les nombres, chaînes et dates. Ces fonctions peuvent apparaître sous deux formes :

- Fonction standards Vb
- Méthodes des classes de type (System.String, System.DateTime …)

De manière générale, il est préférable d'utiliser les méthodes liées aux type de variables dans le sens où elles sont accessibles par simple ajout du point après le nom de la variable (Liaison précoce).

Attention, l'appel des méthodes ne change pas la valeur de la variable mais retourne la valeur modifiée.

### 4.6.1 Fonctions sur les chaînes de caractères

Pour les exemples du tableau ci-dessous, nous utiliserons la variable suivante :

```
Dim ch as string = ‘’tartanpion’’
```

| Méthode | Description | Exemple | Résultat |
|---|---|---|---|
| .Chars(Index) | Retourne le caractère à l'indice Index | Ch.Chars(2) | ‘’r’’ |

| | | | |
|---|---|---|---|
| .IndexOf(Caractère, debut) | Retourne l'indice de la première occurrence de caractère à partir de la position début | Ch.IndexOf(''p'') | 6 |
| .Insert(Indice, chaine) | Insère chaine à la position indice | Ch.Insert(2,''ta'') | Tatartanpion |
| .Length | Retourne le nombre de caractère de la chaîne | Ch.length | 10 |
| .PadLeft(nb, remplissage) | Formate la chaîne sur nb caractères en remplissant les espaces vides à gauche avec remplissage | Ch.PadLeft(15,''s'') | Ssssstartanpion |
| .PadRight(nb, remplissage) | Identique mais à droite | Ch.PadLeft(15,''s'') | Tartanpionssssss |
| .Remove(debut, nombre) | Supprime les caractères de la chaîne à partir de l'indice début sur une longueur de nombre | Ch.remove(2,3) | Tanpion |
| .Replace(recherche, remplace) | Remplace les occurrences de recherche par remplace | Ch.replace(''an'', ''i'') | Tartipion |
| .Split(Séparateur) | Découpe une chaîne selon séparateur et retourne un tableau | Ch.Split(''a'') | T<br>Rt<br>Npion |
| .StartWith(debut) | Retourne un booléen spécifiant si la chaîne commence par debut | Ch.StartWith(''tar'') | True |
| .SubString(Debut, n) | Retourne la sous chaîne à partir de la position Debut sur 1 longueur de n caractères | Ch.SubString(4,2) | Anp |
| .ToLower() | Met la chaîne en minuscule | Ch.tolower | Tartanpion |
| .ToUpper() | Met la chaîne en majuscule | Ch.toUpper | TARTANPION |
| .Trim(caractère) | Supprime les occurrences de caractère en début et en fin de chaine | Ch.trim(''t'') | Artanpion |

D'autres fonctions sont disponibles dans l'espace de nom « `microsoft.VisualBasic.Strings` »

### 4.6.2 Fonctions sur les nombres

L'ensemble des fonctions disponibles pour la gestion des nombres est disponible dans la classe Math.

| Fonction | Description | Exemple | Résultat |
|---|---|---|---|
| .Floor(n) | Tronque la valeur | .Floor(3.25) | 3 |
| .Ceiling(n) | Retourne le nombre entier juste | .Ceiling(3.25) | 4 |

| | | | |
|---|---|---|---|
| | supérieur | | |
| .Pow(n, p) | Retourne n à la puissance p | .pow(2,3) | 8 |
| .abs(n) | Retourne la valeur absolue de n | .abs(-8) | 8 |

### 4.6.3 Fonctions sur les dates

Pour les exemples ci dessous, nous utiliserons la variable suivante comme base :

```
Dim d as datetime = #12/07/2003 16:00#
```

| Méthodes | Description | Exemple | Résultat |
|---|---|---|---|
| .addmonths(n) | Ajoute n mois à la date | d.addmonths(2) | #12/09/2003 16 :00# |
| .addDay(n) | Ajoute n jours à la date | d.addDay(1) | #13/07/2003 16:00# |
| .addHours(n) | Ajoute n heures à la date | d.addHours(1) | #12/07/2003 17:00# |
| .addMinutes(n) | Ajoute n minutes à la date | d.addMinutes(1) | #12/07/2003 16:01# |
| .addSeconds(n) | Ajoute n secondes à la date | d.addSeconds(1) | |
| .addYears(n) | Ajoute n années à la date | d.addyears(1) | #12/07/2004 16:00# |
| .Day | Jour du mois | d.day | 12 |
| .DayOfWeek | Jour de la semaine | d.DayOfWeek | 6 |
| .DayOfYear | Jour de l'année | d.DayOfYear | 341 |
| .DaysInMonth(y, m) | Nombre de jour pour le mois m de l'année y | d.DaysInMonth(1,2003) | 31 |
| .Minute | Retourne la valeur de minute | d.minute | 00 |
| .Hour | Retourne la valeur de heure | d.hour | 16 |
| .Day | Retourne la valeur de jour | d.day | 12 |
| .Month | Retourne la valeur de mois | d.month | 07 |
| .Year | Retourne la valeur de année | d.year | 2003 |
| .Now | Retourne l'heure et la date courante | d.now | |

### 4.6.4 Fonctions sur les tableaux

pour les exemples ci dessous, nous utiliserons la variable suivante :

```
dim t() as integer = {1,9,4,3}
```

| Méthodes | Description | Exemple | Résultat |
|---|---|---|---|
| .Length | Retourne la taille du tableau | t.length | 4 |

| | | | |
|---|---|---|---|
| .Reverse(tab) | Inverse les éléments de tab | t.reverse(t) | 3,4,9,1 |
| .Sort(tab) | Trie les éléments de tab | t.Sort(t) | 1,3,4,9 |
| .GetUpperBound(d) | Retourne l'indice de l'élément le plus grand dans la dimension spécifiée | t.GetUpperBound(0) | 3 |

### 4.6.5 Fonctions de conversion

Les fonctions de conversion permettent de travailler sur plusieurs variables possédant des types de données différents. Dans tous les cas, le type de données retourné est modifié et la valeur, lorsque cela est possible, adaptée au nouveau type. Une erreur sera générée dans les autres cas.

| Fonction | Type de données retourné |
|---|---|
| Cbool | Booléen |
| Cdate | Date |
| CDbl | Double |
| Cint | Entier |
| Cstr | String |

Il existe principalement deux cas d'utilisation de ces fonctions :

- Convertir une donnée lors d'un passage de paramètres
- S'assurer du bon fonctionnement des opérateurs de comparaison

Il existe une autre fonction de formatage dont le fonctionnement est un peu différent : la fonction « ctype » permet de convertir une variable de type « Object » vers un type d'objet précis. Le principal intérêt de cette méthode est de profiter de la liaison précoce lors de l'écriture de programmes.

```
Ctype(objet, classe)
```

```
Dim f as object
Dim g as Personne
F = new personne(''toto'')
G = Ctype(f, personne)
```

### 4.6.6 Fonction de formatage (Format)

Les fonctions de formatage permettent une représentation différentes des données numériques ou dates mais ne sont pas utilisée pour des conversion de types.
Une seule fonction est utilisée en Vb : la fonction format.

```
Format(expression_a_formater, style) as string
```

L'expression à formater peut être une valeur numérique ou de type date / heure.

Le style est un format spécifiant les éléments de représentation de l'expression (par exemple, le nombre de décimal après la virgule …). Deux possibilités sont offertes à l'utilisateur pour le formatage :

- Des formats prédéfinis sous vb (par exemple, la date au format américain)
- Des caractères de formatage permettant à l'utilisateur de définir ses propres formats

#### *4.6.6.1 Caractères de formatage pour les numériques*

- Formats prédéfinis

| Caractère | Description | Exemple |
|---|---|---|
| C | Séparateur de milliers et 2 décimales | 12 846.49 |
| Percent | Multiplie le nombre par 100 et ajoute le signe % | 1284648.61% |
| Scientific | Notation scientifique | $1.28^{E}$+04 |

- Formats définis par l'utilisateur

| Caractère | Description |
|---|---|
| 0 | Affiche le chiffre ou 0 |
| # | Affiche le chiffre ou rien |
| . | Décimal |
| , | Séparateur de milliers |
| \ | Caractère d'échappement |

```
Dim nb as double = 12.263
Msgbox (format(nb, ‘’000,000.00’’))                    ‘affiche 000 012.26
Msgbox (format(nb, ‘’ ###.##’’))                       ‘affiche 12,26
Msgbox (format(nb, ‘’résultat\ : ###,###.##’’))        ‘affiche résultat : 12.26
```

#### *4.6.6.2 Caractères de formatage pour les dates*

- Formats prédéfinis

| Caractère | Description | Exemple |
|---|---|---|
| G | Affiche la date selon les paramètres du système | 4/3/2003 |
| D | Format date longue spécifié au niveau du système | 04/03/2003 |

- Formats définis par l'utilisateur

| Caractère | Description |
|---|---|
| : | Séparateur d'heures |
| / | Séparateur de date |

| | |
|---|---|
| d | Jour sans zéro non significatif |
| dd | Jour sur 2 chiffres |
| ddd | Jour sous la forme d'abbréviation (ex : Lun) |
| dddd | Jour sous la forme du nom complet (ex : Lundi) |
| M | Mois sans zéro non significatif |
| MM | Mois sur 2 chiffres |
| MMM | Mois sous la forme abrégée (ex : Juil) |
| MMMM | Mois sous la forme de nom complet |
| gg | Affiche l'ère (A.D) |
| h | Heure sans zéro significatif |
| hh | Heure sur 2 chiffres |
| m | Minute sans zéro significatif |
| mm | Minute sur 2 chiffres |
| s | Seconde sans zéro significatif |
| ss | Seconde sur 2 chiffres |
| tt | Format 12 heures |
| yy | Année sur 2 chiffres |
| yyyy | Année sur 4 chiffres |
| z | Affiche le décalage horaire |

```
Dim madate as datetime = #12/07/2002 15 :30#
Msgbox(format(madate, ‘’M/d/yy‘’))                    ‘affiche 12/7/02
Msgbox(format(madate, ‘’d-MMMM-yyyy‘’))               ‘affiche 12-Juillet-2002
Msgbox(format(madate, ‘’ddd d MMM yy‘’))              ‘Lun 12 Juil 02
```

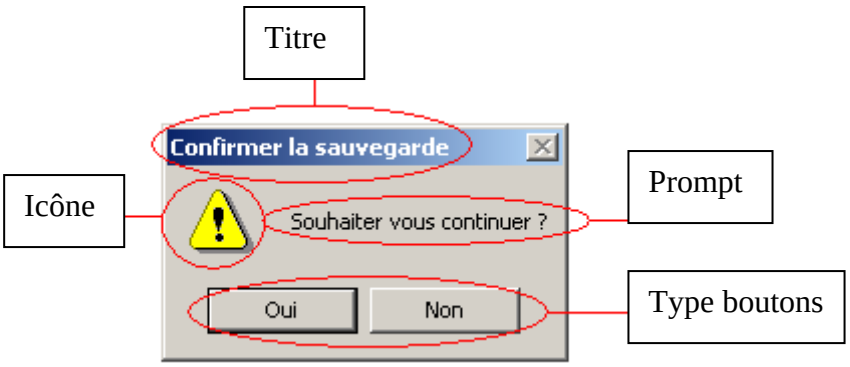
### 4.6.7 Les boîtes de dialogue

Les boîte de dialogue intégrées sont des feuilles modales (c'est à dire qu'elle suspendent l'exécution du programme jusqu'à leur fermeture) destinées à afficher une information ou demander une saisie.

#### *4.6.7.1 Boite de message*

La boîte de message (MsgBox) permet l'affichage d'une information et donne la possibilité de configurer les boutons, le titre et l'icône associée.

```
MessageBox.Show(prompt, titre, type_boutons, icône, bouton_par_defaut)
```




- Types de bouton

| Types de bouton (membres de la classe MessageBoxButtons) | |
|---|---|
| **Constante** | **Description** |
| Ok | Bouton Ok Seulement |
| OkCancel | Boutons Ok et bouton annulé |
| AbortRetryIgnore | Boutons Abandonner, Réessayer, Ignorer |
| YesNoCancel | Boutons Oui, Non et Annuler |
| YesNo | Boutons Oui et Non |
| RetryCancel | Boutons Réessayer et Annuler |

- Constantes de Retour

La définition du type de boutons modifie l'affichage mais permet à la méthode .Show de retourner une valeur correspondant au bouton sélectionner par l'utilisateur.

| Constantes de retour (membres de la classe DialogResult) | |
|---|---|
| **Constante** | **Description** |
| Ok | Bouton Ok |
| Cancel | Bouton Annuler |
| Abort | Bouton Abandonner |
| Retry | Bouton Réessayer |
| Ignore | Bouton Ignorer |
| Yes | Bouton Oui |
| No | Bouton Non |

- Type d'icône

| Types d'icône (membres de la classe MessageBoxIcon) | |
|---|---|
| **Constante** | **Aperçu** |
| Error |  |
| Exclamation |  |
| Information |  |
| Question |  |

- Bouton par défaut

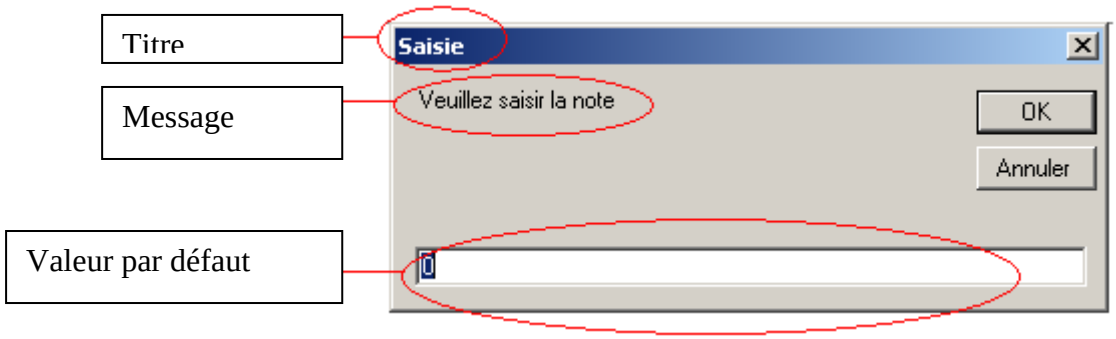
L'option bouton par défaut permet de définir le bouton qui sera sélectionné par défaut.

| **Bouton par défaut (membres de la classe MessageBoxDefaultButton)** | |
|---|---|
| **Constante** | **Description** |
| DefaultButton1 | Premier bouton |
| DefaultButton2 | Second bouton |
| DefaultButton3 | Troisième bouton |

```
MessageBox.Show("Le total de la commande s'élève à 1000 Eur", "Résultat",
MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1)

If MessageBox.Show("Continuer ?", "Confirmation", MessageBoxButtons.YesNo) =
DialogResult.Yes then
        MessageBox.Show("Vous avez choisi Oui ")
Else
        MessageBox.Show("Vous avez choisi Non ")
End if
```

#### *4.6.7.2 Boîte de saisie*

La boîte de saisie (InputBox) permet la saisie d'une chaîne de caractère par l'utilisateur. Il est possible de paramétrer le titre, le texte affiché ainsi que le texte saisi par défaut.




InputBox est une fonction, elle retourne la valeur saisie par l'utilisateur où chaîne vide si le bouton Annuler est utilisé. La valeur retournée est toujours de type String. Les éventuels contrôle de saisie devront être fait par le programme une fois la valeur retournée.

```
Dim note as string
note = InputBox(''Veuillez saisir la note'', ''Saisie'', ''0'')
```

## 4.7 Les collections

Une collection est une classe contenant un ensemble d'objets de même type ainsi que plusieurs propriétés et méthodes afin de gérer cette liste. Suivants vos besoins, il existe plusieurs types de collection tous issu de l'espace de nom « System.Collections ».
Pour imager, une collection est un tableau d'objets.

### 4.7.1 Classe CollectionBase

La classe CollectionBase permet de créer une classe collection contenant tous les éléments de base nécessaires à la création. Il ne reste qu'à définir quelques méthodes supplémentaires pour avoir une collection opérationnelle.

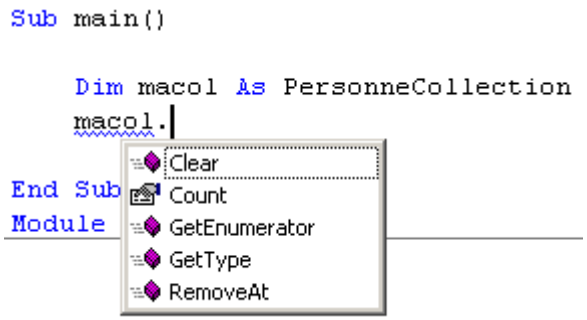
#### *4.7.1.1 Créer la collection*

Dans les exemples suivants, nous nous inspirons de la classe « Personne » décrite dans le chapitre consacrée à la POO.

La création d'une collection se fait à partir d'une classe dans laquelle il faut hériter de « system.collections.collectionbase ».

```
Public class PersonneCollection
        Inherits system.collections.CollectionBase
End class
```

La classe PersonneCollection hérite des éléments suivants :




Ceci n'est pas suffisant : en effet, nous n'avons pas accès au contenu de notre collection. Pour cela, nous devons implémenter la propriété « item » et la méthode « add » :

```
Public Class PersonneCollection
   Inherits System.collections.CollectionBase

   Public Property Item(ByVal index As Integer) As personne
      Get
         Return CType(Me.InnerList(index), personne)
      End Get
      Set(ByVal Value As personne)
         Me.InnerList(index) = Value
```

```
        End Set
    End Property

    Public Sub add(ByVal value As personne)
        Me.InnerList.Add(value)
    End Sub

End Class
```

#### *4.7.1.2 Utilisation*

```
        Dim macol As PersonneCollection
        macol = New PersonneCollection

        macol.add(New personne("toto"))
        macol.add(New personne("titi"))

        macol.RemoveAt(0)

        MsgBox(macol.Item(0).nom)
```

### 4.7.2 Classe Stack

La classe Stack est une collection particulière de type LIFO (Dernier rentré premier sorti) permettant de gérer une pile. Cette dernière se présente comme un tableau mais contient des fonctionnalités différentes. Pour imager, vous empiler des éléments les uns sur les autres et vous ne pouvez retirer que l'élément du haut. Ce genre de liste est utilisé pour gérer les files d'appels.

Pour déclarer une pile, on utilise la classe Stack en précisant le nombre maximum d'éléments :

```
Dim nom_pile as new Stack(nb)
```

```
Dim mapile as new Stack(10)
```

| **Propriétés et méthodes disponibles** | |
|---|---|
| Pop | Supprime la valeur au sommet de la pile et la retourne |
| Peek | Retourne la valeur au sommet de la pile |
| Contains(valeur) | Return true si valeur est présent dans la pile |
| Count | Nombre d'élément de la pile |
| Push | Insére une nouvelle valeur |
| Clear | Vide la pile |

```
Dim mapile as New Stack(10)
```

```
Mapile.push(5)
Mapile.push(8)
Mapile.push(10)

Msgbox (Mapile.peek)              'affiche 5
Msgbox (Mapile.pop)               'affiche 5 et supprime l'élément

Msgbox (Mapile.contains(10))      'affiche true

Msgbox (Mapile.count)             'affiche 2
```

#### 4.7.3 Classe Queue

La classe Queue est également issue de l'espace de nom « System.collections ». Elle permet de gérer des files FIFO (Premier rentré, premier sorti).

Pour déclarer une telle files d'attente, utiliser la classe Queue en précisant deux paramètres : la taille initiale de la liste (par défaut 32) et le facteur d'extension (par défaut 2).

```
Dim maliste as new Queue(taille_initiale, facteur)
```

```
Dim malisteas new Queue(100,1.5)
```

| Propriétés et méthodes disponibles | |
|---|---|
| Enqueue(valeur) | Ajoute un élément à la liste |
| Dequeue | Retourne la valeur et supprime l'élément |
| Peek | Lit la valeur |
| Count | Nombre d'élément |
| Clear | Vide la liste |
| Contains(valeur) | Retourne true si la valeur est présente dans la liste |

### 4.8 Gestion des erreurs

La gestion des erreurs est fondamentale en programmation : une bonne gestion d'erreur permet d'éviter tout plantage brusque avec risque de perte de données, renseigner le développeur sur l'origine du problème et enfin proposer ou choisir des alternatives.

#### 4.8.1 Types d'erreurs

| Erreur | Description |
|---|---|
| De syntaxe | De type d'erreur correspond aux fautes d'orthographe. Elles sont directement captées par l'environnement de développement qui les signale en les soulignant. Noter qu'une erreur de syntaxe empêche la compilation du programme et de fait ne peut survenir lors de l'exécution. |
| D'exécution | Une erreur d'exécution survient lorsque l'environnement |

| | |
|---|---|
| | d'exécution ne correspond pas à l'opération demandée (Accès à un fichier inexistant, dépassement de capacité…). Ces dernière affiche un message d'erreur incompréhensible pour l'utilisateur final avant de stopper le programme. |
| De logique | C'est sûrement le type d'erreur le plus compliqué à corriger : aucun plantage n'est effectué par la machine et pourtant le résultat obtenu est erroné. Sur ce genre d'erreur, il faut généralement « débugger » le programme avec des outils comme les espions. |

### 4.8.2 Gestion en ligne

La gestion en ligne existait déjà sur la version 6. Elle permet de définir les traitements à effectuer dans une procédure lorsqu'une erreur survient.

#### *4.8.2.1 L'instruction On Error*

L'instruction « On error » est la base de la gestion d'erreur : c'est elle qui va spécifier les opérations à réaliser lorsqu'une erreur survient.

Il existe 3 gestion différentes :

- On error resume next

Cette instruction ignore la ligne ayant provoqué l'erreur et continue l'exécution. Cette méthode est peu fiable dans le sens où vous n'êtes pas prévenu d'une erreur et le reste du programme peut en être affecté.

- On error goto étiquette

Cette instruction dirige l'exécution du code vers une étiquette en fin de procédure. C'est dans cette étiquette que vous pourrez ensuite définir les traitements correspondants.

```
    On error goto fin
    …
    exit sub
fin :
    … instructions de gestion
```

Il est important de placer l'instructions « Exit sub » ou « Exit function » juste avant la déclaration de l'étiquette car dans le cas où l'exécution se déroule correctement, il ne faut pas que la gestion d'erreur soit activée.

```
Public sub main()
        On error goto fin
        Dim i as integer
        i = 3 / 0          'plantage, division par 0
        exit sub
```

```
fin :
        msgbox(‘’Une erreur est survenue’’)
end sub
```

- On error goto 0

Cette instruction  désactive simplement les gestionnaires d’erreur.

#### *4.8.2.2 L’instruction Resume*

Une fois que vous avez redirigé le code lors d’une erreur, vous devez définir les actions à entreprendre : Il existe pour cela 2 mots clés permettant de continuer l’exécution du programme :

| Instruction | Description |
|---|---|
| Resume | Cette instruction reprend le programme où il s’était interrompu et essaye de réexécuter l’instruction à l’origine du plantage. |
| Resume Next | Reprend l’exécution du programme à partir de l’instruction suivant l’instruction à l’origine du plantage. |

L’exemple suivant illustre l’utilisation de Resume :

```
        On Error GoTo fin
        Dim i, j As Integer
        j = 0
        i = (3 / j)
        MsgBox("Fin")
        Exit Sub
fin:
        Select Case MessageBox.Show("Une erreur est survenue, souhaitez vous réessayer ?",
"", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning,
MessageBoxDefaultButton.Button1, MessageBoxOptions.RightAlign)
            Case DialogResult.Cancel
                Exit Sub
            Case DialogResult.Yes
                Resume
            Case DialogResult.No
                Resume Next
        End Select
```

#### *4.8.2.3 L’objet Err*

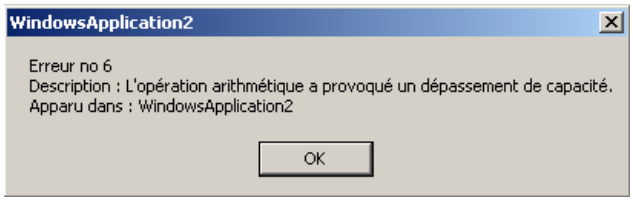
Avec la méthode précédente, vous récupérez toutes les erreur de la procédure et ce quelque soit l’erreur. Pour une gestion d’erreur plus pointue, vous avez la possibilité de récupérer des informations sur la dernière erreur générée : c’est l’objet « Err ».
Cet objet est toujours accessible.

| Propriété | Description |
|---|---|
| Description | Description textuelle de l'erreur |
| Erl | Numéro de la dernière ligne exécutée |
| Number | Numéro de l'erreur |
| Source | Objet ou application à l'origine de l'erreur |

| Méthode | Description |
|---|---|
| Clear | Efface les paramètres de l'objet |
| Raise | Déclenche une erreur |

Le code suivant affiche les informations sur l'erreur :

```
        On Error GoTo fin
        Dim i, j As Integer
        j = 0
        i = (3 / j)
        MsgBox("Fin")
        Exit Sub
fin:
        With Err()
            MsgBox("Erreur no " & .Number & vbCr & "Description : " & .Description & vbCr &
"Apparu dans : " & .Source)
        End With
```




### 4.8.3 Les Exceptions

#### *4.8.3.1 Try, Catch & Finally*

La gestion des exceptions est une méthode plus récente pour la gestion des erreurs. Cette méthode est standardisée pour tous les langages du FrameWork .Net . De plus, elle permet de définir une gestion d'erreur pour un bloc particulier et non pas pour une procédure ou une fonction.

La gestion des exceptions pour un bloc de code est structuré de la manière suivante : le code dangereux doit être compris dans le bloc « try ». Ensuite pour chaque exception suceptible d'être déclenchées, le mot clé « catch » (analogue au select case) permet de définir les traitements à exécuter. Enfin, le bloc « finally » contient du code qui sera exécuté dans tous les cas.

```
Try
        { Instructions dangereuses}
```

```
catch objet1 as type_exception1
        {Instructions à réaliser si exception1}
catch objet2 as type_exception2
        {Instructions à réaliser si exception2}
finally
        {Code à exécuter dans tous les cas}
end Try
```

L'exemple suivant montre l'utilisation des exceptions lors de l'accès à un fichier contenu sur une disquette :

```
Try
        Microsoft.visualbasic.fileopen(1, ''A:\fichier.txt'', openmode.input)
        Catch exception1 as system.io.ioexception
                Msgbox(''Erreur lors de l'ouverture du fichier : '' & exception1.source)
        Finally
                Msgbox(''Fin de la procédure'')
End try
```

A chaque exception est associé un objet contenant les informations sur l'exception levée :

| Propriété | Description |
|---|---|
| Message | Description textuelle de l'exception |
| Source | Nom de l'application ayant générée l'erreur |
| StackTrace | Liste de toutes les méthodes par lesquelles l'application est passée avant le déclenchement de l'erreur |
| TargetSite | Méthode ayant déclenchée l'exception |

# 5 Programmation Objet

Contrairement aux versions précédentes, Visual Basic .net impose un minimum de notions Objets afin de l'utiliser de façon optimale. De plus, de nombreux concepts objets sont venus renforcer la POO sous Visual Basic comme les notions d'héritage, d'interface et de polymorphisme.

## 5.1 Introduction à la POO

Lorsque nous parlons de programmation séquentielle, nous avons un ensemble de variables, de tableaux, de bloc de code, de procédure et de fonction relatifs au même domaine mais sans lien logique entre eux : par exemple, pour gérer un groupe de personne avec un nom et un prénom, nous utiliserons deux tableaux en faisant correspondre les indices pour la même personne. De même, les fonctions comme envoyer un mail ou augmenter le salaire travailleront sur les tableaux sans qu'il y ait de réel lien.

L'idée de la Programmation Orientée Objet est de faire correspondre à chaque élément du monde réel une entité informatique. C'est la notion d'objets. Chaque objet représente un élément du réel et contient ses informations (Couleur, taille ….). De plus, les fonctionnalités

attendues pour les objets seront implémentées sous la forme de méthode : c'est à dire une fonction ou une procédure liée à un type d'objet (les classes) et dépendante. Par exemple, la fonction calculer age perd tout son sens si vous ne l'appelez pas à partir d'une personne existante (objet). De ce fait, nous obtenons un programme structuré logiquement et grandement clarifié.

Le principe des classes, objets, méthodes et l'encapsulation sont les principes de base de la programmation orientée objet. Plusieurs aux concepts viennent enrichir la POO comme l'héritage qui permet de récupérer le comportement d'une classe déjà existante pour en créer une nouvelle, la surcharge qui permet de définir plusieurs fois le comportement d'une même procédure en fonction des paramètres passés et la visibilité (partie de l'encapsulation) permettant de cacher un certain nombre de paramètres inutiles pour l'utilisateur final.

Les objectifs de la POO sont les suivants :

- Regroupement logique des éléments de programmation pour une meilleur compréhension du programme
- Réutilisation des composants (une classe ou une partie d'une classe utilisée dans un programme peut être importée dans une autre, d'où les gains en productivité)
- Maintenance simplifiée car tous les objets ont un modèle (classe) qui une fois modifiée répercutera ses modifications sur tous les objets

## 5.2 Classes & Objets

Nous l'avons dit précédemment, une classe représente un objet du réel : elle sera composée d'attributs (variable contenant les caractéristiques de l'objet), de méthodes (procédures ou fonctions matérialisant leur comportement) et d'événement (bloc de code exécuter lorsque l'objet passe à un état définit).

La définition d'une classe permet de définir un modèle pour les objets qui seront créés.

### 5.2.1 Création d'une classe

Pour créer une nouvelle classe, il est conseillé de créer un nouveau fichier (.vb) dans le projet. La déclaration d'une classe se fait avec les mots clés « class » et « end class ».

```
Public class personne
        …
End class
```

En visual Basic .Net, il est possible de définir une classe à l'intérieur d'une autre classe :

```
Public class personne
        …
        Private class emploi
                …
```

```
        End class
End class
```

### 5.2.2 Visibilité de la classe

La visibilité d'une classe permet de définir la portée de celle ci, c'est à dire les blocs de code qui auront accès à la classe. Cinq niveaux de visibilité existent :

| Visibilité | Description |
|---|---|
| Public | Classe utilisable dans tout les projet |
| Friend | Classe utilisable uniquement dans le projet |
| Private | Classe utilisable uniquement dans le module / classe où elle est définie |
| Protected | Classe utilisable dans les sous classes de celle où elle est définie |
| Protected friend | Union des portées de Protected et Friend |

```
Protected class personne
        …
End class
```

### 5.2.3 Options d'héritage

Ces options permettront de préciser le comportement de la classe vis à vis de l'héritage.

| Option | Description |
|---|---|
| MustInherit | Indique que la classe sert de base dans une relation d'héritage (Super classe). |
| NotInheritable | Indique que la classe ne peut être héritée. Elle devra être au dernier niveau de la hiérarchie d'héritage. |

```
Public MustInherit class personne
        …
End class
```

### 5.2.5 Implements & Inherits

Le mot clé « Inherits » permet d'hériter des membres d'une super classe(Cf section sur l'héritage).

### 5.2.6 Création et utilisation d'objets

Une fois la classe définie, son utilisation devra se faire par la création ou instanciation d'objets. Chaque objet contiendra ses propres caractéristiques (valeurs des propriétés) mais aura les même possibilités (méthodes) que les autres objets.

La création d'un objet se fait en deux étapes :

- D'abord, il faut créer une variable capable de référencer l'objet créé en mémoire : c'est à dire que cette variable ne contiendra que l'adresse de l'objet. Le type de cette variable est le nom de la classe qui instanciera l'objet.

```
Dim MonObjet as MaClasse
```

```
Dim moi as personne
Dim toi as personne
```

- Ensuite, créer l'objet avec l'opérateur new et stocker l'adresse retournée dans la variable

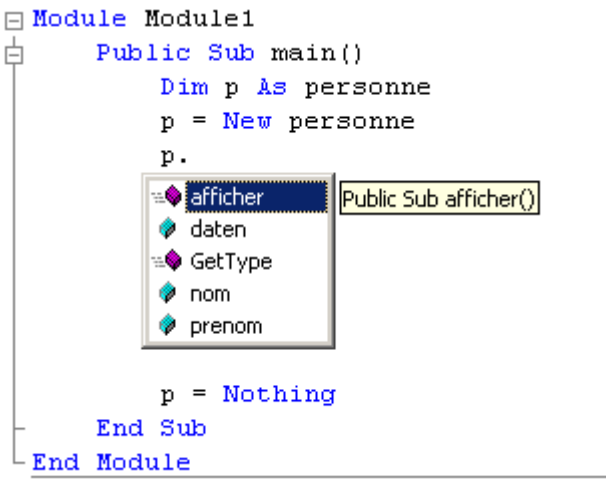
```
MonObjet = New MaClasse()
```

```
Moi = new personne()
Toi = new personne()
```

Le mot clé New effectue plusieurs opérations :

- Réservation de l'espace mémoire nécessaire au stockage de l'objet
- Retourne l'adresse mémoire de l'objet créé
- Le constructeur de la classe est appelé (Cf plus bas), généralement pour initialiser les propriétés de l'objet.

A partir de ce moment, l'objet est utilisable, c'est à dire qu'il possible de modifier ses propriétés et d'appeler ses méthodes. L'environnement .net effectue ce qu'on appel une liaison précoce, c'est à dire qu'à partir de la définition de la classe, il est capable d'afficher l'ensemble des membres disponibles :




Les liaisons précoces sont utilisables en faisant suivre le nom de l'objet par le caractère point « . ». Il est ensuite possible d'utiliser les propriétés de l'objet comme des variables et d'appeler ses méthodes.

Lorsqu'un objet n'est plus utilisé, il est important de le supprimer afin de libérer de la mémoire et, éventuellement, effectuer des opérations d'enregistrement dans une base de données par exemple via le destructeur.

Pour supprimer un objet, il suffit de placer le mot clé « nothing » à l'intérieur de la variable référençant l'objet.

```
MonObjet = Nothing
```

```
Moi = nothing
Toi = nothing
```

## 5.3 Les propriétés

Une propriété permet de définir une caractéristique d'un objet. Une propriété peut être une variable ou un objet dont la valeur sera définie pour chaque objet.

Il existe deux façon d'implémenter une propriété :

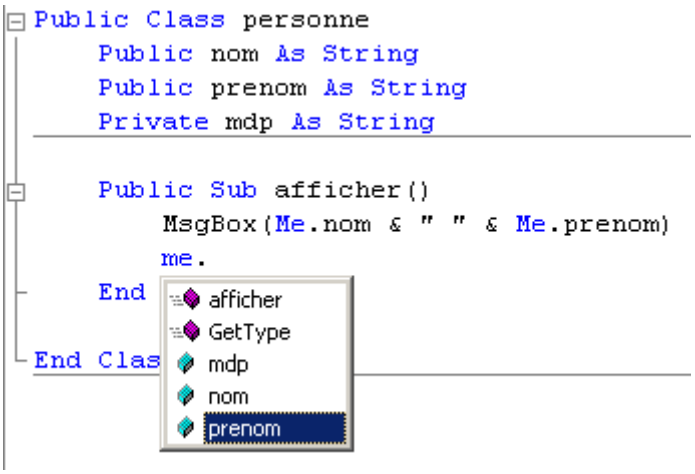
### 5.3.1 Variables

La première solution consiste à déclarer des variables directement à l'intérieur de la classe. En fonction de leur visibilités, celles ci seront accessibles uniquement à l'intérieur de la classe ou alors dans tout le programme

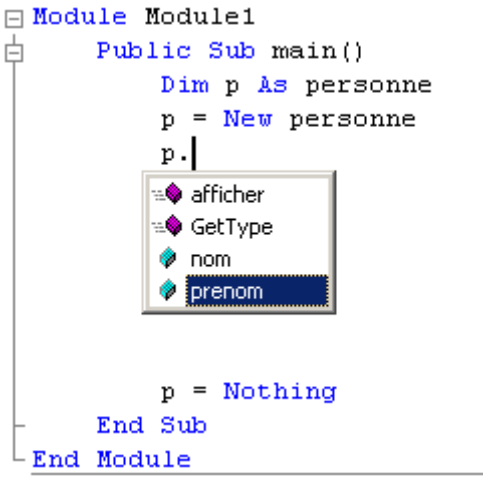
```
Public Class personne
   Public nom As String
   Public prenom As String
   Private mdp As String
```

```
    Public Sub afficher()
        MsgBox(Me.nom & " " & Me.prenom)
    End Sub

End Class
```

- Visibilité à partir de la classe :




- Visibilité à partir d'un autre module :




les propriétés ainsi créées sont accessibles en lecture / écriture :

```
dim p as personne
p = new personne()
p.nom = ''toto''
p.prenom = ''titi''
```

### 5.3.2 Procédures de propriétés

L'utilisation de variables directement déclarées dans la classe n'est pas très souple. En effet il n'est pas possible de :

- Créer des propriétés en lecture seule
- Créer des propriétés en écriture seule
- Effectuer des calculs avant de retourner la valeur d'une propriété (pour le formatage par exemple)
- Effectuer des calculs lors de la modification d'une propriété (Effectuer des vérifications de date de naissance par exemple)

Pour palier à ces problèmes, il existe les procédures de propriétés. L'idée est de lier chaque propriété à deux procédures : GET et SET. La procédure « get » sera appelée lorsque la propriété set utilisée en lecture et « set » lorsque la propriété est utilisée en écriture.
Dans tous les cas, les procédures de propriété agiront sur une variable de stockage qui contiendra la valeur de la propriété.

Pour créer les procédures de propriété, utiliser la syntaxe suivante :

```
Private variable_stockage as type_propriété

Property nom_propriété() as type_propriété
        Get
                …
        End Get

        Set (byval value as type_propriété)
                …
        End set
End property
```

La section Get est en fait une fonction devant contenir le mot clé « return » pour définir la valeur de retour de la propriété.

La section Set est une procédure prenant en paramètre « value », la valeur à assigner à la propriété.

L'exemple suivant implémente la propriété « nom ». Elle vérifie que le nom ne soit pas vide et retourne le nom en majuscule. Dans cet exemple, c'est la variable « v_nom » qui contient la valeur de la propriété.

```
Public Class personne

    Private v_nom As String = ""

    Property nom() As String
        Get
            Return Me.v_nom.ToUpper()
        End Get
        Set(ByVal Value As String)
            If Value.Length > 0 Then v_nom = Value
        End Set
    End Property

End Class
```

Il est également possible via les procédures de propriétés de créer des propriétés en lecture seule ou en écriture seule.

- Pour créer une propriété en lecture seule, il suffit d'omettre la section « set » et d'ajouter le mot clé « Readonly » devant la définition de la propriété
- Pour créer une propriété en écriture seule, il suffit d'omettre la section « get » et d'ajouter le mot clé « WriteOnly » devant la définition de la propriété

L'exemple suivant crée la propriété « mdp » en écriture seule et la propriété « daten » en lecture seule :

```
Public Class personne

    Private v_nom As String = ""
    Private v_mdp As String = ""
    Private v_daten As Date = #12/7/2002#

    Property nom() As String
        Get
            Return Me.v_nom.ToUpper()
        End Get
        Set(ByVal Value As String)
            If Value.Length > 0 Then v_nom = Value
        End Set
    End Property

    ReadOnly Property daten() As Date
        Get
            Return v_daten
        End Get
    End Property


    WriteOnly Property mdp() As String
        Set(ByVal Value As String)
            v_mdp = Value
        End Set
    End Property
End Class
```

Enfin il est possible d'utiliser les mots clés « private » et « public » pour la définition des propriétés.

## 5.4 Les Méthodes

### 5.4.1 Généralités

Les méthodes sont des procédures ou des fonctions déclarées à l'intérieur d'une classe et appelable uniquement à partir d'un objet.

```
Public Class personne

    Private v_nom As String = ""
    Private v_prenom As String = ""
    Private v_daten As Date = #12/7/2002#

    Public Sub afficher()
        MsgBox(Me.v_nom & " " & Me.v_prenom)
    End Sub

End Class
```

l'appel de la méthode se fait ensuite par l'intermédiaire d'un objet issu de la classe.

```
dim p as personne
p = new personne()
p.afficher()
```

### 5.4.2 Constructeur et Destructeur

Les constructeur et destructeur sont 2 méthodes spécifiques dont l'appel est effectué automatiquement respectivement lors de la création et lors de la destruction d'un objet.
Le constructeur est généralement utilisé pour initialiser les propriétés d'une classe tandis que le destructeur permet généralement la sauvegarde des données dans une base par exemple.

#### *5.4.2.1 Constructeur*

Le constructeur se doit d'être « Public » afin qu'il puisse être appelé lors de l'instanciation des objets.

```
Public Sub New()

End Sub
```

Le constructeur suivant permet d'initialiser les propriétés de personne

```
Public Sub New()
      v_nom = "toto"
      v_prenom = "titi"
      v_daten = #12/7/2002#
End Sub
```

Il est également possible de passer des paramètres à un constructeur. Le constructeur suivant prend en paramètre le nom de la personne a créer.

```
   Public Sub New(ByVal lenom As String)
      v_nom = "toto"
      v_prenom = "titi"
      v_daten = #12/7/2002#
   End Sub
```

C'est lors de la création de l'objet que la valeur du paramètre sera définie :

```
Dim p As personne
p = New personne("toto")
```

#### *5.4.2.2 Destructeur*

Le destructeur est lui appelé automatiquement lors de la destruction de l'objet. Il permet généralement de sauvegarder des données ou de supprimer des objets liés en mémoire.

```
Protected Overrides sub Finalize()
        …
end sub
```

Le mot clé « Overrides » définit une réécriture de fonction. Nous détaillerons ce point plus tard.

### 5.4.3 La surcharge

La surcharge de méthode permet de définir plusieurs fois la même méthode mais avec des paramètres différents. Seuls les paramètres peuvent être modifiés dans une surcharge, pas le type de retour. Pour surcharger une méthode, il faut utiliser le mot clé « OverLoads ».

```
Overloads sub nom_procedure(paramètres)

End sub
```

```
Overloads function convertir (byval valeur as integer) as double
        Return valeur / 100 * 19.6
End function

Overloads function convertir (byval valeur as integer, byval taux as double) as double
        Return valeur / 100 * taux
End function
```

Ainsi, deux appels sont possibles :

```
Msgbox convertir(100)
Msgbox convertir(100, 5.5)
```

Ci dessous figure un exemple de surcharge pour le constructeur de la classe personne. Cet exemple est spécifique dans le sens où pour les constructeur, il ne faut pas utiliser le mot clé overload.

```
Public Sub New(ByVal lenom As String)
      v_nom = lenom
      v_prenom = "titi"
      v_daten = #12/7/2002#
End Sub

Public Sub New(ByVal lenom As String, ByVal leprenom As String)
      v_nom = lenom
      v_prenom = leprenom
```

```
        v_daten = #12/7/2002#
    End Sub

Public Sub New(ByVal lenom As String, ByVal leprenom As String, ByVal ladaten As Date)
        v_nom = lenom
        v_prenom = leprenom
        v_daten = ladaten
End Sub
```

Ainsi, il est possible d'appeler tel ou tel constructeur lors de la création de l'objet :

```
Dim p, q, r As personne
p = New personne("toto")
q = New personne("titi", "momo")
r = New personne("tutu", "mimi", #12/7/2002#)
```

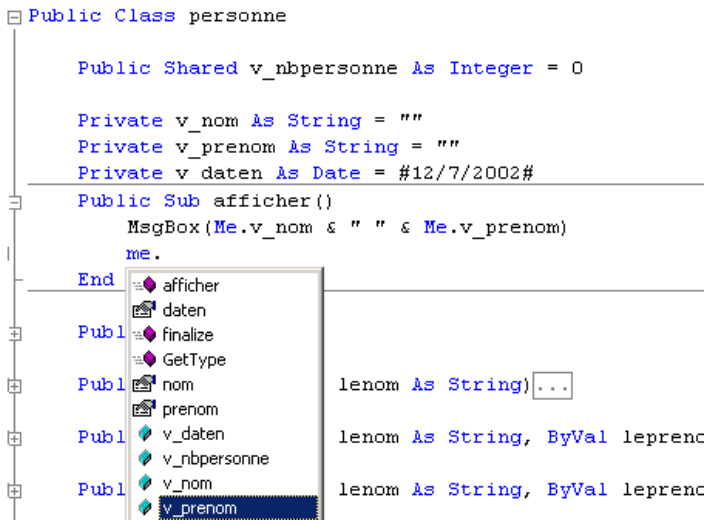
### 5.4.4 L'objet Me

Comme dans la version précédente de Visual Basic, l'objet « Me » est l'objet courant. Lorsque vous écrivez une méthode, vous pouvez faire référence directement aux propriétés de l'objet :

```
Return nom
```

Le compilateur sait alors que vous faîtes référence à la propriété de l'objet courant.

Pour clarifier le code, lever des ambiguités entre plusieurs variables portant le même nom ou profiter du menu de saisie automatique, vous pouvez faire référence à l'objet « Me » qui lors de l'exécution fera référence à l'objet ayant appelé la méthode.




## 5.5 L'Héritage

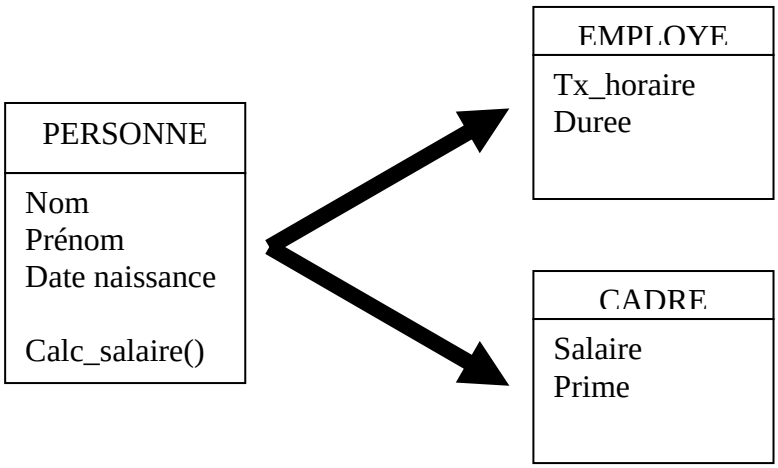
### 5.5.1 Introduction

L'héritage est une fonctionnalité fondamentale de la programmation objet. Il permet d'importer des classes déjà existantes à l'intérieur d'autres classes. L'idée est de réutiliser les composants déjà programmés sans avoir à copier/coller le code ni créer d'autres objets à l'intérieur de la classe.

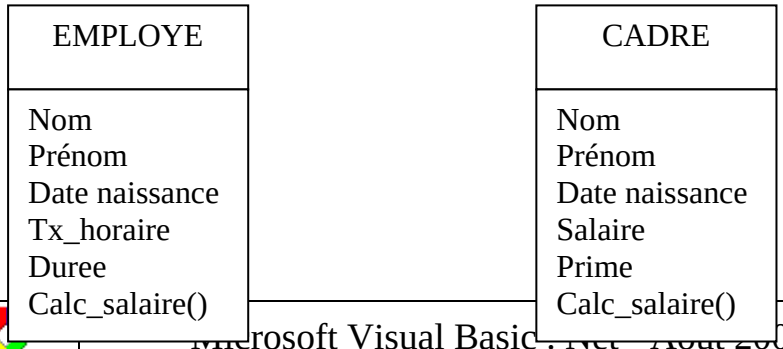
Deux acteurs sont présents lors de l'héritage :

- Super classe (ou classe mère: c'est la classe qui contient les propriétés et méthodes qui seront ensuite réutilisée dans les sous classes. De manière générale, la super classe contient les membres communs aux sous classes.
- Sous classe (ou classe fille) : c'est la classe qui hérite des membres de la super classe. Elle ajoutera ses propres membres qui en feront une classe spécialisée.

Reprenons par exemple notre classe personne. Elle contient des propriétés et des méthodes génériques quelque soit la personne. Cependant, les personnes sont subdivisées en 2 catégories : les employés qui ont un taux horaire et une durée de contrat, les cadres qui ont un salaire fixe plus un montant de primes. Dans ce cas, les employés comme les cadres auront un nom, prénom et date de naissance. Ainsi, les propriétés communes sont regroupées dans la super classe (Personne) tandis que les propriétés spécifiques comme le taux horaire ou le montant des primes seront eux stockés dans la sous classe.




Il sera possible (dans certaines conditions) d'instancier la classe PERSONNE. Cependant, lorsque nous instancierons les classes EMPLOYE ou CADRE, nous obtiendrons réellement les classes suivantes :



L'intérêt de l'héritage est donc de définir les éléments communs une seule fois et de pouvoir les réutiliser dans plusieurs autres classes.

### 5.5.2 Mise en place

Pour définir qu'une classe hérite d'une autre classe, vous devez utiliser le mot clé « Inherits » dans la définition de la sous classe. Attention, la classe que vous utilisez en tant que super classe ne doit pas posséder l'option « NotInheritable », auquel cas vous ne pourrez hériter de ses membres.

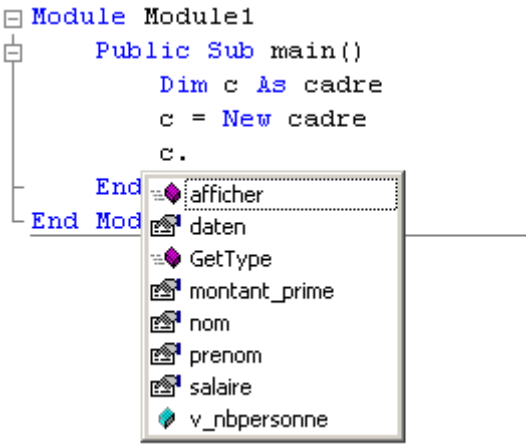
```
Public class sous_classe
        Inherits super_classe
        …
End class
```

Dans l'exemple suivant, la classe cadre hérite des membres de la classe personne.

```
Public Class cadre
   Inherits personne
   Private v_salaire As Double
   Private v_montant_prime As Double


   Public Property montant_prime() As Double
      Get
         Return v_montant_prime
      End Get
      Set(ByVal Value As Double)
         v_montant_prime = Value
      End Set
   End Property


   Public Property salaire() As Double
      Get
         Return v_salaire
      End Get
      Set(ByVal Value As Double)
         v_salaire = Value
      End Set
   End Property

   Public sub new()
        Me.nom = ‘’’
```

```
        Me.prenom = ‘’’
        Me.daten = #12/07/2002#
        Me.salaire = 0
        Me.montant_prime = 0
    End sub

End Class
```

Ainsi, lors de l’instanciation de la classe cadre, nous retrouvons l’ensemble des propriétés :




### 5.5.3 Objet MyBase

L’objet MyBase est utilisable uniquement dans les sous classes. Il fait référence aux super classes.

Si nous reprenons notre exemple précédent, le constructeur de la sous classe « cadre » fait directement référence aux propriétés de la super classe « personne ». Dans le cas ou rien n’est précisé, le compilateur appellera le constructeur de la super classe par défaut, c’est à dire celui qui ne possède aucun paramètre.

Si vous souhaitez spécifié le constructeur, vous devez l’appeler de manière explicite dans le corps du constructeur de la sous classe :

```
Public Sub New()
    MyBase.New("tyty", "marine")
    Me.salaire = 0
    Me.montant_prime = 0
End Sub
```

### 5.5.4 Le remplacement

Le remplacement est une autre fonctionnalité de l’héritage qui permet de réécrire le comportement d’une méthode déclarée dans la super classe. L’intérêt est de pouvoir récupérer

les éléments déjà définis d'une classe sans pour autant être bloqué par le comportement d'une fonction qui ne conviendrait pas à notre programme.

Par exemple, nous allons modifier le code de telle façon à ce que, lorsque nous appelons la méthode « afficher » à partir d'un objet de type cadre, celle ci affiche le nom, prénom et salaire fixe. Cependant, le fonctionnement de la méthode ne sera pas modifié pour les objets de type « Personne » ou « Employe ».

Pour remplacer la fonction « afficher », nous devons la réécrire dans la sous classe cadre et utiliser le mot clé « overrides ». Attention, la signature des deux classes doit être identique (paramètres et valeur de retour).

```
Overrides sub procedure_de_la_super_classe()
```

```
Overrides Sub afficher()
   MsgBox(Me.salaire)
End Sub
```

Quelques contraintes doivent cependant être respectées afin de mener à bien le remplacement :

- La méthode de remplacement doit avoir la même signature que la méthode remplacée.
- La méthode remplacée (dans la super classe) doit comporter l'option « Overridable »

Dans le cas d'une méthode de la super classe comportant l'option « Mustoverride », le remplacement est obligatoire.

## 5.6 Méthodes et variables partagées

Jusqu'ici, nous avons vu que les méthodes et les propriétés n'étaient utilisables que par l'intermédiaire d'objet instanciés. Les méthodes et variables partagées sont elles accessibles à partir de la classe et non d'un objet instancié. Ainsi, dans le cas d'une variable partagée, nous avons à faire à une globale au niveau de la classe.

Pour déclarer une propriété ou une méthode partagée, il suffit de rajouter le mot clé « shared » (partagé ☺) devant le nom du membre :

```
Public shared nom_variable as type_variable
Public shared function nom_fonction () as type_fonction
Public shared sub nom_procédure ()
```

Ces variables et procédures seront ensuite accessibles soit via un objet, soit via le nom de la classe.

L'exemple suivant crée une variable partagée contenant le nombre d'objet instanciés à partir de la classe « personne ». Nous créons également la propriété « nb_personne » retournant la valeur de v_nbpersonne.

```
Public class personne
        Private shared v_nbpersonne as integer = 0

        Sub New()
                v_nbpersonne++
        End Sub

        Public shared ReadOnly nb_personne() as integer
                Get
                        Return v_nbpersonne
                End get
        End sub

End class
```

A chaque instanciation d'objet, le constructeur « new » incrémente la variable v_nbpersonne.

Ensuite, pour accéder à la propriété nb_personne, il est possible d'utiliser un objet ou directement la classe :

```
Dim p as personne
P = new personne
Msgbox (p.nb_personne)
Msgbox (personne.nb_personne)
```

## 5.7 Les Interfaces

Une interface est une classe contenant des propriétés et dont les méthodes possèdent toutes l'option « MustOverride », c'est à dire qu'elles ne contiennent aucun code.
Les interfaces sont utilisées lorsque plusieurs classes doivent implémenter la même méthode.
L'avantage des interfaces est que le codeur s'engage à fournir le comportement de toutes les méthodes liées à l'interface (Le compilateur sera là pour le vérifier).
Il n'est pas possible d'instancier directement une classe.

### 5.7.1 Création

La création d'une interface ressemble à celle d'une classe, elle est définie par les mots clé « interface » et « end interface ».

```
Public interface nom_interface
        …
End interface
```

Eléments susceptibles d'intervenir dans une interface

- Déclaration de variables
- Héritage d'une autre interface
- Synopsis des méthodes

Eléments ne devant pas apparaître dans une interface

- mots clé « end xxx »
- lignes de code, opérations, appels…

Dans notre cas nous allons définir l'interface « comparer » implémentant la méthode « plus_grand » prenant en paramètre un Objet.

```
Public Interface comparer
        Function plus_grand(obj as object) as boolean
End Interface
```

### 5.7.2 Utilisation

Pour utiliser une interface, il suffit de l'implémenter dans une classe et de définir le comportement des différentes méthodes. Plusieurs éléments doivent apparaître dans la classe implémentant l'interface :

- Le mot clé « implements nom_interface » afin de préciser que la classe va implémenter l'interface
- Le mot clé « implements nom_interface.nom_méthode » après les méthodes qui définissent le comportement d'une méthode de l'interface.

Dans l'exemple suivant, la classe employé va implémenter l'interface comparer. Nous nous baserons sur l'age de la personne pour effectuer la comparaison.

```
Public Class personne

        Implements comparer
        Public Function plus_grand(ByVal obj As Object) As Boolean Implements _
comparer.plus_grand
                If Me.daten < obj.daten Then
                        Return True
                Else
                        Return False
                End If
        End Function
…
```

Il est maintenant possible d'utiliser la méthode à partir des objets « personne » :

```
Public Sub main()
   Dim a, b As personne
   a = New personne("toto", "titi", #1/1/1901#)
   b = New personne("vuvu", "baba", #1/1/1951#)
   MsgBox(a.plus_grand(b))          'affiche true
End Sub
```

### 5.7.3 Exploiter les interfaces .Net

Jusqu'ici, nous avons créé nos propres interfaces. Le framework .net met à disposition plusieurs interfaces dont le développeur doit savoir tirer avantage pour simplifier la programmation.

Nous baserons notre exemple sur l'interface Icomparable qui permet ensuite d'utiliser la méthode « Sort » de la classe « Array ». Cette interface ne comporte qu'une seul méthode « CompareTo » qui prend en paramètre un objet et retourne –1, 0 ou 1 selon que l'objet en cours est inférieur, égal ou supérieur à l'objet passé en argument. A partir de cette méthode de Tri, la méthode « Sort » sera capable de trier tous types d'objets, à condition qu'ils implémente l'interface.

Plus simplement, nous devons définir dans la classe « personne » la méthode « CompareTo » qui implémente l'interface « Icomparable » et plus particulièrement la méthode « CompareTo ». Ainsi, lorsque nous appellerons la méthode « Sort » de la classe « Array », le compilateur utilisera notre méthode afin de comparer les différents Objets et classer notre tableau.

L'intérêt des Interfaces est de pouvoir utiliser des méthodes déjà définies et de modifier leur comportement afin qu'elles puissent être utilisées avec des classes utilisateurs.

#### *5.7.3.1 Implémenter l'interface*

Pour implémenter l'interface, il suffit de la déclarer et d'écrire le comportement de la méthode « CompareTo » :

```
Public Class personne

   Implements comparer
   Implements IComparable

Private Function CompareTo(ByVal obj As Object) As Integer Implements _
IComparable.CompareTo

      If obj Is Nothing Then Return 1

      Dim autre As personne
      autre = CType(obj, personne)

      If Me.nom < autre.nom Then
         Return -1
      ElseIf Me.nom > autre.nom Then
         Return 1
      Else
         Return 0
      End If

   End Function
```

```
….
```

#### *5.7.3.2 Utiliser l'interface*

Une fois l'interface implémentée, nous pouvons utiliser la méthode « Sort » avec un tableau d'objets de type « personne » :

```
Sub main()

    Dim t_personne(2) As personne
    Dim p As personne

    t_personne(0) = New personne("toto")
    t_personne(1) = New personne("titi")
    t_personne(2) = New personne("tata")

    Array.Sort(t_personne)

    For Each p In t_personne
        MsgBox(p.nom)
    Next

End Sub
```

Cet exemple affiche la liste des personne  triées grâce à l'appel de la méthode « Sort ».

## 5.8 La délégation

La délégation est un mécanisme qui permet de définir une variable pointant sur une fonction. En réalité, cette variable contiendra l'adresse de la fonction ou de la procédure.

L'intérêt de la délégation est de permettre à tous moments de modifier le comportement du programme en le faisant pointer sur une autre méthode.

Dans les exemples suivants, nous utiliserons les deux fonctions suivantes :

```
Public function ajouter(byval x as integer, byval y as integer) as integer
        Return (x + y)
End function

Public function multiplier (byval x as integer, byval y as integer) as integer
        Return (x * y)
End function
```

### 5.8.1 Signature de la méthode

Afin de pouvoir stocker une ou plusieurs références vers des fonctions, il faut définir un type de donnée correspondant à la signature des fonctions qui seront appelées. Nous déclarerons ensuite une variable de ce type qui contiendra l'adresse de la fonction déléguée.

```
Public delegate function nom_type (parametre) as type_retour
```

Nous allons créer le type calcul qui sera un pointeur vers fonction :

```
Public delegate function calcul (byval x as integer, byval y as integer) as integer
```

Une fois le type défini, il faut lui assigner une valeur à l'aide de l'opérateur « Address Of »

```
Calcul = addressof multiplier
```

A ce moment, le type calcul fait référence à la fonction « multiplier ».

#### 5.8.2 Appel du Delegate

Une fois le type faisant référence à la fonction, il suffit d'appeler la méthode « invoke » afin d'exécuter la fonction.

```
Nom_type.invoke(paramètres)
```

```
Msgbox calcul.invoke(1,3)          'affiche 3
```

### 5.9 Les évènements

Les évènements sont des méthodes qui seront automatiquement appelées par les objets afin de prévenir d'un état donné. Il est également possible de définir des paramètres à l'événement : ces derniers permettront de préciser les conditions dans lesquelles il se déclenche.

#### 5.9.1 Création

La création d'un événement permet d'en définir ses paramètres ainsi que les conditions dans lesquelles il sera déclenché. C'est l'utilisateur de la classe qui définira ensuite les actions à réaliser lorsque l'événement survient.

Pour créer un événement, il faut tout d'abord le déclarer dans la classe :

```
Public event nom_evenement(paramètres)
```

Dans l'exemple suivant, nous créons un événement lorsque le montant des primes d'un cadre dépasse 10 000.

```
Public event depassement_plafond_prime (byval montant as double)
```

Une fois l'événement défini, il faut placer à l'intérieur de la classe les instructions qui permettront de le déclencher. Pour cela, on utilise le mot clé « RaiseEvent » en précisant les valeurs des différents paramètres attendus par l'événement.

```
Raiseevent nom_evenement (valeur_parametre)
```

Dans notre cas, nous devons placer le code nécessaire lors de la modification de la propriété « montant_salaire ». Pour cela nous allons placer le code dans la procédure de propriété « Set » de « montant_prime ».

```
Public Property montant_prime() As Double
   Get
      Return v_montant_prime
   End Get
   Set(ByVal Value As Double)
      v_montant_prime = Value
      If v_montant_prime > 10000 Then
         RaiseEvent depassement_plafond_prime(v_montant_prime)
      End If
   End Set
End Property
```

### 5.9.2 Utilisation

Il existe deux méthodes pour traiter les évènements : soit en utilisant le mot clé « withEvents », soit en utilisant un gestionnaire d'évènements.

#### *5.9.2.1 Utilisation de With Events*

La première solution consiste à utiliser le mot clé « WithEvents » lors de la déclaration de l'objet :

```
Dim WithEvents nom_variable as Classe
```

```
Dim withEvents violaine as cadre
```

Une fois la variable créée, vous devez définir le bloc de code à exécuter lors du déclenchement de l'événement : pour cela, placer le code dans la procédure d'événement correspondant. Pour qu'une procédure soit déclenchée lorsqu'un événement survient, vous devez ajouter à la fin le mot clé « handles » suivi du nom de la classe et de l'événement

```
Public sub nom_procedure(parametre) handles objet.nom_evenement
        …
End sub
```

Dans notre cas, nous afficherons un simple message :

```
Public sub violaine_depassement_plafond_prime(byval montant as double) handles violaine.
depassement_plafond_prime
        Msgbox(''Attention, le montant des primes s'élève à '' & montant)
End sub
```

Par convention, le nom de la procédure est le nom de l'objet suivi de l'événement mais il n'y à aucune obligation : l'exemple suivant fonctionne très bien.

```
Public sub toto(byval montant as double) handles violaine.depassement_plafond_prime
        Msgbox(''Attention, le montant des primes s'élève à '' & montant)
End sub
```

L'utilisation du mot clé « withevents » comporte cependant quelques limitations car les variables déclarées avec l'option WithEvents ne doivent pas se trouver dans des procédures ou fonctions et il est impossible de modifier dynamiquement le comportement d'un événement en le faisant pointer sur une autre procédure par exemple.

#### *5.9.2.2 Utilisation du gestionnaire d'événement*

La seconde méthode utilise un gestionnaire d'événement : il n'est plus nécessaire d'utiliser le mot clé « WithEvents » et il faudra associer dynamiquement une procédure pour gérer l'événement. Pour cela, on utilise le mot clé « AddHandler » prenant en paramètre l'événement et la procédure.

```
AddHandler objet.evenement, adresseOf procedure
```

Dans l'exemple suivant, nous avons 2 procédure qui affiche le montant des primes en mode texte et en mode graphique.

```
Public sub affiche_txt(byval montant as double)
        Console.writeline (montant)
End sub

Public sub affiche_graph(byval montant as double)
        Msgbox (montant)
End sub
```

Nous créons ensuite deux objets de type cadre (c1 et c2) et nous lions à chaque objet une procédure différente pour la gestion de l'évènement depassement_plafond_prime.

```
Dim c1, c2 as Cadre

C1 = new Cadre()
C2 = new Cadre()
```

```
AddHandler c1.depassement_plafond_prime, addressOf affiche_txt
AddHandler c2.depassement_plafond_prime, addressOf affiche_graph
```

Ainsi, le traitement des événements sera différent selon l'objet utilisé.

A l'inverse, pour supprimer un gestionnaire d'événement mis en place, il faut utiliser le mot clé « RemoveHandler »

```
RemoveHandler objet.événement, AddressOf procédure
```

```
RemoveHandler c1.depassement_plafond_prime, AddressOf affiche_txt
```

Si aucune procédure n'est liée à l'événement, ce dernier sera tout simplement ignoré.

## 5.10 Classes d'exemple

### 5.10.1 Classe personne

```
Public Class personne

    Implements comparer
    Implements IComparable

    Private Function CompareTo(ByVal obj As Object) As Integer Implements
IComparable.CompareTo

        If obj Is Nothing Then Return 1

        Dim autre As personne
        autre = CType(obj, personne)

        If Me.nom < autre.nom Then
            Return -1
        ElseIf Me.nom > autre.nom Then
            Return 1
        Else
            Return 0
        End If

    End Function

    Public Function plus_grand(ByVal obj As Object) As Boolean Implements
comparer.plus_grand
        If Me.daten < obj.daten Then
            Return True
        Else
```

```
        Return False
      End If
   End Function

   Public Shared v_nbpersonne As Integer = 0

   Private v_nom As String = ""
   Private v_prenom As String = ""
   Private v_daten As Date = #12/7/2002#




   Public Overridable Sub afficher()
      MsgBox(Me.v_nom & " " & Me.v_prenom)
   End Sub

   Public Sub New()
      v_nom = "toto"
      v_prenom = "titi"
      v_daten = #12/7/2002#
      v_nbpersonne = v_nbpersonne + 1
   End Sub

   Public Sub New(ByVal lenom As String)
      v_nom = lenom
      v_prenom = "titi"
      v_daten = #12/7/2002#
      v_nbpersonne = v_nbpersonne + 1
   End Sub

   Public Sub New(ByVal lenom As String, ByVal leprenom As String)
      v_nom = lenom
      v_prenom = leprenom
      v_daten = #12/7/2002#
      v_nbpersonne = v_nbpersonne + 1
   End Sub

   Public Sub New(ByVal lenom As String, ByVal leprenom As String, ByVal ladaten As
Date)
      v_nom = lenom
      v_prenom = leprenom
      v_daten = ladaten
      v_nbpersonne = v_nbpersonne + 1
   End Sub

   Protected Overrides Sub finalize()
      MsgBox("Destruction de l'objet personne")
   End Sub
```

```
    Public Property nom() As String
        Get
            Return v_nom
        End Get
        Set(ByVal Value As String)
            v_nom = Value
        End Set
    End Property

    Public Property prenom() As String
        Get
            Return v_prenom
        End Get
        Set(ByVal Value As String)
            v_prenom = Value
        End Set
    End Property

    Public Property daten() As String
        Get
            Return v_daten
        End Get
        Set(ByVal Value As String)
            v_daten = Value
        End Set
    End Property

End Class
```

### 5.10.2 Classe Cadre

```
Public Class cadre
    Inherits personne
    Private v_salaire As Double
    Private v_montant_prime As Double

    Public Event depassement_plafond_prime(ByVal montant As Double)

    Public Property montant_prime() As Double
        Get
            Return v_montant_prime
        End Get
        Set(ByVal Value As Double)
            v_montant_prime = Value
            If v_montant_prime > 10000 Then
                RaiseEvent depassement_plafond_prime(v_montant_prime)
```

```
        End If
      End Set
   End Property


   Public Property salaire() As Double
      Get
         Return v_salaire
      End Get
      Set(ByVal Value As Double)
         v_salaire = Value
      End Set

   End Property

   Public Sub New()
      MyBase.New("tyty", "marine")
      Me.salaire = 0
      Me.montant_prime = 0
   End Sub

   Overrides Sub afficher()
      MsgBox(Me.salaire)
   End Sub
End Class
```

# 6 Applications Windows

Le FrameWork Visual Basic .Net permet la création de formulaires Windows afin d'établir des interfaces graphiques entre l'utilisateur et le code. Ces formulaires sont des fenêtre qui contiendront des contrôles (Champs texte, boutons, liste déroulantes ….).

## 6.1 Les formulaires

les formulaires sont les éléments de base des applications graphiques Windows.

### 6.1.1 Différents types

Il existe 2 solutions pour la création de formulaires sous le FrameWork .Net :

#### *6.1.1.1 Windows Forms*

Ce sont les formulaires dont disposait Visual basic 6. Les applications basées sur ces formulaires sont utilisés pour le développement d'applications pour lesquelles la plupart des traitements se font sur la machine cliente et qui ont besoin d'accéder aux ressources de la machine (fichiers, lecteurs, imprimantes …).

#### *6.1.1.2 Web forms*

Les applications à base de Web Forms sont destinés à être utilisées sur le Web par le biais d'un navigateur. Ce genre d'application présente plusieurs avantages comme un déploiement facile dans le sens où seul les composants du navigateur doivent être installés, une maintenance simplifiée car le programme est stocké sur le serveur et, enfin, les applications développées sont indépendantes de toutes plateformes dans le sens où elles n'utilisent que les ressources du navigateur.

#### *6.1.1.3 Modes de présentation*

En fonction de l'application à réaliser, plusieurs modes de présentation des feuilles peuvent être utilisés :

- Mono document : Ce genre d'application appelée SDI (Single Document Interface) ne permet l'affichage que d'une fenêtre à la fois. L'outil Paint en est un bon exemple.

- Multi document : Les applications MDI (Multiple Document Interface) sont constituées d'une fenêtre principale (Fenêtre mère) contenant à son tour plusieurs documents (fenêtre fille). Microsoft Word est une application MDI.

- Explorateur : C'est le mode de présentation le plus utilisé. Il permet un affichage hiérarchique des menus sur la partie gauche et l'affichage des éléments sous forme de liste sur la partie droite. L'outil « Gestion de l'ordinateur » sous Windows 2000 en est un exemple.

### 6.1.2 Membres de la classe Form

Nous l'avons vu au chapitre 4, un formulaire à sa création contient déjà plusieurs lignes de code. Maintenant que nous connaissons la programmation orientée objet ainsi que ces concepts, voyons d'un peu plus près son contenu :

- Héritage de la classe « system.windows.forms.form »

```
Inherits System.Windows.Forms.Form
```

- Définition du constructeur « new » avec notamment l'appel du constructeur de MyBase (donc de windows.forms)

```
Public Sub New()
    MyBase.New()
    InitializeComponent()
End Sub
```

- Surcharge de méthode Dispose afin de « nettoyer » les composants ajoutés par l'utilisateur

```
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub
```

- Ajout de la propriété « components » donnant au formulaire toutes les caractéristiques pour ensuite contenir des contrôles

```
Private components As System.ComponentModel.IContainer
```

- Définition des Objets placé sur le formulaire. (remarquez le mot clé withevents permettant la gestion des événements). Dans cet exemple, l'élément ajouté est un champs texte.

```
Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
```

- Ajout de la méthode « InitializeComponent » dont le rôle est d'initialiser les différentes propriétés du formulaire et des contrôles. Dans l'exemple suivant, nous avons rajouté un champs texte.

```
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.TextBox1 = New System.Windows.Forms.TextBox
    Me.SuspendLayout()
    '
    'TextBox1
    '
    Me.TextBox1.Location = New System.Drawing.Point(112, 72)
    Me.TextBox1.Name = "TextBox1"
    Me.TextBox1.Size = New System.Drawing.Size(128, 20)
    Me.TextBox1.TabIndex = 0
    Me.TextBox1.Text = "toto"
    '
    'Form1
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(292, 273)
    Me.Controls.Add(Me.TextBox1)
    Me.Name = "Form1"
    Me.Text = "Form1"
    Me.ResumeLayout(False)

End Sub
```

#### *6.1.2.1 Propriétés*

- AcceptButton

Lorsque l'utilisateur appuie sur la touche entrée, la méthode liée à l'événement « click » du bouton d'acceptation sera automatiquement déclenchée. Généralement, c'est le bouton « ok » ou « sauvegardé » qui est paramétré comme AcceptButton.

- AllowDrop

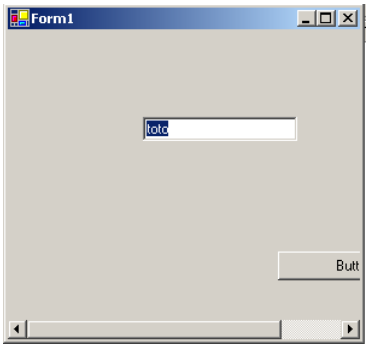
Spécifie si le formulaire gère le Drag and Drop (Glisser déposé).

- AutoScale

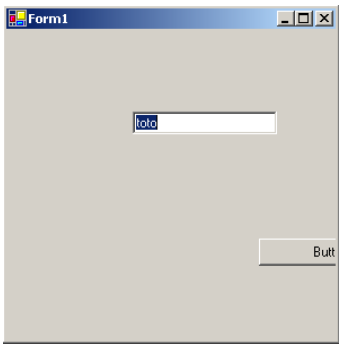
Si cette option est activée, la taille des contrôles et du formulaire sont automatiquement redimensionnés lors d'un changement dynamique de la police d'écran.

- AutoScroll

L'option AutoScroll est une nouveauté de la version .Net : elle permet de placer automatiquement des barres de défilement lorsque la taille du formulaire ne permet pas l'affichage de tous les contrôles qu'il contient.




*Avec AutoScroll*




*Sans AutoScroll*

- BackColor

La propriété backColor définit la couleur de fond du formulaire.

- BackgroundImage

Il est possible de définir une image comme fond pour le formulaire. L'image sera automatiquement répétée en mosaique.

- CancelButton

Le bouton d'annulation réalise l'opération inverse du bouton d'acceptation. Il permet déclencher l'événement « click » d'un bouton du formulaire lorsque l'utilisateur appuie sur touche escape.

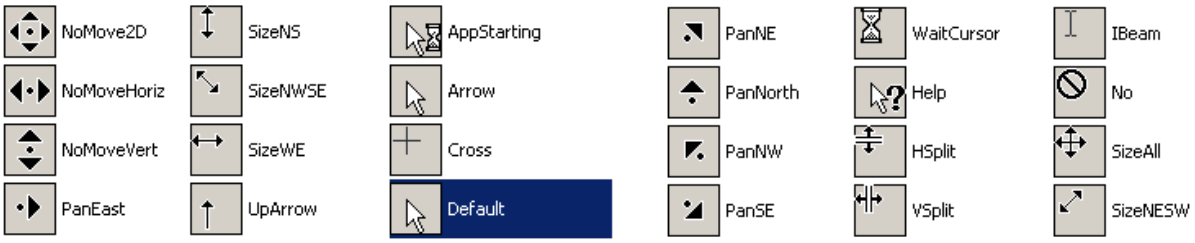
- ControlBox

La propriété ControlBox définit si le menu système apparaît au niveau de la barre de titre du formulaire :


Le menu système peut également être modifié avec les propriétés « MinimizeBox », « MaximizeBox » et « HelpButton ».

- Cursor

Définit l'apparence par défaut du curseur sur le formulaire. Cette option peut également être paramétrée au niveau des contrôles.




Les différentes valeurs sont disponibles dans la classe « System.Windows.Forms.Cursors »

- Enabled

Définit si le formulaire est disponible (True) ou non (False). Dans ce dernier cas, aucun des contrôles et menus du formulaires ne seront accessibles (grisés).

- Font

Cette propriété définit les paramètres de formatage du texte. Cette propriété sera automatiquement appliquée par défaut au texte des différents contrôles. Cette propriété est elle même décomposée en plusieurs autres propriétés :

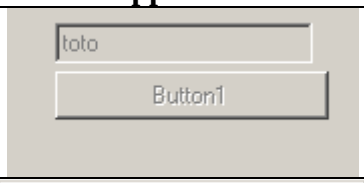
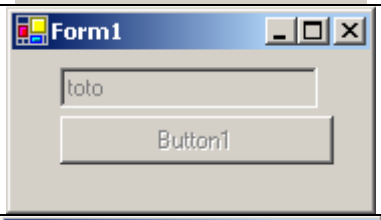
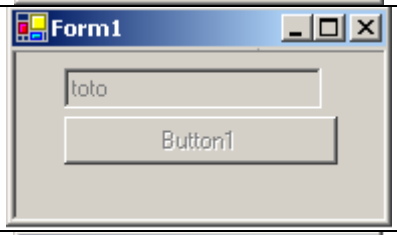
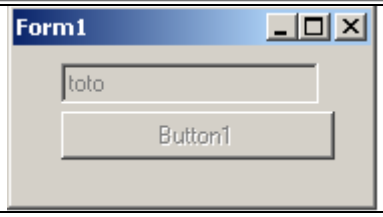
| Propriété | Description | Exemple |
|---|---|---|
| Name | Nom de la police utilisée | Garamond |
| Size | Taille de la police | 12.5 |
| Unit | Unité de mesure pour la police (Sachez que la plupart des logiciels utilisent l'unité « point ») | Point |
| Bold | Texte en gras | True |
| GdiXXXX | Paramètres sur le jeu de caractère utilisé | |
| Italic | Texte en italique | True |
| Strikeout | Texte barré | False |
| Underline | Texte souligné | True |

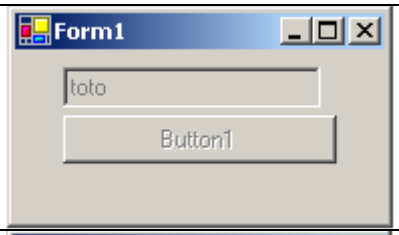
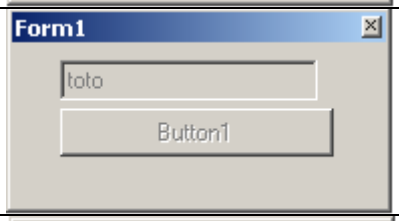
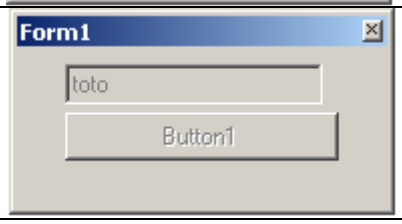
- ForeColor

Couleur d'affichage par défaut pour les textes et graphismes du formulaire.

- FormBorderStyle

Style de bordure du formulaire :

| Valeur | Apparence | Dimensionnable |
|---|---|---|
| None |  | Non |
| FixedSingle |  | Non |
| Fixed3d |  | Non |
| FixedDialog |  | Non |

| | | |
|---|---|---|
| Sizable |  | Oui |
| FixeToolWindow |  | Non |
| SizableToolWindow |  | Oui |

- HelpButton

Affiche le bouton d'aide à gauche de la barre de titre. Attention, le bouton ne sera pas affiché si les boutons min et max sont activés.


Pour déclencher un bloc d'instruction lorsque l'utilisateur demande l'aide (Soit à partir de la touche F1, soit à partir du bouton d'aide, vous devez créer une méthode implémentant l'événement :

```
Private sub nom_méthode (ByVal sender As Object, ByVal hlpevent As
System.Windows.Forms.HelpEventArgs) Handles objet.HelpRequested
```

Le code suivant permet d'afficher une boite de dialogue lorsque l'utilisateur demande l'aide sur le champs texte « text1 » qui doit être déclaré avec le mot clé « WithEvents ». La procédure suivante implémente l'événement :

```
Private Sub textBox_HelpRequested(ByVal sender As Object, ByVal hlpevent As
System.Windows.Forms.HelpEventArgs) Handles TextBox1.HelpRequested
        'converti le paramètre passé en control
    Dim requestingControl As Control = CType(sender, Control)
        'affiche le nom du controle
    MsgBox(CStr(requestingControl.name))
        'valide la gestion de l'événement
    hlpevent.Handled = True
End Sub
```

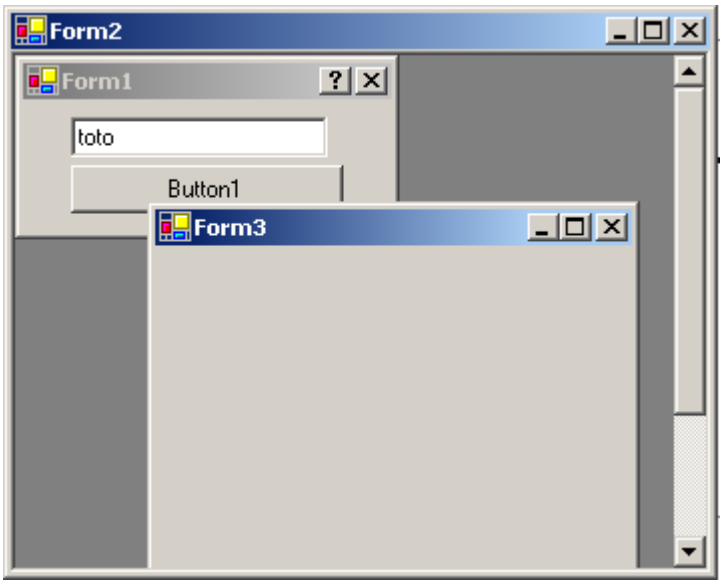
L'objet « sender » passé en paramètre référence l'objet à l'origine de la demande d'aide.

- Icon

Définit l'icône liée au formulaire : cette dernière apparaît dans la barre de titre.


- IsMDIContainer

Détermine si le formulaire est un conteneur MDI, c'est à dire s'il est capable de contenir d'autre fenêtres.




Dans le cas d'un formulaire MDI, vous devez spécifier le code afin d'afficher d'autres formulaires à l'intérieur. Le code suivant permet l'affichage d'un formulaire fils : dans cet exemple, Form2 est le formulaire MDI, Form1 et Form3 sont les formulaires enfant. Il faut également paramétré l'option « IsMdiContainer » du Form2 à True.

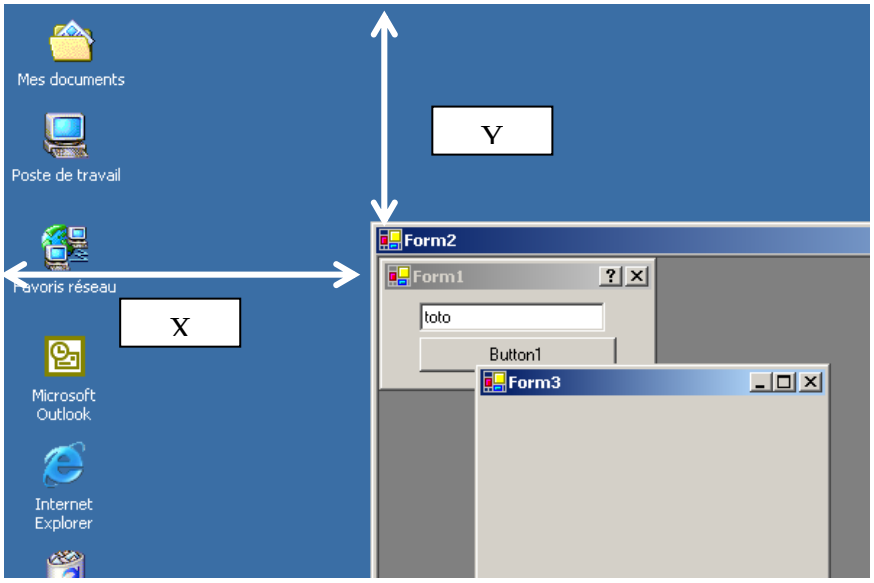
```
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    Dim f1 As New Form1
    Dim f2 As New Form3
    f1.MdiParent = Me
    f1.Show()

    f2.MdiParent = Me
    f2.Show()
```

```
End Sub
```

- Location

Cette objet défini la position du formulaire par rapport à son conteneur (c'est à dire l'écran ou le formulaire parent dans le cas d'application MDI). Deux propriétés permettent de définir la position :

- X : distance entre le bord gauche du conteneur et le bord gauche du formulaire
- Y : distance entre le haut du conteneur et le haut du formulaire




- Locked

Détermine si le formulaire est verrouillé ou non : cette propriété est identique à « enabled » mais elle ne grise pas l'apparence du formulaire.

- MinimizeBox, MaximizeBox

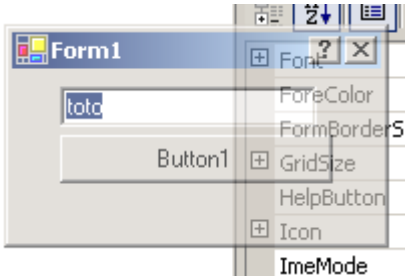
Détermine si les boutons « Agrandir » et « Réduire » sont visibles. Leur affichage empêchera l'affichage du bouton d'aide.


- MinimumSize, MaximumSize

Cet objet définit la taille minimale et maximale que peut avoir le formulaire. Cet objet est généralement utilisé pour éviter que l'utilisateur réduise la fenêtre au point de ne plus avoir accès aux contrôles. Pour chaque objet, deux propriété sont disponibles : width (largeur) et height (hauteur).

- Opacity

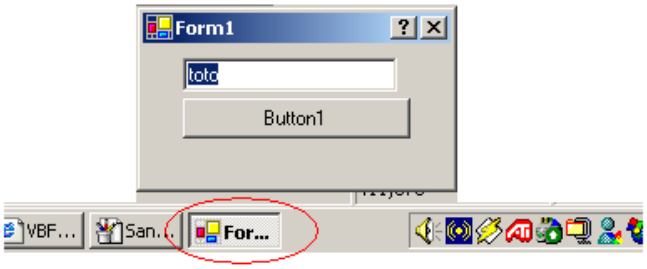
Définit un pourcentage d'opacité pour la fenêtre. Une valeur de 100% rend la fenêtre opaque.



*(Exemple avec 65%)*

- ShowInTaskBar

Détermine si un nouveau bouton est ajouté dans la barre des tâches lorsque la fenêtre est ouverte :




- Size

Cet objet définit la taille du formulaire à l'aide de deux propriétés : width (largeur) et height (hauteur).

- Startposition

Définit la position de départ lorsque la fenêtre est ouverte :

| Valeur | Description |
|---|---|
| Manual | Position définie par la propriété location |
| CenterScreen | Centré par rapport à l'écran |
| WindowsDefaultlocation | Situé à l'emplacement par défaut de Windows et possède la taille définie dans size |
| WindowsDefaultBounds | Situé à l'emplacement par défaut de Windows et possède la taille par défaut de Windows |
| Centerparent | Centré par rapport à la fenêtre ayant déclenché l'ouverture. |

- Text

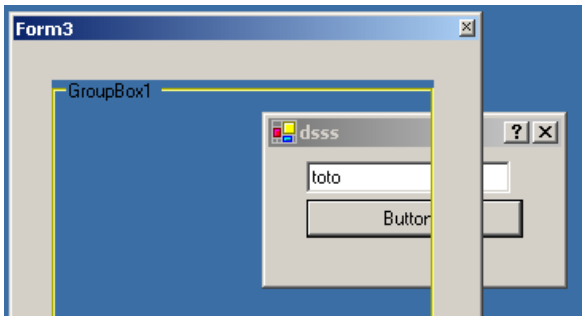
Détermine le texte affiché dans la barre de titre


- TopMost

Si cette option est activée (true) le formulaire sera toujours au dessus de tous les autres formulaires, même s'il n'est pas activé. Cette option se prête particulièrement bien pour les boîtes à outils qui doivent toujours rester accessibles.

- TransparencyKey

Du meilleur effet, cette propriété définit la couleur de transparence du formulaire : si vous spécifiez la couleur jaune en tant que TransparencyKey, toutes les zones du formulaire jaune seront transparentes :




- WindowState

Détermine l'état du formulaire lors de l'ouverture :

| Valeur | Description |
|---|---|
| Normal | Le formulaire apparaît avec sa taille standard |
| Minimize | Le formulaire est réduit lors de l'ouverture |
| Maximize | Le formulaire est en plein écran lors de l'ouverture |

#### *6.1.2.2 Méthodes*

- Activate

La méthode activate permet de mettre le formulaire au premier plan et de lui donner le focus.

- Close

Ferme le formulaire

- ShowDialog

Affiche le formulaire en tant que feuille modale, c'est à dire qu'au niveau de l'application, la fenêtre restera au premier plan tant qu'elle n'est pas fermée.

#### *6.1.2.2 Evénements*

Les événements correspondent au cycle de vie de l'objet formulaire. Ils sont listés dans l'ordre chronologique.

- New

L'objet formulaire est en cours de création

- Load

Le formulaire ainsi que ses composants sont chargés mais il n'est pas visible.

- Paint

Se produit lorsque le formulaire est redessiné. Cet événement peut apparaître plusieurs fois : par exemple au démarrage et lorsque le formulaire réapparait devant un autre.

- Activated

Le formulaire récupère le focus.

- Deactivate

Le formulaire perd le focus

- Closing

Le formulaire est en cours de fermeture, les différents éléments le composant sont détruits. Le formulaire est cependant encore visible.

- Closed

Le formulaire est fermé et maintenant invisible.

- Dispose

L'objet formulaire est détruit.

- Resize

Cet événement survient lorsque le formulaire est redimensionné. Généralement utilisé pour modifier la taille des contrôles le composant.

- Click

L'utilisateur clique sur le fond du formulaire

- DoubleClick

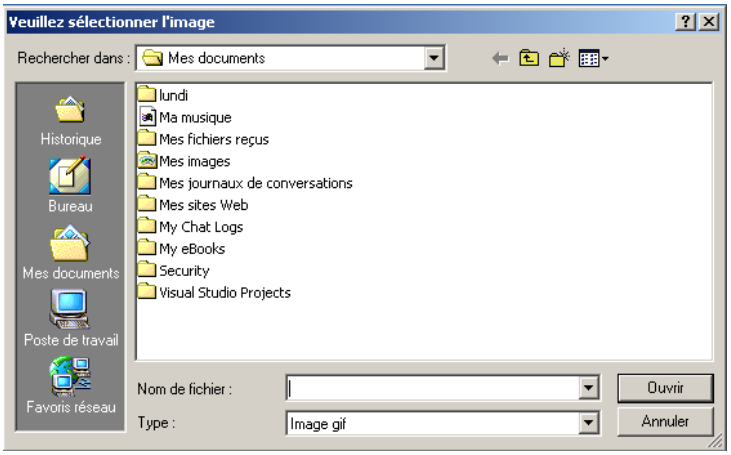
L'utilisateur double clique sur le fond du formulaire

### 6.1.3 Boîtes de dialogue

Si vous observez les différentes applications tournant sous windows, vous vous apercevrez qu'elles utilisent toutes les même boîtes de dialogue standard (Enregistrer, Ouvrir Imprimer …). Visual Basic permet l'utilisation de ces boîte de dialogue standard.

#### *6.1.3.1 Ouverture*

La boîte de dialogue d'ouverture permet la sélection d'un ou plusieurs fichiers physiques. La classe « OpenFileDialog » permet la gestion de cette boîte de dialogue.




| Propriété | Description | Valeur |
|---|---|---|
| InitialDirectory | Dossier initial | ''C:\winnt'' |
| Title | Titre de la boîte | ''Sélection du fichier'' |
| Filter | Extension des fichiers acceptés | ''tous\|*.*\|Fichier texte\|*.txt'' |
| DefaultExt | Extension par défaut | ''gif'' |
| AddExtension | Booléen indiquant si l'extension par défaut doit être automatiquement ajoutée à la fin du fichier lors de l'enregistrement | True |
| MultiSelect | Permet la sélection de plusieurs fichiers | True |
| CheckFileExist | Vérifie l'existence du fichier | True |
| FileName | Chemin du fichier sélectionné | |
| FileNames | Chemin des fichiers selectionnés | |

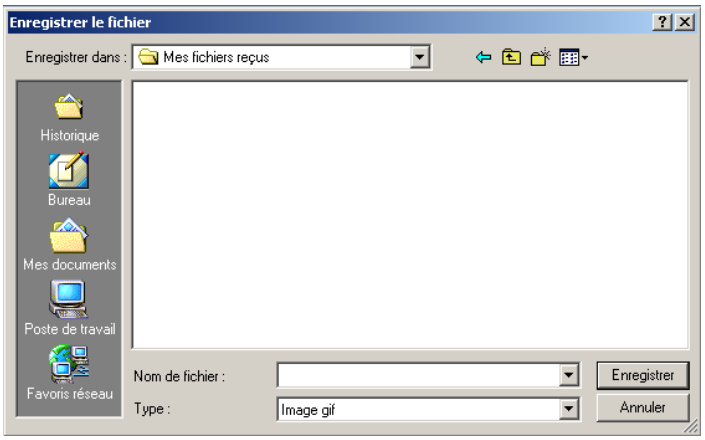
| Méthode | Description |
|---|---|
| ShowDialog | Affiche la fenêtre |

Le code suivant paramètre la boîte de dialogue en acceptant uniquement les fichiers images (gif ou jpg). La sélection multiple est autorisée et la liste des fichiers est affichée à la fin.

```
Dim dlg As OpenFileDialog
dlg = New OpenFileDialog

'paramétrage de la boîte
dlg.Title = "Veuillez sélectionner l'image"
dlg.DefaultExt = "gif"
dlg.Filter = "Image gif|*.gif|Image Jpeg|*.jpg"
dlg.Multiselect = True
dlg.CheckFileExists = True

'affichage de la boite
dlg.ShowDialog()

'affichage des fichiers selectionnés
Dim fichier As String
For Each fichier In dlg.FileNames
   MsgBox(fichier)
Next
```

#### *6.1.3.2 Enregistrement*

La boîte de dialogue d'enregistrement est identique à la boîte de dialogue d'ouverture exceptée la propriété « Méthode » qui disparaît. Pour ouvrir une boîte d'enregistrement, utiliser la classe « SaveFileDialog ».




```
Dim dlg As SaveFileDialog
dlg = New SaveFileDialog
```

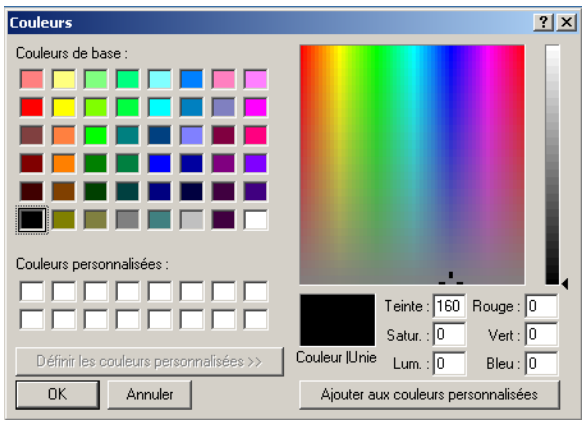
```
'paramétrage de la boîte
dlg.Title = "Enregistrer le fichier"
dlg.DefaultExt = "gif"
dlg.Filter = "Image gif|*.gif|Image Jpeg|*.jpg"

'affichage de la boite
dlg.ShowDialog()

'affichage des fichiers selectionnés
MsgBox("le fichier sera enregistré dans: " & dlg.FileName)
```

#### *6.1.3.3 Choix d'une couleur*

Cette boîte de dialogue permet à l'utilisateur de choisir une couleur dans un panel. Deux versions de cette boîte de dialogue existent :

- Version « simple »


- Version « complète »




Dans les deux cas, vous devez utiliser la classe « ColorDialog ».

| Propriété | Description |
|---|---|

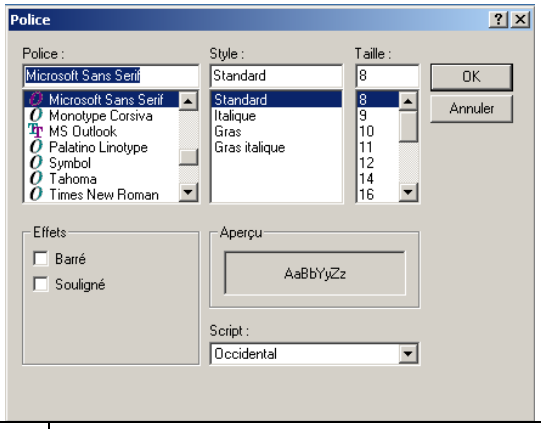
| | |
|---|---|
| Color | Définit la couleur par défaut affichée et retourne la couleur sélectionnée par l'utilisateur |
| Fullopen | Booléen définissant si la boîte de dialogue s'affiche en mode complet ou non |
| SolidColorOnly | Booléen n'affichant que les couleurs gérées par la carte graphique |

L'exemple suivant affiche une boîte de dialogue et modifie la couleur de fond du formulaire en fonction du choix de l'utilisateur :

```
Dim c As New ColorDialog
c.FullOpen = False
c.Color = Me.BackColor
c.ShowDialog()
Me.BackColor = c.Color
```

#### *6.1.3.4 Choix d'une police*

Cette boîte de dialogue permet la sélection de tous les paramètres concernant le formatage de chaîne de caractère (police, taille, gras…).



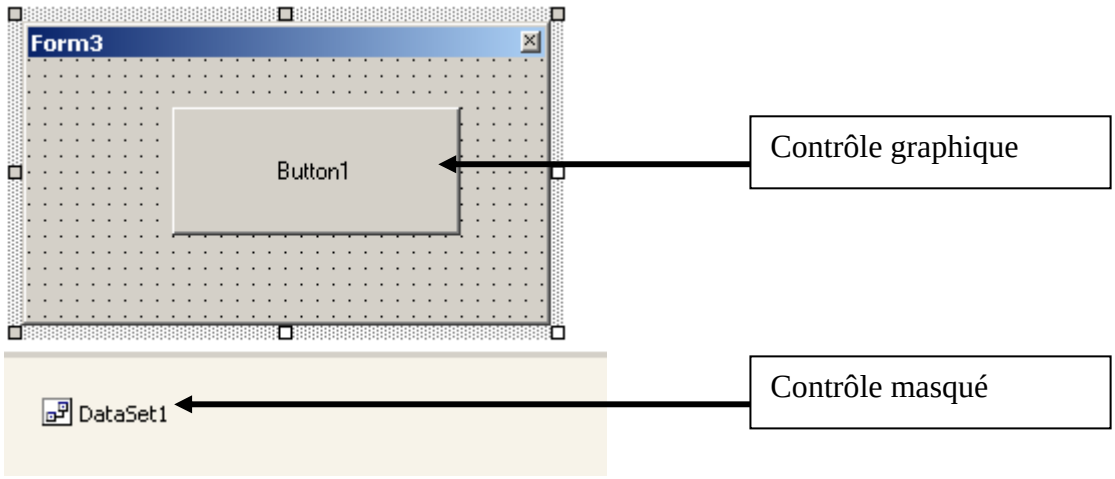

| Propriété | Description |
|---|---|
| ShowEffects | Booléen spécifiant si l'utilisateur peut définir les effets (gras …) |
| ShowColor | Booléen spécifiant si l'utilisateur peut définir la couleur |
| MinSize | Taille minimale des caractères |
| MaxSize | Taille maximale des caractères |
| Font | Police par défaut et police retournée |
| Color | Couleur par défaut et couleur retournée |

Pour afficher la boîte de dialogue, utiliser la méthode « ShowDialog ».

## 6.2 Les contrôles

Les contrôles permettent de créer l'interface entre l'utilisateur et notre application. C'est via les contrôles que l'utilisateur pourra saisir des données, effectuer des sélection et déclencher des actions par l'intermédiaires des événements.

De manière générale, les contrôles sont des objets graphiques, c'est à dire qu'il seront placés et visibles sur le formulaire. Cependant, certains contrôles offrant des fonctionnalités de programmation n'apparaîtront pas sur le formulaire mais dans une zone située en bas et uniquement en mode « Design ».




### 6.2.1 Membres communs

Les contrôles Visual Basic .Net sont des classes issues de la classe de base « control ». Cette dernière assure les fonctions élémentaires comme le placement sur une feuille, leur position … A cette classe est ajoutée une classe dérivée permettant la personnalisation des différents contrôles.

#### *6.2.1.1 propriétés*

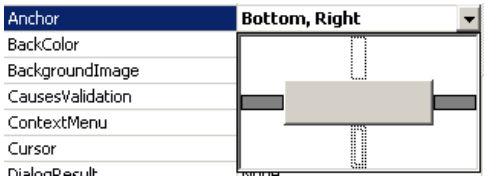
- Name

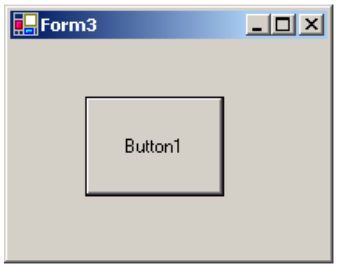
Nom du contrôle. Ce nom ne comporter que des lettres et le caractère underscore « _ ».

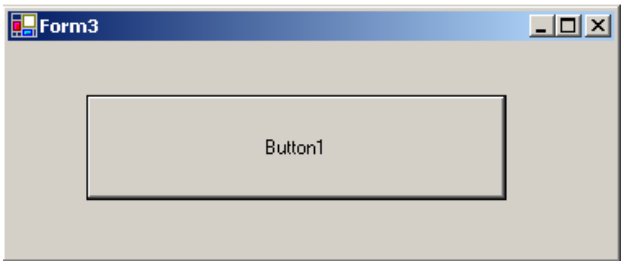
- Anchor

Les ancres permettent de modifier automatiquement la taille d'un contrôle lors du redimensionnement d'un formulaire. Chaque contrôle possède sa propre ancre.

Lors du paramétrage, vous devez définir sur quels bords du conteneur est ancré le contrôle. Dans l'exemple suivant, nous créons un contrôle ancré à gauche et à droite :




- CanFocus

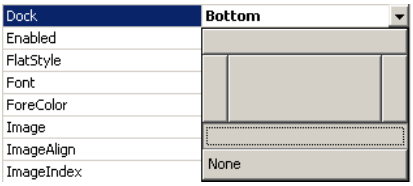
Booléen spécifiant si le contrôle peut recevoir le focus.

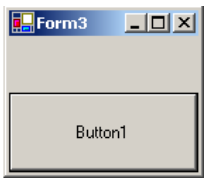
- CanSelect

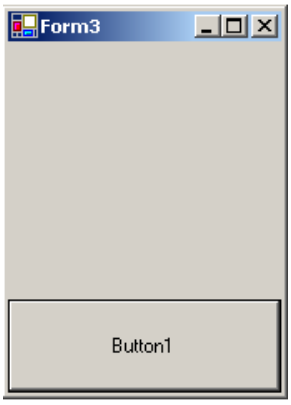
Booléen spécifiant si le contrôle peut être sélectionné.

- Dock

Dans le même esprit, la propriété « Dock » permet d'ancrer un contrôle aux à un bord du conteneur. Dans l'exemple suivant, le bouton est ancré en bas :




- Enabled

Cette propriété est une valeur booléenne spécifiant si le contrôle est accessible ou non. Dans le second cas, le contrôle apparaîtra grisé.

- Location

La propriété Location est un objet permettant de définir l'emplacement du contrôle par rapport à son conteneur. Il est composé de deux propriétés (X et Y) qui définissent ses coordonnées par rapport au coin supérieur gauche du conteneur.

- Locked

Contrairement à la version précédente, cette propriété ne bloque pas le contrôle lors de l'exécution mais lors de la conception. Il permet d'éviter de modifier les propriétés d'un contrôle.

- Modifiers

Cette propriété paramètre la visibilité au niveau programmation de l'objet. Elle peut prendre les valeurs suivantes :

| Valeur | Description |
|---|---|
| Public | Accessible à partir de tous les éléments de la solution |
| Protected | Accessible à partir des membres de la classe et des sous classes |
| Protected Friend | Correspond à l'union des visibilités Friend et Protected |
| Friend | Accessible à partir du programme et des assemblages liés |
| Private | Accessible à partir des membres de la classe |

Par défaut, la visibilité est friend.

- Size

Cet objet permet de définir la taille du contrôle. Il est composé de deux propriétés, width (largeur) et height (hauteur).

- TabIndex

Indice définissant l'ordre de tabulation du contrôle par rapport à son conteneur.

- Text

Cet propriété référence le texte contenu ou affiché dans un contrôle (Par exemple, le texte affiché sur un bouton).

- Visible

Cet propriété détermine si le contrôle est visible lors de l'exécution. Attention, aucun changement n'est visible lors de la conception.

#### *6.2.1.2 Méthodes*

| Méthode | Description |
|---|---|
| Focus | Donne le focus au contrôle |

#### *6.2.1.3 Evénements*

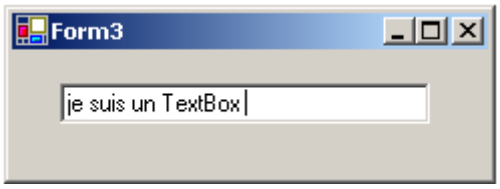
| Evénements | Description |
|---|---|
| Click | Activé lors du clic sur le contrôle |
| DoubleClick | Activé lors du double clic sur le contrôle |
| Enter | Activé lorsque l'utilisateur entre sur le contrôle |
| GotFocus | Activé lorsque le contrôle reçoit le focus |
| KeyDown | Touche enfoncée |
| KeyPress | Touche enfoncée et relachée |
| KeyUp | Touche relachée |
| LostFocus | Activé lorsque le contrôle perd le focus |
| MouseDown | Bouton souris enfoncé |
| MouseUp | Bouton souris relaché |
| MouseMove | Souris déplacée sur le contrôle |
| MouseWheel | Déplacement de la roulette |
| Resize | Déclenché lorsque le contrôle est redimensionné |

### 6.2.2 Principaux Contrôles

Nous ne listerons dans cette partie que les principaux contrôles.

#### *6.2.2.1 TextBox*

Le contrôle TextBox est certainement le contrôle le plus utilisé : il permet de saisir des chaînes de caractère de 2000 à 32 000 caractères en fonction de la configuration.




| Propriété | Description |
|---|---|
| CanFocus | Détermine si le contrôle peut recevoir le focus |
| CharacterCasing | Détermine la casse du texte : majuscules (upper) ou minuscules (lower) |
| Focused | Indique si le contrôle détient le focus |
| ForeColor | Couleur du texte |
| HideSelection | Définit si le contrôle masque la sélection lorsqu'il perd le focus |
| Lines | Tableau correspondant aux lignes du contrôle |
| MaxLength | Nombre de caractères maximum du contrôle |
| Modified | Spécifie si le contenu du champs a été modifié depuis sa création |
| MultiLine | Définit si le contrôle est multi lignes |
| PasswordChar | Définit le caractère servant à masquer un mot de passe |
| Readonly | Contenu du champs en lecture seule |

| | |
|---|---|
| Scrollbars | Affiche ou masque les barres de défilement |
| Selectionlength | Longueur de la sélection |
| SelectionStart | Indice de début de la sélection dans le champs |
| Text | Contenu du champs |
| TextLength | Longueur du texte dans le contrôle |

| Méthode | Description |
|---|---|
| Clear | Efface le contenu du champs texte |
| Copy / Cut | Copie / coupe la sélection dans le presse papier |
| Focus | Donne le focus au contrôle |
| ResetText | Rétabli la valeur initiale du champs |

| Evénement | Description |
|---|---|
| TextChanged | Déclenché lorsque le texte change |

L'exemple suivant permet de copier dans le presse papier tout le texte contenu dans le champs « textbox1 » et de vider ce dernier lorsqu'il reçoit le focus.

```
Private Sub TextBox1_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs)
Handles TextBox1.GotFocus
    With Me.TextBox1
        .SelectionStart = 0
        .SelectionLength = .TextLength
        .Copy()
        .Text = ""
    End With
End Sub
```

#### *6.2.2.2 Label*

Le contrôle label est utilisé pour afficher du texte qui ne sera pas éditable par l'utilisateur. Il est généralement utilisé pour afficher le rôle des différents contrôles.


| Propriété | Description |
|---|---|
| BorderStyle | Style de bordure |
| AutoSize | Le contrôle s'adapte à la taille du texte |
| Text | Contenu du label |

L'exemple suivant affiche successivement « Bonjour » en gras et « Au revoir » en rouge lorsque l'utilisateur double clic sur le contrôle label1 :

```
Private Sub Label1_DoubleClick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Label1.DoubleClick
    With Me.Label1
      If .Text = "Bonjour" Then
        .Text = "Au revoir"
        .Font = New Font(.Font, FontStyle.Regular)
        .ForeColor = System.Drawing.Color.Red
      Else
        .Text = "Bonjour"
        .Font = New Font(.Font, FontStyle.Bold)
        .ForeColor = System.Drawing.Color.Black
      End If
    End With
End Sub
```

#### *6.2.2.3 CheckBox*

Le contrôle Checkbox (Case à cocher) est utilisé pour proposer plusieurs options à l'utilisateur parmi lesquelles il pourra effectuer plusieurs choix.


| Propriété | Description |
|---|---|
| Checked | Valeur booléenne indiquant si la case est cochée ou non |
| CheckState | Retourne ou modifie la valeur de la case à cocher en gérant le 3ème mode (grisé). |
| ThreeState | En standard, une case à cochée peut être cochée ou non. Il existe cependant un 3ème état « Indéterminé » permettant de grisé la case. Cette propriété permet d'activer ce 3ème état. |
| CheckAlign | Alignement de la case à cocher par rapport au contrôle |
| Text | Texte associé au contrôle |

| Evénement | Description |
|---|---|
| CheckedChanged | Se produit lorsque la propriété « Checked » change |
| CheckStateChanged | Se produit lorsque la propriété « CheckState » change |

L'exemple suivant comporte 3 cases à cocher (pour une sélection d'options voiture) :

- ch1 libellée « Décapotable »
- ch2 libellée « Toit ouvrant »
- ch3 libellée « Ailerons »

Pour des raisons logiques, il n'est pas possible de choisir ch2 et ch3 si ch1 est sélectionné. Le code suivant permet de décocher et griser les cases.

```
    Private Sub ch1_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ch1.CheckedChanged
        If Me.ch1.CheckState = CheckState.Checked Then
            Me.ch2.Checked = False
            Me.ch3.Checked = False
            Me.ch2.Enabled = False
            Me.ch3.Enabled = False
        Else
            Me.ch2.Enabled = True
            Me.ch3.Enabled = True
        End If
    End Sub

    Private Sub ch2_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ch2.CheckedChanged
        If Not Me.ch2.Checked = True Then Me.ch1.CheckState = CheckState.Unchecked
        If Not Me.ch2.Enabled Then Me.ch2.Enabled = True
        If Not Me.ch3.Enabled Then Me.ch3.Enabled = True
    End Sub

    Private Sub ch3_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ch3.CheckedChanged
        If Not Me.ch3.Checked = True Then Me.ch1.CheckState = CheckState.Unchecked
        If Not Me.ch2.Enabled Then Me.ch2.Enabled = True
        If Not Me.ch3.Enabled Then Me.ch3.Enabled = True
    End Sub
```

#### *6.2.2.4 RadioButton*

Contrairement aux cases à cocher, les boutons radio permettent à l'utilisateur d'effectuer un seul choix parmi plusieurs options. Cette dernière contrainte impose donc qu'il n'y ait jamais deux boutons cochés en même temps : Visual basic s'occupe de faire basculer l'état des boutons pour les boutons présents dans le même conteneur. Dans l'exemple suivant, c'est le formulaire qui est conteneur. Nous verrons plus loin les conteneurs « GroupBox » et « Panel ».


Les boutons radios possèdent les même propriétés et événements que les cases à cocher.

L'exemple suivant travaille avec 3 boutons radio (rad_blue, rad_red, rad_black) qui modifie la couleur de fond du formulaire en fonction de celui sélectionné. Notez qu'une seule

procédure est utilisée pour implémenter les l'événement « CheckedChanged » de chaque bouton radio.

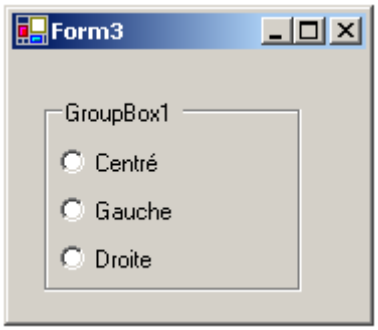
```
Private Sub rad_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles rad_blue.CheckedChanged, rad_black.CheckedChanged,
rad_red.CheckedChanged
    If Me.rad_black.Checked Then
        Me.BackColor = System.Drawing.Color.Black
    ElseIf Me.rad_blue.Checked Then
        Me.BackColor = System.Drawing.Color.Blue
    Else
        Me.BackColor = System.Drawing.Color.Red
    End If
End Sub
```

#### *6.2.2.5 GroupBox et Panel*

Au même titre qu'un formulaire, les contrôles GroupBox et Panel sont des conteneurs, c'est à dire qu'il contiennent eux même d'autres contrôles. Ces contrôles présentent deux intérêts majeurs :

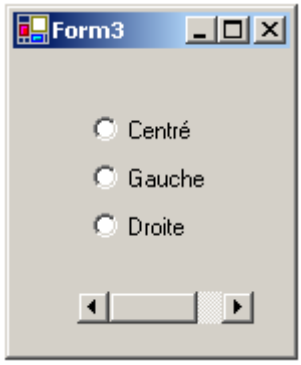
- Regrouper de manière logique des contrôles afin de les isoler (pour les boutons radio par exemple)
- Faciliter le placement de plusieurs contrôles car en modifiant la position du conteneur, vous modifiez la position de tous les contrôles contenus

- Le GroupBox




Ce contrôle possède une seule propriété particulière« text » qui correspond au texte affiché.

- Le Panel




Le contrôle panel reprend les fonctionnalités du contrôle GroupBox avec en plus la possibilité de gérer les barres de défilement (propriété AutoScroll).

#### *6.2.2.6 Button*

Le contrôle « button » est principalement utilisé pour déclencher une action lors du clic.

#### *6.2.2.7 ListBox*

Le contrôle ListBox permet l'affichage d'une liste de choix, généralement des chaînes de caractères, dans laquelle l'utilisateur pour effectuer un ou plusieurs choix.

| Propriété | Description |
|---|---|
| MultiColumn | Permet un défilement horizontal de la liste |
| Integralheight | Evite l'affichage d'une partie d'un élément de la liste |
| Items | Collection représentant les éléments contenus dans la liste |
| Sorted | Eléments classés par dans l'ordre |
| Itemheight | Hauteur d'un élément de la liste |
| SelectedIndex | Indice de l'élément sélectionné |
| SelectedIndices | Indices des éléments sélectionnés |
| SelectionMode | Mode de sélection des éléments (« MultiExtended » permet une sélection multiple, « One » permet une seule sélection et « None » aucune. |

| Méthode | Description |
|---|---|
| FindString | Retourne l'indice de l'élément commençant par le texte recherché |
| SetSelected | Définit un élément en tant que sélectionné ou non |
| GetSelected | Retourne un booléen permettant de savoir si un élément est sélectionné ou non |

| Evénement | Description |
|---|---|
| SelectedIndexChange | Déclenché lorsque la propriété « SelectedIndex » change |

L'exemple suivant remplit un ListBox à l'aide d'une boucle.

```
Dim i As Int16
Me.ListBox1.Items.Clear()
```

```
    For i = 1 To 50
       Me.ListBox1.Items.Add("Element no " & i)
    Next
```

Celui ci affiche l'élément sélectionné lors d'un double clic sur le ListBox.

```
   Private Sub ListBox1_DoubleClick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ListBox1.DoubleClick
      Dim elt As String
      Dim indice As Int16
      indice = Me.ListBox1.SelectedIndex
      elt = Me.ListBox1.Items(indice)
      MsgBox("Elément sélectionné: " & elt)
   End Sub
```

Enfin, le code permettant d'afficher la liste des éléments sélectionnés :

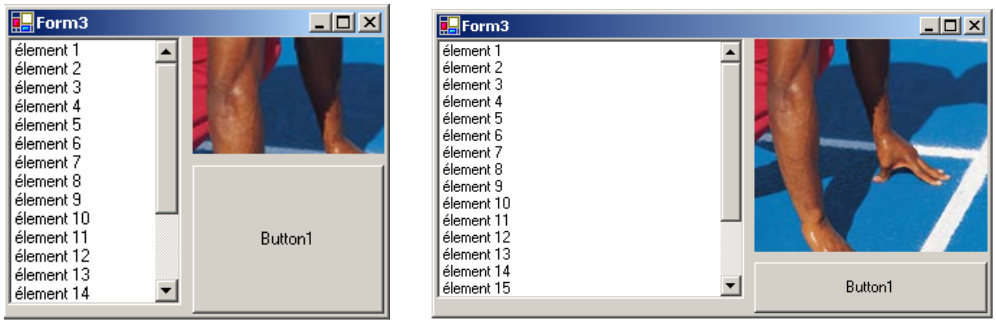
```
    Dim elt As String
    For Each elt In Me.ListBox1.SelectedItems
       MsgBox("Libellé: " & elt)
    Next
```

#### *6.2.2.8 ComboBox*

Le contrôle ComboBox est l'association du contrôle listBox et TextBox : il permet à l'utilisateur de sélectionner une valeur dans une liste ou de saisir une nouvelle valeur. Cependant, ce contrôle n'accepte pas les sélections multiples.

#### *6.2.2.9 Splitter*

Le contrôle « Splitter » est une nouveauté de la version .Net. Il permet de créer des barres de séparation redimensionnables pour distribuer l'espace du formulaire entre les différents contrôles. Splitter est particulièrement utilisé dans les interfaces de type « explorateur ».




Plutôt qu'un long discours, ci-dessous figure le mode opératoire afin de réaliser l'interface montrée en exemple.

- Placer le contrôle « liste » et paramétrer la propriété « Dock » à « left »
- Placer le contrôle « splitter » à droite de la liste et paramétrer sa propriété « dock » à left
- Placer le contrôle « image » à droite et paramétrer la propriété « Dock » à « top »
- Placer le second splitter en dessous de l'image et paramétrer la propriété « Dock » à « top »
- Enfin, placer le contrôle « bouton » en dessous et paramétrer la propriété « Dock » à « fill »

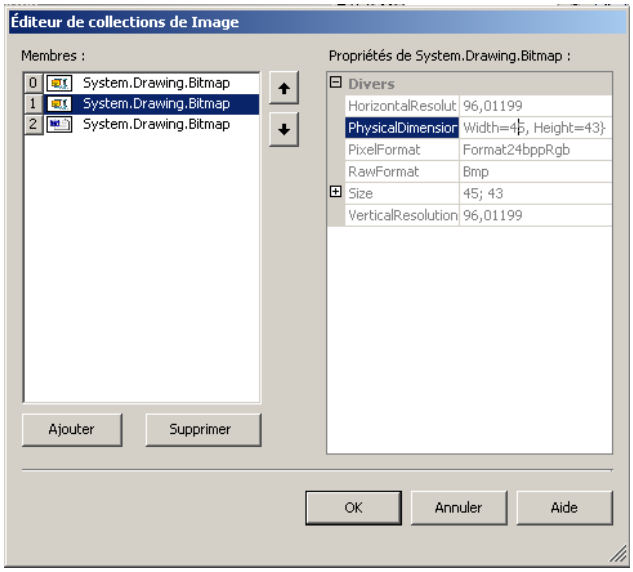
| Propriété | Description |
|---|---|
| MinSize | |
| MinExtra | |

#### *6.2.2.10 ImageList*

Le contrôle « ImageList » est un conteneur d'images destinées à être utilisée dans l'application ou alors par d'autre contrôles (Listview, TreeView …). Ce contrôle n'est pas visible sur le formulaire et peut contenir tous types d'images (Gif, Jper, Bmp …).

| Propriété | Description |
|---|---|
| ColorDepth | Nombre de couleurs à utiliser pour les images |
| ImageSize | Taille en pixels des images |
| Transparent | Définit la couleur de transparence |
| Images | Collection contenant les images |

Chacune des images possède un Index qui sera ensuite utilisé pour les lier aux autres contrôles. La gestion des images se fait à l'aide d'un assistant :

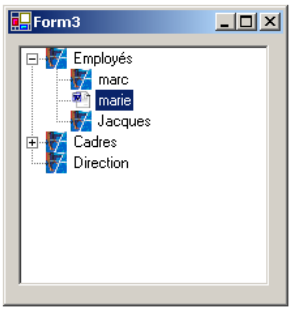



Le code suivant permet d'ajouter une image dans un Imagelist à partir d'un fichier physique et la supprimer du contrôle :

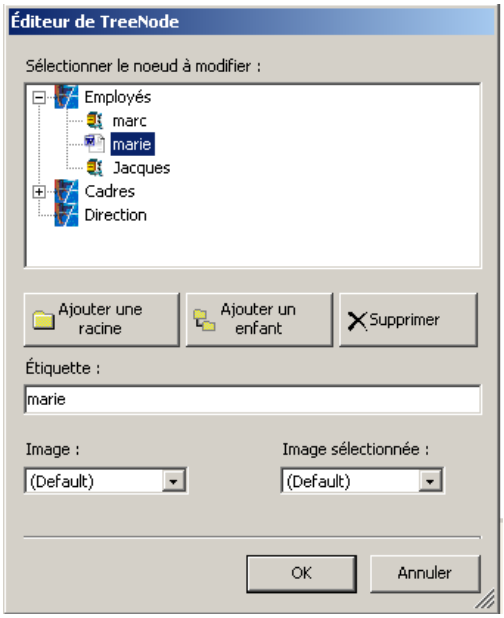
```
Dim chemin as string
Chemin = ‘’C:\mes documents\toto.gif’’
Me.Imagelist1.images.add(Image.fromfile(chemin))
Me.Imagelist1.images.RemoveAt(0)
```

#### *6.2.2.11 Treeview*

Le contrôle TreeView permet un affichage hiérarchique des données à la façon de l’explorateur Windows. Chaque élément du treeview est un nœud pouvant à son tour contenir d’autres nœuds.




Pour remplir le Treeview, vous pouvez utiliser les méthodes liées au contrôle ou utiliser l’assistant fourni par le framework. Pour l’ouvrir, utiliser le bouton situé à droite de la propriété « Nodes » :




- Ajouter une racine : Ajoute un élément à la racine (Employés par exemple)
- Ajouter un enfant : Ajoute un nœud enfant au nœud sélectionné
- Etiquette : Texte affiché au niveau de l’élément sélectionné
- Image : Image du contrôle imagelist lié.
- Image sélectionnée : Image affichée lorsque l’élément est sélectionné

| Propriété | Description |
|---|---|
| CheckBoxes | Afficher les cases à cocher au niveau des éléments |
| FullrowSelect | La surbrillance s'étend sur toute la largeur du contrôle |
| ImageIndex | Indice de l'image par défaut du contrôle ImageList |
| ImageList | Contrôle ImageList contenant les images utilisées par le treeview |
| Indent | Valeur en pixel de l'indentation |
| LabelEdit | Permet à l'utilisateur de modifier l'étiquette |
| Nodes | Collection de nœuds |
| SelectedImageIndex | Indice de l'image par défaut pour les éléments sélectionnés |
| ShowPlusMinus | Affiche les signes + et – devant les nœuds parents |
| Sorted | Indique si les nœuds sont triés |

| Méthode | Description |
|---|---|
| Nodes.add | Ajoute un élément |

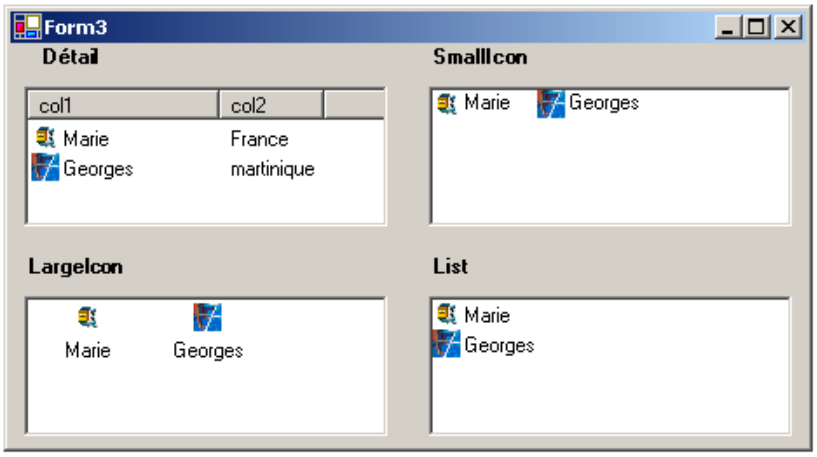
Le code suivant remplit un ImageList, insère à l'intérieur du TreeView 2 catégories principales (Renault & Peugeot) et place ensuite à l'intérieur les différents modèle en leur affectant des images :

```
Me.ImageList1.Images.Add(Image.FromFile("C:\peugeot.jpg"))
Me.ImageList1.Images.Add(Image.FromFile("C:\renaud.bmp"))

Me.TreeView1.ImageList = Me.ImageList1
Me.TreeView1.ImageIndex = 0

Dim noeud1, noeud2 As TreeNode
noeud1 = New TreeNode
noeud2 = New TreeNode

With noeud1
   .Text = "Peugeot"
   .ImageIndex = 0
   .Nodes.Add("307")
   .Nodes.Add("806")
   .Nodes.Add("309")
End With
With noeud2
   .Text = "Renault"
   .ImageIndex = 1
   .Nodes.Add("Mégane")
   .Nodes.Add("4L")
   .Nodes.Add("Laguna")
End With

With Me.TreeView1
   .Nodes.Add(noeud1)
   .Nodes.Add(noeud2)
```

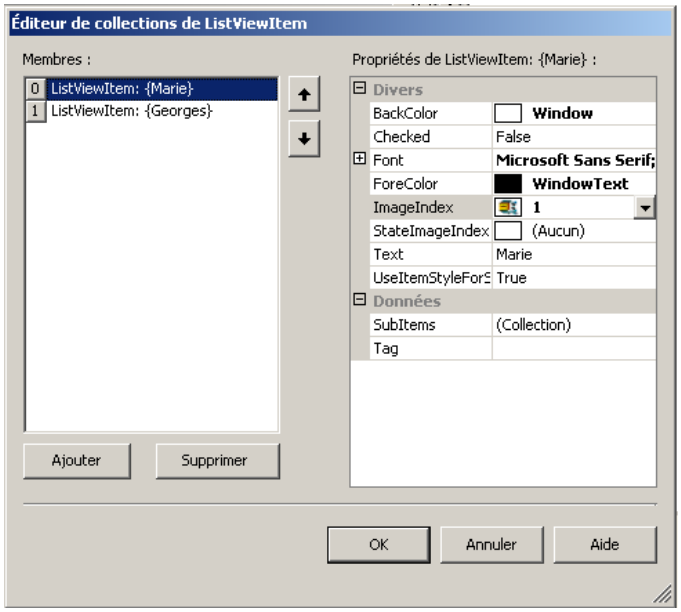
```
End With
```

### *6.2.2.12 ListView*

Le contrôle listview permet l'affichage d'une liste plate selon les 4 modes de présentation de l'explorateur Windows :




Au même titre que le TreeView, le ListView posséde des assistants afin de définir leur contenu. Pour ouvrir l'assistant permettant de gérer la liste, cliquer sur le bouton à droite de la propriété Items :

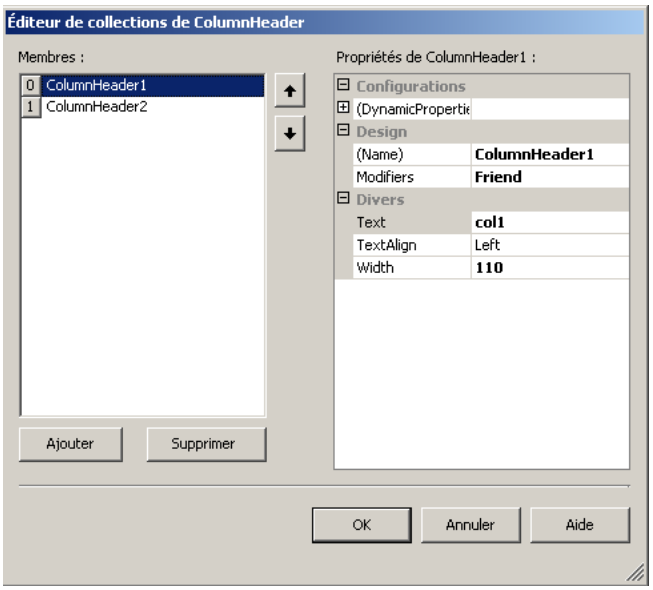



Pour chaque élément, il est possible de paramétrer :

- Checked : définit si l'élément apparaît coché par défaut
- Font : police de l'élément
- ForeColor : couleur d'affichage du texte
- ImageIndex : indice de l'image liée à l'élément
- Text : libellé de l'élément
- UseItemStyleForSubitems : répercute les propriété de l'élément sur les sous éléments

- SubItem : dans un affichage par détail, correspond aux colonnes à partir de la seconde. Lors du clic sur le bouton correspondant à cette propriété, un nouvel assistant est lancé pour définir les autres colonnes.

Il existe également un assistant pour les colonnes : pour l'ouvrir, utiliser le bouton à droite de la propriété « Columns ». Celles ci n'apparaîtront que lors d'un affichage au détail.




| Propriété | Description |
|---|---|
| AllowColumnReorder | Autorise ou non la modification de l'ordre des colonnes |
| AutoArrange | Organise automatiquement la présentation des éléments |
| Columns | Collection de colonnes |
| FullrowSelect | La surbrillance s'étend sur toute la largeur du contrôle |
| HeaderStyle | Style des entête de colonne |
| Items | Collection d'éléments |
| LabelEdit | Permet à l'utilisateur de modifier l'étiquette des éléments |
| LargeImageList | ImageList utilisé pour la présentation « LargeIcon » |
| MultiSelect | Permet la sélection multiple |
| SmallImageList | ImageList utilisé pour toutes les présentation (sauf LargeIcon) |
| Sorting | Mode de tri |
| View | Mode de représentation de la liste |

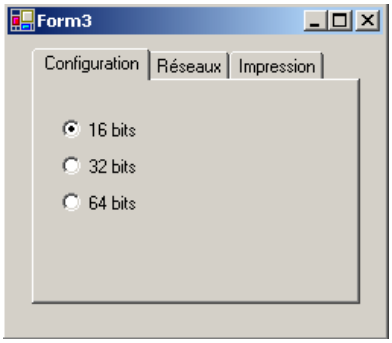
Le code suivant affiche pour chaque caractère son code ascii, le caractère en minuscule et le caractère en majuscule. Il crée également les colonnes :

```
Dim elt As ListViewItem
Dim i As Byte

With Me.ListView1
   .View = View.Details
   .Columns.Add("Ascii", 50, HorizontalAlignment.Center)
```

```
        .Columns.Add("Min", 50, HorizontalAlignment.Center)
        .Columns.Add("Maj", 50, HorizontalAlignment.Center)
        For i = 0 To 255
            elt = New ListViewItem
            elt.Text = CType(i, String)
            elt.SubItems.Add(LCase(Chr(i)))
            elt.SubItems.Add(UCase(Chr(i)))
            Me.ListView1.Items.Add(elt)
        Next

    End With
```

### *6.2.2.13 TabControl*

Le contrôle « TabControl » permet l'affichage d'onglet contenant chacun plusieurs contrôles. Ce dernier est généralement utilisé pour regrouper logiquement des contrôle ou pour placer beaucoup de contrôles dans la même formulaire.




TabControl possède également un assistant permettant de le configurer. Pour ouvrir l'assistant, utiliser le bouton à droite de la propriété « TabPages » :

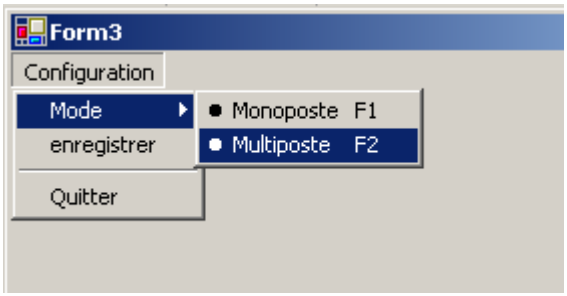

Pour chaque « page », il est possible de configurer les options d'apparence qui sont analogues à celles d'un formulaire.

| Propriété | Description |
|---|---|
| Alignment | Définit la position des onglets par rapport aux pages |
| HotTrack | Modifie l'apparence des onglets lorsque la souris passe dessus |
| Imagelist | ImageList lié pour les icônes d'onglets |
| Multiline | Permet l'affichage des onglets sur plusieurs lignes |
| TabPages | Collection de pages. |

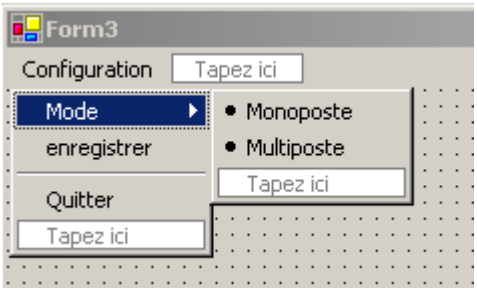
#### *6.2.2.14 Menus*

les menus permettent d'offrir à l'utilisateur un ensemble de fonctionnalités sans pour autant surcharger la présentation du formulaire. Il existe 2 types de menu :

- Menu d'application situé en haut du formulaire
- Menu contextuel activé généralement lors d'un clic droit




Pour créer un ou plusieurs menus, vous devez ajouter à votre formulaire le contrôle « MainMenu ». A ce moment, le contrôle apparaît en dessous du formulaire et un menu est ajouté. Il n'y a pas d'assistants particulier : pour ajouter un élément, cliquer sur les zones « Tapez ici » :




Pour définir une barre de séparation, créer un élément avec « - » (tiret) en libellé.

| Propriété | Description |
|---|---|
| Checked | Indique si l'élément est coché |
| DefaultItem | Définit l'élément en tant qu'élément par défaut |
| MdiList | Affiche la liste des fenêtres enfants dans le cas d'une fenêtre MDI |
| RadioCheck | Indique si l'élément est activé |
| ShortCut | Permet de définir un raccourci pour le menu |
| ShowShorcut | Affiche le raccourci |
| Text | Libellé du menu |

#### *6.2.2.15 DateTimePicker*

Le contrôle « DateTimePicker » associe une zone de texte et un calendrier permettant la sélection d'une date.


| Propriété | Description |
|---|---|
| Checked | Si activé, spécifie lorsque l'utilisateur a sélectionné une date |
| CustomFormat | Chaine de format pour l'affichage de la date sélectionnée |
| Format | Mode d'affichage de la date |
| MaxDate | Date maximale sélectionnable |
| MinDate | Date minimale sélectionnable |

| Evénement | Description |
|---|---|
| ValueChanged | Déclenché lorsque la valeur change |

Il existe un second contrôle pour la gestion des dates : MonthCalendar. Ce dernier reprend les fonctionnalités du contrôle DateTimePicker avec en plus la possibilité de définir des jours fériés ou des périodes sélectionnées.

#### *6.2.2.16 Timer*

Le contrôle Timer permet de déclencher un événements à intervals réguliers.

| Propriété | Description |
|---|---|
| Interval | Définit l'interval en milliseconde. La valeur doit être comprise entre 1 et 65536 |
| Enabled | Active ou désactive le timer |

| Evénement | Description |
|---|---|
| Tick | Déclenché à chaque interval. |

### 6.2.3 Le Drag and Drop

Le drag and Drop (ou Glisser Déposer) est une des fonctionnalités en terme d'ergonomie qui fit le succès de Windows. Il permet de déplacer des informations (Fichiers, Images, Texte, Objet) au sein d'une même application ou entre plusieurs en accrochant un élément au curseur pour ensuite le déposer sur l'objet de destination (Liste, champs texte…).

#### *6.2.3.1 Démarrer le drag and drop*

Pour démarrer le drag and drop, vous devez détecter lorsque l'utilisateur quitte un contrôle avec un bouton enfoncé. Pour cela, on utilise l'événement « MouseMove » :

```
  Private Sub TextBox1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles TextBox1.MouseMove
      If e.Button = MouseButtons.Left Then
         …
      End If
   End Sub
```

Ensuite, vous activer le drag and drop en spécifiant l'élément à déplacer ainsi que les effets. L'élément à déposer est de type Object (donc ce peut être n'importe quel élément). Pour cela, on utilise la méthode « DoDragDrop » prenant deux arguments : l'objet à déplacer et les effets voulus. Les effets sont membres de la classe « DragDropEffects ».

| Effet | Description |
|---|---|
| All | Les données sont copiées, supprimées et parcourues dans la zone cible |
| Copy | Les données sont copiées dans la zone de déplacement |
| Link | Les données sont liées à la cible de déplacement |
| Move | Les données sont déplacées vers la cible de déplacement |

#### *6.2.3.3 Contrôler la réception*

En premier lieu, vous devez spécifier que le contrôle cible peut recevoir des éléments : pour cela, paramétrez la propriété « AllowDrop » à « True ».

Le contrôle de la réception consiste à modifier l'apparence du curseur et du contrôle cible en vérifiant que l'élément déplacé corresponde aux besoins. Pour cela, nous disposons de plusieurs événements (ces 3 événements sont valides tant que l'utilisateur ne lâche pas le bouton de la souris).

| Evénement | Description | Utilisation |
|---|---|---|
| DragEnter | Se produit lorsque le curseur entre dans la zone du contrôle. | Utilisé pour modifier l'apparence du contrôle cible et du curseur et vérifier la validité de l'élément déplacé |
| DragOver | Se produit tant que le curseur reste au dessus de la zone du contrôle | |
| DragLeave | Se produit lorsque le curseur quitte la zone du contrôle | Utilisé pour rétablir l'apparence du contrôle cible et du curseur |

Sur chacun des événement est passé en paramètre « e » de type DragEventsArg contient toutes les informations sur le déplacement :

| Propriété | Description |
|---|---|

| | |
|---|---|
| Data | Correspond à l'élément déplacé |
| Effect | Défini l'action autorisée par le contrôle de destination et permet de modifier l'apparence de la souris |
| KeyState | Permet de connaître l'état des touches Shift, Alt, Ctrl |
| X, Y | Position du curseur sur le contrôle |

Pour récupérer l'élément ou l'objet déplacé, vous devez utiliser la méthode suivante :

```
e.Data.GetData(DataFormats.type_données)
```

Cette méthode est utilisée pour vérifier le type d'objet déplacé (dans la méthode DragEnter).

#### *6.2.3.3 Récupérer l'élément*

La récupération de l'élément se fait lorsque l'événement « DragDrop » est déclenché.

Ci dessous figure un exemple complet permettant de déplacer du texte d'une textbox à une autre :

```
   Private Sub TextBox1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles TextBox1.MouseMove
      If e.Button = MouseButtons.Left Then
         Me.TextBox1.DoDragDrop(Me.TextBox1.Text, DragDropEffects.Move)

      End If
   End Sub

   Private Sub TextBox2_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox2.DragEnter
      e.Effect = DragDropEffects.Move
      Me.TextBox2.BorderStyle = BorderStyle.FixedSingle
   End Sub

   Private Sub TextBox2_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox2.DragDrop
      Me.TextBox2.Text = e.Data.GetData(DataFormats.Text)
      Me.TextBox2.BorderStyle = BorderStyle.FixedSingle
      If CBool(e.KeyState And 32) Then
         Me.TextBox1.Clear()
      End If
   End Sub

   Private Sub TextBox2_DragLeave(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox2.DragLeave
      Me.TextBox2.BorderStyle = BorderStyle.Fixed3D
   End Sub
```

# 7 ActiveX Data Object .Net

L'accès aux données dans le développement d'applications est une étape fondamentale qui influera ensuite sur la rapidité et l'évolutivité de votre application. Dans les versions précédentes de Visual Basic, il existait plusieurs méthodes d'accès aux données en fonction des configurations (Bases de données distantes ou locales, type de fichiers…) : DAO, RDO, ADO. Dans la nouvelle mouture seule la technologie ADO .Net est gardé. Elle permet un accès à différentes sources de données par l'intermédiaire de fournisseurs OLE DB. La grande force de cette technologie est qu'elle permet une manipulation identique quelque soit la source de données (en dehors des paramètres de connexion).

Cette nouvelle version voit apparaître de nouvelles fonctionnalités :

- Gestion des données en mode déconnecté : les DataSet et DataAdapter permettent de fusionner la base de données avec des fichiers de données en intégrant les modifications effectuées de chaque bord et, le cas échéant, mettre en avant les conflits
- Mise à disposition de classe dépendante de la source de données afin d'optimiser les traitement et ainsi éviter les couches successives
- Intégration du format XML pour l'échange et le rapatriement de données

Au final, l'ADO est un ensemble de classe mettant à disposition les objets, méthodes et évènements nécessaire à l'interfaçage avec une base de données. Ces classes sont disponibles dans plusieurs espaces de nom :

| Espace de noms | Description |
|---|---|
| System.Data | Contient les objets ADO n'appartenant pas à un fournisseur spécifique (DataSet, DataTable …) |
| System.data.Common | Contient les classes de base pour plusieurs objets des autres espaces de noms |
| System.Data.OleDB | Contient les objets associés au fournisseurs OLEDB .Net |
| System.Data.SqlClient | Contient les objets associés au fournisseurs Sql Server .Net |

## 7.1 Mode connecté et déconnecté

Lors de l'utilisation de données dans une application, deux modes sont opposés :

### 7.1.1 Mode connecté

On parle de mode connecté lorsque l'application client à un accès direct à la source de données. Dans ce cas, vous devez dans un premier vous connecter à la source avant d'effectuer n'importe quelle tâche. Ce mode est utilisé pour les applications « résidante » au sein d'une entreprise dans laquelle la base de données est toujours accessible.

Dans ce mode, vous utiliserez les objets suivants :

- Connection : permet d'établir une connexion avec la source de données

- Command : cet objet permet d'exécuter des traitements où de modifier / lire des données sur la source
- DataReader : permet de parcourir un ensemble d'enregistrement récupéré

### 7.1.2 Mode déconnecté

Le mode déconnecté est une nouveauté de l'ADO .Net. Ce mode permet de travailler sur des données sans avoir un accès direct et permanent à la base. Il est principalement utilisé pour les applications « nomades » qui ne disposent pas toujours d'un accès à la source de données comme par exemple, les portables des commerciaux dans une société.

Dans cette configuration, le traitement des données se fait en 3 étapes :

- Récupération des données à partir de la source
- Modification des données en mode déconnecté
- Intégration des données modifiées à la source avec, le cas échéant, résolution des conflits (généralement les contraintes d'intégrité)

Si vous utilisez le mode déconnecté, vous disposerez des objets suivants :

- DataSet : jeu de données
- DataAdapter : classe permettant la fusion entre un jeu de données et la source

## 7.2 Les fournisseurs d'accès

Les fournisseurs d'accès permettent la communication avec une source de données. En fonction des sources de données, le provider sera différent. Il existe 3 types de Providers en ADO.Net :

- Les fournisseurs ODBC permettant de se connecter aux sources ODBC
- Le fournisseur OLE DB (permettant de se connecter à toutes les sources ayant un provider OLE Db). Voir http://www.able-consulting.com/MDAC/ADO/Connection/OLEDB_Providers.htm pour plus d'informations
- Les fournisseurs natifs (Pour Sql Server ou Oracle)

| Source de données | Fournisseur | Chaine de connexion |
|---|---|---|
| Microsoft Access | OLEDB | Microsoft.Jet.OLEDB.4.0 |
| SQL Server | SQL | SQLOLEDB.1 |
| Oracle | OLEDB | OraOLEDB.Oracle |
| ODBC | ODBC | MSDASQL |
| MySQL | OLEDB | MySQLProv |

Le choix du fournisseur aura deux impacts majeurs dans le développement de votre application :

- Influe sur la rapidité des traitements avec la source de données : lorsque que vous le pouvez, utiliser le fournisseur dédié à votre base de données

- Utilisation des fonctionnalités propres à la source : certaines fonctionnalités ne sont accessibles que pour certains fournisseurs. Choisir un fournisseurs trop générique ne vous permettra pas d'exploiter complètement les fonctionnalités de votre source.

## 7.3 L'objet Connection

L'objet connection constitue la première étape dans l'accès aux données : elle permet de se connecter à la source et de connaître le niveau de sécurité et les différents droits de l'utilisateur sur la source.

En fonction du type de fournisseur choisi, la création de l'objet Connection différera :

| Fournisseur | Classe |
|---|---|
| ODBC | Odbc.OdbcConnection |
| OLEDB | OLEDB.oledbConnection |
| SQL Server | Sqlclient.SQLconnection |

L'exemple suivant crèe 3 connexions en fonction du fournisseur :

```
Dim cn_oledb As New OleDb.OleDbConnection
Dim cn_odbc As New Odbc.OdbcConnection
Dim cn_sql As New SqlClient.SqlConnection
```

### *7.3.1 Propriétés*

| Propriété | Description |
|---|---|
| ConnectionString | Chaîne utilisée pour la connexion |
| ConnectionTimeOut | Délai en seconde maximum pour l'établissement de la connexion |
| State | Etat de la connexion (Closed, connecting, Open, Executing, Fetching, Broken) |
| Provider | Fournisseur utilisé |
| DataSource | Emplacement de la base de données |

La principale propriété est « connectionString ». C'est une chaîne de caractères définissant les principaux attributs nécessaires à la connexion. Parmi ces informations figurent :

- Le fournisseur d'accès
- L'emplacement de la base
- Informations d'authentification
- La base de données initiale

L'exemple suivant ouvre une connexion avec une base Microsoft Access :

```
    Dim cn As New OleDb.OleDbConnection
    cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\Support\vbnet\bd2.mdb;"
```

```
cn.Open()
Select Case cn.State.ToString
   Case ConnectionState.Open
      MsgBox("Ouvert")
   Case ConnectionState.Broken
      MsgBox("Interrompue")
   Case ConnectionState.Closed
      MsgBox("Fermée")
   Case ConnectionState.Connecting
      MsgBox("En cours de connexion")
   Case ConnectionState.Executing
      MsgBox("En exécution")
   Case ConnectionState.Fetching
      MsgBox("Extraction de données")
End Select
cn.Close()
```

L'exemple ci-dessous ouvre une connexion avec un serveur SQL Server, spécifie un compte SQL pour l'authentification et paramètre le « timeout » :

```
Dim cn as new SqlClient.SqlConnection
Cn.connectionString = "Provider=SQLOLEDB.1;Data Source=(local) ;User Id=sa ;Initial
Catalog =NorthWind ;Connection TimeOut=50"
Cn.open
Msgbox(cn.state)
Cn.close
```

Enfin, un exemple de connexion SQL Server avec authentification Windows, c'est à dire que l'utilisateur doit être authentifié au niveau du domaine afin d'accéder aux données.

```
Dim cn as new SqlClient.SqlConnection
cn.ConnectionString = "Integrated Security=SSPI;Persist Security Info=False;Initial
Catalog=planning;Data Source=zeus;Workstation ID=OLIVIER;"
cn.Open()
…
cn.close
```

### *7.3.2 Méthodes*

| Méthode | Description |
|---|---|
| Open | Ouvre la connexion |
| Close | Ferme la connexion et libère les ressources |
| BeginTransaction | Débute une transaction avec un niveau d'isolation |
| CreateCommand | Crée un objet Command |

### *7.3.3 Evénements*

| Evénement | Description |
|---|---|

| | |
|---|---|
| StateChange | Déclenché lors du changement de la propriété State |
| InfoMessage | Déclenché lorsque la BD envoie un message |

L'exemple suivant permet d'afficher un message lorsque la connexion se ferme :

```
Dim WithEvents cn as new OleDb.OleDbConnection

Private sub cn_State_change(byval sender as Object, ByVal e as
System.Data.StateChangeEventArgs) handles cn.StateChange
        If (e.CurrentState = ConnectionState.closed) then
                MsgBox("Connexion fermée ! ")
        End if
End sub
```

## 7.4 Objet Command

Une fois la connexion établie avec la source de données, vous devez communiquer avec cette dernière pour gérer vos traitements. Trois types de traitements peuvent être effectués :

- Requête de sélection pour extraire des informations
- Requête d'exécution
- Procédures stockées (scripts stockés sur le serveur)

Lors de la création d'un objet commande, vous devez définir le type d'opération qu'il devra réaliser ainsi que la connexion à laquelle il est rattaché.

| Propriété | Description |
|---|---|
| Commandtext | Texte SQL de la requête ou nom de la procédure stockée |
| CommandType | Type de la commande (requête, table, procédure) |
| Connection | Connexion liée à la commande |
| Transaction | Objet transaction lié (voir plus bas) |
| CommandTimeOut | Nombre de seconde pour l'exécution de la commande |
| Parameters | Collection de paramètres à envoyer avec la commande |

| Méthode | Description |
|---|---|
| Cancel | Annule l'exécution de la commande |
| ExecuteNonQuery | Exécute la requête d'action et retourne le nombre de ligne affectées |
| ExecuteReader | Exécute la requête de sélection et retourne un objet de type DataReader |
| ExecuteScalar | Exécute la requête et retourne la valeur scalaire (1ère ligne, 1ère colonne) |
| CreateParameter | Crée un objet paramètre |
| ExecuteXMLReader | Exécute la requête de sélection et retourne un objet de type XmlReader |

L'exemple suivant permet de récupérer le résultat d'une requête :

```
Dim com_sql As New SqlClient.SqlCommand
Dim dr_sql As SqlClient.SqlDataReader
```

```
com_sql.Connection = cn_sql
com_sql.CommandType = CommandType.Text
com_sql.CommandText = "select * from STAGIAIRE"

dr_sql = com_sql.ExecuteReader
```

L'exemple suivant permet de modifier les enregistrements d'une table et d'afficher le nombre d'enregistrement modifiés :

```
Dim com_sql As New SqlClient.SqlCommand

com_sql.Connection = cn_sql
com_sql.CommandType = CommandType.Text
com_sql.CommandText = "update STAGIAIRE set sta_nom = 'toto' where sta_num = -55"
MsgBox(com_sql.ExecuteNonQuery)
```

L'exemple suivant permet de récupérer le nombre de stagiaires :

```
Dim com_sql As New SqlClient.SqlCommand

com_sql.Connection = cn_sql
com_sql.CommandType = CommandType.Text
com_sql.CommandText = "select count(*) from STAGIAIRE"
MsgBox(com_sql.ExecuteScalar)
```

L'exemple suivant permet l'exécution d'une procédure stockée :

```
Dim com_sql As New SqlClient.SqlCommand
Dim dr_sql As SqlClient.SqlDataReader

com_sql.Connection = cn_sql
com_sql.CommandType = CommandType.StoredProcedure
com_sql.CommandText = "sp_who"
dr_sql = com_sql.ExecuteReader
```

## 7.5 Objet DataReader

De manière générale, il existe deux types de résultat pour un objet command : soit il retourne un seul résultat (c'est le cas lorsque vous utilisez les méthodes ExecuteScalar ou ExecuteNonQuery), soit il retourne un ensemble d'enregistrements (méthode ExecuteReader).

L'objet DataReader permet de lire (seulement en avant) les enregistrements issus d'une requête.

| Propriété | Description |
|---|---|
| FieldCount | Nombre de champs (colonne) |
| HasRows | Détermine si le DataReader comporte 1 ou plusieurs lignes |

| | |
|---|---|
| RecordsAffected | Nombre de ligne affectée lors d'une opération en Transact SQL |

| Méthode | Description |
|---|---|
| Close | Ferme le DataReader et remplit les paramètres de retour |
| Read | Avance au prochain enregistrement |
| GetValue(i) | Retourne la valeur du champs à l'indice « i » |
| GetName(i) | Retourne le nom du champs à l'indice « i » |
| GetType(i) | Retourne le type du champs à l'indice « i » |

L'exemple suivant affiche tous les enregistrement du datareader en séparant chaque champs par un pipe « | ». Si aucun enregistrement n'est retourné, un message est affiché.

```
Dim com_sql As New SqlClient.SqlCommand
Dim dr_sql As SqlClient.SqlDataReader

com_sql.Connection = cn_sql
com_sql.CommandType = CommandType.Text
com_sql.CommandText = "select * from FORMATEUR"
dr_sql = com_sql.ExecuteReader

if not dr_sql.HasRows then msgBox(''Aucun enregistrement'')

Dim contenu As String
Dim i As Int16
Do While dr_sql.Read
   For i = 0 To dr_sql.FieldCount - 1
      contenu &= dr_sql.GetValue(i)
   Next
   contenu &= vbCr
Loop
MsgBox(contenu)
```

## 7.6 Objet DataSet

Un DataSet regroupe un ensemble de classe, collections et objets permettant de reproduire une source de données relationnelle avec des objets « Tables » et « Relation ». L'intérêt d'un tel objet est de pouvoir travailler sur des données sans pour autant être connecté à la base ce qui permet une meilleur monter en charge des bases de données qui se verront soulagées d'un grand nombre d'opérations.

La création d'un DataSet se fait par simple déclaration en spécifiant son nom. Attention, deux DataSet ne peuvent avoir le même nom.

```
Dim monds as DataSet
Monds = new DataSet("monds")
```

Une fois créé, vous devez lui ajouter des relations  et des tables contenant elle même des champs.

| Méthode | Description |
|---|---|
| AcceptChanges | Valide toutes les modifications effectuées dans le DataSet |
| RefuseChanges | Refuse toutes les modifications effectuées dans le DataSet |

| Propriétés | Description |
|---|---|
| HasChanges | Retourne vrai si le DataSet contient des tables dont les enregistrements ont été modifiés, supprimés, ajoutés |

### 7.6.1 Objet DataTable

L'objet DataTable correspond à une table.

| Propriété | Description |
|---|---|
| Columns | Collection des colonnes de la table |
| Constraints | Collection des contraintes de la table |
| Rows | Collection des lignes de la table |
| MinimumCapacity | Taille initiale pour les enregistrements |
| CaseSensitive | Mode de comparaison (Respect de la casse ou non) |
| PrimaryKey | Tableau de colonnes faisant parti de la clé primaire |

L'exemple suivant crée un objet DataTable en spécifiant ces paramètres et le lie au DataSet

```
Dim donnees As New DataSet("donnees")
Dim personne As New DataTable("personne")
personne.CaseSensitive = False
personne.MinimumCapacity = 50
donnees.Tables.Add(personne)
```

### 7.6.2 Objet DataColumn

Les objets DataColumn correspondent aux différentes colonnes d'une table. En créant une colonne, vous devez spécifier plusieurs informations :

- Le nom de la colonne
- Le type de la colonne (ci dessous figure la liste des types utilisables

| Type | Description |
|---|---|
| Boolean | Valeur booléene |
| Byte | Octets (Entier de 0 à 254) |
| Char | Représente 1 caractère |
| DateTime | Heure & Date |
| Decimal | Décimal |
| Double | Nombre à virgule flottante à double précision |
| Int16 | Entier signé de XX Bits |

| | |
|---|---|
| Int32<br>Int64 | |
| SByte | Entier compris entre –127 et 128 |
| Single | Nombre à virgule flottante |
| String | Chaîne de caractère |
| TimeSpan | Représente un intervalle de temps |
| UInt16<br>UInt32<br>UInt64 | Entier non signé codé sur 16 bits |

- Les propriétés de la colonne

| Propriété | Description |
|---|---|
| AllowDBNull | Autorise la valeur Null |
| AutoIncrement | Définit si la colonne est un compteur dont la valeur s'incrémente automatiquement (Attention, le champs doit être de type Integer ) |
| AutoIncrementSeed | Valeur de départ pour l'autoIncrement |
| AutoIncrementStep | Pas de l'incrément |
| Defaultvalue | Valeur par défaut de la colonne |
| MaxLength | Longueur maximale pour le champs texte |
| Unique | Définit si les valeurs de la colonne doivent être uniques |

L'exemple suivant ajoute des champs à la table « personne » en utilisant les différentes propriétés et spécifie la clé primaire :

```
Dim donnees As New DataSet("donnees")
Dim personne As New DataTable("personne")
personne.CaseSensitive = False
personne.MinimumCapacity = 50
donnees.Tables.Add(personne)

Dim col1 As New Data.DataColumn("pers_num", GetType(Integer))
col1.AutoIncrement = True
col1.AutoIncrementSeed = 1
col1.AutoIncrementStep = 1
donnees.Tables("personne").Columns.Add(col1)
donnees.Tables("personne").PrimaryKey = New DataColumn() {col1}

Dim col2 As New Data.DataColumn("pers_nom", GetType(String))
col2.Unique = True
col2.MaxLength = 255
col2.DefaultValue = "inconnu"
donnees.Tables("personne").Columns.Add(col2)
```

### 7.6.3 Objet DataRelation

Un DataSet est une représentation en objets d'une base de données. Un des points fondamental des bases de données est la mise en place de relation permettant ensuite de vérifier la cohérence des données saisies dans plusieurs tables.

La création d'une relation se fait par rapport à un DataSet en créant un objet DataRelation et en spécifiant les champs des tables qui sont liés. Une relation concerne deux champs. Lors de la création de la relation, vous devez spécifier son nom, le champs « clé primaire » et le champs « clé étrangère ». L'exemple suivant crée les tables « personne » et « categorie » et met en place une relation entre les deux :

```
Dim donnees As New DataSet("donnees")

'creation de la table personne et ajout des champs
Dim personne As New DataTable("personne")
personne.CaseSensitive = False
personne.MinimumCapacity = 50
donnees.Tables.Add(personne)

Dim col1 As New Data.DataColumn("pers_num", GetType(Integer))
col1.AutoIncrement = True
col1.AutoIncrementSeed = 1
col1.AutoIncrementStep = 1
donnees.Tables("personne").Columns.Add(col1)
donnees.Tables("personne").PrimaryKey = New DataColumn() {col1}

Dim col2 As New Data.DataColumn("pers_nom", GetType(String))
col2.Unique = True
col2.MaxLength = 255
col2.DefaultValue = "inconnu"
donnees.Tables("personne").Columns.Add(col2)

Dim col3 As New Data.DataColumn("pers_cat_num", GetType(Integer))
donnees.Tables("personne").Columns.Add(col3)

'creation de la table catégorie et ajout des champs
Dim categorie As New DataTable("categorie")
categorie.CaseSensitive = False
categorie.MinimumCapacity = 50
donnees.Tables.Add(categorie)

Dim col4 As New Data.DataColumn("cat_num", GetType(Integer))
col4.AutoIncrement = True
col4.AutoIncrementSeed = 1
col4.AutoIncrementStep = 1
donnees.Tables("categorie").Columns.Add(col4)
donnees.Tables("categorie").PrimaryKey = New DataColumn() {col4}

Dim col5 As New Data.DataColumn("cat_lib", GetType(String))
col5.MaxLength = 50
```

```
        donnees.Tables("categorie").Columns.Add(col5)

        'creation de la relation
        Dim rel As New DataRelation("personne_categorie",
donnees.Tables("categorie").Columns("cat_num"),
donnees.Tables("personne").Columns("pers_cat_num"))
        donnees.Relations.Add(rel)

        'Ajout des contraintes
        Dim fk As ForeignKeyConstraint = rel.ChildKeyConstraint
        fk.DeleteRule = Rule.None
        fk.UpdateRule = Rule.None
```

Notez que la dernière partie définit le comportement des enregistrements de la relation lors d'opération d'insertion ou de suppression : dans le cas précédent, la suppression d'une catégorie de personne sera refusée si elle contient des personnes.

### 7.6.4 Travailler avec les données

Une fois que la structure du DataSet est en place, vous devez être en mesure d'ajouter des données aux DataTable. Chaque DataTable contient une collection de DataRow qui correspond aux différents enregistrements de la table.

#### *7.6.4.1 Parcourir les données*

Le parcourt des données permet de lire les enregistrements stockés dans le DataTable. Il existe deux modes de parcourt :

- Le parcourt linéaire qui consiste à accéder aux enregistrements d'une seule table à partir de leurs indices (éléments inexistants dans les bases de données relationnelles)
- Le parcourt hiérarchique permettant d'utiliser les relations mises en place entre les tables d'un DataSet

L'exemple suivant affiche tous les éléments de la table « personne » :

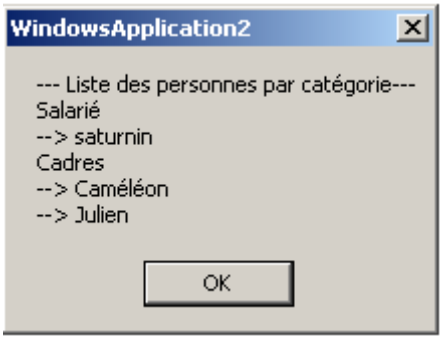
```
        Dim i As Int16
        Dim contenu As String
        With donnees.Tables("personne")
            For i = 0 To .Rows.Count - 1
                contenu &= .Rows(i).Item("pers_num") & " - " & .Rows(i).Item("pers_nom") &
vbCr
            Next
        End With
        MsgBox(contenu)
```

Le résultat obtenu :


Le second exemple permet d'afficher pour chaque catégorie, toutes les personnes correspondantes. Le code va dans un premier temps parcourir toutes les catégories et, avec la méthode « GetChildRows » récupérer tous les enregistrements correspondant dans la table personne. Cette méthode prend en argument la relation pour laquelle vous souhaitez récupérer les enregistrements enfants.

```
Dim contenu2 As String = "--- Liste des personnes par catégorie---" & vbCr
Dim elt As DataRow
With donnees.Tables("categorie")
   For i = 0 To .Rows.Count - 1
      contenu2 &= .Rows(i).Item("cat_lib") & vbCr
      For Each elt In .Rows(i).GetChildRows("personne_categorie")
         contenu2 &= "--> " & elt.Item("pers_nom") & vbCr
      Next
   Next
End With
MsgBox(contenu2)
```

Le résultat obtenu :

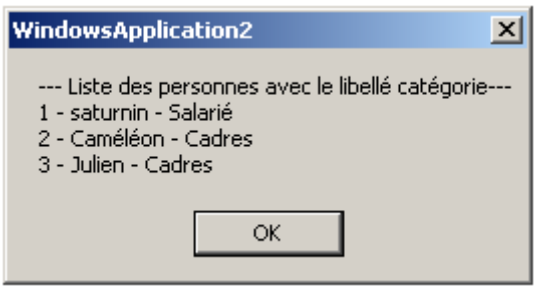



A l'inverse, il est possible de récupérer l'enregistrement « parent » d'un enregistrement faisant parti d'une relation avec la méthode « GetParentRow ». Cette méthode prend également en paramètre la relation pour laquelle vous souhaitez récupérer l'enregistrement père. L'exemple suivant affiche la liste des personnes en affichant également le libellé de la catégorie à laquelle ils appartiennent :

```
Dim contenu3 As String = "--- Liste des personnes avec le libellé catégorie---" & vbCr
With donnees.Tables("personne")
   For i = 0 To .Rows.Count - 1
```

```
            contenu3 &= .Rows(i).Item("pers_num") & " - " & .Rows(i).Item("pers_nom") & "
- " & .Rows(i).GetParentRow("personne_categorie").Item("cat_lib") & vbCr
        Next
    End With
    MsgBox(contenu3)
```

Le résultat obtenu :




### *7.6.4.2 Insertion de données*

Pour insérer un nouvel enregistrement, vous devez créer un objet DataRow à partir du Datatable, configurer les valeurs des différentes colonnes et enfin ajouter le DataRow au DataTable.

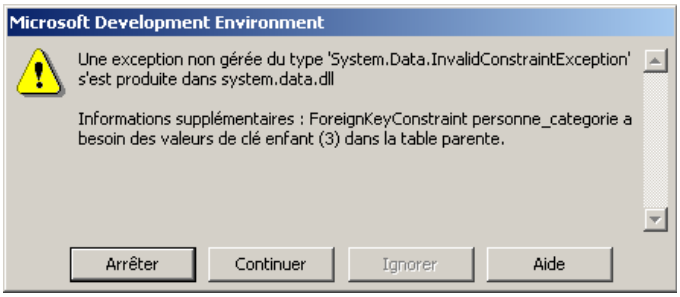
L'exemple suivant ajoute un enregistrement dans la table categorie :

```
    Dim dr As DataRow = donnees.Tables("categorie").NewRow()
    dr("cat_lib") = "Salarié"
    donnees.Tables("categorie").Rows.Add(dr)
```

L'exemple suivant ajoute une personne en lui attribuant la catégorie no 3 (catégorie inexistante !) :

```
    Dim dr2 As DataRow = donnees.Tables("personne").NewRow()
    dr2("pers_nom") = "Victor"
    dr2("pers_cat_num") = "3"
    donnees.Tables("personne").Rows.Add(dr2)
```

A ce moment, une exception est levée car les enregistrements ne respectent pas la relation « personne_categorie » :




#### *7.6.4.3 Modification de données*

La modification de données est possible en modifiant la collection Rows de l'objet DataTable. La modification d'un enregistrement se fait en 3 étapes :

- Appel de la méthode « BeginEdit » sur l'objet DataRow : cette méthode permet de commencer une opération de modification
- Modification des données
- Appel de la méthode « EndEdit » pour valider les modifications ou « CancelEdit » pour les annuler.

L'exemple suivant modifie le premier enregistrement de la table « Personne » :

```
With donnees.Tables("personne").Rows(0)
   .BeginEdit()
   .Item("pers_nom") = "saturnin"
   .EndEdit()
End With
```

#### *7.6.4.4 Suppression de données*

La suppression de données consiste à supprimer des Objets DataRow de l'objet DataTable. Pour cela, on utilise la méthode « Delete » de l'objet DataRow.

L'exemple suivant permet de supprimer une personne :

```
donnees.Tables("personne").Rows(0).Delete()
```

### 7.6.5 Objet DataView

L'objet DataView permet d'extraire, filtrer et trier des données issues d'un DataTable. De plus, il est possible de modifier, ajouter, supprimer des données directement à partir du DataView et les modifications seront automatiquement retranscrites dans le DataTable lié. Un DataView peut être vu comme une vue au niveau base de données.

| Propriété | Description |
|---|---|
| AllowDelete | Autorise la suppression d'enregistrements |
| AllowEdit | Autorise la modification d'enregistrements |

| | |
|---|---|
| AllowNew | Autorise l'ajout d'enregistrements |
| Count | Nombre d'enregistrement |
| RowFilter | Définit un filtre pour les données : chaîne de texte équivalente à la clause « where » d'une requête SQL |
| Sort | Définit le tri : Equivalant à la clause « order by » d'une requête SQL |

| Méthode | Description |
|---|---|
| Addnew | Ajoute un enregistrement |
| Delete | Supprime un enregistrement |
| Find | Retourne l'indice de l'enregistrement correspondant aux paramètres de recherche par rapport au champs spécifié dans l'attribut « sort » |
| FindRows | Retourne un ensemble de DataRow correspondant à la recherche. |

L'exemple suivant crée un DataView à partir du DataTable personne et ne garde que les personnes dont le numéro de catégorie est 1. les enregistrements sont classés par nom décroissant. Enfin, nous recherchons l'enregistrement dont le nom est « Julien » et nous affichons les informations.

```
Dim dv As New DataView
dv.Table = donnees.Tables("personne")
dv.AllowDelete = False
dv.AllowNew = False
dv.AllowEdit = True
dv.Sort = " pers_nom"

Dim indice As Integer
indice = dv.Find("Julien")
MsgBox(dv(i).Item("pers_nom"))
```

#### 7.6.6 Les évènements

La modification des données d'un DataTable entraîne la levée de plusieurs événements qui permettront par exemple de valider les valeurs saisies dans les enregistrements.

| Evénement | Description |
|---|---|
| ColumnChanged | Valeur de colonne changée |
| ColumnChanging | Valeur de colonne en cours de modification |
| RowChanged | Ligne de la table modifiée |
| RowChanging | Ligne de la table en cours de modification |
| RowDeleted | Ligne de la table supprimée |
| RowDeleting | Ligne de la table en suppression |

## 7.7 Objet DataAdapter

Les Objets DataSet et DataTable permettent de définir la structure des données au sein d'une application comme mode temporaire de stockage. Ceux ci peuvent ensuite être alimentés en données par un fichier texte par exemple. La réalité est cependant différente, la plupart des applications nécessitent l'interrogation et le stockage des informations dans une base de données distante (Oracle, Sql Server) ou local (Access).

L'objet DataAdapter est un connecteur entre la source de données et l'objet DataSet. L'intérêt d'un tel composant est de pouvoir dissocier la zone de stockage des donnéees (Une BDR par exemple) de la zone de travail (Le DataSet).

Son utilisation se fait en plusieurs étapes :

- Connection à la source de données
- Récupération des données de la base et insertion dans les DataTable
- Déconnexion de la source de données
- Modification des données par l'application
- Connexion à la source de données
- Envoi des modifications effectuées vers la source

### 7.7.1 Création

Lors de la création d'un DataAdapter, il est nécessaire de spécifier la connexion utilisée ainsi que la requête Select. En fonction du Provider utilisé, l'objet DataAdapter sera différent car il doit être en mesure de modifier les données d'une source.
2 objets existent :

- OleDbDataAdapter
- SqlDataAdapter

```
Dim da As New OleDb.OleDbDataAdapter("select * from Table1", cn)
```

Ainsi, le DataAdapter sait sur quelles données il devra travailler.

### 7.7.2 Importer des données

L'importation des données consiste à remplir les DataTable d'une DataSet à partir d'un DataAdapter défini.

#### *7.7.2.1 Remplir un DataSet*

Le remplissage des éléments d'un DataSet se fait à l'aide de la méthode Fill de l'objet DataAdapter. Plusieurs configurations sont possibles pour la récupération des données en raison de la surcharge de la méthode Fill :

Quelque soit la surcharge utilisée, la méthode Fill a la capacité d'ouvrir et de fermer la connexion utilisée pour la création du DataAdapter. Attention, si vous accéder plusieurs à des tables via un DataAdapter, il est préférable d'ouvrir et fermer vous même la connexion.

- Remplir un DataTable existant

La première possibilité consiste à remplir une DataTable déjà existant : dans ce cas, la structure du DataTable doit correspondre au jeu de résultat retourné. Nous verrons plus tard qu'il est possible de passer outre cette possibilité en utilisant un Mappage des données. La méthode Fill prend en paramètres un DataSet ainsi que le nom de la table à remplir.

```
'definition dataset et datatable
Dim ds As New DataSet("monds")
Dim t As New DataTable("matable")
Dim col1 As New Data.DataColumn("num", GetType(Integer))
Dim col2 As New Data.DataColumn("lib", GetType(String))
t.Columns.Add(col1)
t.Columns.Add(col2)
ds.Tables.Add(t)

'connexion (définition mais pas ouverture)
Dim cn As New OleDb.OleDbConnection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\bd2.mdb;"

'remplissage datatable à partir du DataAdapter
Dim da As New OleDb.OleDbDataAdapter("select * from Table1", cn)
da.Fill(ds, "matable")
```

- Créer et remplir un nouveau DataTable

La seconde possibilité permet de laisser au DataAdapter le soin de créer le DataTable ainsi que sa structure. Dans ce cas, il suffit d'appeler la méthode Fill en passant en paramètre le nom d'un DataTable inexistant.

```
'definition dataset et datatable
Dim ds As New DataSet("monds")

'connexion (définition mais pas ouverture)
Dim cn As New OleDb.OleDbConnection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\bd2.mdb;"

'remplissage datatable à partir du DataAdapter
Dim da As New OleDb.OleDbDataAdapter("select * from Table1", cn)
da.Fill(ds, "matable")
```

- Récupérer un certain nombre d'enregistrements

Il est possible lors de l'appel de la méthode Fill de spécifier quelles sont les lignes de la requête à récupérer. Pour cela, on utilise deux paramètres qui sont l'indice de départ et le nombre maximal d'enregistrement à récupérer. Cette signature permet en particulier de générer un affichage par page en récupérant tour à tour les 10 premiers enregistrements, les 10 suivants etc…

L'exemple suivant récupère 10 enregistrements à partir du 50 ème.

```
Dim da As New OleDb.OleDbDataAdapter("select * from Table1", cn)
da.Fill(ds, 50, 10, "toto")
```

- Remplissage de Datatable multiples

Il est également possible de remplir plusieurs DataTable à partir d'un seul DataAdapter en passant en paramètres plusieurs requêtes séparées par des points virgules :

Attention, lors de la création des DataTable, la méthode Fill les nomme « Table1 », Table2 … Il est donc vivement conseillé de renommer les tables après exécution.

```
'connexion (définition mais pas ouverture)
Dim cn As New OleDb.OleDbConnection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\bd2.mdb;"

'remplissage datatable à partir du DataAdapter
Dim da As New OleDb.OleDbDataAdapter("select * from Client;Select * from
Produit;Select * from Commande", cn)
da.Fill(ds, "matable")
ds.Tables(1).TableName = “Client”
ds.Tables(2).TableName = “Produit”
ds.Tables(3).TableName = “Commande”
```

#### *7.7.2.2 Mappage des données*

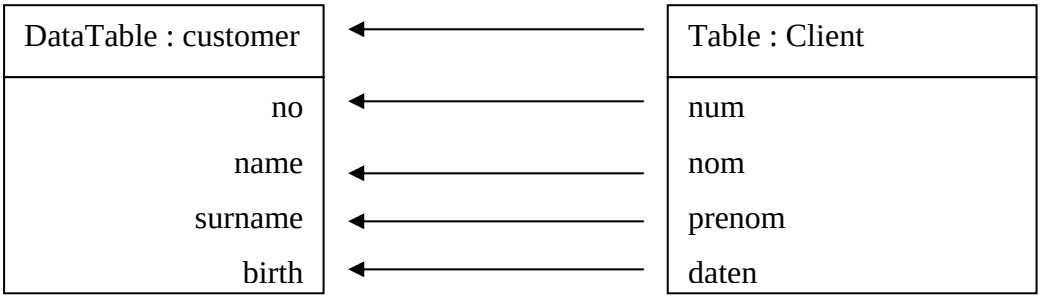
Jusqu'à présent, pour remplir un DataTable déjà existant, nous devions une structure identique à celle définit dans la source de données, en particulier au niveau des noms et des types de champs. Le mappage des données permet de définir pour chaque table et colonne de la source leur correspondance au niveau du DataTable.

Une telle méthode peut être utile afin de :

- Modifier le nom d'une table
- Renommer les champs d'une table pour améliorer la lisibilité du code
- Récupérer des informations pour des colonnes ne portant pas de nom (les opérations d'aggrégation par exemple)

La définition d'un mappage se fait au niveau du DataAdapter par le biais de la collection « TableMapping » et « «ColumnMapping » qui contiennent les équivalences en terme de tables et de champs.

L'exemple suivant définit un mappage correspondant au schéma ci dessous :




```
'definition dataset et datatable
Dim ds As New DataSet("monds")
Dim t As New DataTable("customer")
Dim col1 As New Data.DataColumn("name", GetType(String))
Dim col2 As New Data.DataColumn("surname", GetType(String))
Dim col3 As New Data.DataColumn("birth", GetType(Date))
Dim col4 As New Data.DataColumn("no", GetType(Int32))
t.Columns.Add(col1)
t.Columns.Add(col2)
t.Columns.Add(col3)
t.Columns.Add(col4)
ds.Tables.Add(t)

'connexion
Dim cn As New OleDb.OleDbConnection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\Support\vbnet\bd2.mdb;"

'definition du DataAdapter
Dim da As New OleDb.OleDbDataAdapter("select num, nom, prenom, daten from
Client", cn)

'définition des mappings de table & champs
da.TableMappings.Add("Client", "customer")
da.TableMappings(0).ColumnMappings.Add("num", "no")
da.TableMappings(0).ColumnMappings.Add("nom", "name")
da.TableMappings(0).ColumnMappings.Add("prenom", "surname")
da.TableMappings(0).ColumnMappings.Add("daten", "birth")

'remplissage
da.MissingMappingAction = MissingMappingAction.Ignore
da.Fill(ds.Tables("customer"))
```

#### *7.7.2.3 Importer la structure*

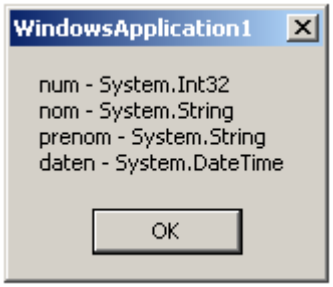
Dans certains cas, il peut être interressant de ne récupérer que la structure d'une source de données sans pour autant récupérer les enregistrements qui la composent. Par exemple, un programme ne faisant que de l'insertion aura besoin de la structure mais pas des données.
Il faut dans ce cas, utiliser la méthode FillSchema et passer les arguments suivants :

| Paramètre | Description |
|---|---|
| DataSet | Le dataSet vers lequel exporter les données |
| SchemaType | Spécifie si les mappings définit doivent être utilisés :<br>SchemaType.Mapped → oui<br>SchemaType.Source → non |
| SrcTable | Nom du DataTable vers lequel envoyer la structure |

L'exemple suivant importe la structure de la table « Client » vers le DataSet « ds » et affiche la structure de la table :

```
'connexion
Dim cn As New OleDb.OleDbConnection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\bd2.mdb;"

'definition du DataAdapter & DataSet
Dim da As New OleDb.OleDbDataAdapter("select * from Client", cn)
Dim ds As New DataSet("monds")
da.FillSchema(ds, SchemaType.Source, "dt_client")

'affichage de la structure de la table
Dim chps As DataColumn
Dim contenu As String
For Each chps In ds.Tables("dt_client").Columns
   contenu &= chps.ColumnName & " - " & chps.DataType.ToString & vbCr
Next
MsgBox(contenu)
```

Résultat obtenu :




### 7.7.3 Exporter des données

Dans la plupart des applications exploitant les bases de données, vous devrez tôt ou tard mettre à jour la source de données à partir des données modifiées dans un DataSet ou un DataReader : c'est également le rôle du DataAdapter qui doit pouvoir mettre à jour les enregistrement insérés, modifiés ou supprimés.

Dans un premier temps, vous devrez définir le comportement du DataAdapter pour chaque type de modification sur la table du DataSet (Insert, Update, Delete). Il est également possible d'effectuer un mappage pour les données du DataSet et de la source.

Cette mise à jour peut cependant lever des conflits au niveau des enregistrements : imaginez un enregistrement que vous modifiez et qui est supprimé par un utilisateur juste avant la demande de mise à jour ! Pour réglez ces conflits, le DataAdapter possède un ensemble d'objets et d'évènements permettant de cibler et résoudre les problèmes.

### *7.7.3.1 Mise à jour de la source de données*

- Propriétés utilisées

Afin de mener à bien les opérations de mise à jour vers la source de données, l'objet DataAdapter doit être en mesure de connaître l'état des enregistrement (DataRow) des différents DataTable.

Pour cela, chacune des lignes possède la propriété « RowState » pouvant prendre les valeurs suivantes :

| Valeur | Description |
|---|---|
| 1 | Detached : la ligne n'est encore intégrée à aucun DataTable |
| 2 | Unchanged : Aucun changement depuis le dernier appel de AcceptChanges |
| 4 | Added : la ligne a été ajoutée |
| 8 | Deleted : La ligne a été supprimée |
| 16 | Modified : La ligne a été modifiée |

### *7.7.3.1 Définition des requêtes d'actualisation*

En fonction de l'état des lignes, l'objet DataAdapter exécutera des requêtes différentes afin de mettre à jour la source de données. Le comportement du DataAdapter est définit par trois objets command :

- InsertCommand
- UpdateCommand
- DeleteCommand

Pour créer ses commandes, vous pouvez soit utiliser la classe CommandBuilder correspondant à votre fournisseur de données qui générera automatiquement les commandes ou alors les paramétrer vous même :

- **Classe CommandBuilder**

L'exemple suivant crèe les commandes d'actualisation à partir de la classe CommandBuilder :

```
        'connexion
        Dim cn As New OleDb.OleDbConnection
        cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\Support\vbnet\bd2.mdb;"

        'dataset, dataadapter
        Dim ds As New DataSet
        Dim da As New OleDb.OleDbDataAdapter("select * from Client", cn)
        da.Fill(ds, "Client")

        'creation des commandes
        Dim cmdbuilder As New OleDb.OleDbCommandBuilder(da)
        da.InsertCommand = cmdbuilder.GetInsertCommand
        da.DeleteCommand = cmdbuilder.GetDeleteCommand
        da.UpdateCommand = cmdbuilder.GetUpdateCommand
```

Les commandes créées sont des requêtes SQL contenant plusieurs paramètres (représentés par des ?).

Dans le cas d'une commande Delete ou Update, les commandes générés prennent en compte le fait que l'enregistrement cible à mettre à jour doit avoir les même valeurs que lors de la lecture initiale. Si une des valeurs de l'enregistrement à été modifiée sur la source de données, la mise à jour ne sera pas validée et une erreur sera retournée.

Voici les commandes générées par la classe CommandBuilder :

- Commande Insert

```
INSERT INTO Client( nom , prenom , daten ) VALUES ( ? , ? , ? )
```

Notez que dans le cas d'une clé primaire avec numéroauto ou AutoIncrement, le champs n'est pas spécifiée car il est généré par la base de données.

- Commande Delete

```
DELETE FROM  Client WHERE ( (num = ?) AND ((? = 1 AND nom IS NULL) OR (nom =
?)) AND ((? = 1 AND prenom IS NULL) OR (prenom = ?)) AND ((? = 1 AND daten IS
NULL) OR (daten = ?)) )
```

- Commande Update

```
UPDATE Client SET nom = ? , prenom = ? , daten = ? WHERE ( (num = ?) AND ((? = 1
AND nom IS NULL) OR (nom = ?)) AND ((? = 1 AND prenom IS NULL) OR (prenom = ?))
AND ((? = 1 AND daten IS NULL) OR (daten = ?)) )
```

- **Générer ses propres commandes**

De manière générale, la commande d'insertion correspond aux besoins de l'entreprise. Cependant, concernant les commandes Update et Delete, elles imposent que l'enregistrement à mettre à jour n'est pas été modifié entre sa lecture et sa mise à jour par votre application.

Dans l'exemple suivant, la commande de suppression ne filtre les enregistrements que par rapport à leur clé primaire : ainsi, si un autre utilisateur a modifié les enregistrements entre temps, la suppression se fera quand même.

```
        'creation de la commande personnalisée de suppression
        Dim cmddelete As New OleDb.OleDbCommand("Delete from Client where
num = ?", cn)
        'definition du parametre numéro
        With cmddelete.Parameters.Add("@p1", GetType(Integer))
            .SourceColumn = "num"   'colonne liée
            .SourceVersion = DataRowVersion.Original 'valeur à récupérer
        End With
        'définition de la requete comme DeleteCommand du DataAdapter
        da.DeleteCommand = cmddelete
```

### *7.7.3.2 Déclencher la mise à jour des données*

Pour déclencher la mise a jour des données, il suffit d'appeler la méthode Update de l'objet DataAdapter en passant en paramètre le DataSet ainsi que le nom de la table du DataSet à valider.

```
        da.Update(ds, "Client")
```

Afin de ne pas lancer de mise à jour injustifiée, la propriété « HasChanges » de l'objet DataSet peut être consultée :

```
        If ds.HasChanges Then
            da.Update(ds, "Client")
        Else
            MsgBox("aucun changement")
        End If
```

Il est également possible de récupérer à partir d'un DataTable uniquement les lignes ayant subies des modifications (ajout ou suppression ou modification). De cette façon, vous êtes en mesure d'actualiser qu'une partie des lignes modifiées.

La méthode GetChanges des objets DataTable et DataSet permet de ne récupérer que les enregistrement modifiés.

L'exemple ci dessous ne met à jour que les enregistrements supprimés de la table « Client » : pour cela, on utilise la méthode « GetChanges » permettant de ne récupérer que les enregistrement ayant subit des modifications (ajout, suppression, modification).

```
        'maj des enregistrements supprimés :
        Dim tbl2 As DataTable =
ds.Tables("Client").GetChanges(DataRowState.Deleted)
```

```
        da.Update(tbl2)
```

Une fois la mise à jour effectuée, la propriété « RowState » des différents DataRow concernés est passée à « UnChanged » spécifiant que la ligne est validé.

#### *7.7.3.3 Gestion des conflits*

Lors de la mise à jour de la base de données avec la méthode Update, plusieurs conflits peuvent apparaître. Un conflit est une exception de mise à jour de la source.

```
Try
        Da.update(ds, ‘’Clients’’)
Catch ex as Exceptio
        Msgbox(‘’Erreur de mise à jour ‘’)
End Try
```

Dans le cas d’une erreur de mise à jour, la mise à jour des lignes suivantes est ommise. Afin de forcer la mise à jour de toutes les lignes (même celles suivants un conflit), vous devez paramétrer le comportement du DataAdapter :

```
Da.ContinueUpdateOnErrors = true
```

- **Afficher les lignes conflictuelles**

Après une mise à jour, toutes les lignes voit leur propriété « RowState » à « Unchanged » sauf les lignes conflictuelles : la méthode consiste donc à parcourir les RowState et récupérer seulement les lignes dont « RowState » est différent de « UnChanged ».

L’exemple suivant illustre cette méthode :

```
        da.ContinueUpdateOnError = True
        da.Update(ds, "Client")

        'affichage des lignes conflictuelles
        chaine = ""
        If ds.HasChanges Then
            Dim dr As DataRow
            For Each dr In ds.Tables("Client").Rows
                If dr.RowState <> DataRowState.Unchanged Then
                    Select Case dr.RowState
                        Case DataRowState.Deleted
                            dr.RejectChanges()  'retablit l'état initial
afin de pouvoir accéder à la PK
                            chaine &= "Suppression impossible pour la clé "
& dr.Item(0)
                        Case DataRowState.Modified
                            chaine &= "Modification impossible pour la clé
" & dr.Item(0)
                    End Select
                End If
            Next
            MsgBox(chaine)
```

```
        Else
            MsgBox("Toutes les mises à jour ont été effectuées")
        End If
```

La seconde méthode consiste à récupérer les enregistrements non validés par la méthode Update par l'intermédiaire de la méthode « GetChanges » : vous créez à partir de cette méthode un nouvel objet « DataTable » et vous pouvez ensuite parcourir les enregistrements de cette table afin d'identifier les conflits.

- **Evénements et propriétés liés**

Pour la gestion des conflits, il est également possible d'utiliser les évènements « RowUpdating » et «RowUpdated » respectivement déclenchés avant et après la mise à jour d'un enregistrement (c'est à dire l'envoi de la commande correspondante).

L'événement RowUpdating est utilisé pour prévenir les sources de conflits : par exemple, dans une application ou deux utilisateurs peuvent simultanément modifier différents champs d'un même enregistrement, un conflit sera automatiquement déclenché pour le dernier faisant la mise à jour. L'événement RowUpdating peut vous permettre de modifier l'UpdateCommande et n'y inclure que les champs modifiés (ceux dont l'ancienne valeur est différente de la nouvelle).

L'événement RowUpdated quand à lui permet de résoudre les conflits une fois que ceux ci sont déclenchés afin de ne pas avoir d'autres répercutions sur les enregistrements suivants.

Ces deux évènements prennent en paramètre « e » de type System.Data.FillErrorEventArgs :

| Propriété | Description |
|---|---|
| StatementType | Type de la commande |
| Command | Objet commande |
| Row | Objet DataRow à actualiser |
| Status | Règle de gestion des lignes suivantes |
| Errors | Erreurs générées (seulement dans RowUpdated) |

L'exemple suivant utilise l'événement « RowUpdated » afin de résoudre les conflits de maj lorsqu'ils interviennent :

```
    Private Sub da_RowUpdated(ByVal sender As Object, ByVal e As
System.Data.OleDb.OleDbRowUpdatedEventArgs) Handles da.RowUpdated
        'gestion des conflits de maj
        If e.Row.RowState <> DataRowState.Unchanged Then
            Select Case e.Row.RowState
                Case DataRowState.Deleted
                    If Not TypeOf e.Errors Is DBConcurrencyException Then
                        'erreur d'integrite referentielle
                        MsgBox("Erreur suppression" & vbCr & "type erreur =
" & e.Errors.Message)
                    Else
                        'conflit d'actualisation, l'enregistrement n'existe
plus
```

```
                        MsgBox("L'enregistrement à supprimer n'existe plus
dans la source de données")
                    End If

                Case DataRowState.Modified
                    If Not TypeOf e.Errors Is DBConcurrencyException Then
                        'erreur d'integrite referentielle
                        MsgBox("Erreur de mise à jour" & vbCr & "type
erreur = " & e.Errors.Message)
                    Else
                        'conflit d'actualisation, l'enregistrement n'existe
plus
                        MsgBox("L'enregistrement à mettre à jour n'existe
plus dans la source de données")
                    End If

            End Select
        End If
    End Sub
```

- **Revenir en arriere sur plusieurs actualisations**

Dans le cas d'un problème d'actualisation bloquant, il peut être interressant d'annuler toutes les actualisations déjà réalisées afin de laisser la base de données dans un état cohérent. Pour annuler toutes les actualisations déjà faîtes, il suffit de lier les commandes à une transaction (regroupement de commande) car l'objet transaction possède les méthodes « commit » et « rollback » qui permettent respectivement de valider et d'annuler toutes les commandes d'une transaction.

L'exemple suivant effectue les mises à jour et annule l'ensemble des commandes si une erreur survient.

```
        'connexion
        cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\Support\vbnet\bd2.mdb;"

        'dataset, dataadapter
        Dim ds As New DataSet
        da = New OleDb.OleDbDataAdapter("select * from Client", cn)
        da.Fill(ds, "Client")

        'creation des commandes et rattachement à la transaction
        Dim cmdbuilder As New OleDb.OleDbCommandBuilder(da)
        da.InsertCommand = cmdbuilder.GetInsertCommand
        da.DeleteCommand = cmdbuilder.GetDeleteCommand
        da.UpdateCommand = cmdbuilder.GetUpdateCommand

        'transaction & initialisation
        Dim transac As OleDb.OleDbTransaction
        cn.Open()
        transac = cn.BeginTransaction

        da.InsertCommand.Transaction = transac
        da.DeleteCommand.Transaction = transac
        da.UpdateCommand.Transaction = transac
```

```
        'modification des données
        ds.Tables("Client").Rows(0).Delete()
        ds.Tables("Client").Rows(1).Item(1) = "tutu"

        'effectue la mise à jour
        da.ContinueUpdateOnError = True
        da.Update(ds, "Client")

        'si erreur, on annnule tout
        If ds.HasChanges Then
            MsgBox("Erreur(s) lors de la mise à jour. Toutes les opérations
sont annulées.")
            transac.Rollback()
        Else
            MsgBox("Mise à jour réussie !")
            transac.Commit()
        End If

        'fermeture de la connexion
        cn.Close()
```

## 7.8 Liaison de données aux contrôles

La liaison de données permet de lier un contrôle de formulaire à 1 ou plusieurs champs d'une source de données.

Deux types de liaison de données existent :

- Liaison de données simple : une seule propriété du contrôle est liée à une colonne de la source de données (exemple : champs texte)

- Liaison de données complexe : plusieurs propriétés du contrôle sont liées à une ou plusieurs colonnes de la source de données (exemple : liste déroulante)

De manière générale, le fonctionnement est le suivant

- Créer un objet connexion
- Créer un objet DataAdapter
- Créer l'objet DataSet et le remplir à l'aide du DataAdapter
- Lier les propriétés des contrôles aux champs du DataSet
- Manipuler et naviguer entre les enregistrements avec l'objet BindingManagerBase

### 7.8.1 Objets utilisés

Lors de la mise en place de contrôles liés, les objets suivants sont mis à contribution :

#### *7.8.1.1 DataBinding*

Cet objet permet de lier une propriété d'un contrôle à un champs de la source de données.

Au niveau d'un contrôle, la collection « DataBindings » contient l'ensemble des liens « source de données » vers « propriété » du contrôle. Ainsi, lorsque vous souhaitez lier par le code un contrôle à un champs, utilisez la méthode Add.

Le code suivant lie la propriété « text » du contrôle textbox1 au champs « nom » de la table « Client » du DataSet11 :

```
Me.TextBox4.DataBindings.Add("text", Me.DataSet11, "Client.nom")
```

#### *7.8.1.2 ControlBindingCollection*

Cet objet contient l'ensemble des objets Binding pour un contrôle (par exemple dans le cas d'un TextBox dont le contenu serait définit par le champs « nom » et la couleur de fond par le champs « color »).

#### *7.8.1.3 BindingManagerBase*

Cet objet contient l'ensemble des objets Binding liés à la même source de données. C'est grâce à lui que vous pourrez par exemple parcourir les données (suivant, précédent), ajouter ou supprimer des enregistrements etc…

| Propriété | Description |
|---|---|
| Count | Nombre de ligne gérées par le BMB |
| Position | Indice de la ligne en cours |

| Méthode | Description |
|---|---|
| AddNew | Ajoute un nouvel enregistrement |
| CancelCurrentEdit | Annule l'édition ou l'ajout en cours |
| RemoveAt(indice) | Supprime l'enregistrement à la position Indice |

#### *7.8.1.4 BindingContext*

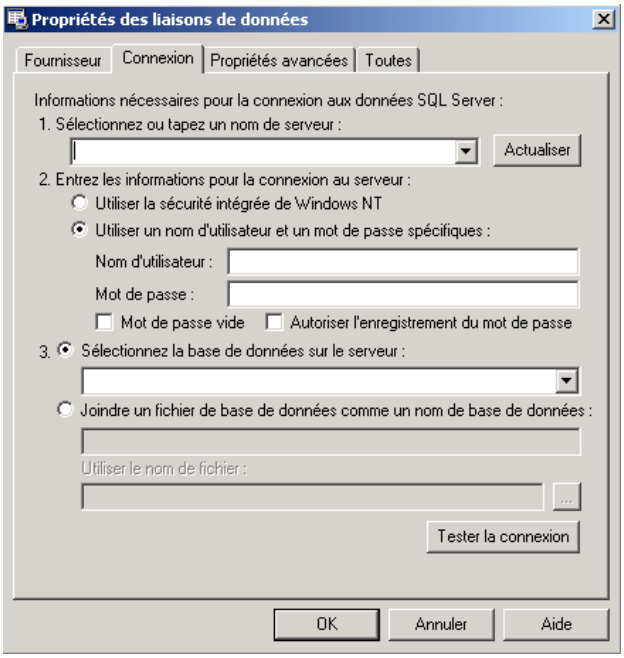
Contient l'ensemble des objets BindingManagerBase pour les contrôles d'un formulaire par exemple.

### 7.8.2 Liaison de données par Interface Graphique

L'environnement de développement Visual Studio offre un ensemble d'outils permettant de générer la liaison entre contrôle et champs de la source de données.

#### *7.8.2.1 Définir la connexion*

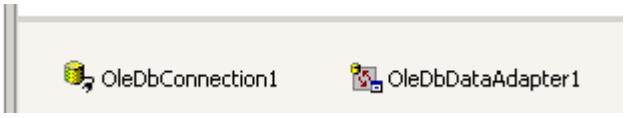
Pour définir la connexion, ouvrir l'explorateur de serveur, clic droit sur « Connexions de données », Ajouter une connexion :




Les paramètres de votre connexion dépendent de la source de données à laquelle vous souhaitez vous connecter. Pour plus de renseignement, consulter la partie « Connexion » de ce chapitre.

#### *7.8.2.2 Création des objets connexion et DataAdapter*

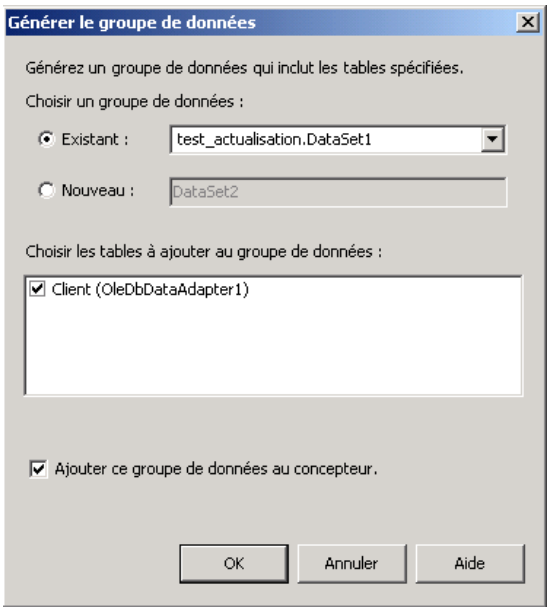
Une fois la connexion définie, vous devez générer les objets Connexion et DataAdapter qui seront utilisés pour lier vos contrôles a la source de données : pour cela, faîtes glisser la table voulue sur le formulaire : les objets connexion et DataAdapter sont alors créés.



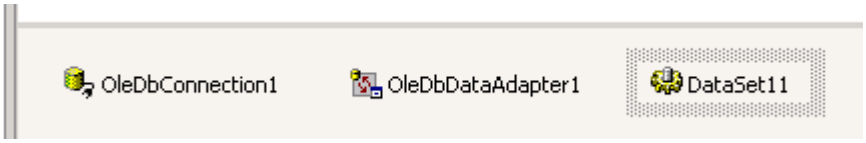

#### *7.8.2.3 Générer le groupe de données*

Afin de pouvoir faire références aux données de la table sélectionnée dans les différents contrôles, vous devez générer le groupe de données. Cette opération permet de générer un DataSet contenant les informations des différentes tables.

Pour générer le groupe de données, clic droit sur l'objet DataAdapter précédemment créé, « Générer le groupe de données » :



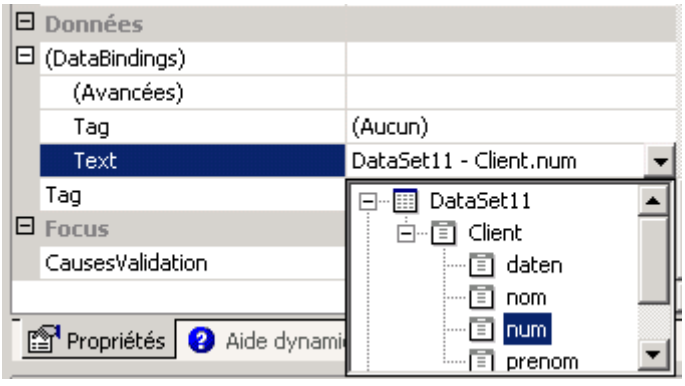

Une fois les informations de la boite de dialogue paramétrée, un nouvel objet DataSet est créé :



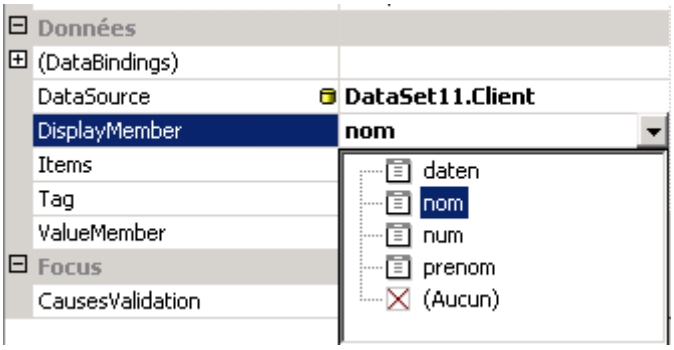

#### *7.8.2.4 Lier les contrôles*

Une fois toutes ces opération réalisées, vous êtes en mesure de lier les différents contrôle aux données du DataSet. Cette opération est différente suivant les contrôles utilisés :

- Pour un TextBox




- Pour un ListBox



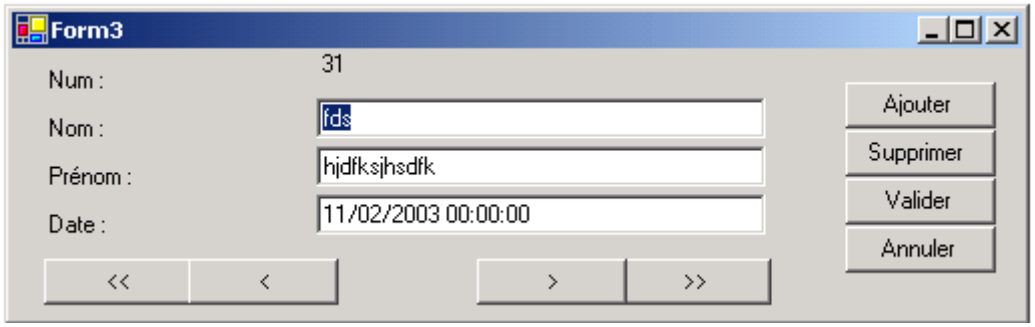

#### *7.8.2.5 Finalisation par le code*

La dernière étape consiste à placer dans l'événement « Form_load » le code permettant d'une part de remplir le DataSet généré et d'autre part récupérer une référence à l'objet BindingManagerBase afin de naviguer dans la source de données :

```
Dim WithEvents bmb As BindingManagerBase

    Private Sub Form_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'remplissage du dataset
        Me.OleDbDataAdapter1.Fill(Me.DataSet11, "Client")
        'recuperation de la référence au BindingManagerBase
        bmb = Me.BindingContext(Me.DataSet11, "Client")
    End Sub
```

### 7.8.3 Exemple d'application

Le code suivant permet d'implémenter l'application suivante :




```
      Public Class Form3
    Inherits System.Windows.Forms.Form

    'declaration globale des objets
    Public cn As New OleDb.OleDbConnection
    Public WithEvents da As OleDb.OleDbDataAdapter
    Public ds As New DataSet
    Public bmb As BindingManagerBase

#Region " Code généré par le Concepteur Windows Form "
```

```
    Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'initialisation des objets
        cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\Support\vbnet\bd2.mdb;"
        da = New OleDb.OleDbDataAdapter("select * from Client", cn)
        da.Fill(ds, "Client")

        'liaison des propriétés des controles aux champs de la base
        Me.lbl_num.DataBindings.Add("text", ds, "Client.num")
        Me.txt_daten.DataBindings.Add("text", ds, "Client.daten")
        Me.txt_nom.DataBindings.Add("text", ds, "Client.nom")
        Me.txt_prenom.DataBindings.Add("text", ds, "Client.prenom")

        bmb = Me.BindingContext(Me.ds, "Client")

    End Sub

    Private Sub btn_premier_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_premier.Click
        Me.bmb.Position = 0
    End Sub

    Private Sub btn_precedent_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btn_precedent.Click
        Me.bmb.Position -= 1
    End Sub

    Private Sub btn_suivant_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_suivant.Click
        Me.bmb.Position += 1
    End Sub

    Private Sub btn_dernier_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_dernier.Click
        Me.bmb.Position = Me.bmb.Count - 1
    End Sub

    Private Sub btn_ajouter_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_ajouter.Click
        Me.bmb.AddNew()
    End Sub

    Private Sub btn_supprimer_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btn_supprimer.Click
        If MsgBox("Supprimer ?", MsgBoxStyle.YesNo) = MsgBoxResult.Yes Then
            Try
                Me.bmb.RemoveAt(Me.bmb.Position)
                MsgBox("Element supprimé")
            Catch ex As Exception
                MsgBox("Erreur de suppression :" & vbCr & ex.Message)
            End Try
        End If
    End Sub

    Private Sub btn_valider_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_valider.Click
        Me.bmb.EndCurrentEdit()
```

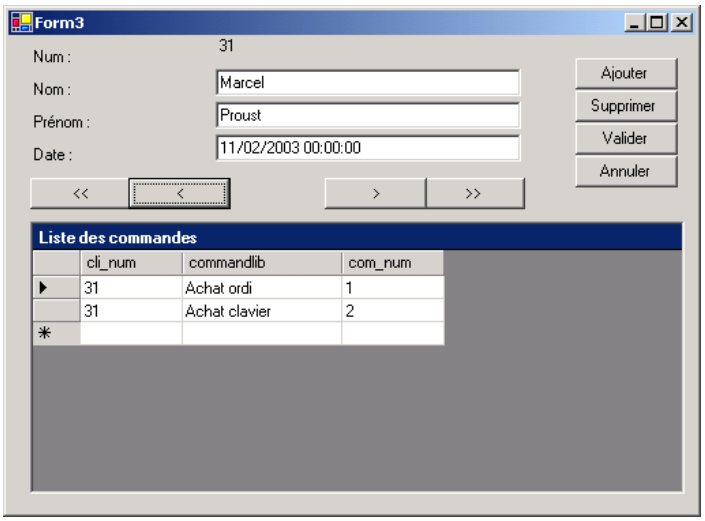
```
    End Sub

    Private Sub btn_annuler_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_annuler.Click
        Me.bmb.CancelCurrentEdit()
    End Sub
End Class
```

### 7.8.4 Formulaires de données Maitre / Détail

Les formulaires de données permettent d'afficher en même temps le contenu de deux tables liées par une relation : par exemple, si vous affichez une fiche client avec en dessous la liste de toutes les commandes passées, la liste des commandes doit être mise à jour lors de la sélection d'un autre client.

Pour cela, il suffit de créer dans le même DataSet deux tables liées par une relation et de modifier la propriété « DataMember » en lui passant en paramètre le nom de la table maître et le nom de la relation.

L'exemple suivant reprend l'application de gestion de client précédente en ajoutant l'affichage des commandes pour le client affiché :




Les seules modifications à appliquer au code précédent se situe dans le « Form_Load ». Voici la version modifiée :

```
        'initialisation des objets
        cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\Support\vbnet\bd2.mdb;"
        da = New OleDb.OleDbDataAdapter("select * from Client", cn)
        da.FillSchema(ds, SchemaType.Source, "Client")
        da.Fill(ds, "Client")
        da2 = New OleDb.OleDbDataAdapter("select * from Commande", cn)
        da2.FillSchema(ds, SchemaType.Source, "Commande")
        da2.Fill(ds, "Commande")
```

```
        'creation de la relation entre les tables commande et client
        Dim r As New DataRelation("client_commande",
ds.Tables("Client").Columns("num"),
ds.Tables("Commande").Columns("cli_num"))
        ds.Relations.Add(r)

        'liaison des propriétés des controles aux champs de la base
        Me.lbl_num.DataBindings.Add("text", ds, "Client.num")
        Me.txt_daten.DataBindings.Add("text", ds, "Client.daten")
        Me.txt_nom.DataBindings.Add("text", ds, "Client.nom")
        Me.txt_prenom.DataBindings.Add("text", ds, "Client.prenom")

        'liaison du DataGrid
        Me.DataGrid1.CaptionText = "Liste des commandes"
        Me.DataGrid1.DataSource = Me.ds
        Me.DataGrid1.DataMember = "Client.client_commande"

        'recupere le context
        bmb = Me.BindingContext(Me.ds, "Client")
```