Licence Professionnelle

Distribution Electrique
et Automatismes

Microsoft Visual Basic .Net

B.Delinchant / J.-P. Guiramand / P.-A. Degryse

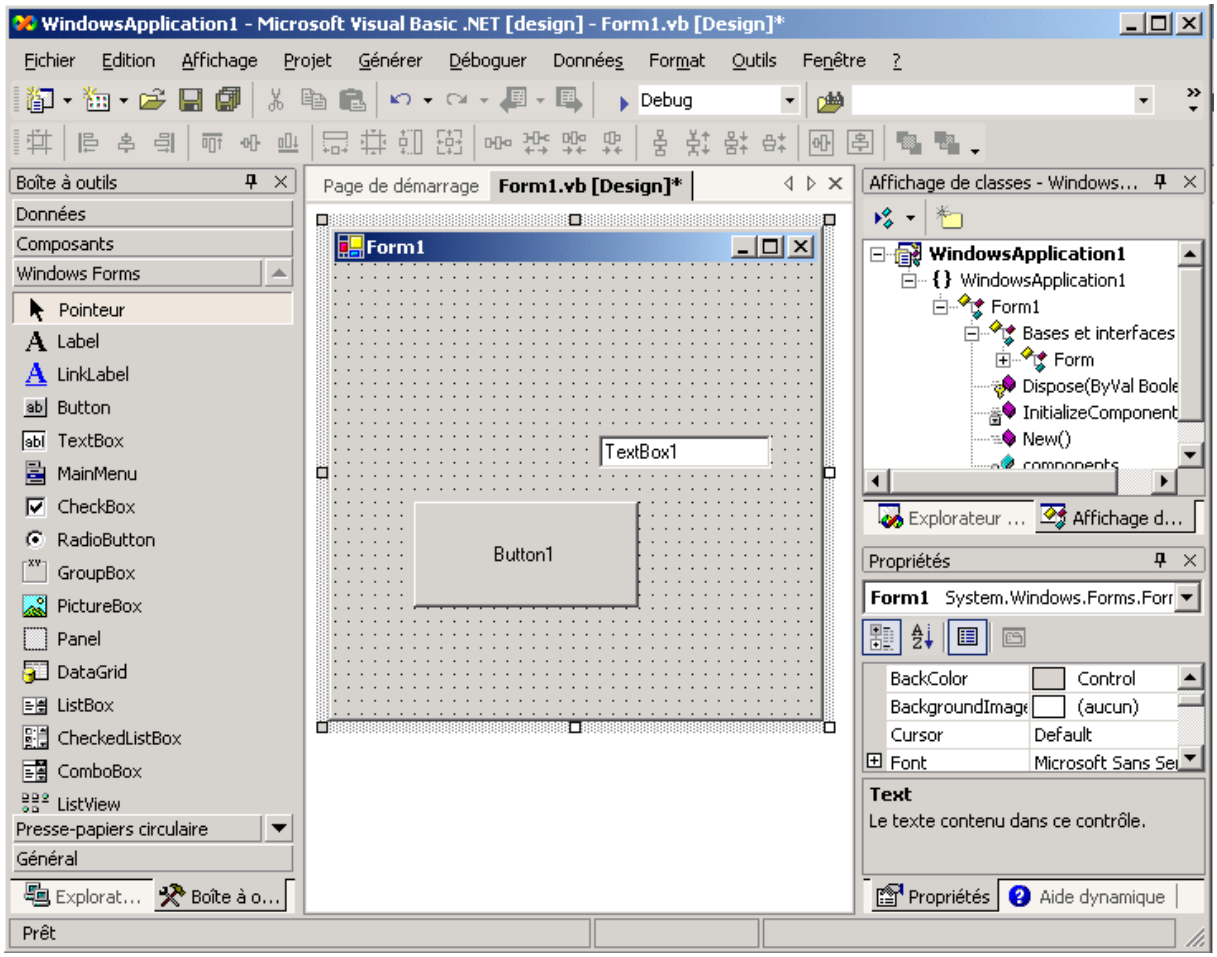
A partir du cours d'Olivier Zimmermann :
http://www.vbfrance.com/tutoriaux/BASICS-VISUAL-BASIC-NET_116.aspx

1 Environnement Visual Studio

1.1 L'interface

Palettes standards :

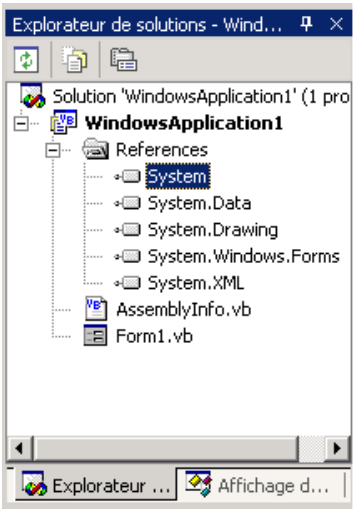
- la barre d'outils regroupe les différents contrôles par catégories
- La zone centrale permet tour à tour d'écrire le code et de définir les interfaces graphiques utilisateurs
- A droite, l'explorateur de solutions et la fenêtre de propriétés



1.2 Fenêtres d'exploration

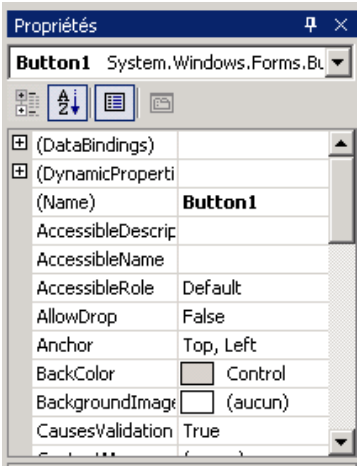
1.2.1 Explorateur de Solutions

L'explorateur de solutions référence l'ensemble des éléments du projet (Fichier de configuration pour l'assemblage, super classes héritées, Feuilles….). Une solution contient les fichiers suivants :



.sln : fichier de configuration de la solution
.vbproj : fichier projet, (ancien .vbp)
.vb : fichiers contenant du code. Un double click sur un objet « form » l'ouvre, pour avoir le source VB : clic droit.
.resx : associé à une feuille, contient les ressources

1.2.2 Fenêtre de propriétés

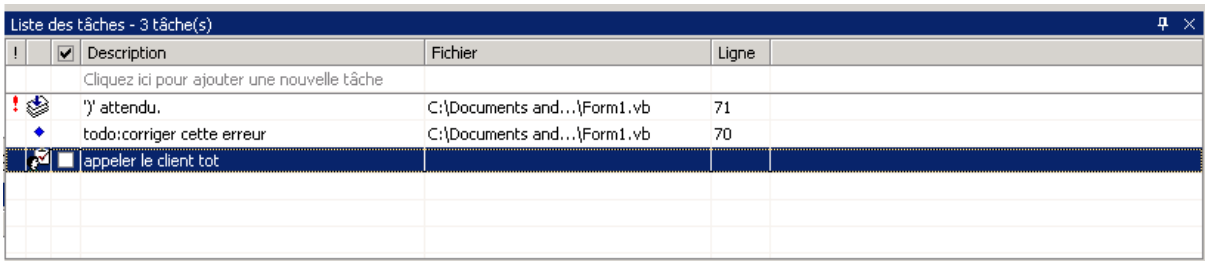


Cette fenêtre recense toutes les propriétés relatives à l'objet sélectionné. Il s'agit en général de propriétés sur les objets graphiques d'une fenêtre « form » telles que le texte affiché, la couleur, la police, la taille, ... Mais aussi le nom de l'objet qui devra être utilisé dans le code source VB (ici « Button1 »).

1.2.3 La liste des tâches

La fenêtre liste des tâches est un outil pour gérer votre projet, elle permet de recenser l'ensemble des tâches 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



2.1 Gestion de projets

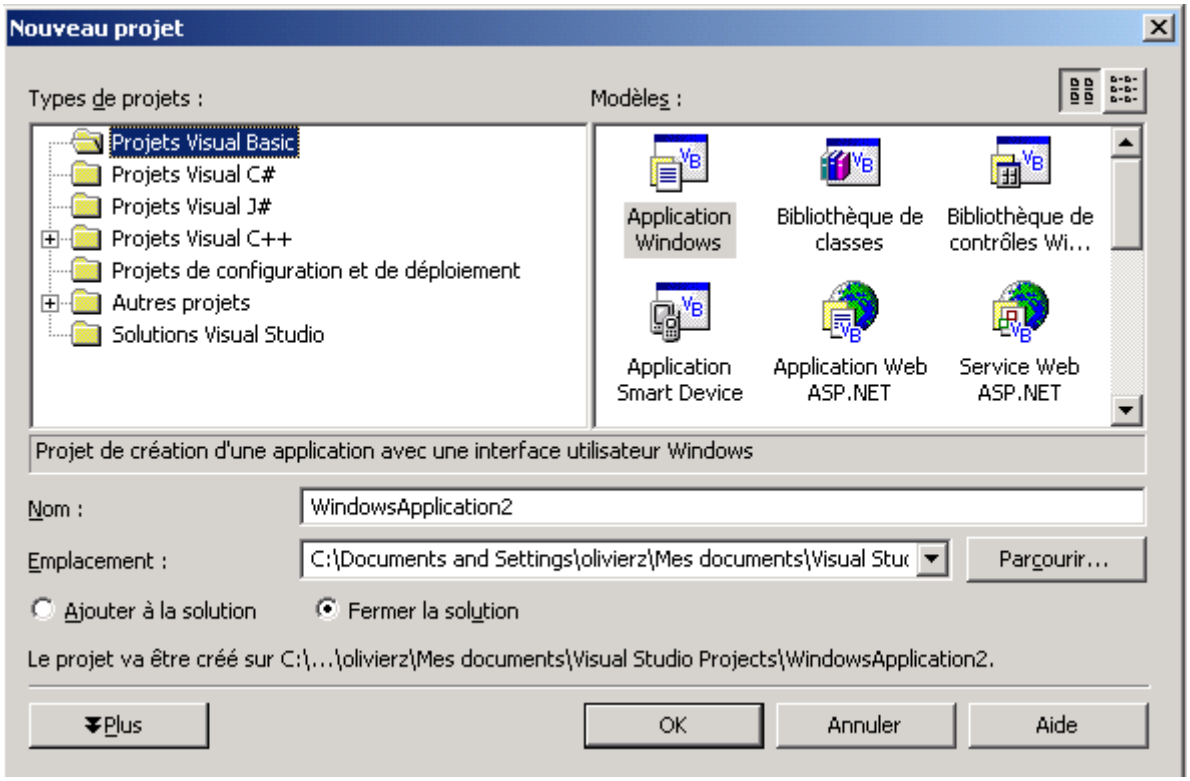
1.2.4 Les solutions

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

1.2.5 Création d'un projet

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

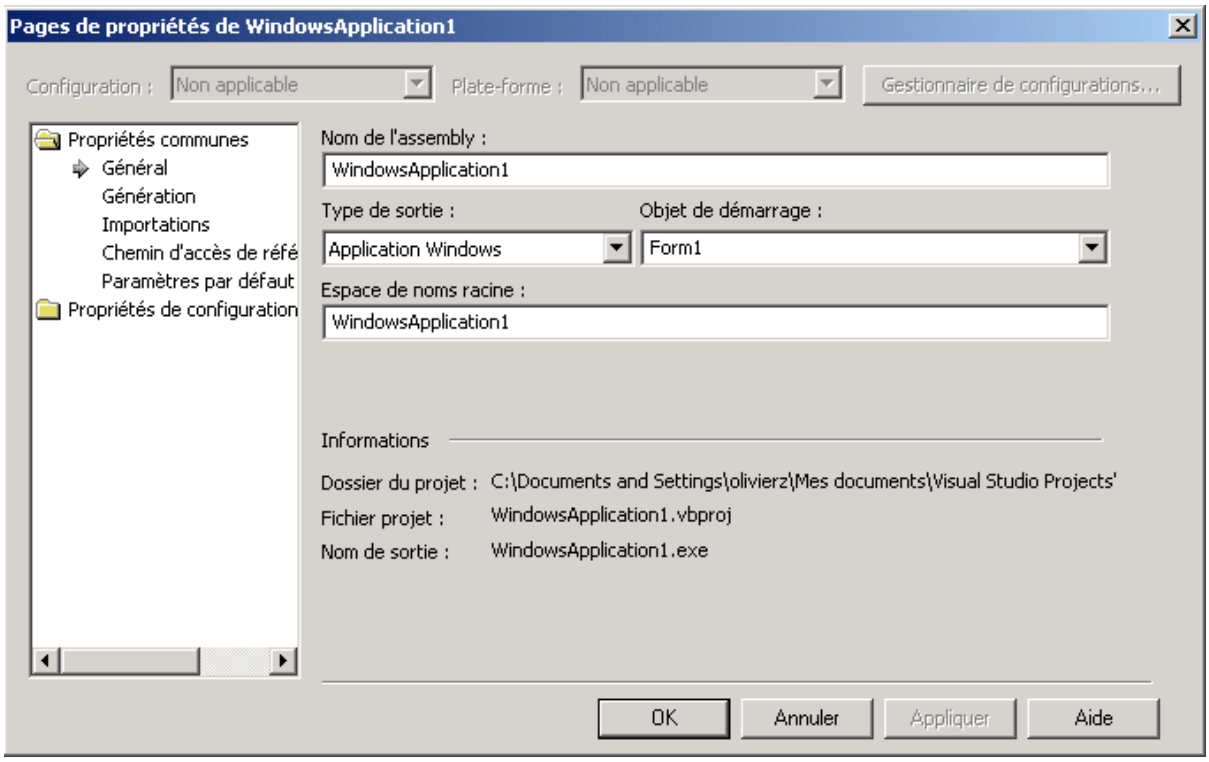
- Menu Fichier > Nouveau > Projet



Principaux projets Visual Basic	
Type	**Description**
Application Windows	Programme classique sous windows
Application console	Programme en mode console (texte)
Bibliothèque de classe	Fonctionnalités réutilisables
Bibliothèque de contrôle Windows	Objets graphiques réutilisables
Application Smart Device, Application Web ASP.Net, Service Web ASP.Net, Service Windows…	

1.2.6 Configuration d'un projet

Pour chaque projet de votre solution, des 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
Type de sortie	Type d'application à 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 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

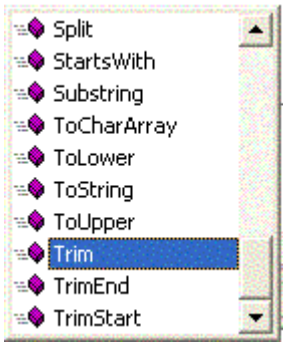
2.2 VB propose des AIDES.

2.2.1 Durant la saisie

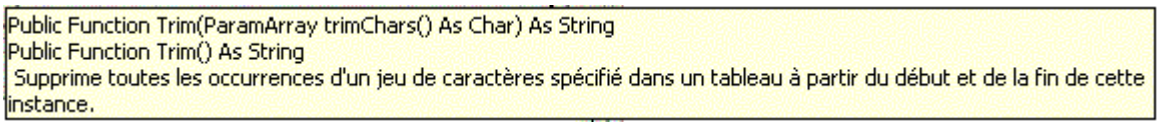
Quand on tape du code, VB affiche, quand il le peut, des aides:

- VB permet de choisir dans une liste une des propriétés d'un objet :

  Exemple: Si on crée une variable chaîne de caractères et qu'on tape le nom de la variable suivi d'un point: 'Chaîne.' la liste des méthodes possibles s'affiche :



Quand on pointe dans la liste un des membres (propriété ou méthode) un carré jaune affiche la définition de la fonction avec ses paramètres et une explication :



- VB aide à retrouver les paramètres d'une fonction :

Si on tape le nom d'une fonction et (, VB affiche les paramètres possibles dans un cadre.

En plus il affiche les différentes manières d'utiliser les paramètres (les différentes signatures), on peut les faire défiler avec les petites flèches du cadre jaune.

- VB aide à compléter des mots :

Si je tape App puis sur le bouton 'A->', Vb affiche la liste des mots commençant pas App

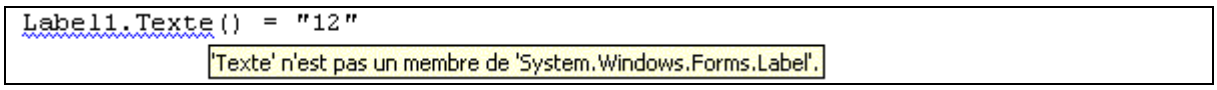
AppActivate
AppDomain

2.2.2 La documentation

- VB donne accès à l'aide sur un mot Clé. Si le curseur passe sur un mot clé, un carré affiche la définition de la fonction. Si je clique sur un mot et que je tape F1 l'aide s'ouvre et un long texte donne toutes les explications.
- VB donne accès à l'aide sur les contrôles. Si le curseur est sur un contrôle et que je tape F1 l'aide s'ouvre pour donner accès à la description des différents membres de cet objet.
- L'aide dynamique est constamment mise à jour. Pour la voir, il faut cliquer sur l'onglet 'Aide Dynamique' en bas à droite (même fenêtre que la fenêtre propriété). Elle donne une liste de liens en rapport avec le contexte en cours.
- Enfin il est toujours possible de rechercher des informations en passant par les 3 onglets de la fenêtre en haut à droite :
  - Sommaire (plan , arbre de l'aide)
  - Index (liste des mots)
  - Recherche (rechercher un mot)

2.2.3 Erreur.

S'il existe une erreur dans le code au cours de la conception, celle-ci est soulignée en bleu ondulé. Un carré donne la cause de l'erreur si le curseur passe sur la zone ou se trouve l'erreur.

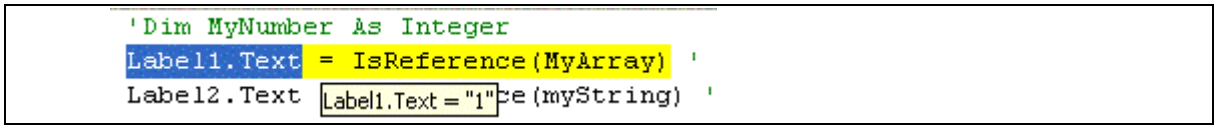


Ici la propriété 'Text' a été mal orthographiée.
Si on lance le programme en mode 'Run' et qu'il y a des erreurs, Vb le signale et répertorie les erreurs dans la liste des taches en bas.

2.2.4 Mode déboguage (mode BREAK):

Une fois lancée l'exécution (F5), puis stoppée (Ctrl +Alt +Pause), on peut:

- Voir la valeur d'une propriété d'un objet en la sélectionnant avec la souris:

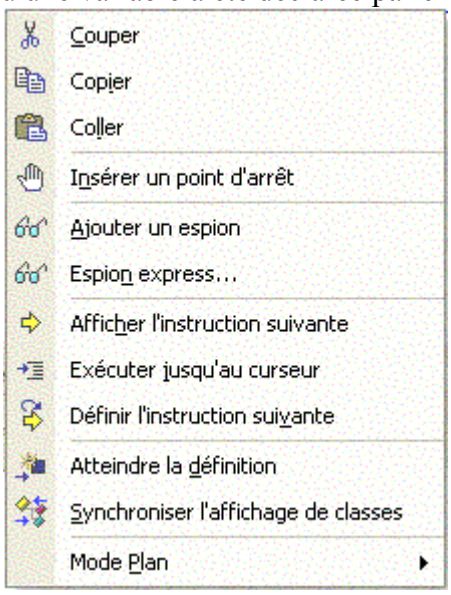


Il s'affiche un petit cadre donnant la valeur de la propriété d'un objet.

- Voir la valeur d'une variable, simplement en positionnant le curseur sur cette variable.

- F11 permet l'exécution pas à pas (y compris des procédures appelées)
- F10 permet le pas à pas (sans détailler les procédures appelées)
- Maj+F11 exécute jusqu'à la fin de la procédure en cours.

En cliquant sur le bouton droit de la souris, on peut **afficher ou définir l'instruction suivante, exécuter jusqu'au curseur, voir la définition de ce qui est sous le curseur** (La définition, c'est l'instruction ou une variable à été déclarée par exemple).



On peut grâce au menu déboguage puis Fenêtre ouvrir les fenêtres:

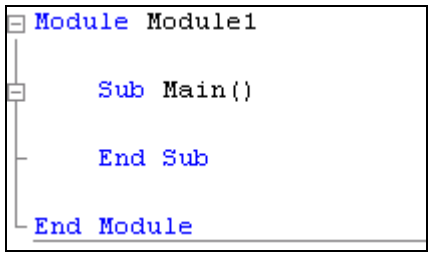
- **Automatique**, qui affiche les valeurs des variables de l'instruction en cours et des instructions voisines.
- **Immédiat** où il est possible de taper des instructions ou expressions pour les exécuter ou voir des valeurs.
- **Espions** permettant d'afficher le contenu de variables ou d'expressions.
- **Espions Express** permet d'afficher la valeur de l'expression sélectionnée.
- **Points d'arrêts** permet de modifier les propriétés des points d'arrêts. on peut mettre un point d'arrêt en cliquant dans la marge grise à gauche: l'instruction correspondante s'affiche en marron et l'exécution s'arrêtera sur cette ligne.
- **Me** affiche les données du module en cours.
- **Variables locales** affiche les variables locales.
- **Modules** affiche les dll ou .exe utilisés.
- **Mémoire, Pile d'appels, Thread, Registres, Code Machine** permettent d'étudier le fonctionnement du programme à un niveau plus spécialisé et technique.

Il est possible de mettre des **points d'arrêt, des espions** pour arrêter l'exécution et suivre la valeur de certaines expressions.

2 Bases du langage

2.1 Structure du code

2.1.1 Module



Lors de la création d'un nouveau module, le code ci-contre est automatiquement généré. Tout le code contenu dans le module sera inséré entre « module » et « end module ».

2.1.2 Les commentaires

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
```

2.2 Les variables

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

2.2.1 Types de variables

2.2.1.1 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	$-3.402823 \times 10^{+38}$	$3.402823 \times 10^{+38}$	32 bits
Double	$-1.79769313486231 \times 10^{+308}$	$-1.79769313486232 \times 10^{+308}$	64 bits
Decimal	-79228162514264337593543950335	79228162514264337593543950335	16 Octets

2.2.1.2 Types chaîne de caractère

Les chaînes de caractère sont enregistrées au format **Unicode**, c'est à dire que chaque caractère utilise **2 octets**, alors que le code ASCII est lui sur 1 octet uniquement.

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''

2.2.1.3 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# #12/07/02# #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	

2.2.2 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 interprétée 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
```

2.2.3 Portée et visibilité des variables

2.2.3.1 Portée des variables

La portée d'une variable est équivalente 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 :

- Au niveau d'un bloc (module, fonction, boucle, condition, ...)
- Au niveau du 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 ».

2.2.3.2 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 du bloc peuvent y accéder

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

2.2.4 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"
```

2.2.5 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)
```

Ces redimensionnements sont à éviter au maximum car ils nécessitent des traitements mémoire lourds. Il est préférable de surdimensionner un peu le tableau dès leur déclaration.

Des outils sont mis à disposition pour travailler sur les tableaux par l'objet « Array » (voir paragraphes sur les fonctions intégrés).

2.2.6 Les structures

Une structure permet de regrouper des données pouvant être de différents types.

Exemple : vous voulez définir une variable contenant une adresse composée d'un numéro, de la rue, de la ville. Il faut d'abord définir la structure :

```
Public Structure Adresse
  Dim Numero  As Integer
  Dim Rue             As String
  Dim Ville      As String
End Structure
```

Puis dans une procédure il faut déclarer la variable :

```
Dim MonAdresse As Adresse
```

La variable MonAdresse est déclaré comme une adresse, elle contient donc:

- un numéro 'stocké' dans MonAdresse.Numero
- une nom de rue 'stocké' dans MonAdresse.Rue
- un nom de ville 'stocké' dans MonAdresse.Ville

On pourra enfin l'utiliser :

```
MonAdresse.Numero=2
MonAdresse.Rue= "Grande rue"
MonAdresse.Ville= "Lyon"
```

On peut aussi utiliser le mot clé With pour ne pas avoir à répéter le nom de la variable.

```
With MonAdresse
   .Rue= "Grande rue"
   .Ville= "Lyon"
End With
```

With est utilisable sur tous les objets.

2.3 Les opérateurs

il existe 5 types d'opérateurs

2.3.1 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 arithmétiques lors d'une affectation.

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

2.3.2 Opérateurs Arithmétiques

les opérateurs arithmétiques 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
```

2.3.3 Opérateurs de comparaison

Ils 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

2.3.4 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'esperluette « & » : 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 !
```

2.3.5 Opérateurs logiques

Les opérateurs logiques permettent de combiner des expressions logiques (renvoyant True ou False) à l'intérieur de conditions.

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

2.4 Les structures de contrôle

Les structures de contrôle permettent de modifier le nombre d'exécution d'une instruction. Elle exploite une « condition » qui est une comparaison entre deux membres dont le résultat retourne True (Vrai) ou False (Faux).

2.4.1 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.

2.4.1.1 Structure If

Plusieurs syntaxes sont possibles :

- Forme simple

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

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

2.4.1.2 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 numériques, permet de définir un intervalle
- Case is > constante : pour les valeurs numériques, définition d'un intervalle 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
```

2.4.2 Structures répétitives

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

2.4.2.1 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
```

2.4.2.2 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
```

2.4.2.3 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
```

2.4.2.4 Structure For each

La structure For Each permet de parcourir un tableau sans se préoccuper de l'indice.

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

2.5 Procédures et fonctions

Dans une application Visual Basic, les instructions doivent obligatoirement figurer dans une procédure ou une fonction. Ce sont des unités logiques 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 imposée par Vb.

- 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

2.5.1 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 !!!

2.5.1.1 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 mêmes règles que pour les variables
- Il possible de préciser la visibilité de la procédure en remplaçant « public » par « private » ou « friend »

2.5.1.2 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)
```

2.5.2 Création de fonction

Les fonctions sont des procédures retournant un résultat.

2.5.2.1 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
```

2.5.2.2 Appel

Nous l'avons dit plus haut, une 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'une 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))
```

2.5.3 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.

2.5.3.1 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
```

2.5.3.2 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")
```

2.5.3.3 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 travaillent 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
```

2.6 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 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.

2.6.1 Fonctions sur les chaînes de caractères

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

```
Dim ch as string = ''tartampion''
```

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 chaîne 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'')	Tartanpionsssss

.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 Rt 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 chaîne	Ch.trim(''t'')	Artanpion

2.6.2 Fonctions sur les nombres

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

Fonction	Description	Exemple	Résultat
Math.Floor(n)	Tronque la valeur	Math.Floor(3.25)	3
Math .Ceiling(n)	Retourne le nombre entier juste supérieur	Math.Ceiling(3.25)	4
Math .Pow(n, p)	Retourne n à la puissance p	Math.pow(2,3)	8
Math .abs(n)	Retourne la valeur absolue de n	Math.abs(-8)	8
Math.sqrt(n)	Retourne la racine carrée de n	Math.sqrt(4)	2
Math.sin(n)	Retourne le sinus de n	Math.sin(Math.PI / 2)	1
...			

2.6.3 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	

2.6.4 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

2.6.5 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.

2.6.5.1.1 Fonction	Type de donnée 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

2.6.6 Fonction de formatage (Format)

Les fonctions de formatage permettent une représentation différente des données numériques ou dates mais ne sont pas utilisées pour des conversions 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

2.6.6.1 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
```

2.6.6.2 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
```

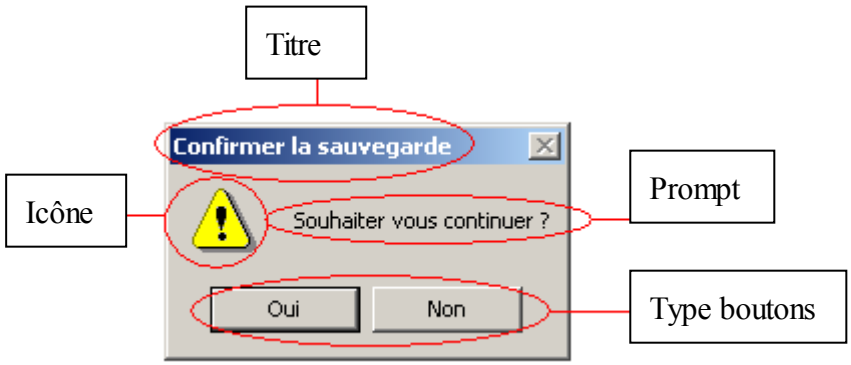
2.6.7 Les boîtes de dialogue

Les boîtes 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.

2.6.7.1 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

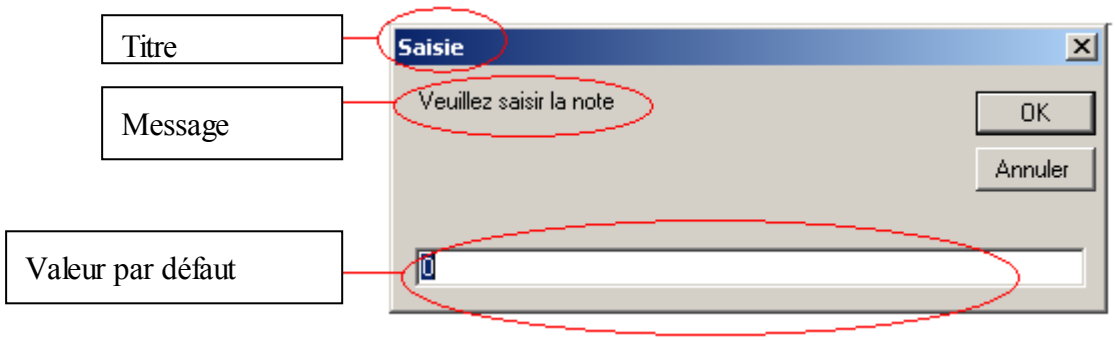
L'option bouton par défaut permet de définir le bouton qui sera sélectionné par défaut.

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
```

2.6.7.2 Boîte de saisie

La boîte de saisie (InputBox) permet la saisie d'une chaîne de caractères 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 contrôles 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")
```

2.7 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.

2.7.1 Types d'erreurs

Erreur	Description
De syntaxe	Type d'erreur correspondant 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 dernières affichent 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.

2.7.2 Gestion en ligne

La gestion en ligne permet de définir les traitements à effectuer dans une procédure lorsqu'une erreur survient.

2.7.2.1 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 gestions 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'instruction « 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.

2.7.2.2 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
```

2.7.2.3 L'objet Err

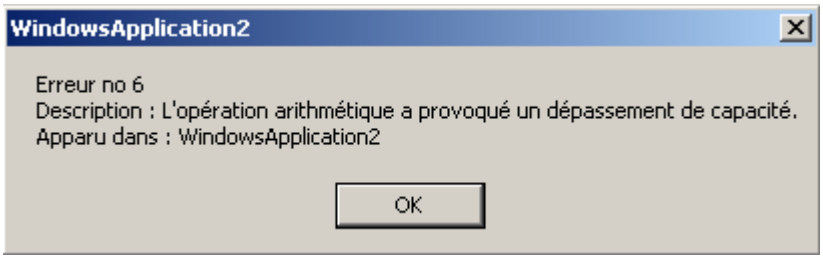
Avec la méthode précédente, vous récupérez toutes les erreurs de la procédure et ce quelque soit l'erreur. Pour une gestion d'erreur plus pointue, vous avez la possibilité de récupérer des informations sur la dernière erreur générée : c'est l'objet « Err ».
Cet objet est toujours accessible.

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

Méthode	Description
Err.Clear	Efface les paramètres de l'objet
Err.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
```



2.7.3 Les Exceptions

2.7.3.1 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 structurée de la manière suivante : le code dangereux doit être compris dans le bloc « try ». Ensuite pour chaque exception susceptible d'être déclenchée, 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

2.8 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.

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

2.8.1 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
```

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
```

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'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.

2.8.2 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 procédures qui affichent un nombre en mode texte et en mode graphique.

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

```
End sub

Public sub affiche_graph(byval nombre as double)
        Msgbox (nombre)
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 « nom_evenement ».

```
Dim c1, c2 as Cadre

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

AddHandler c1.nom_evenement, addressOf affiche_txt
AddHandler c2.nom_evenement, 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
```

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

3 Applications Console

L'interface utilisateur (UI) ou interface homme-machine est la partie du programme qui permet à l'utilisateur du logiciel de communiquer avec le programme. Quand on crée une application VB, on a le choix sur le type d'entrée et de sortie. Où est-il possible de saisir des données au clavier, où seront affichés les résultats ?

Le type d'application le plus basique est le mode « console » ouvrant une fenêtre en mode texte. Il y est possible d'écrire du texte et de saisir du texte. Ce mode est utilisé pour des applications légères ou pour mettre au point des parties de programmes.

3.1 Fonction de lecture (clavier)

Les données provenant du clavier proviennent de l'objet « Console.In » de type « StreamReader ». Ce type d'objets permet de lire une ligne de texte avec la méthode ReadLine :

```
Dim ligneTapee As String = Console.In.ReadLine()
```

La ligne tapée au clavier se retrouve dans la variable « ligneTapee » et peut ensuite être exploitée par le programme.

3.2 Fonction d'écriture (écran)

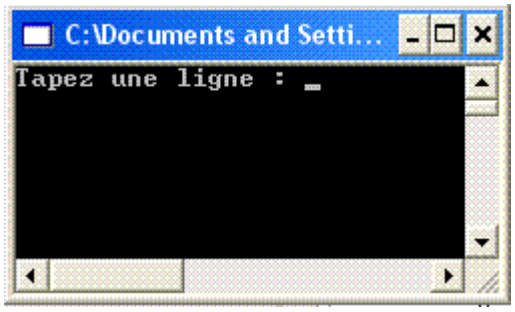
Les données à afficher son envoyée à l'objet « Console.Out »

```
Console.Out.WriteLine("ligne à afficher")
```

Exemple en 'Sortie console':

Affiche 'Tapez une ligne'; quand l'utilisateur a tapé du texte puis a validé, affiche: 'Ligne tapée=..'

```
Console.Out.Write("Tapez une ligne : ")
Dim ligne As String = Console.In.ReadLine()
Console.Out.WriteLine(("ligne tapée=" + ligne))
Console.WriteLine( ) est aussi accepté.
```



4 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 ….).

4.1 Les formulaires

Les formulaires (« forms ») sont les éléments de base des applications graphiques Windows.

4.1.1 Différents types

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

4.1.1.1 Windows Forms

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 …).

4.1.1.2 Web forms

Les applications à base de Web Forms sont destinées 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.

4.1.1.3 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 (fenêtres filles). 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 « Explorer » sous Windows en est un exemple.

4.1.1.4 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

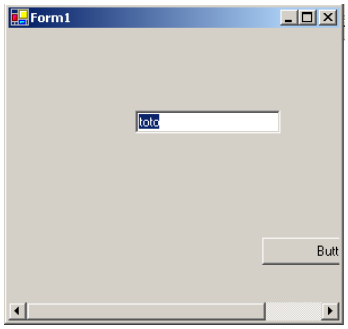
Spécifie si le formulaire gère le Drag and Drop (Glisser-Déposer).

- AutoScale

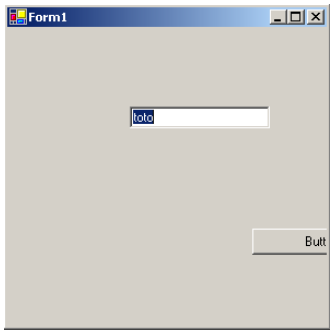
Si cette option est activée, la taille des contrôles et du formulaire sont automatiquement redimensionnés lors d'un changement dynamique de la police d'écran.

- AutoScroll

L'option AutoScroll 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 mosaïque.

- CancelButton

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

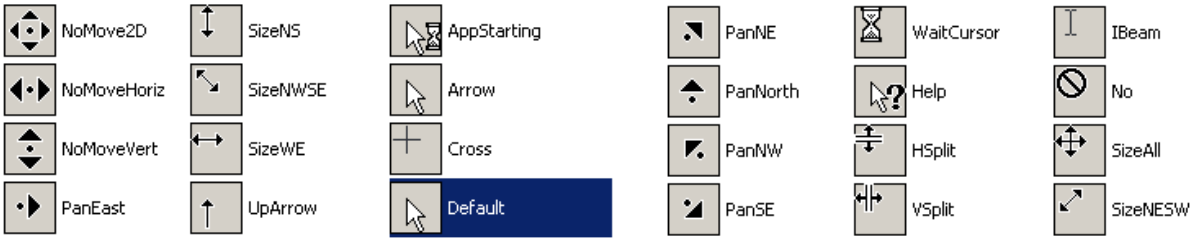
- ControlBox

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 :

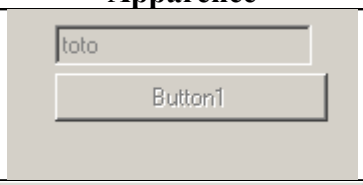
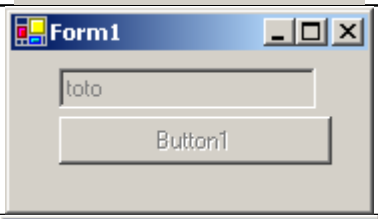
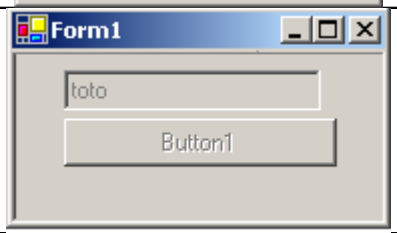
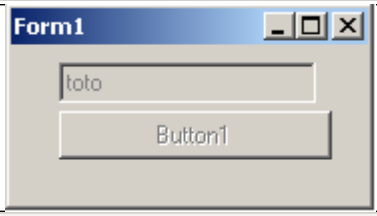
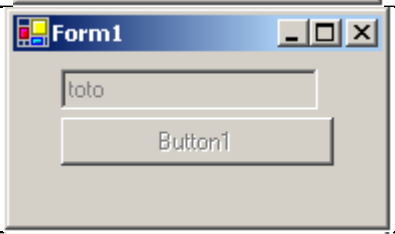
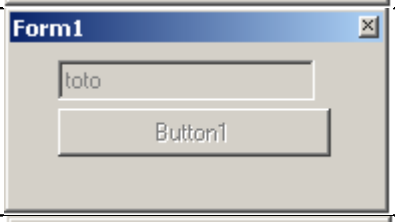
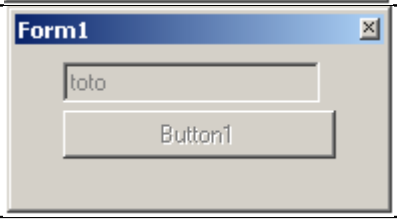
Propriété	Description	Exemple
Name	Nom de la police utilisée	Garamond
Size	Taille de la police	12.5
Unit	Unité de mesure pour la police (Sachez que la plupart des logiciels utilisent l'unité « point »)	Point
Bold	Texte en gras	True
GdiXXXX	Paramètres sur le jeu de caractère utilisé	
Italic	Texte en italique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

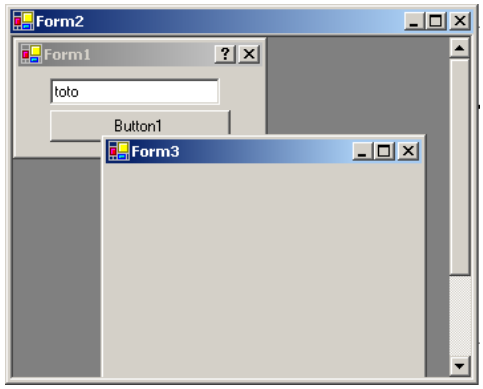
- Icon

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'autres 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 paramétrer l'option « IsMdiContainer » du Form2 à True.



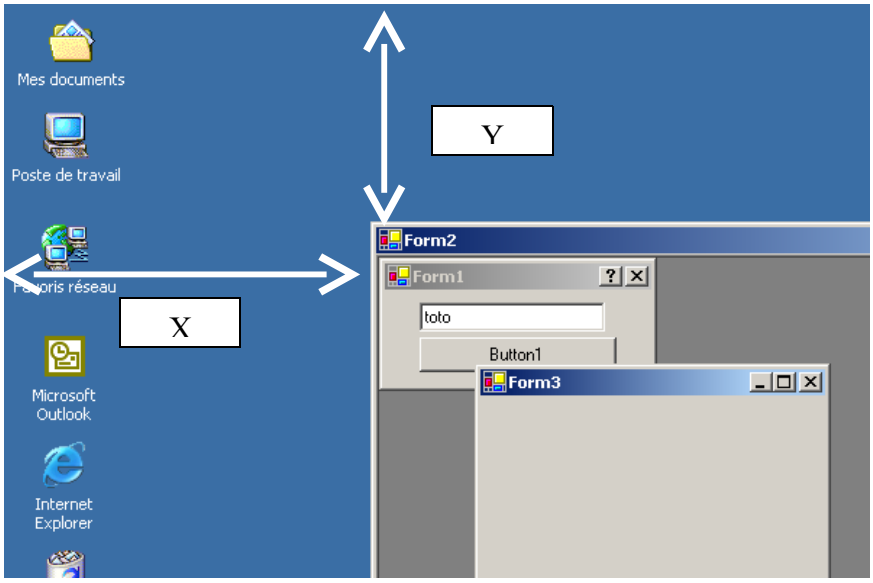

```
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
```

```
    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

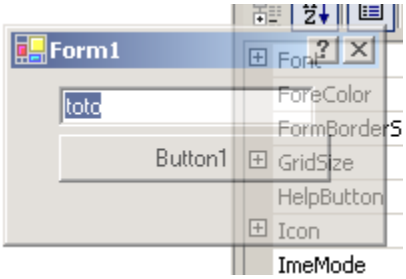
Détermine si les boutons « Agrandir » et « Réduire » sont visibles. Leur affichage empêchera l'affichage du bouton d'aide.

- 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 propriétés sont disponibles : width (largeur) et height (hauteur).

- Opacity

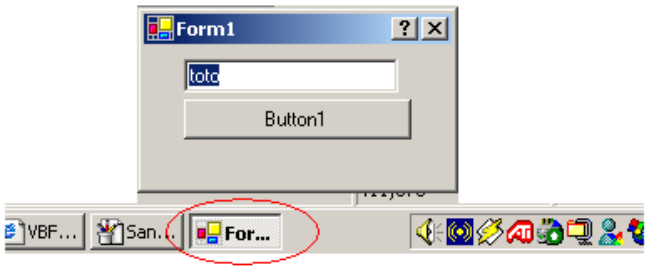
Définit un pourcentage d'opacité pour la fenêtre. Une valeur de 100% rend la fenêtre opaque.



(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

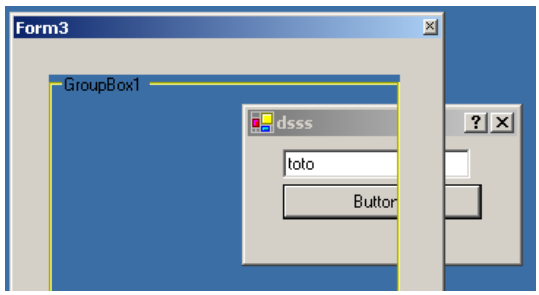
Détermine le texte affiché dans la barre de titre

- 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

4.1.1.5 Méthodes

- 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.

4.1.1.6 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 réapparaît 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

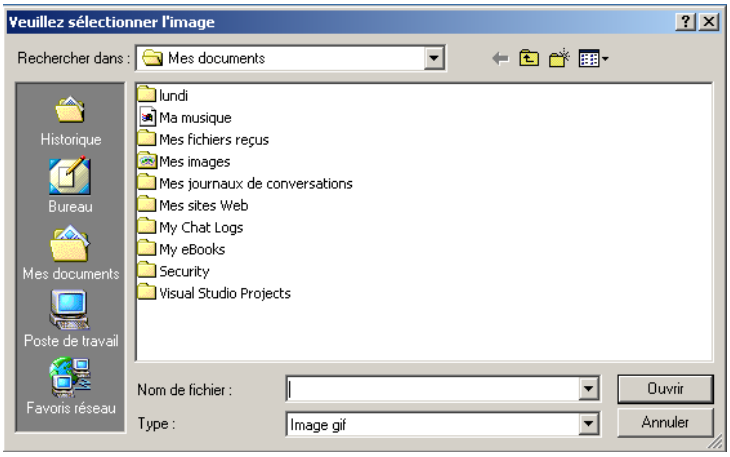
L'utilisateur double clique sur le fond du formulaire

4.1.2 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 boîtes de dialogue standard.

4.1.2.1 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:\windows"
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 sélectionnés	

Méthode	Description
ShowDialog	Affiche la fenêtre

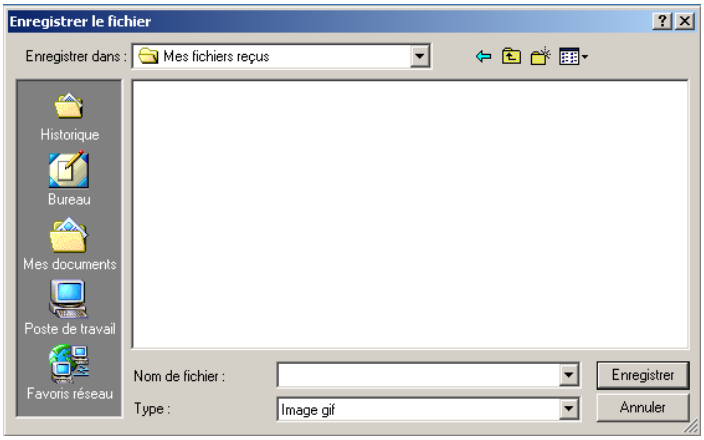
Le code suivant paramètre la boîte de dialogue en acceptant uniquement les fichiers images (gif ou jpg). La sélection multiple est autorisée et la liste des fichiers est affichée à la fin.

```
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 sélectionnés
Dim fichier As String
For Each fichier In dlg.FileNames
   MsgBox(fichier)
Next
```

4.1.2.2 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 ».



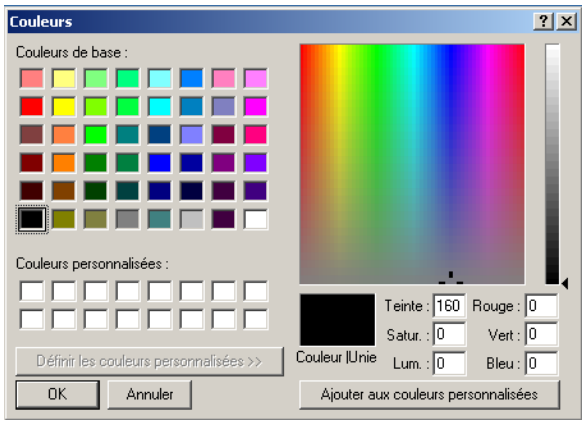

```
Dim dlg As SaveFileDialog
dlg = New SaveFileDialog

'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 sélectionnés
MsgBox("le fichier sera enregistré dans: " & dlg.FileName)
```

4.1.2.3 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

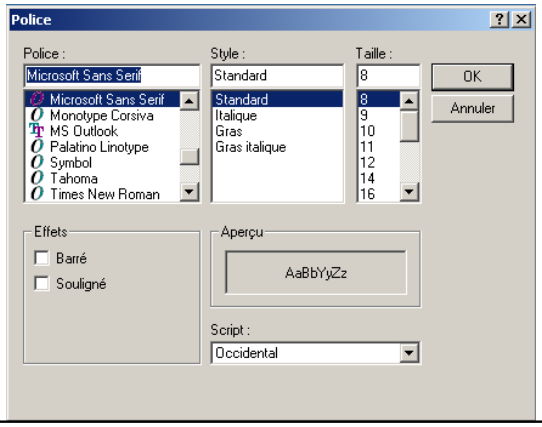
L'exemple suivant affiche une boîte de dialogue et modifie la couleur de fond du formulaire en fonction du choix de l'utilisateur :

```
Dim c As New ColorDialog
```

```
c.FullOpen = False
c.Color = Me.BackColor
c.ShowDialog()
Me.BackColor = c.Color
```

4.1.2.4 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…).



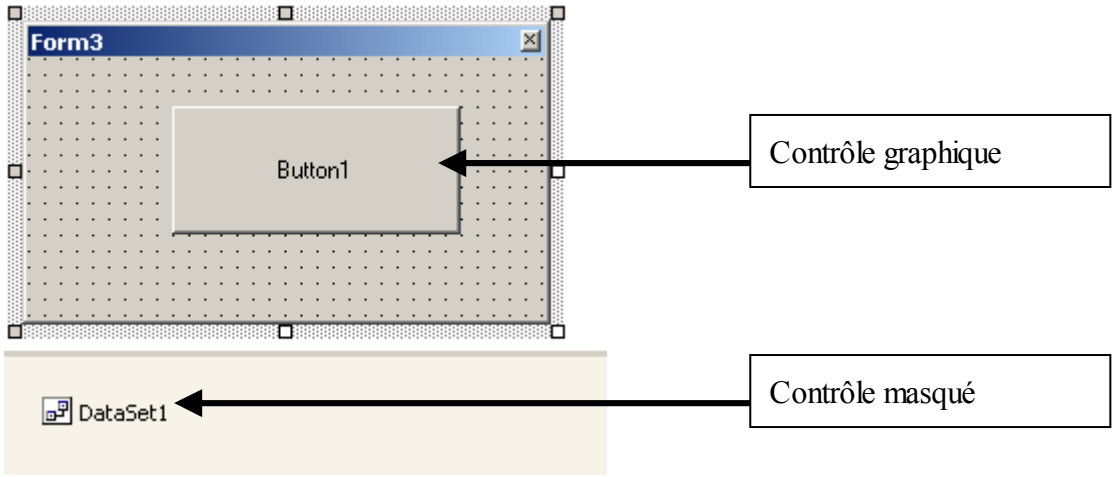
Propriété	Description
ShowEffects	Booléen spécifiant si l'utilisateur peut définir les effets (gras …)
ShowColor	Booléen spécifiant si l'utilisateur peut définir la couleur
MinSize	Taille minimale des caractères
MaxSize	Taille maximale des caractères
Font	Police par défaut et police retournée
Color	Couleur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

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

4.2 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 sélections et déclencher des actions par l'intermédiaires des événements.

De manière générale, les contrôles sont des objets graphiques, c'est à dire qu'ils 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 ».



4.2.1 Membres communs

4.2.1.1 propriétés

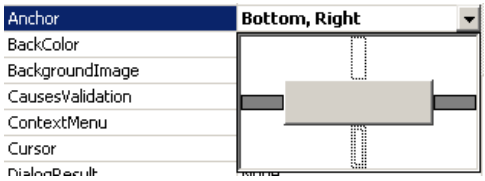
- Name

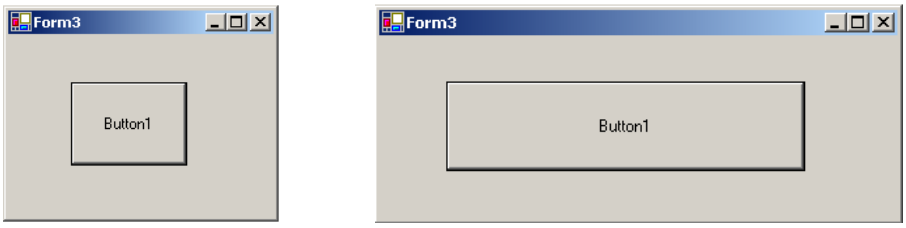
Nom du contrôle. Ce nom ne comporter que des lettres et le caractère underscore « _ ».

- 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

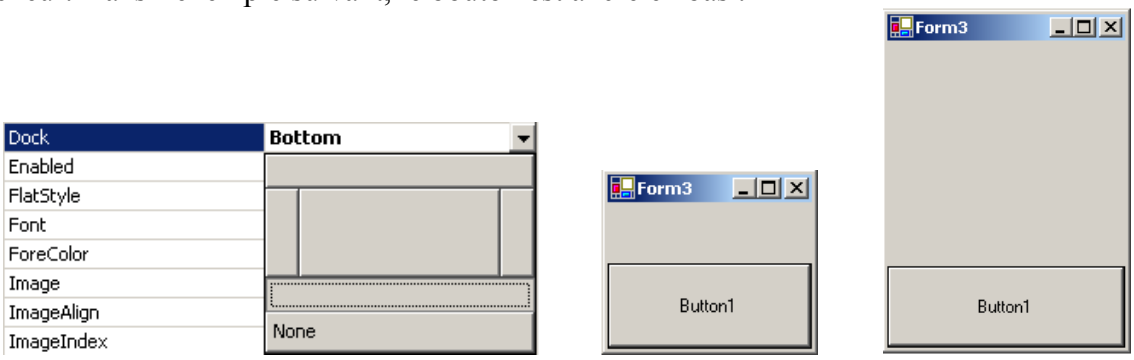
Booléen spécifiant si le contrôle peut recevoir le focus.

- 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

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

4.2.1.2 Méthodes

Méthode	**Description**
Focus	Donne le focus au contrôle

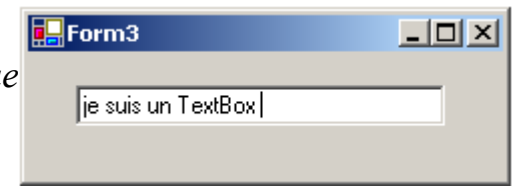
4.2.1.3 Evénements

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 relâchée
KeyUp	Touche relâchée
LostFocus	Activé lorsque le contrôle perd le focus
MouseDown	Bouton souris enfoncé
MouseUp	Bouton souris relâché
MouseMove	Souris déplacée sur le contrôle
MouseWheel	Déplacement de la roulette
Resize	Déclenché lorsque le contrôle est redimensionné

4.2.2 Principaux Contrôles

Nous ne listerons dans cette partie que les principaux contrôles.

4.2.2.1 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 champ 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 champ 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 champ
Text	Contenu du champ
TextLength	Longueur du texte dans le contrôle

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

Evénement	**Description**
TextChanged	Déclenché lorsque le texte change

L'exemple suivant permet de copier dans le presse-papiers tout le texte contenu dans le champ « 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
```

4.2.2.2 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
```

4.2.2.3 CheckBox

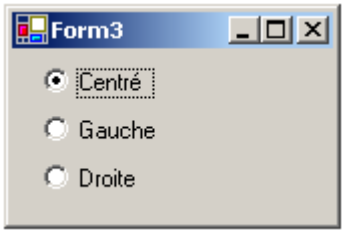
Le contrôle Checkbox (Case à cocher) est utilisé pour proposer plusieurs options à l'utilisateur parmi lesquelles il pourra effectuer plusieurs choix.

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

4.2.2.4 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 ».

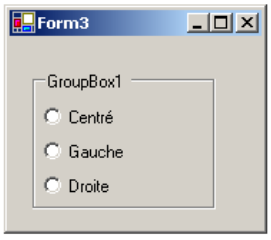
Les boutons radios possèdent les même propriétés et événements que les cases à cocher.

4.2.2.5 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 :

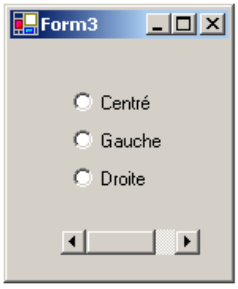
- Regrouper de manière logique des contrôles afin de les isoler (pour les boutons radio par exemple)
- Faciliter le placement de plusieurs contrôles car en modifiant la position du conteneur, vous modifiez la position de tous les contrôles contenus

<u>Le GroupBox</u>



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

<u>Le Panel</u>



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

4.2.2.6 Button

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

4.2.2.7 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 :

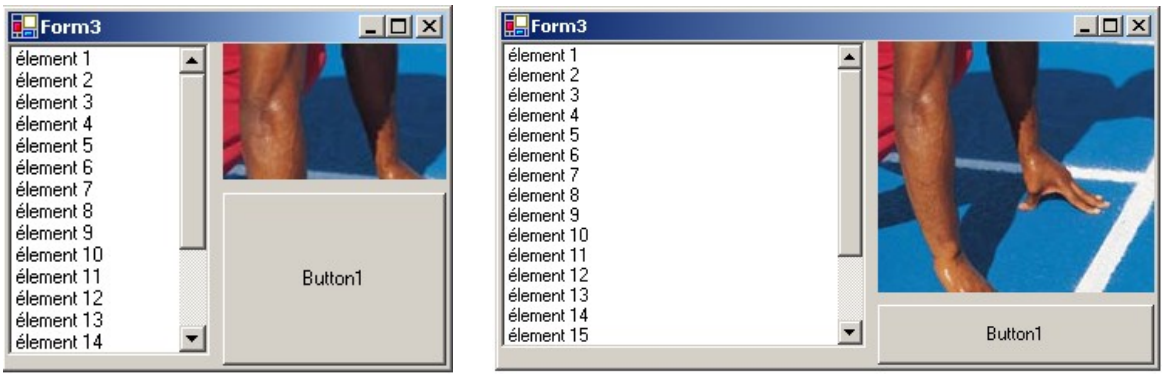
```
Dim elt As String
For Each elt In Me.ListBox1.SelectedItems
   MsgBox("Libellé: " & elt)
Next
```

4.2.2.8 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.

4.2.2.9 Splitter

Le contrôle « Splitter » 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.

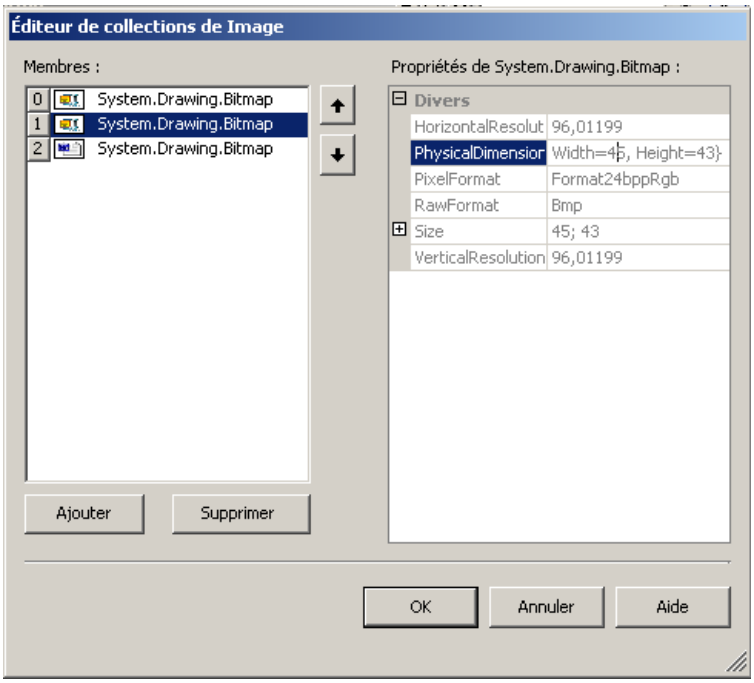
- Placer le contrôle « liste » et paramétrer la propriété « Dock » à « left »
- Placer le contrôle « splitter » à droite de la liste et paramétrer sa propriété « dock » à left
- Placer le contrôle « image » à droite et paramétrer la propriété « Dock » à « top »
- Placer le second splitter sous de l'image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

4.2.2.10 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

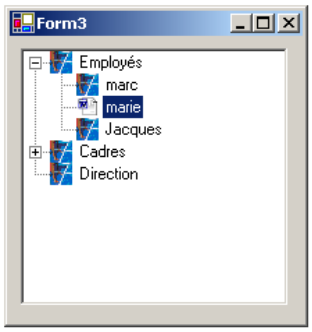
Chacune des images possède un index qui sera ensuite utilisé pour les lier aux autres contrôles. La gestion des images se fait à l'aide d'un assistant :



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

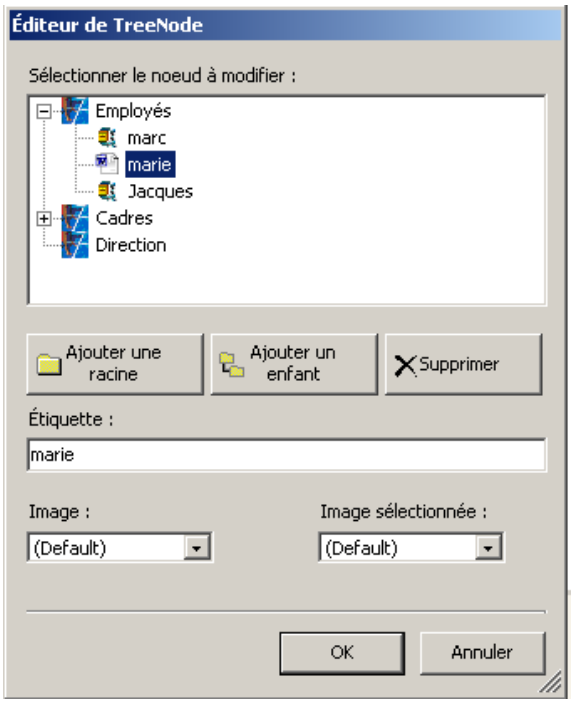
```
Dim chemin as string
Chemin = "C:\mes documents\toto.gif"
Me.Imagelist1.images.add(Image.fromfile(chemin))
Me.Imagelist1.images.RemoveAt(0)
```

4.2.2.11 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

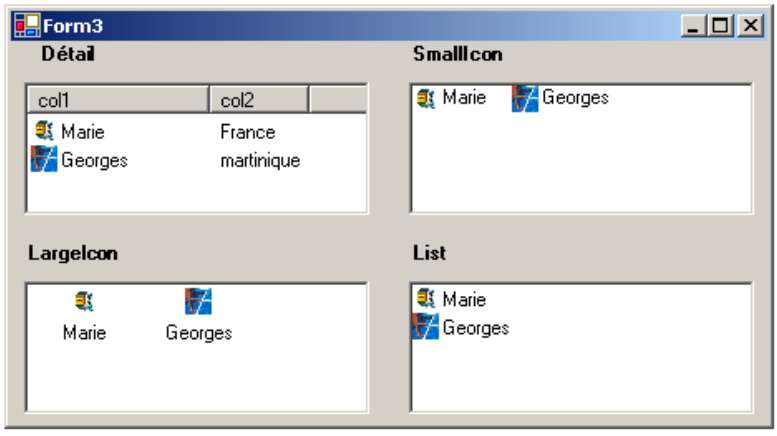
Le code suivant remplit un « ImageList », insère à l'intérieur du « TreeView » 2 catégories principales (Renault & Peugeot) et place ensuite à l'intérieur les différents modèles en leur affectant des images :

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

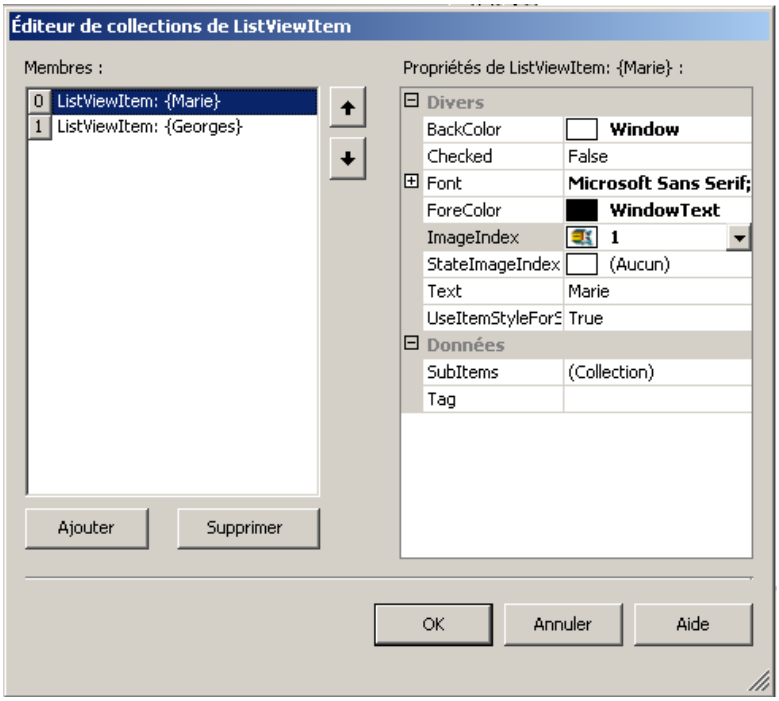
```
Me.TreeView1.ImageList = Me.ImageList1
Me.TreeView1.ImageIndex = 0

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
```

4.2.2.12 ListView

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



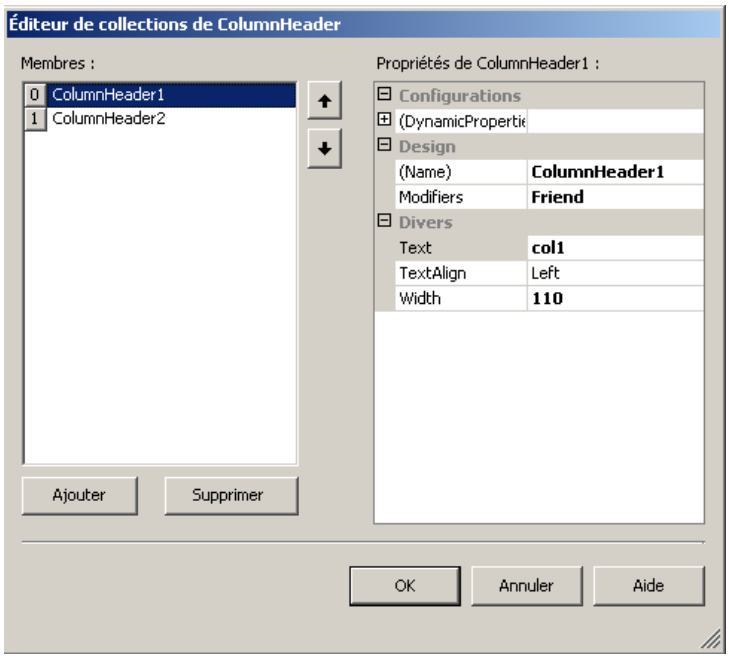
Au même titre que le « « TreeView », le « ListView » possède des assistants afin de définir leur contenu. Pour ouvrir l'assistant permettant de gérer la liste, cliquer sur le bouton à droite de la propriété Items :



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

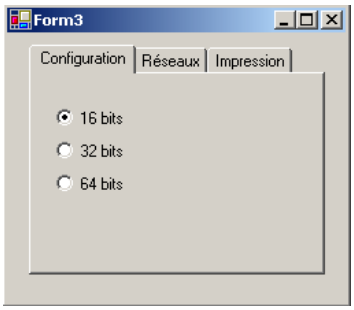
Le code suivant affiche pour chaque caractère son code ascii, le caractère en minuscule et le caractère en majuscule. Il crée également les colonnes :

```
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
```

4.2.2.13 TabControl

Le contrôle « TabControl » permet l'affichage d'onglet contenant chacun plusieurs contrôles. Ce dernier est généralement utilisé pour regrouper logiquement des contrôles 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.

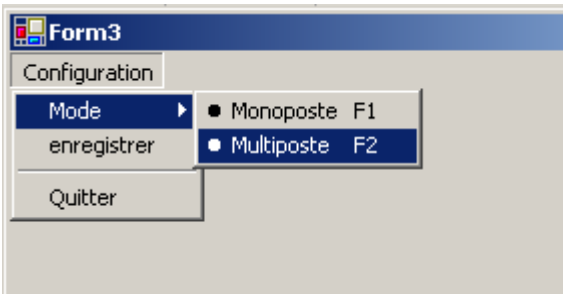
Propriété	Description
Alignment	Définit la position des onglets par rapport aux pages
HotTrack	Modifie l'apparence des onglets lorsque la souris passe dessus

Imagelist	ImageList lié pour les icônes d'onglets
Multiline	Permet l'affichage des onglets sur plusieurs lignes
TabPages	Collection de pages.

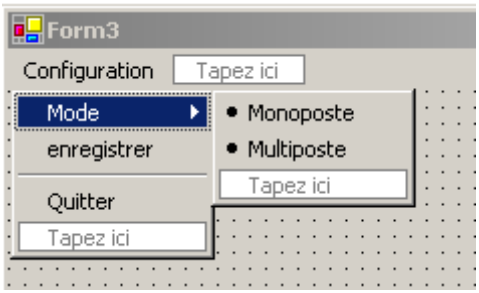
4.2.2.14 Menus

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'assistant 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

4.2.2.15 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	Chaîne 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.

4.2.2.16 Timer

Le contrôle Timer permet de déclencher un événement à intervalles réguliers.

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

Evénement	Description
Tick	Déclenché à chaque intervalle.